

Введение в машинное обучение

Библиотека языка Питон Scikit-Learn

Дьяконов А.Г.

**Московский государственный университет
имени М.В. Ломоносова (Москва, Россия)**

Установка

<http://scikit-learn.org/stable/install.html>

ВХОДИТ ВО МНОГИЕ ДИСТРИБУТИВЫ

```
import sklearn as sk
import numpy as np
import matplotlib.pyplot as plt
```

Что есть

`sklearn.datasets`

генерация / загрузка данных

алгоритмы классификации, регрессии, кластеризации

`model_selection`

**организация экспериментов для выбора модели,
перебор параметров**

`preprocessing`
`feature_extraction`

предобработка / подготовка данных / генерация признаков

Что есть, кроме хороших алгоритмов...

Перемешивание

Раньше

```
rng = np.random.RandomState(0)
permutation = rng.permutation(len(X))
X, y = X[permutation], y[permutation]
```

Сейчас

```
from sklearn.utils import shuffle
X, y = shuffle(X, y)
```

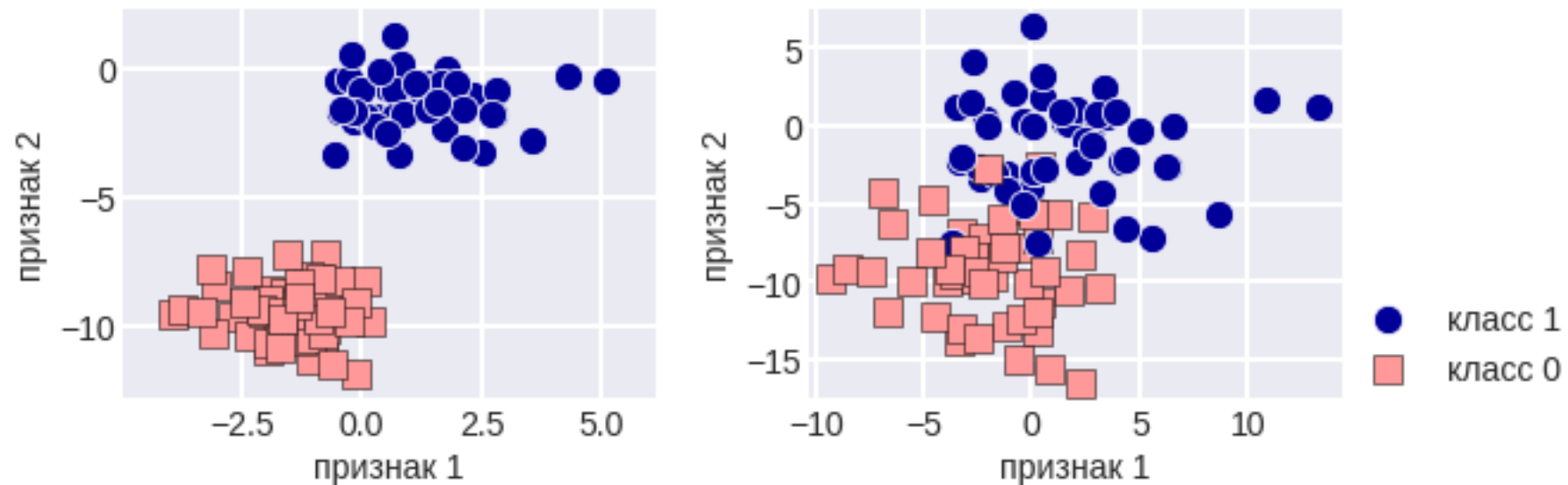
Разбиение на обучение и контроль – одна строчка

```
from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split # с версии 0.20

train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.5,
                                                    random_state=1999)
```

Встроенные датасеты (генераторы)

«Кучки»



```
from sklearn.datasets import make_blobs  
X, y = make_blobs(centers=2, random_state=2)  
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
```

`n_samples`, `n_features` – **размеры**

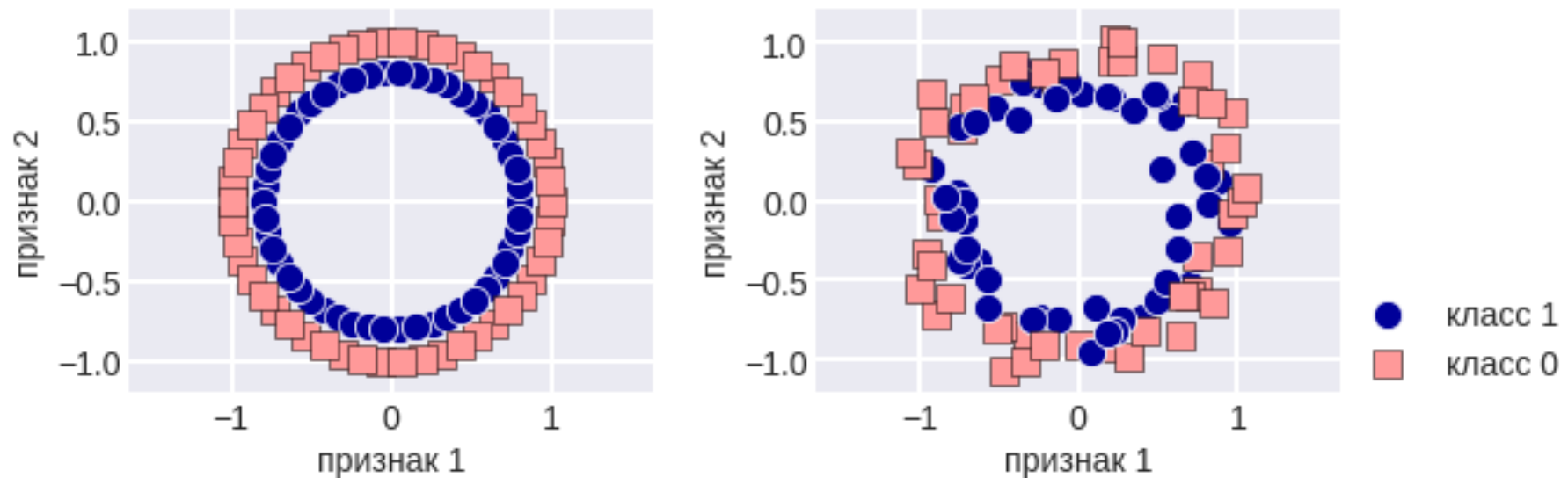
`centers` – **сколько кучек**

`cluster_std` – **дисперсия**

`random_state` – **инициализация генератора**

Встроенные датасеты (генераторы)

«Кольца»



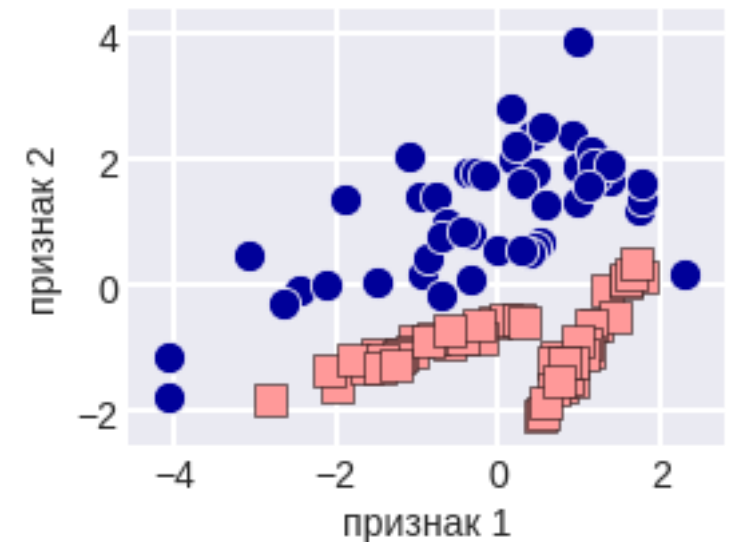
```
from sklearn.datasets import make_circles  
X, y = make_circles(noise=0.1, random_state=1)  
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
```

`n_samples` – размеры
`shuffle` – перемешивание
`noise` – дисперсия

Встроенные датасеты (генераторы)

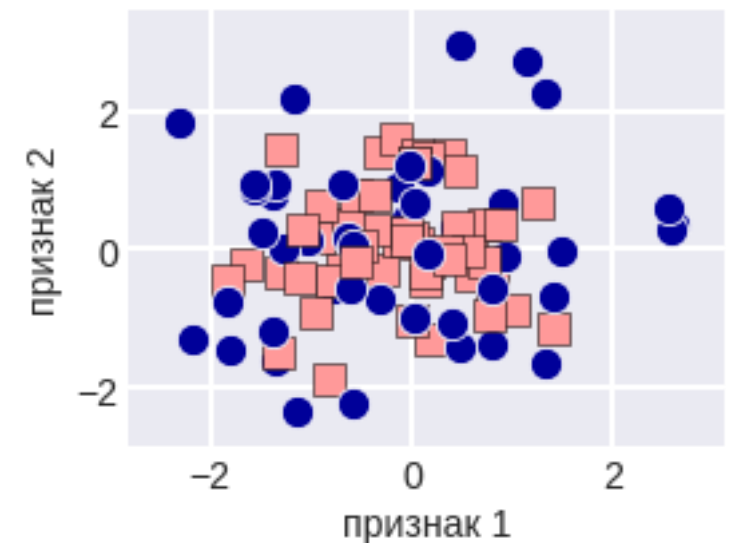
«Классификация»

```
make_classification(n_samples=100,  
                  n_features=2,  
                  n_informative=2,  
                  n_redundant=0,  
                  n_repeated=0,  
                  n_clusters_per_class=2)
```



Из книги Хасты

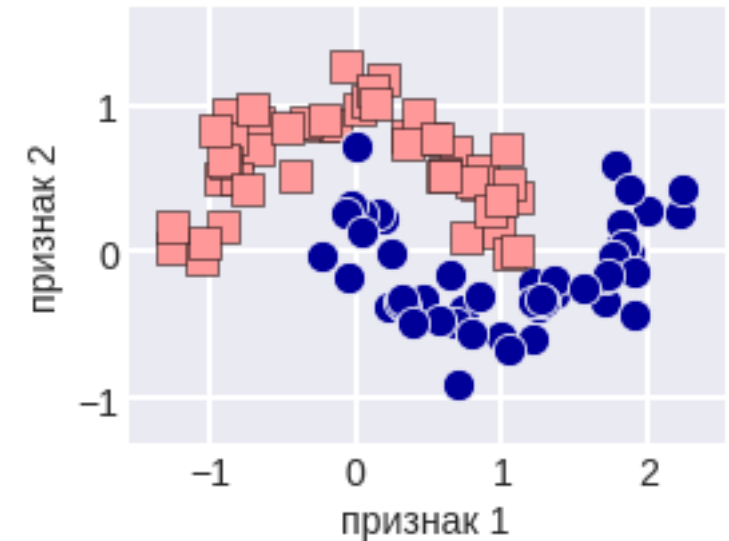
```
make_hastie_10_2(n_samples=100)
```



Встроенные датасеты (генераторы)

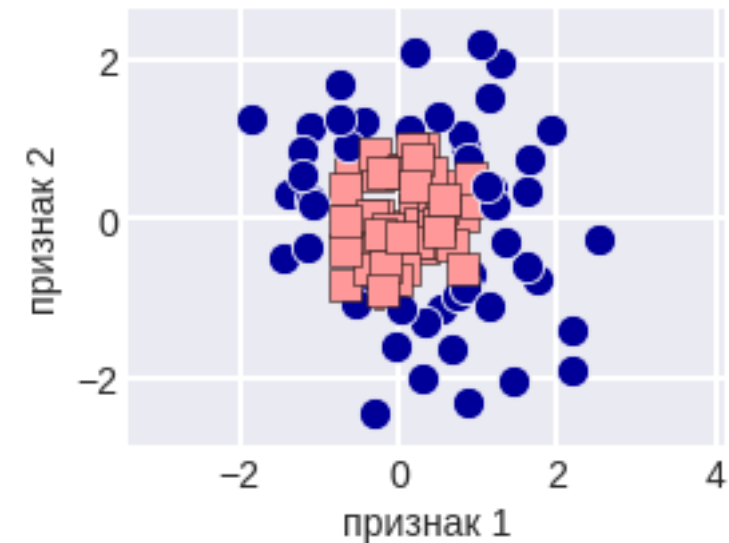
Два месяца

```
make_moons(n_samples=100,  
           noise=0.15,  
           random_state=1)
```



Квантили нормального распределения

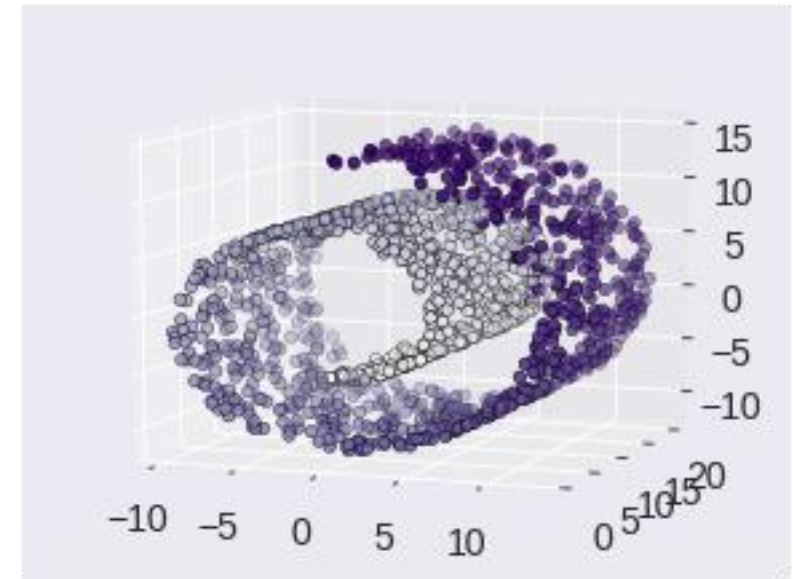
```
make_gaussian_quantiles(n_classes=2,  
                        random_state=1)
```



Встроенные датасеты (генераторы)

```
from sklearn.datasets import make_swiss_roll
n_samples = 1500
noise = 0.05
X, y = make_swiss_roll(n_samples, noise)
import mpl_toolkits.mplot3d.axes3d as p3

fig = plt.figure(figsize=(4, 3))
ax = p3.Axes3D(fig)
ax.view_init(10, -70)
ax.scatter(X[:, 0], X[:, 1], X[:, 2],
           s=20, c=y, cmap='Purples',
           edgecolor='k', linewidth=0.5)
```



Встроенные датасеты (генераторы)

Регрессия

```
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=100, n_features=1,
                      n_informative=1, n_targets=1,
                      bias=0.0, effective_rank=None,
                      tail_strength=0.5, noise=7.0,
                      shuffle=True, coef=False, random_state=1)
```



разные регрессии:

```
datasets.make_friedman1
datasets.make_friedman2
datasets.make_friedman3
```

Встроенные датасеты (генераторы)

`make_multilabel_classification`

– **многоклассовая задача с пересекающимися классами**

`make_spd_matrix`

`make_sparse_spd_matrix`

– **случайная симметричная положительно определённая матрица**

`make_low_rank_matrix`

– **матрица малого ранга**

`make_sparse_uncorrelated`

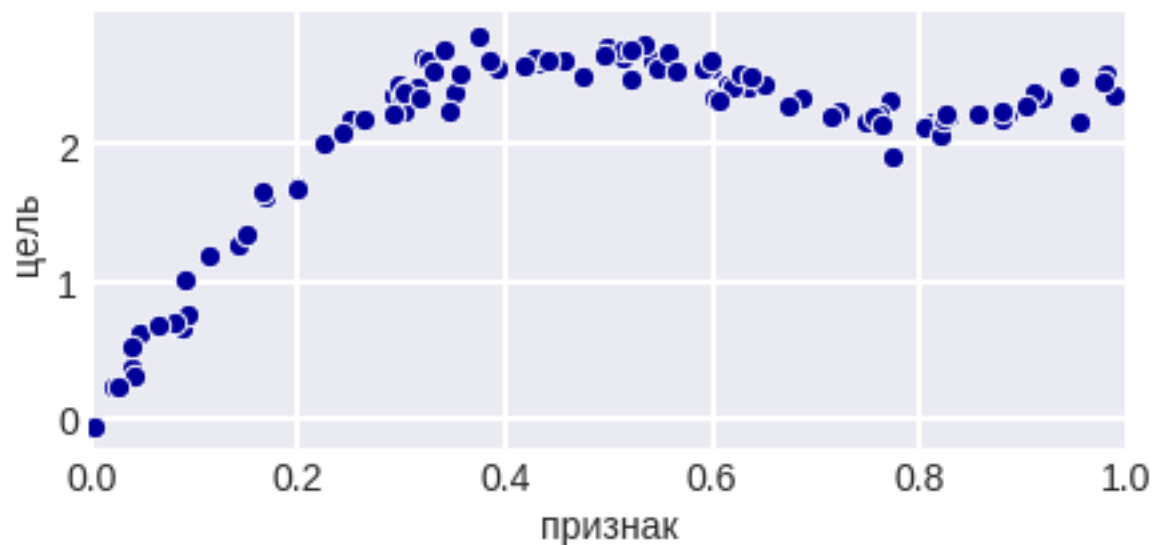
– **регрессия**

$X \sim N(0, 1)$

$y(X) = X[:, 0] + 2 * X[:, 1] - 2 * X[:, 2] - 1.5 * X[:, 3]$

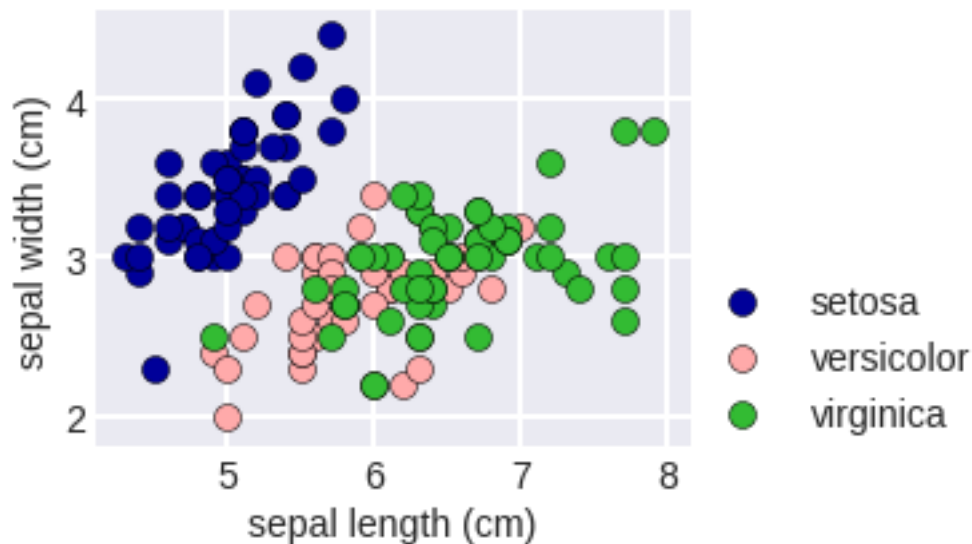
Ручная генерация данных

```
n_samples = 100  
np.random.seed(10)  
X = np.random.rand(n_samples)  
Y = np.sin(5 * X) + 5 * np.log1p(X) + 0.1 * np.random.randn(n_samples)
```



Встроенные датасеты (загрузчики)

```
from sklearn import datasets
iris = datasets.load_iris()
X_iris, y_iris = iris.data, iris.target
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Встроенные датасеты (загрузчики)

```
from sklearn.datasets import fetch_olivetti_faces
```

```
faces = fetch_olivetti_faces()
```

```
print (faces.keys())
```

```
print (faces.images.shape)
```

```
print (faces.data.shape)
```

```
dict_keys(['data', 'images',  
          'target', 'DESCR'])
```

```
(400, 64, 64)
```

```
(400, 4096)
```

```
print_faces(faces.images,  
            faces.target, 20)
```

```
def print_faces(images, target, top_n):
```

```
    fig = plt.figure(figsize=(5, 5))
```

```
    fig.subplots_adjust(bottom=0, top=1, hspace=0.05, wspace=0.05)
```

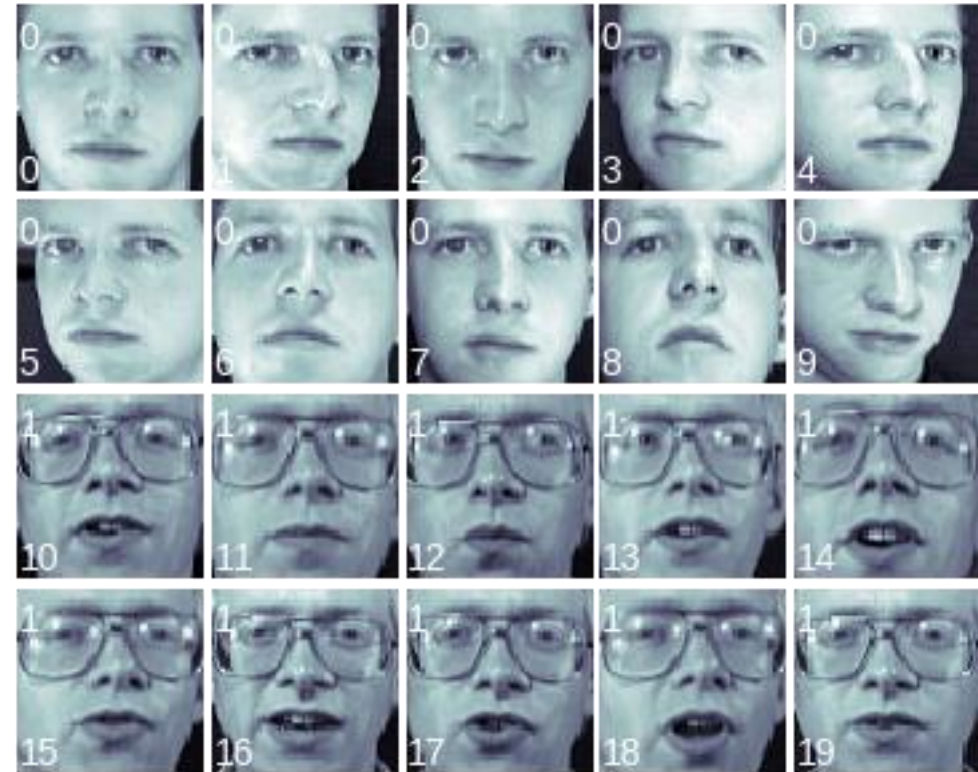
```
    for i in range(top_n):
```

```
        p = fig.add_subplot(5, 5, i + 1, xticks=[], yticks=[])
```

```
        p.imshow(images[i], cmap=plt.cm.bone)
```

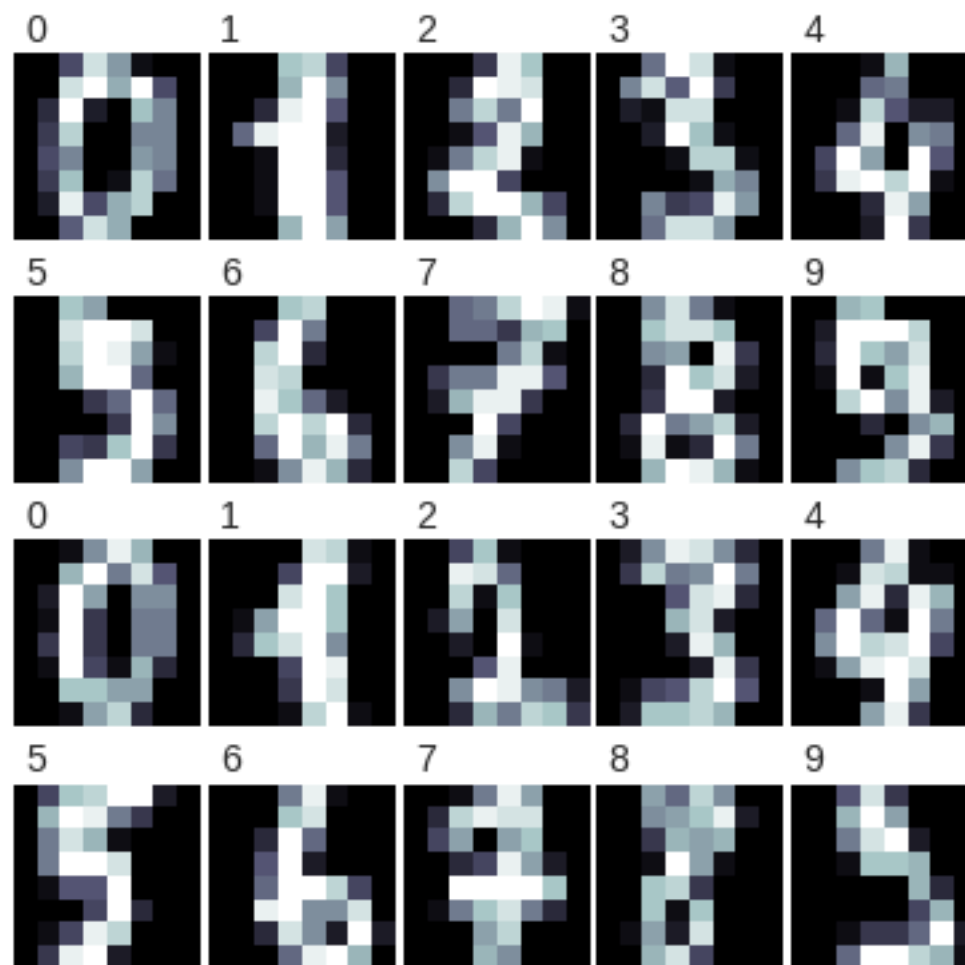
```
        p.text(0, 14, str(target[i]), color='white')
```

```
        p.text(0, 60, str(i), color='white')
```



Встроенные датасеты (загрузчики)

```
from sklearn.datasets import load_digits
digits = load_digits()
X_digits, y_digits = digits.data, digits.target
```



Встроенные датасеты (загрузчики)

```
from sklearn.datasets import fetch_20newsgroups
news = fetch_20newsgroups(subset='all')
print (type(news.data), type(news.target), type(news.target_names))
<class 'list'> <class 'numpy.ndarray'> <class 'list'>
```

```
print (news.target_names)
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', ...]
```

```
print (news.data[5])
From: tell@cs.unc.edu (Stephen Tell)
Subject: Re: subliminal message flashing on TV
Organization: The University of North Carolina at Chapel Hill
Lines: 25
NNTP-Posting-Host: rukbat.cs.unc.edu
```

In article <7480237@hpfcso.FC.HP.COM> myers@hpfcso.FC.HP.COM (Bob Myers) writes:

```
>> Hi. I was doing research on subliminal suggestion for a psychology
>> paper, and I read that one researcher flashed hidden messages on the
>> TV screen at 1/200ths of a second. Is that possible?
```


Интерфейсы

У Scikit-learn единый способ использования всех методов.

Для всех моделей (estimator object**) доступны следующие методы.**

`model.fit()` – настройка на данные (обучение)

`model.fit(X, y)` – для обучения с учителем (supervised learning)

`model.fit(X)` – для обучение без учителя (unsupervised learning)

<code>model.predict</code>	<code>model.transform</code>
Classification	Preprocessing
Regression	Dimensionality Reduction
Clustering	Feature Extraction
	Feature selection

Для обучения с учителем:

`model.predict(X_test)` – предсказать значения целевой переменной

`model.predict_proba()` – выдать «степень уверенности» в ответе (вероятность) – для некоторых моделей

`model.decision_function()` – решающая функция – для некоторых моделей

`model.score()` – в большинстве моделей встроены методы оценки их качества работы

`model.transform()` – для отбора признаков (feature selection) «сжимает» обучающую матрицу. Для регрессионных моделей и классификаторов (linear, RF и т.п.) выделяет наиболее информативные признаки

Для обучения без учителя

`model.transform()` – преобразует данные

`model.fit_transform()` – не во всех моделях – эффективная настройка и трансформация обучения

`model.predict()` – для кластеризации (не во всех моделях) – получить метки кластеров

`model.predict_proba()` – Gaussian mixture models (GMMs) получают вероятности принадлежности к компонентам для каждой точки

`model.score()` – некоторые модели (KDE, GMMs) получают правдоподобие (насколько данные соответствуют модели)

Совет – для 2D-визуализации

Напишите подобную функцию... (см. дальше результаты работы)

```
def plot_2d_separator(classifier, X, fill=False, line=True, ax=None, eps=None):
    if eps is None:
        eps = 1.0 #X.std() / 2.
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy = np.linspace(y_min, y_max, 100)

    X1, X2 = np.meshgrid(xx, yy)
    X_grid = np.c_[X1.ravel(), X2.ravel()]
    try:
        decision_values = classifier.decision_function(X_grid)
        levels = [0]
        fill_levels = [decision_values.min(), 0, decision_values.max()]
    except AttributeError:
        # no decision_function
        decision_values = classifier.predict_proba(X_grid)[:, 1]
        levels = [.5]
        fill_levels = [0, .5, 1]

    if ax is None:
        ax = plt.gca()
    if fill:
        ax.contourf(X1, X2, decision_values.reshape(X1.shape),
                    levels=fill_levels, colors=['cyan', 'pink'])
    if line:
        ax.contour(X1, X2, decision_values.reshape(X1.shape), levels=levels,
                   colors="black")
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())
```

Работа с моделями (1)

данные

```
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2, random_state=0)
```

разбивка: обучение - контроль

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=0)
```

обучение модели и предсказание

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
prediction = classifier.predict(X_test)
```

качество

```
print (np.mean(prediction == y_test))           0.8
print (classifier.score(X_test, y_test)) # более удобная 0.8
print (classifier.score(X_train, y_train))       0.93
```

Работа с моделями (2)

визуализация

```
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], c=y, s=70)
```

матрица несоответствий

```
from sklearn.metrics import confusion_matrix
print (confusion_matrix(y_test, prediction))
```

```
[[12  1]
 [ 4  8]]
```

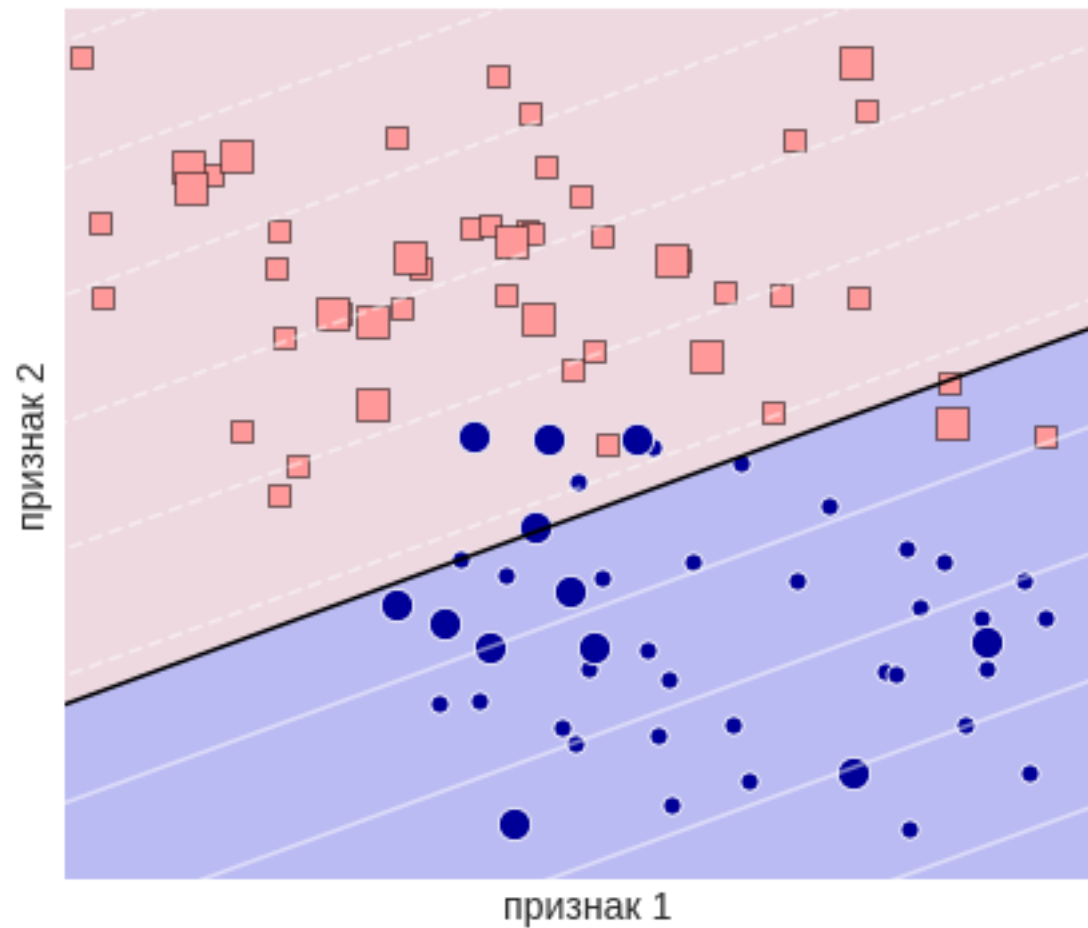
отчёт о точности

```
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction))
```

	precision	recall	f1-score	support
0	0.75	0.92	0.83	13
1	0.89	0.67	0.76	12
micro avg	0.80	0.80	0.80	25
macro avg	0.82	0.79	0.79	25
weighted avg	0.82	0.80	0.80	25

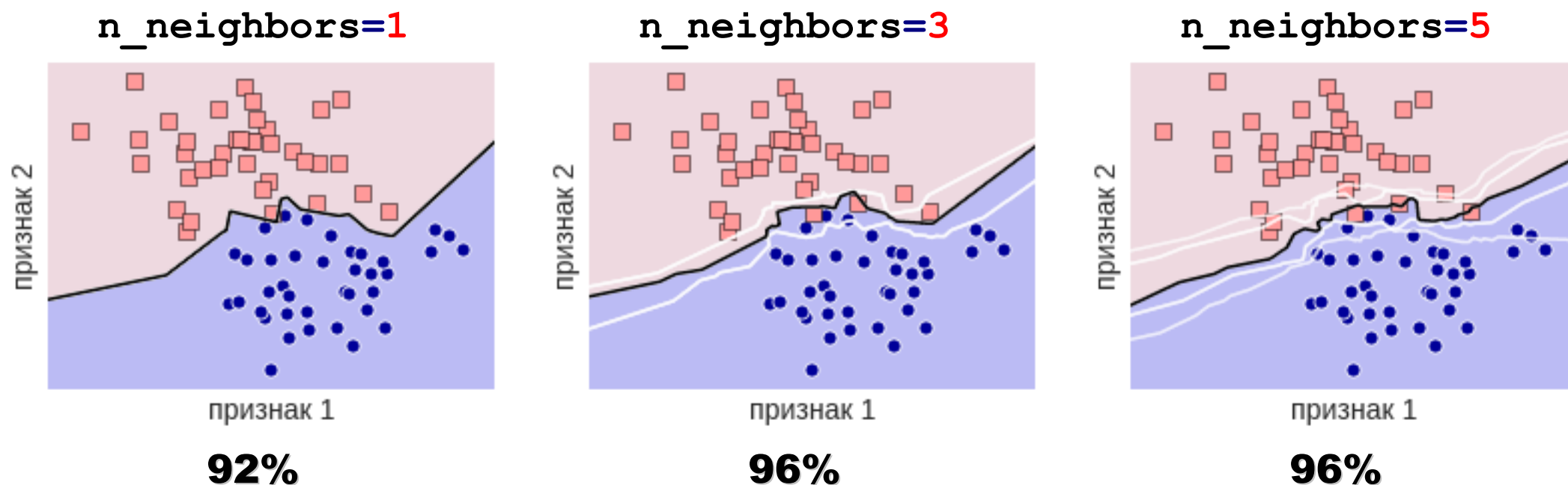
Работа с моделями

Что получилось (логистическая регрессия)



«Метод ближайшего соседа»

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=1)  
knn.fit(X_train, y_train)
```



Замечание: здесь разный масштаб по осям

«Метод ближайшего соседа»

`n_neighbors` – число соседей

`weights` – веса («uniform», «distance», функция)

`algorithm` – алгоритм для эффективного нахождения соседей
(«auto», «ball_tree», «kd_tree», «brute»)

`leaf_size` – для `BallTree` / `KDTree`

`p` – параметр для метрики Минковского

`metric` – метрика («minkowski»)

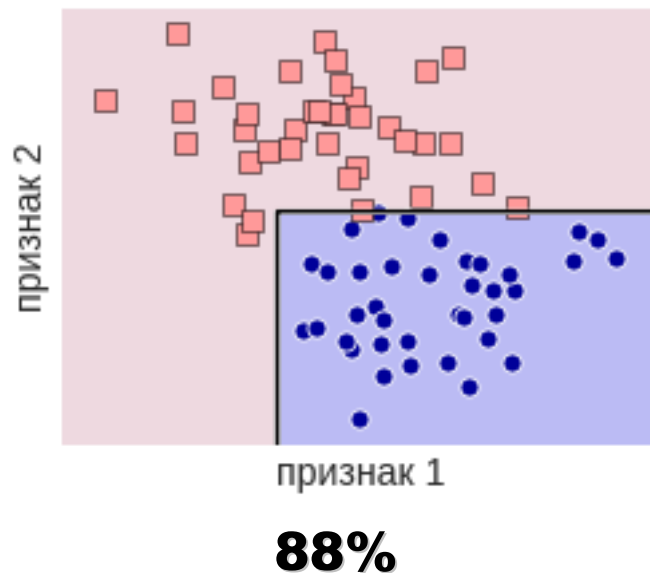
`metric_params` – параметры для метрики

`n_jobs` – число процессов для нахождения соседей

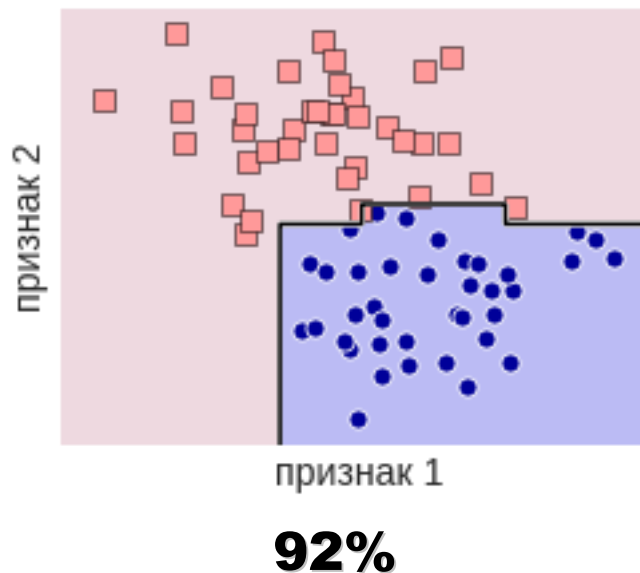
«Решающее дерево»

```
from sklearn.tree import DecisionTreeClassifier  
tree = DecisionTreeClassifier(max_depth=10)  
tree.fit(X_train, y_train)
```

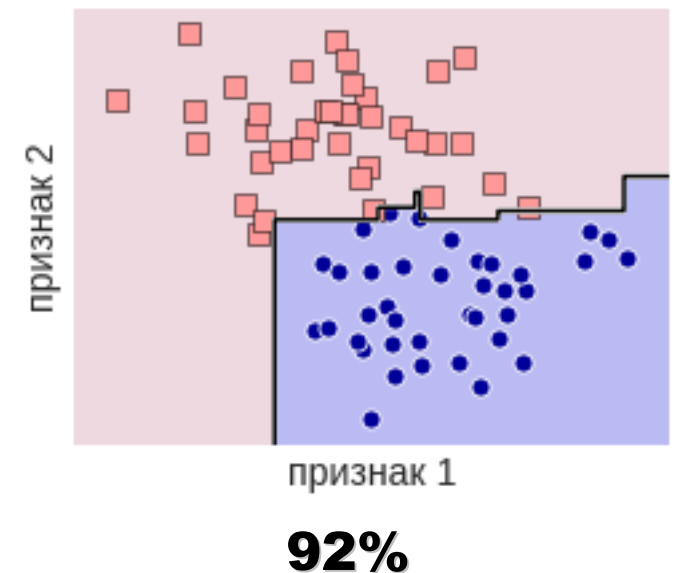
max_depth=10



splitter='random'



splitter='random'
max_features=1



«Решающее дерево»

`criterion` – критерий расщепления «gini» / «entropy»

`splitter` – разбиение «best» / «random»

`max_depth` – допустимая глубина

`min_samples_split` – минимальная выборка для разбиения

`min_samples_leaf` – минимальная мощность листа

`min_weight_fraction_leaf` – аналогично с весом

`max_features` – число признаков, которые смотрим для нахождения разбиения

`random_state` – инициализация генератора случайных чисел

`max_leaf_nodes` – допустимое число листьев

`min_impurity_decrease` – порог «зашумлённости» для разбиения

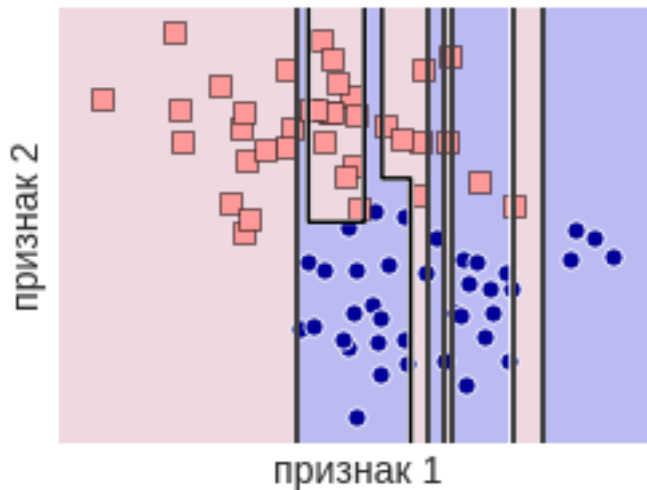
`min_impurity_split` – порог «зашумлённости» для останова

`class_weight` – веса классов («balanced» или словарь, список словарей)

«Случайный лес»

```
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier(n_estimators=1)  
rf.fit(X_train, y_train)
```

n_estimators=1



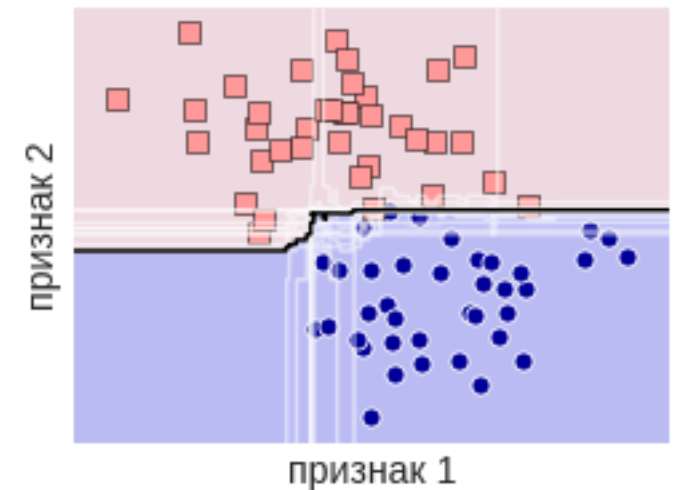
86%

n_estimators=5



84%

n_estimators=100



88%

«Случайный лес»

`n_estimators` – **число деревьев**

`criterion`

`max_depth`

`min_samples_split`

`min_samples_leaf`

`max_features`

`max_leaf_nodes`

`min_impurity_decrease`

`min_impurity_split`

`bootstrap` – **делать ли бутстреп**

`oob_score` – **вычислять ли ООВ-ошибку**

`n_jobs`

`random_state`

`verbose` – **контроль процесса**

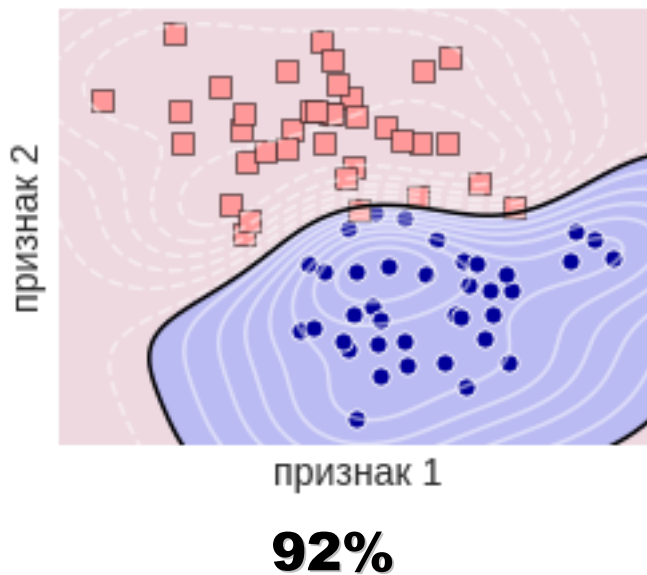
`warm_start` – **использовать ли существующий лес, чтобы его
дополнить или учить заново**

`class_weight`

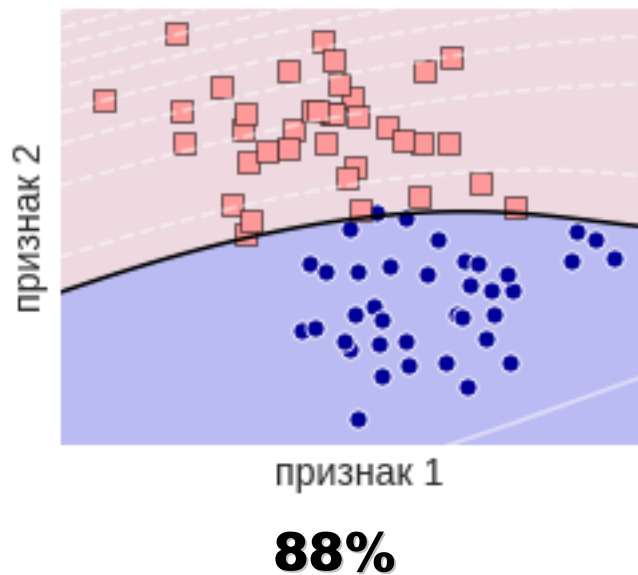
«Метод опорных векторов»

```
from sklearn.svm import SVC  
svm = SVC(kernel='rbf')  
svm.fit(X_train, y_train)
```

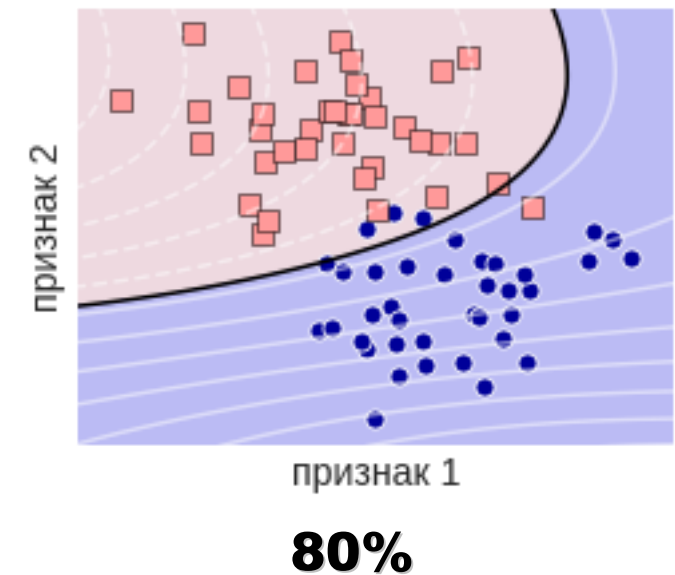
kernel='rbf'



kernel='poly'



kernel='sigmoid'
gamma=0.05



«Метод опорных векторов»

C – параметр регуляризации

kernel – ядро («linear», «poly», «rbf», «sigmoid», «precomputed», функция)

degree – степень полинома для poly

gamma – коэффициент для «rbf», «poly», «sigmoid»

coef0 – коэффициент для «poly», «sigmoid»

shrinking – «shrinking heuristic»

probability – вычислять ли вероятность

tol – порог для остановки

cache_size

class_weight

verbose

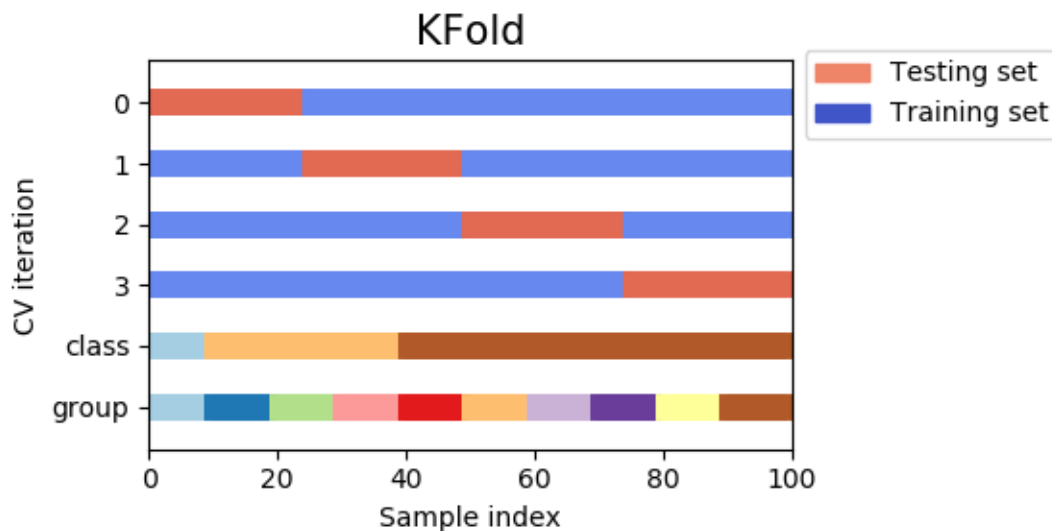
max_iter – ограничение на число итераций

decision_function_shape – ovr (one-vs-rest), ovo (one-vs-one)

Разбиения выборок: `model_selection` (ex: `cross_validation`)

KFold: разбиение на фолды

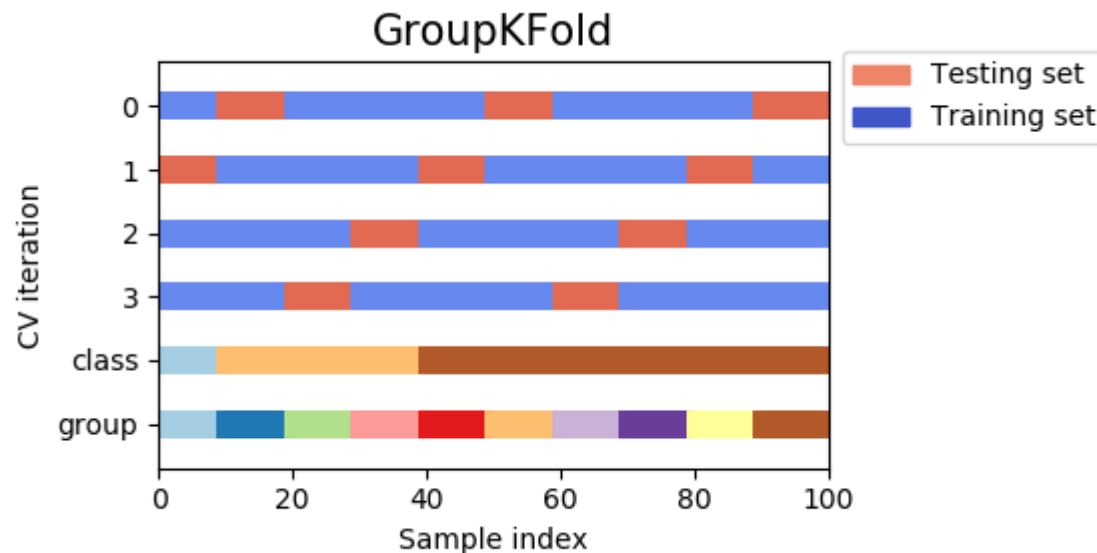
```
sklearn.model_selection.KFold(n_splits='warn',  
                               shuffle=False,  
                               random_state=None)
```



```
from sklearn.model_selection import KFold  
kf = KFold(n_splits=2)  
kf.get_n_splits(X)  
print(kf)  
for train_index, test_index in kf.split(X):  
    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]
```

GroupKFold: разбиение на фолды без разбиения групп

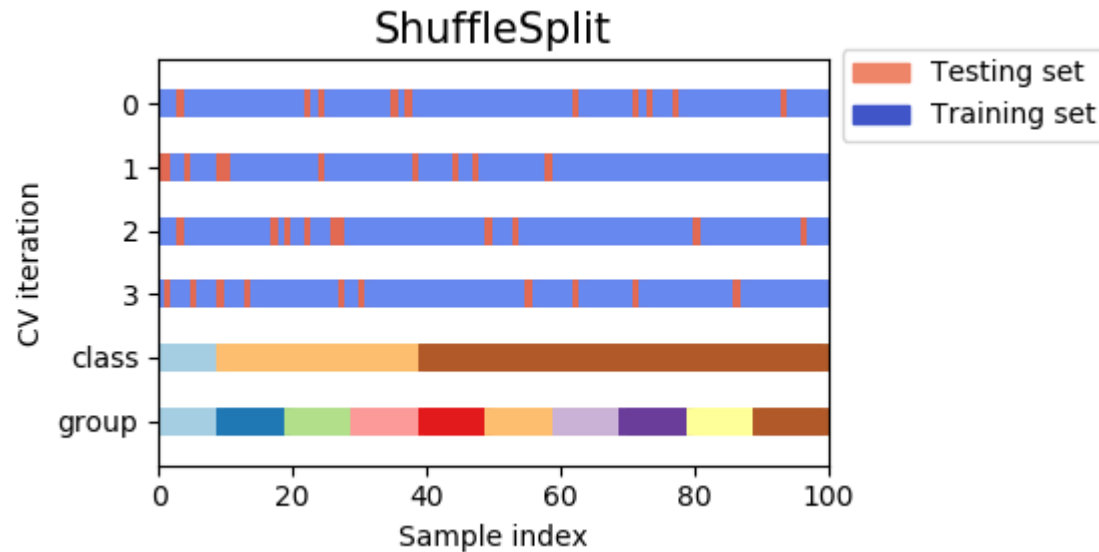
```
sklearn.model_selection.GroupKFold(n_splits='warn')
```



```
from sklearn.model_selection import GroupKFold
group_kfold = GroupKFold(n_splits=2)
group_kfold.get_n_splits(X, y, groups)
print(group_kfold)
i=0
for train_index, test_index in group_kfold.split(X, y, groups):
    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]
    print(X_train, X_test, y_train, y_test)
```

ShuffleSplit: случайные разбиения

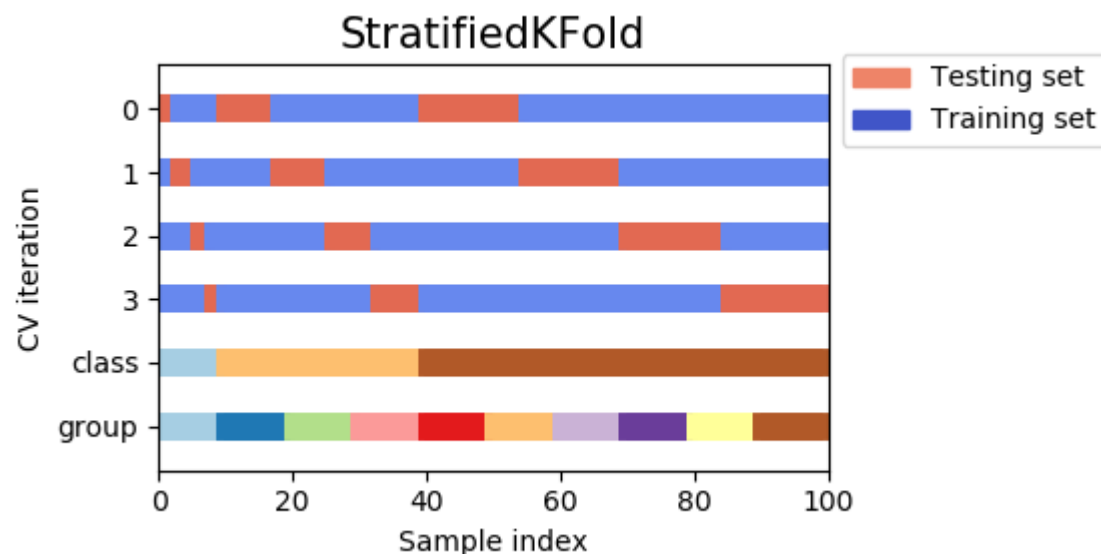
```
sklearn.model_selection.ShuffleSplit(n_splits=10, test_size='default',  
                                     train_size=None, random_state=None)
```



```
from sklearn.model_selection import ShuffleSplit  
rs = ShuffleSplit(n_splits=5, test_size=.25, random_state=0)  
rs.get_n_splits(X)  
print(rs)  
ShuffleSplit(n_splits=5, random_state=0, test_size=0.25, train_size=None)  
for train_index, test_index in rs.split(X):  
    print("TRAIN:", train_index, "TEST:", test_index)
```

StratifiedKFold: сохраняет пропорцию классов

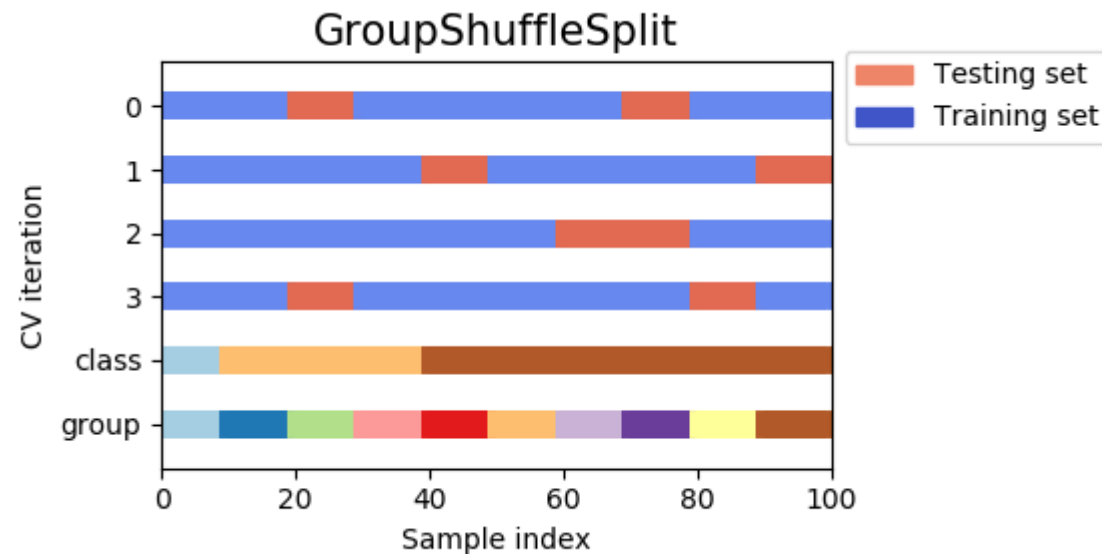
```
sklearn.model_selection.StratifiedKFold(n_splits='warn', shuffle=False,  
                                         random_state=None)
```



```
from sklearn.model_selection import StratifiedKFold  
skf = StratifiedKFold(n_splits=2)  
skf.get_n_splits(X, y)  
print(skf)  
for train_index, test_index in skf.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]
```

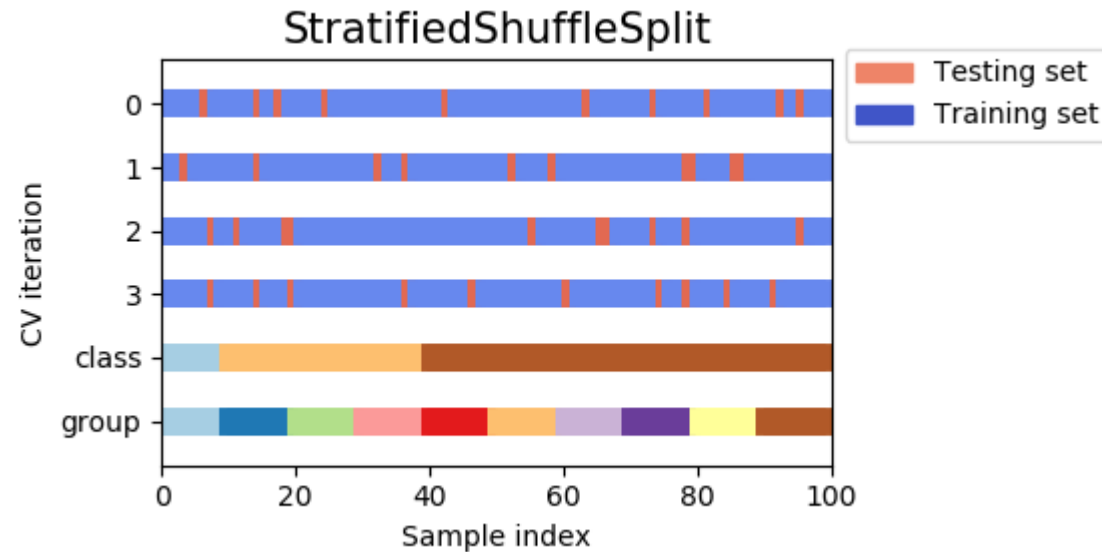
GroupShuffleSplit: случайные разбиения без разбиения групп

```
sklearn.model_selection.GroupShuffleSplit(n_splits=5, test_size='default',  
                                          train_size=None, random_state=None)
```



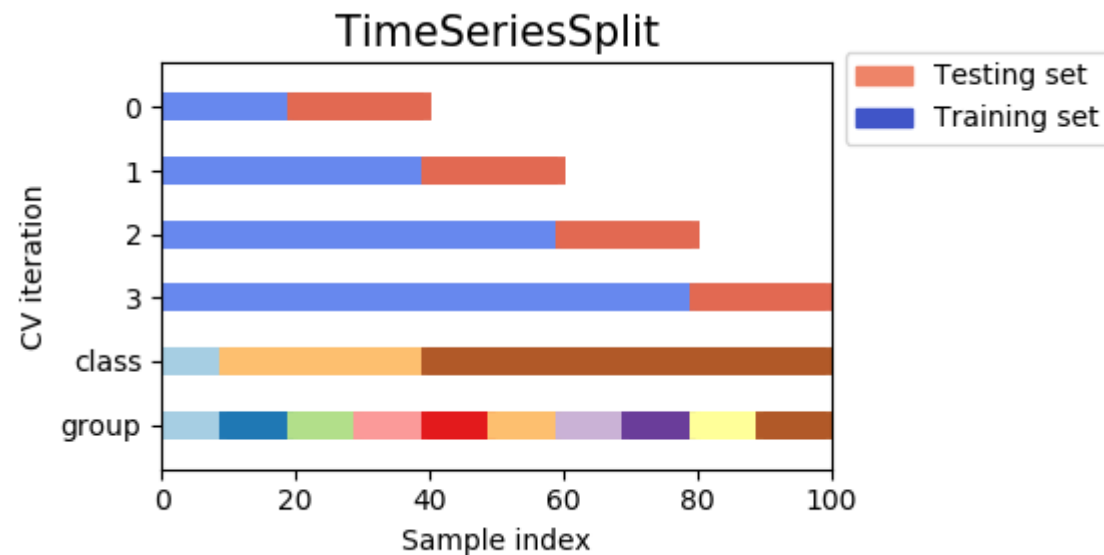
GroupShuffleSplit: случайные разбиения сохраняя пропорции классов

```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=10,  
test_size='default', train_size=None, random_state=None)
```



TimeSeriesSplit: разбиения временных рядов

`sklearn.model_selection.TimeSeriesSplit(n_splits='warn',
max_train_size=None)`



LeaveOneOut: Контроль по одному

`sklearn.model_selection.LeaveOneOut`

LeaveOneGroupOut: Контроль по одной группе

```
from sklearn.model_selection import LeaveOneGroupOut
logo = LeaveOneGroupOut()
logo.get_n_splits(X, y, groups)
logo.get_n_splits(groups=groups)
print(logo)
for train_index, test_index in logo.split(X, y, groups):
    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]
    print(X_train, X_test, y_train, y_test)
```


Какие бывают разбиения

```
from sklearn.model_selection import LeavePOut
# всевозможные тройки
cv = LeavePOut(3)
```

```
for train, test in cv.split(X):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[test]))
```

```
from sklearn.model_selection import ShuffleSplit
# n случайных разбиений
cv = ShuffleSplit(n_splits=3, test_size=0.1,
train_size=None, random_state=None)
```

```
for train, test in cv.split(X):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[test]))
```

```
from sklearn.model_selection import PredefinedSplit
# заданные разбиения
g = np.array([1, 2, 1, 2, 1, 2, 1, 2, 1, 4, 4, 4])
cv = PredefinedSplit(g)
```

```
for train, test in cv.split(X, y, g):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[test]))
    print('метки групп теста = ' + str(g[test]))
```

```
индексы теста = [0 1 2]
классы теста = [1 1 2]
индексы теста = [0 1 3]
классы теста = [1 1 2]
индексы теста = [0 1 4]
...
```

```
индексы теста = [11  0]
классы теста = [3 1]
индексы теста = [2 4]
классы теста = [2 2]
индексы теста = [0 1]
классы теста = [1 1]
```

```
индексы теста = [0 2 4 6 8]
классы теста = [1 2 2 3 3]
метки групп теста = [1 1 1 1 1]
индексы теста = [1 3 5 7]
классы теста = [1 2 2 3]
метки групп теста = [2 2 2 2]
индексы теста = [ 9 10 11]
классы теста = [3 3 3]
метки групп теста = [4 4 4]
```

Ещё в `sklearn.model_selection`

`train_test_split` – от матрицы (пример был)

`cross_val_score` – оценка с помощью CV (см. ниже)

`cross_val_predict` – формирование cv-мета-признаков

Оценка модели (`cross_val_score`)

```
from sklearn.model_selection import cross_val_score      array([0.8, 0.8 ,
from sklearn.model_selection import ShuffleSplit          1.  ])
from sklearn.linear_model import LogisticRegression

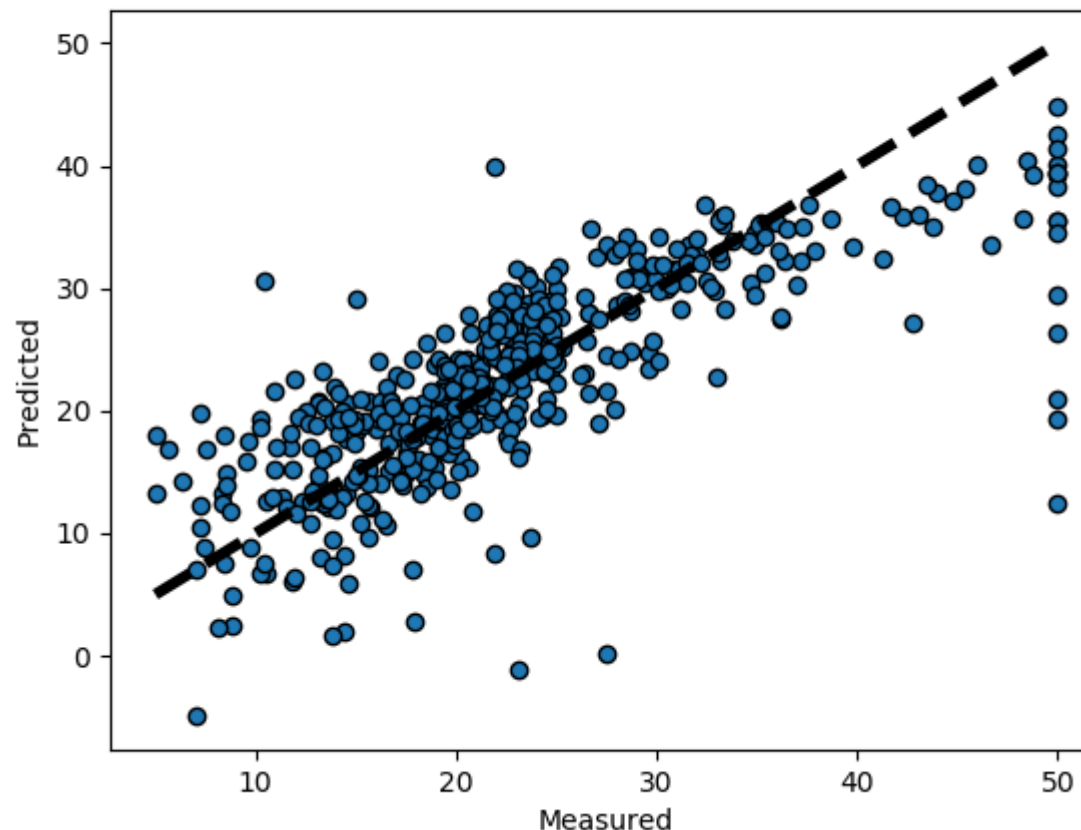
logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                 train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

У этих функций много параметров...

Они (функции) «понимают» друг друга

Пока не указываем скорер – используется встроенный (в модель)

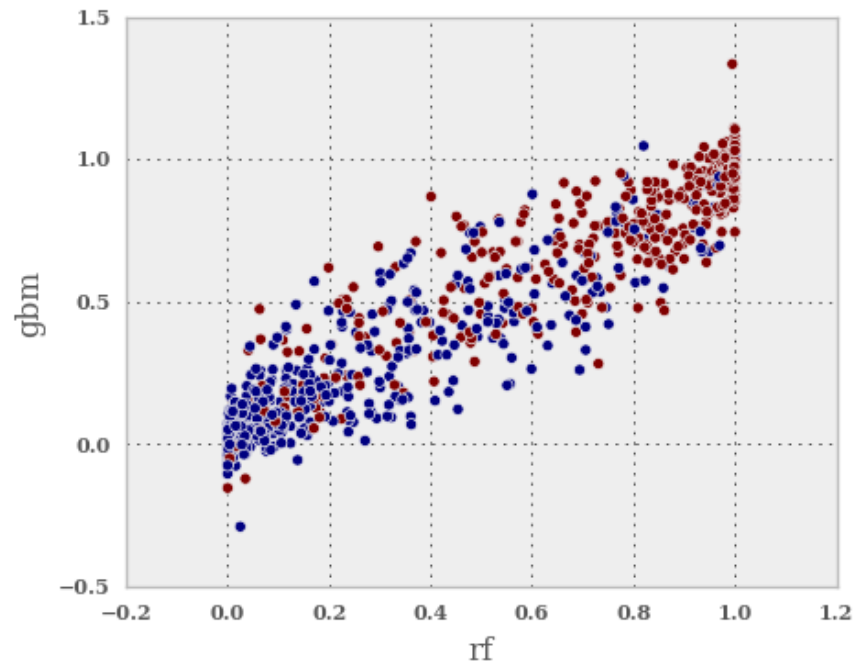
Ответы алгоритма: `model_selection.cross_val_predict`



Для визуальной оценки работы алгоритма

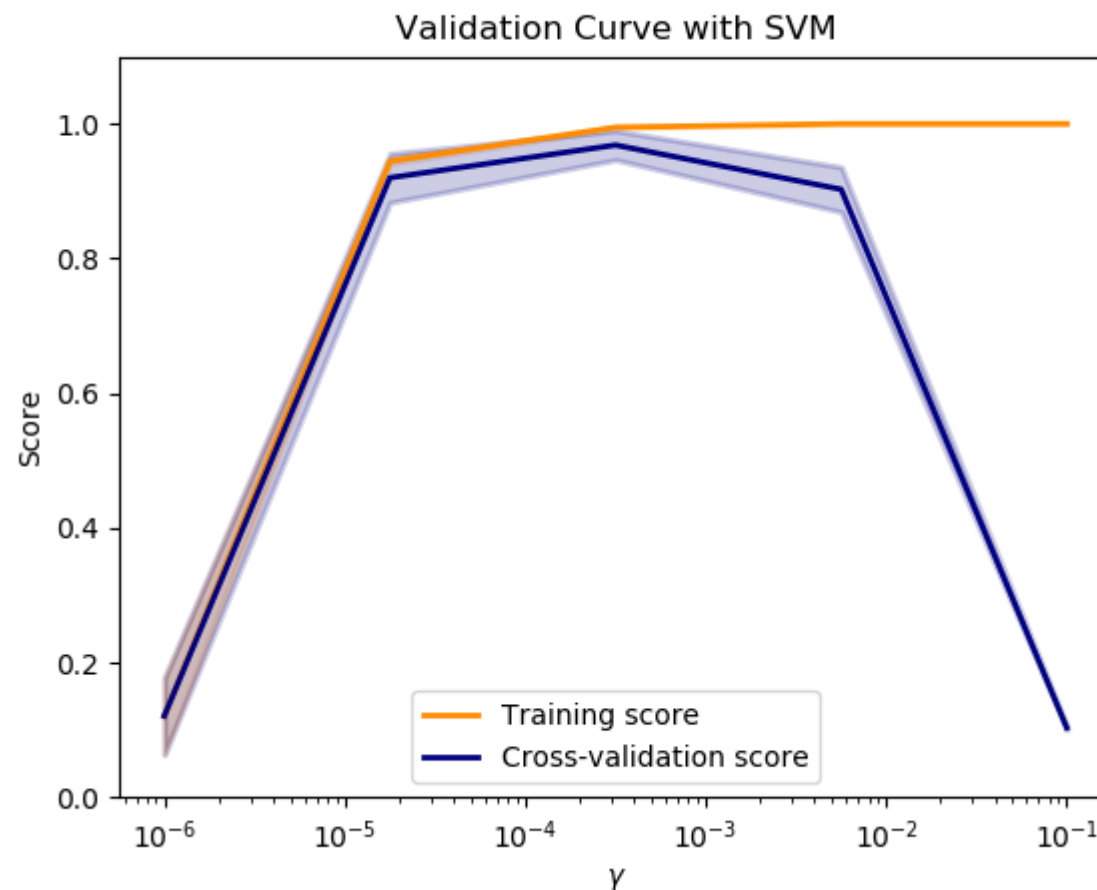
```
from sklearn.model_selection import cross_val_predict  
predicted = cross_val_predict(model, X, y, cv=10)  
plt.scatter(y, predicted)
```

Формирование метапризнаков: `model_selection.cross_val_predict`



```
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
cv = KFold(n_splits=10, shuffle=True, random_state=1)
# ответы rf на скользящем контроле
a_rf = cross_val_predict(rf, X, y, cv=cv)
# ответы gbm на скользящем контроле
a_gbm = cross_val_predict(gbm, X, y, cv=cv)
plt.scatter(a_rf, a_gbm, c=y)
plt.xlabel('rf')
plt.ylabel('gbm')
```

качество при варьировании параметра: `model_selection.validation_curve`



качество при варьировании параметра:`model_selection.validation_curve``estimator` – модель`X, y` – данные`param_name` – какой параметр варьировать`param_range` – какие значения перебирать`groups` – для организации CV`cv` – CV-генератор или итерируемый объект`scoring` – вычисление качества (строка, список, функция, `None` – по умолчанию)`n_jobs``pre_dispatch` – число заданий во время распараллеливания`verbose``error_score` – что делать при ошибке

«raise» / число

Пример: качество при варьировании параметра, validation_curve

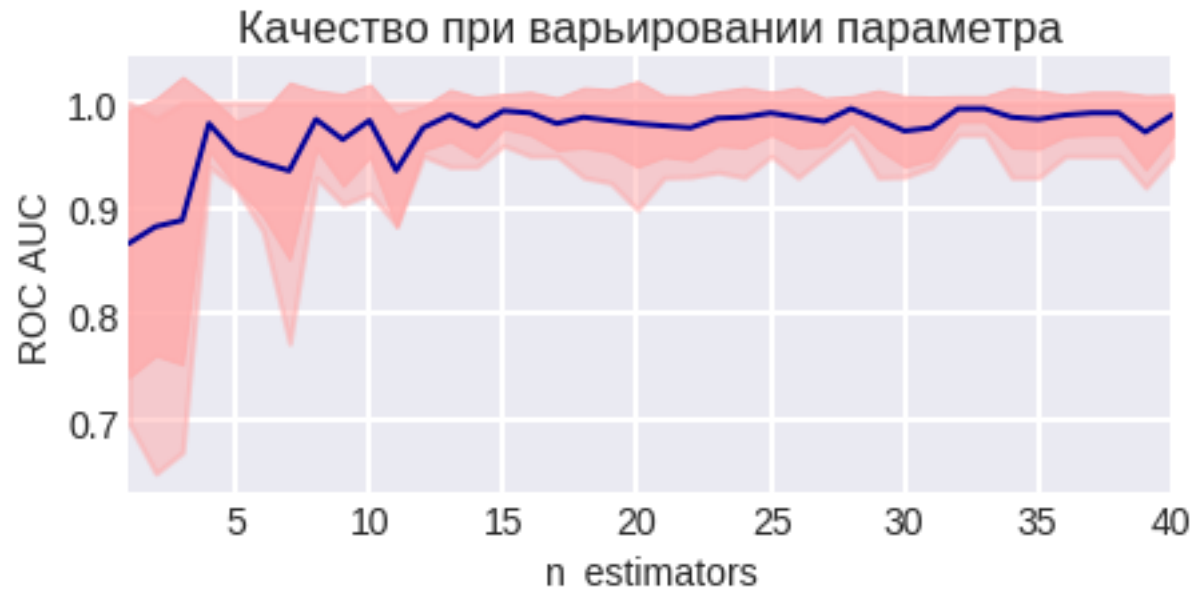
```
from sklearn.model_selection import validation_curve
from sklearn.model_selection import KFold # cross_validation
from time import time

tm = time()
# cv-контроль
cv = KFold(n_splits=5, shuffle=True, random_state=2)
# модель
rf = RandomForestClassifier() # можно прописать параметры
# параметр
param_name = "max_features"
# его значения
pars = np.linspace(5, 100, 20).astype(int).tolist()
# сделать тест
train_errors, test_errors = validation_curve(rf, X, y,
                                             param_name=param_name, param_range=pars,
                                             cv=cv.split(X), scoring='roc_auc', n_jobs=-1)
print ('Время = ' + str(time() - tm))
```

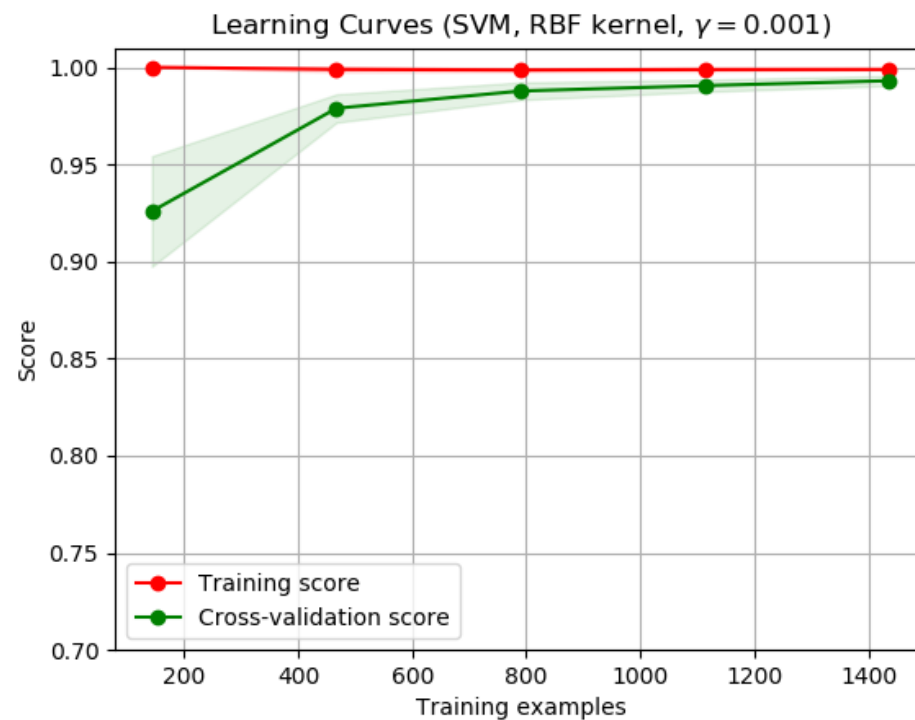
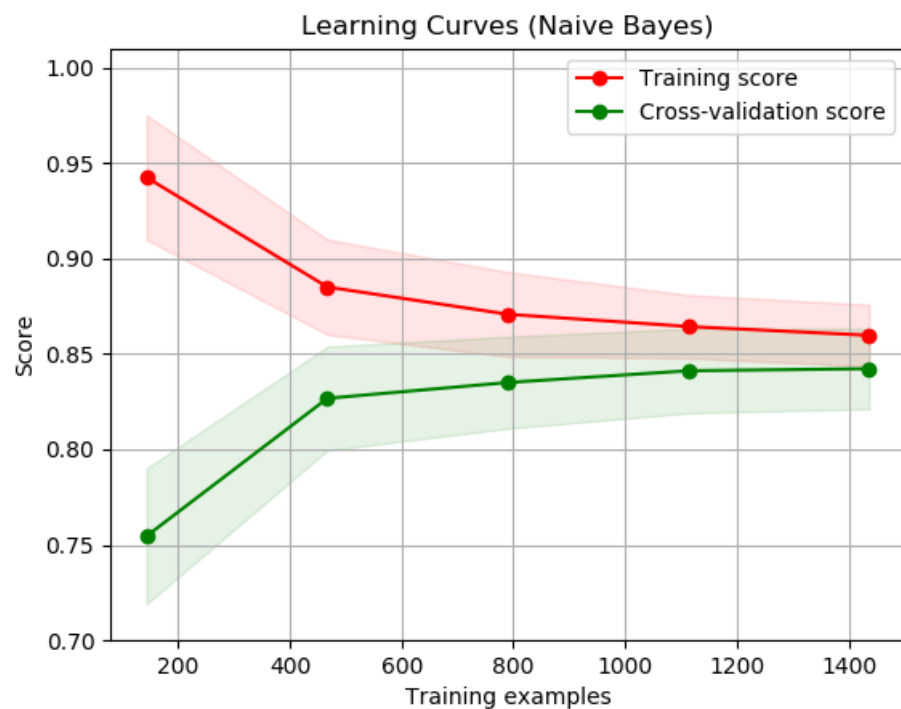
Пример: качество при варьировании параметра, `validation_curve`

```
plt.plot(pars, test_errors.mean(axis=1), label="cv", lw=2)
plt.fill_between(pars, test_errors.min(axis=1), test_errors.max(axis=1),
                alpha=0.2, color="darkorange", lw=2)
plt.fill_between(pars, test_errors.mean(axis=1) - test_errors.std(axis=1),
                test_errors.mean(axis=1) + test_errors.std(axis=1),
                alpha=0.1, color="darkblue", lw=1)

plt.legend(loc="best")
plt.ylabel('ROC AUC')
plt.xlabel(param_name)
plt.title(u'Качество при варьировании параметра')
```



Кривая обучения: `model_selection.learning_curve`



Перебор параметров: `model_selection.GridSearchCV`

```
from sklearn.model_selection import GridSearchCV
parameters = {'kernel': ('linear', 'rbf'), 'C': [1, 10]}
svc = svm.SVC(gamma="scale")
clf = GridSearchCV(svc, parameters, cv=5)
clf.fit(iris.data, iris.target)
```

Результаты перебора в

```
clf.cv_results_
```

Есть также случайный поиск

`model_selection.RandomizedSearchCV`
(тут есть «число итераций»)

Перебор параметров: `model_selection.GridSearchCV`

`estimator` – модель

`param_grid` – словарь или список словарей
(перебираемые параметры)

`scoring` – вычисление качества (строка, список, функция,
`None` – по умолчанию)

`n_jobs`

`pre_dispatch` – число заданий во время распараллеливания

`iid` – взвешивается ли качество по мощности фолдов

`cv` – CV-генератор или итерируемый объект

`refit` – перенастроить ли модель,

используя лучшие найденные параметры (\Rightarrow `best_estimator_`)
`verbose`

`error_score` – что делать при ошибке
«raise» / число

`return_train_score` – будут ли в `cv_results_` ошибки на обучении

Ещё в `model_selection`

Надёжность CV-контроля

`model_selection.permutation_test_score`

Генерация значений параметров из специальных распределений

`model_selection.ParameterSampler`

Скореры в оценке модели (sklearn.metrics)

```
from sklearn.metrics.scorer import SCORERS
# какие скореры есть
print(SCORERS.keys())
# пишем свой скорер
def my_accuracy_scoring(est, X, y):
    return np.mean(est.predict(X) == y)

cross_val_score(knn, X, y, scoring=my_accuracy_scoring, cv=4)
array([ 0.95,  0.9 ,  1.,  0.95])

# другой способ
from sklearn.metrics import make_scorer
# ф-я сравнения
def cmp(a, y):
    return (np.mean(np.abs(a - y) < 0.1))

# скорер на её основе
scorer = make_scorer(cmp, greater_is_better=False,
                      needs_proba=False, needs_threshold=False)

# можно использовать так:
cross_val_score(rf, X, y, scoring=scorer, cv=2) # а не 'roc_auc'
```

Скореры в оценке модели (sklearn.metrics)

```
'f1',  
'f1_weighted',  
'f1_samples',  
'neg_mean_squared_error',  
'precision_weighted',  
'recall_samples',  
'recall_micro',  
'adjusted_rand_score',  
'recall_macro',  
'mean_absolute_error',  
'precision_macro',  
'neg_log_loss',  
'neg_mean_absolute_error',  
'f1_macro',  
  
'recall_weighted',  
'accuracy',  
'precision_samples',  
'median_absolute_error',  
'precision',  
'log_loss',  
'precision_micro',  
'average_precision',  
'roc_auc',  
'r2',  
'recall',  
'mean_squared_error',  
'f1_micro',  
'neg_median_absolute_error'
```

Есть много скореров
Можно написать свой

Качество (sklearn.metrics)

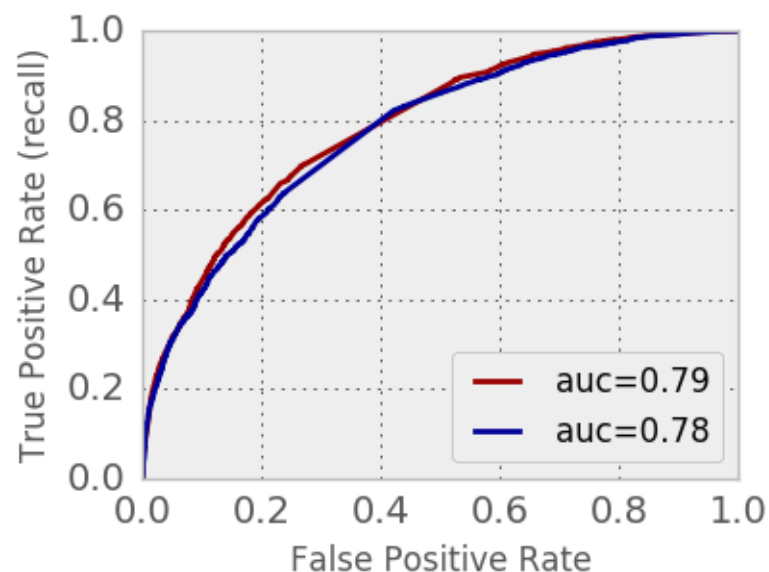
```
from sklearn.metrics import classification_report
```

```
print(classification_report(y, a))
```

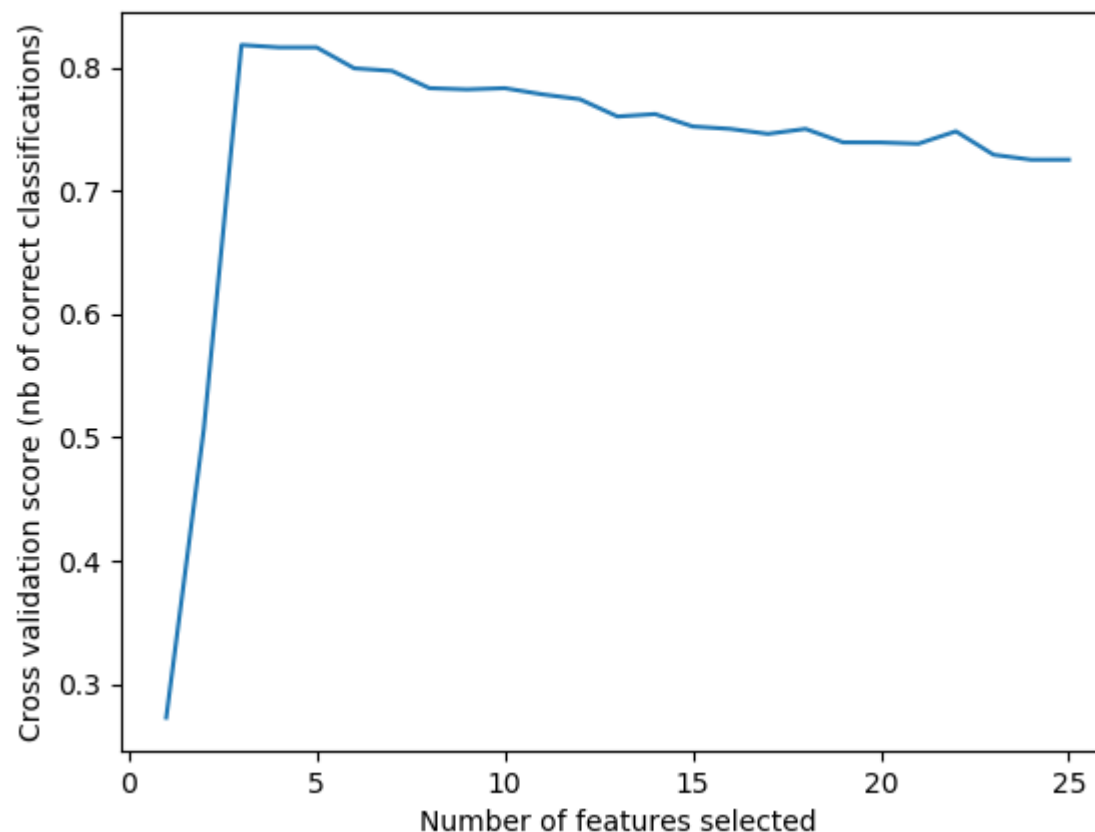
	precision	recall	f1-score	support
0.0	0.82	1.00	0.90	6694
1.0	1.00	0.73	0.84	5306
avg / total	0.90	0.88	0.88	12000

Качество (sklearn.metrics)

```
plt.figure(figsize=(6,5))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (recall)")
fpr, tpr, _ = roc_curve(y, a)
fpr2, tpr2, _ = roc_curve(y, a2)
auc1 = roc_auc_score(y, a)
auc2 = roc_auc_score(y, a2)
plt.plot(fpr, tpr, label=("auc=%.2f" % auc1), linewidth=2,
color='#990000')
plt.plot(fpr2, tpr2, label=("auc=%.2f" % auc2), linewidth=2,
color='#000099')
plt.legend(loc="best")
```



Отбор признаков: `sklearn.feature_selection`



https://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html#sphx-glr-auto-examples-feature-selection-plot-rfe-with-cross-validation-py

Отбор признаков

```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV

svc = SVC(kernel="linear")
rfecv = RFECV(estimator=svc,
              step=1,
              cv=StratifiedKFold(2),
              scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features : %d" % rfecv.n_features_)
plt.plot(range(1, len(rfecv.grid_scores_) + 1),
         rfecv.grid_scores_)
```

Отбор признаков: `sklearn.feature_selection.RFECV`

`estimator`

`step` – сколько признаков удалять на каждом шаге

`min_features_to_select` – минимальное число признаков,
которые хотим отобрать

`cv`

`scoring`

`verbose`

`n_jobs`

Отбор признаков: стратегии отбора

Мало изменяющиеся
VarianceThreshold

Фильтры
SelectKBest

Рекурсивное удаление
RFECV

Из модели
SelectFromModel

Последовательность операторов: pipeline

```
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(TfidfVectorizer(), LogisticRegression())
pipeline.fit(text_train, y_train)
pipeline.score(text_test, y_test)
0.5
```

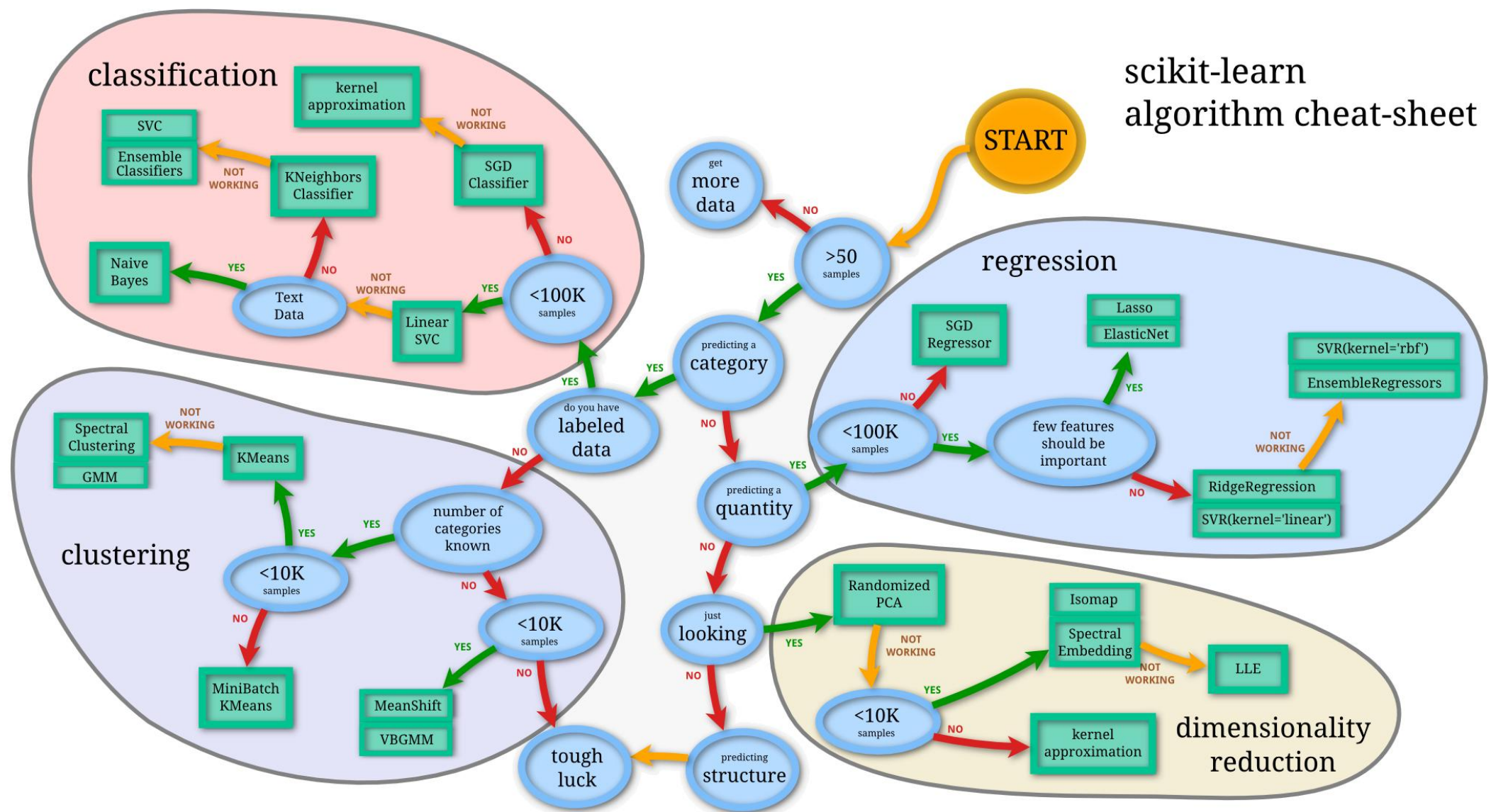
Оптимизация параметров

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

pipeline = make_pipeline(TfidfVectorizer(), LogisticRegression())

params = {'logisticregression__C': [.1, 1, 10, 100],
          "tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (2, 2)]}
grid = GridSearchCV(pipeline, param_grid=params, cv=5)
grid.fit(X, y)
print(grid.best_params_)
grid.score(X, y)
```

Выбор алгоритма (модели)



Предобработка данных: preprocessing

Нормировка данных

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

Перенумерация

```
f = ['a', 'bb', 20, 'bb', 'a', 'a']
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
encoder.fit(f)
encoder.transform(f)

array([1, 2, 0, 2, 1, 1], dtype=int64)
```

Предобработка данных: preprocessing

Характеристическая матрица

```
f = ['a', 'bb', 'c', 'bb', 'a', 'a']  
from sklearn.preprocessing import LabelBinarizer  
encoder = LabelBinarizer()  
encoder.fit(f)  
encoder.transform(f)
```

```
array([[1, 0, 0],  
       [0, 1, 0],  
       [0, 0, 1],  
       [0, 1, 0],  
       [1, 0, 0],  
       [1, 0, 0]])
```


Предобработка данных: preprocessing

Характеристическая матрица для группы вещественных признаков

```
f = [[1, 2], [1, 1], [2, 2]]  
from sklearn.preprocessing import OneHotEncoder  
encoder = OneHotEncoder()  
encoder.fit(f)  
encoder.transform(f).toarray()
```

```
array([[ 1.,  0.,  0.,  1.],  
       [ 1.,  0.,  1.,  0.],  
       [ 0.,  1.,  0.,  1.]])
```

Предобработка данных: preprocessing

Полиномиальные признаки

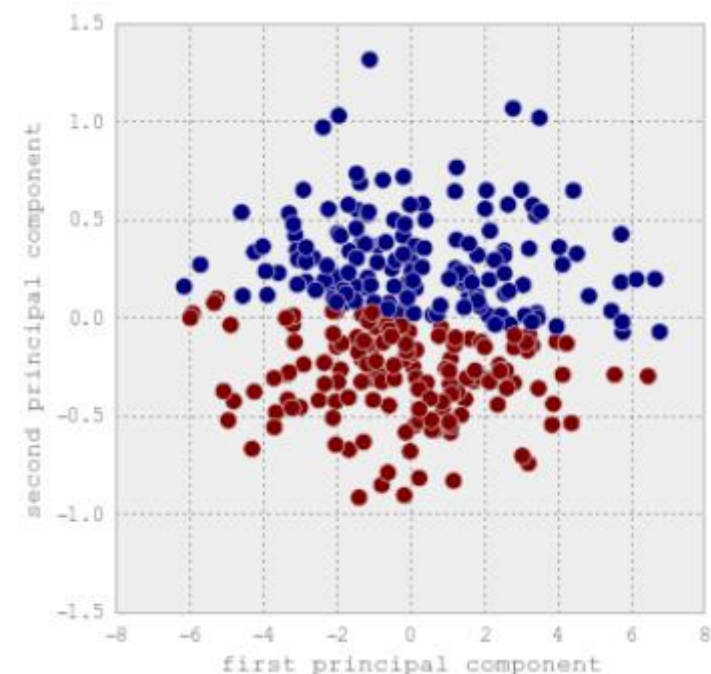
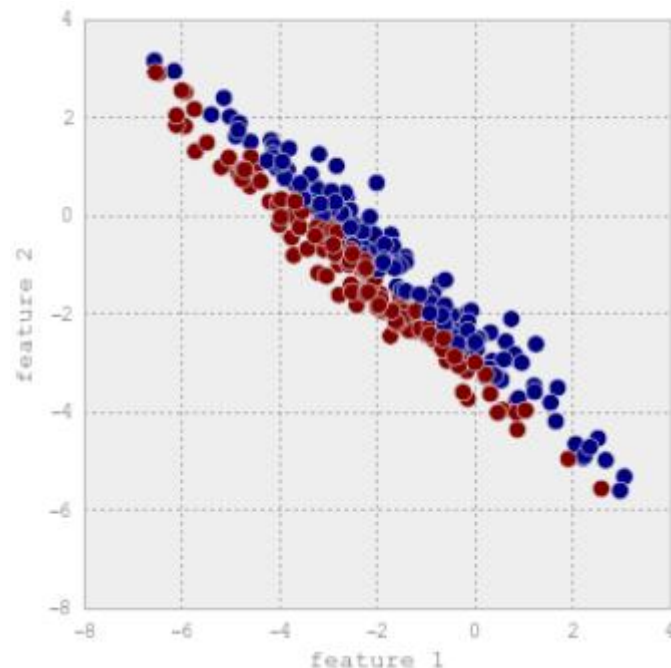
```
from sklearn.preprocessing import PolynomialFeatures
pf = PolynomialFeatures(degree=2)
f = [[1, 0], [2, 1], [3, 2]]
pf.fit(f)
pf.transform(f)

array([[1, 1, 0, 1, 0, 0],
       [1, 2, 1, 4, 2, 1],
       [1, 3, 2, 9, 6, 4]])
```

Декомпозиции матриц: decomposition

Приведём лишь пример с SVD (есть ещё ICA, NMF и т.п.)

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X_blob)
X_pca = pca.transform(X_blob)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, linewidths=0, s=70)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
```



Сокращение размерности

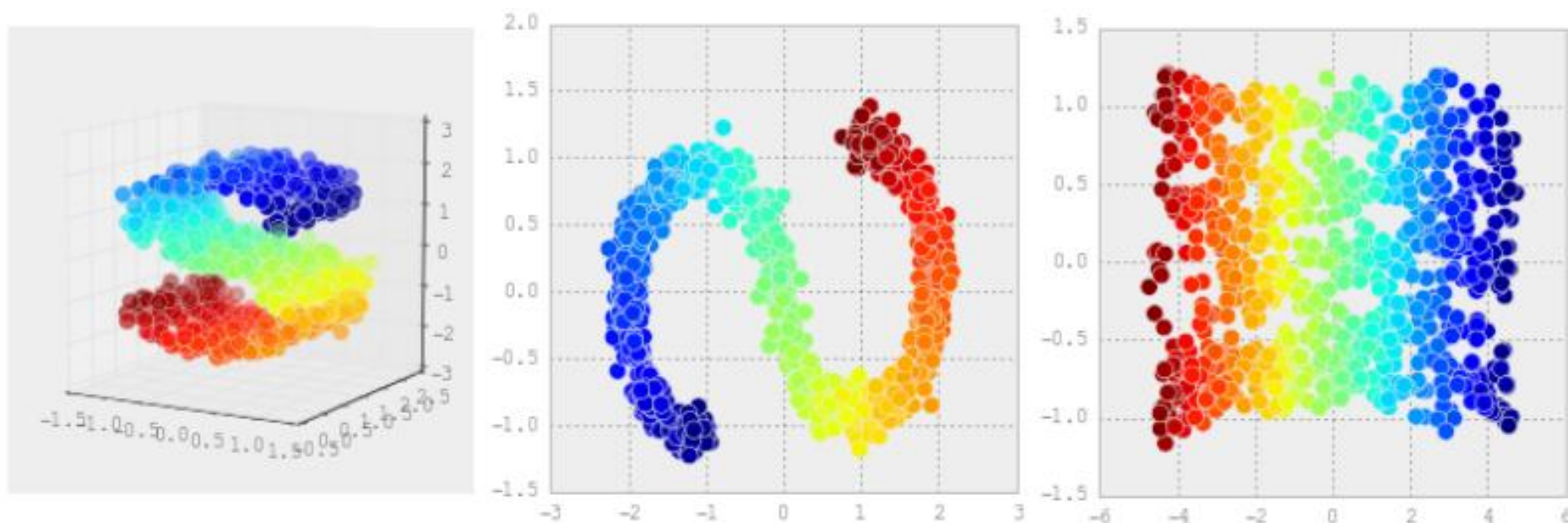
```
from sklearn.datasets import make_s_curve
X, y = make_s_curve(n_samples=1000, noise=0.1)
```

```
from mpl_toolkits.mplot3d import Axes3D
ax = plt.axes(projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], X[:, 2], c=y, s=70)
ax.view_init(10, -60)
```

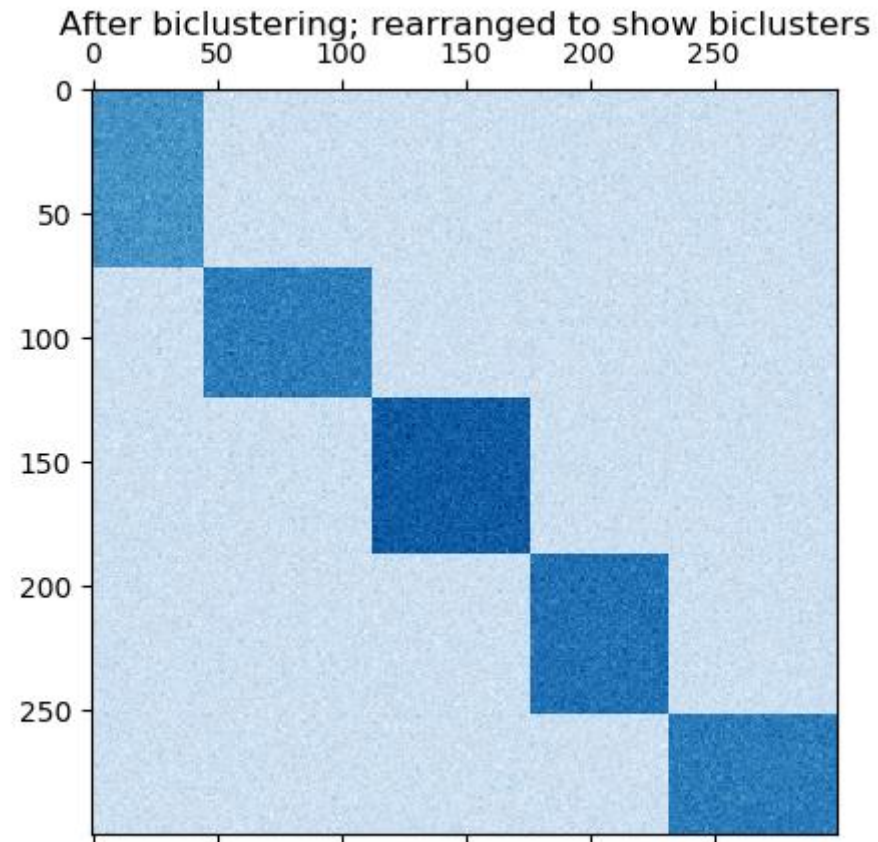
```
X_pca = PCA(n_components=2).fit_transform(X)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, s=70)
```

```
from sklearn.manifold import Isomap
iso = Isomap(n_neighbors=15, n_components=2)
X_iso = iso.fit_transform(X)
plt.scatter(X_iso[:, 0], X_iso[:, 1], c=y, s=70)
```

Сокращение размерности



Бикластеризация: `sklearn.cluster.bicluster`



Одновременная кластеризация строк и столбцов

Работа с текстами

Как всегда – всё просто...

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

vectorizer = TfidfVectorizer()
vectorizer.fit(text_train)

X_train = vectorizer.transform(text_train)
X_test = vectorizer.transform(text_test)

clf = LogisticRegression()
clf.fit(X_train, y_train)

clf.score(X_test, y_test)
```

Работа с текстами: чуть подробнее

```
X = ["Some say the world will end in fire,",
     "Some say in ice."]

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X)
vectorizer.vocabulary_
{'end': 0,
 'fire': 1,
 'ice': 2,
 'in': 3,
 'say': 4,
 'some': 5,
 'the': 6,
 'will': 7,
 'world': 8}

X_bag_of_words = vectorizer.transform(X) # sparse-матрица
X_bag_of_words.toarray()
array([[1, 1, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 1, 1, 1, 1, 0, 0, 0]], dtype=int64)
```


Работа с текстами: чуть подробнее

```
vectorizer.get_feature_names()  
['end', 'fire', 'ice', 'in', 'say', 'some', 'the', 'will', 'world']
```

```
vectorizer.inverse_transform(X_bag_of_words)  
[array(['end', 'fire', 'in', 'say', 'some', 'the', 'will', 'world'],  
      dtype='<U5'), array(['ice', 'in', 'say', 'some'],  
      dtype='<U5')]
```

```
from sklearn.feature_extraction.text import TfidfVectorizer  
tfidf_vectorizer = TfidfVectorizer() # другой "векторайзер!"  
tfidf_vectorizer.fit(X)  
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',  
               dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',  
               lowercase=True, max_df=1.0, max_features=None, min_df=1,  
               ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,  
               stop_words=None, strip_accents=None, sublinear_tf=False,  
               token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,  
               vocabulary=None)
```

```
print(tfidf_vectorizer.transform(X).toarray())  
[[ 0.39  0.39  0.      0.28  0.28  0.28  0.39  0.39  0.39]  
 [ 0.      0.      0.63  0.45  0.45  0.45  0.      0.      0.  ]]
```

Работа с текстами: чуть подробнее

```
bigram_vectorizer = CountVectorizer(ngram_range=(1, 2))
# от какого до какого ранга

bigram_vectorizer.fit(X)
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 2), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w+\\b',
                tokenizer=None, vocabulary=None)

bigram_vectorizer.get_feature_names()
['end',
 'end in',
 ...
 'world will']

bigram_vectorizer.transform(X).toarray()
array([[1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0]],
dtype=int64)
```

Модели для sparse-матриц

```
linear_model.Ridge()  
linear_model.Lasso()  
linear_model.ElasticNet()  
linear_model.LinearRegression()  
linear_model.Perceptron()  
linear_model.PassiveAggressiveRegressor()  
linear_model.PassiveAggressiveClassifier()  
linear_model.SGDRegressor()  
linear_model.SGDClassifier()  
svm.SVR()  
svm.NuSVR()  
naive_bayes.MultinomialNB()  
naive_bayes.BernoulliNB()  
neighbors.KNeighborsRegressor()
```

**На вход можно подавать разреженную матрицу – всё работает
(не во всех моделях быстро на больших матрицах).**

Подробнее о кластеризации...

Метод	Параметры	Использование	Геометрия
k-means	Число кластеров	Популярный Когда немного кластеров Плоская кластеризация	Расстояния Гипотеза «кластеры – шары»
Affinity propagation	Фактор забывания, вероятность стать экземпляром	Много кластеров Не плоская геометрия Не масштабируется	Граф расстояний
Mean-shift	Параметр ядра	Много кластеров Не плоская геометрия Не масштабируется	Расстояния
Spectral clustering	Число кластеров	Мало кластеров Не плоская геометрия Плохо масштабируется	Граф расстояний
Ward hierarchical / Agglomerative clustering	Число кластеров / тип слияния и метрика	Много кластеров Можно добавить ограничения на связи (топологию слияния) Хорошо масштабируется	Расстояния
DBSCAN	Размер соседства	Не плоская геометрия Разномощные кластеры Хорошо масштабируется	Расстояния между ближайшими точками
Gaussian mixtures	много	Специальный случай Не масштабируется	Расстояния Махаланобиса
Birch	Мета-кластеризатор, порог, максимальное число подкластеров	Для больших данных Удаление выбросов Сжатие данных Хорошо масштабируется	Евклидовы расстояния между точками

k-Means

`sklearn.cluster.KMeans`

`n_clusters` – **число кластеров**

`init` – **метод инициализации :«k-means++», «random», ndarray**

`n_init` – **сколько раз алгоритм запускается (выбирается лучший ответ)**

`max_iter` – **максимальное число итерация для каждого алгоритма**

`tol` – **порог для определения сходимости**

`precompute_distances` – **предварительное вычисление расстояний
«auto», True, False (для скорости, но требует память)**

`verbose` – **отчёт о работе**

`random_state`

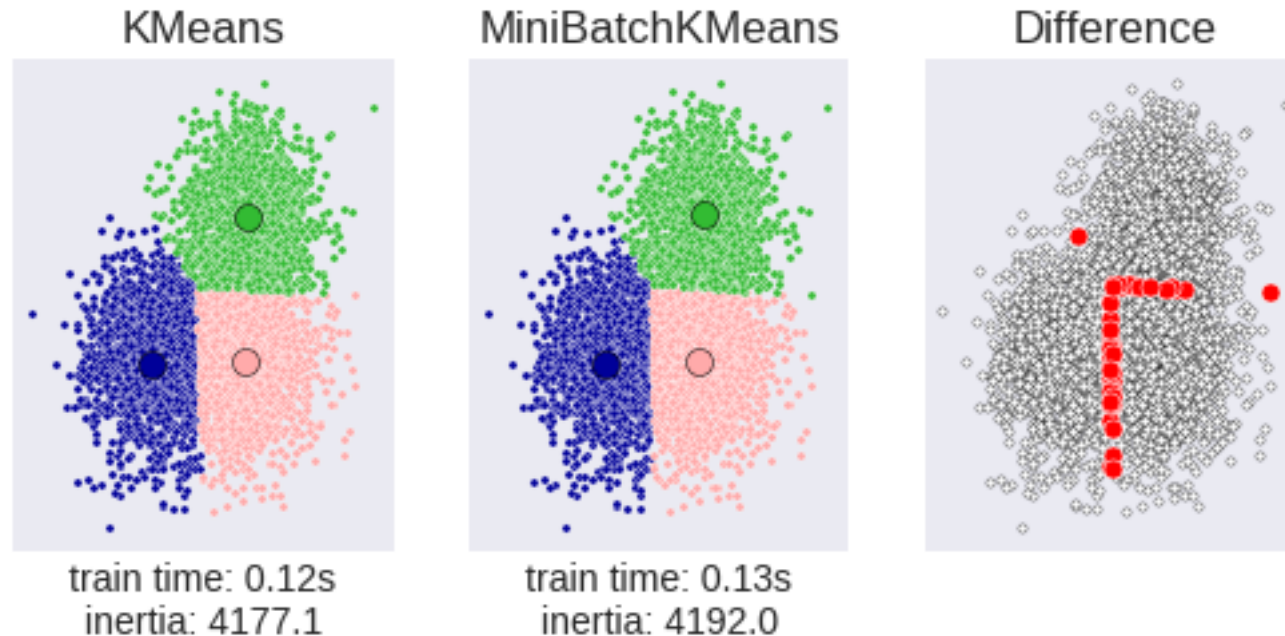
`copy_x` – **когда precompute_distances можно ли портить данные**
`n_jobs`

`algorithm` – **какой алгоритм использовать:**

«auto», «full», «elkan» (использует неравенство треугольника, но не подходит для разреженных данных)

k-Means с мини-батчами

`sklearn.cluster.MinibatchKMeans`



D. Sculley «Web-Scale K-Means Clustering» // WWW 2010: Proceedings of the 19th Annual International World Wide Web Conference. April, 2010.

<http://www.eecs.tufts.edu/~dsculley/papers/fastkmeans.pdf>

k-Means с мини-батчами

`sklearn.cluster.MinibatchKMeans`

`batch_size` – **размер батча**

`compute_labels` – **вычислять ли метки и качество 1 раз в конце (?)**

`max_no_improvement` – **для остановки**

`init_size` – **число сэмплов для ускорения инициализации**

`n_init` – **число случайных инициализаций**

метод запускается один раз (!) на лучшей инициализации

`reassignment_ratio` – **максимальное число вычислений для пересчёта центра (?)**

Кластеризация сообщениями между точками: **AffinityPropagation**

`sklearn.cluster.AffinityPropagation`

`damping` – **фактор забывания значений параметров**

`max_iter` – **максимальное число итераций**

`convergence_iter` – **число итераций без изменений,
после которого останов**

`copy`

`preference` – **вероятность для каждой точки стать экземпляром (т.е.
породить кластер)**

`affinity` – **п.в. эвклидово расстояние**

`verbose`

Обнаружение мод плотности: MeanShift

`sklearn.cluster.MeanShift`

`bandwidth` – параметр RBF-ядра

`seeds` – для инициализации ядер

`bin_seeding` – если включить опцию, изначально алгоритм работает в
огрублённом пространстве (дискретизация исходных данных)

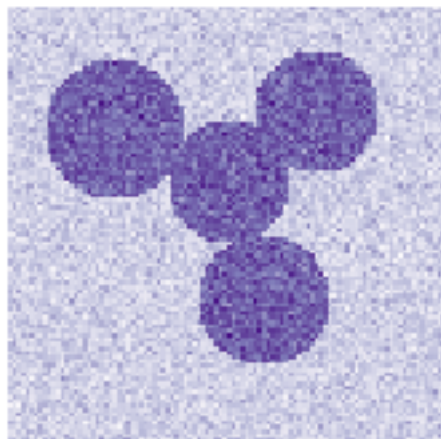
`min_bin_freq` – для увеличения скорости принимать бины только с
таким количеством точек

`cluster_all` – кластеризовать ли все точки

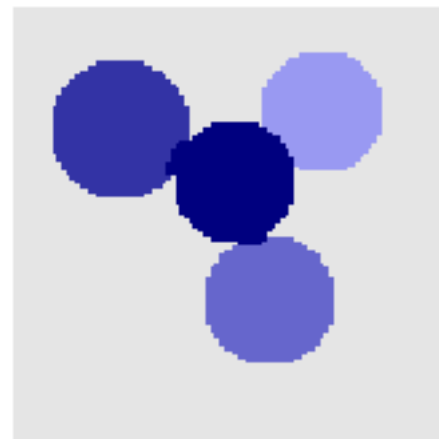
Есть возможность оставлять «сирот» (orphans) без кластеров

`n_jobs`

Спектральная кластеризация



изображение



спектральная кластеризация

**Рекомендуется, когда матрица сходства разрежена
надо выбрать `eigen_solver="amg"`
предварительно установив `ruamg`
(library of Algebraic Multigrid)**

Спектральная кластеризация графов

```
from sklearn.cluster import SpectralClustering
sc = SpectralClustering(3, affinity='precomputed', n_init=100,
                        assign_labels='discretize')
sc.fit_predict(adjacency_matrix)
```

Спектральная кластеризация

`sklearn.cluster.SpectralClustering`

`n_clusters` – **число кластеров**

`eigen_solver` – **метод вычисления собственных векторов (?)**: `None`,
«`arpack`», «`lobpcg`», «`amg`»

`random_state`

`n_init` – **число запусков k-means**

`gamma` – **коэффициент для ядер**: `rbf`, `poly`, `sigmoid`, `laplacian`, `chi2`

`affinity` – **(сходство)** «`nearest_neighbors`» / «`precomputed`» / «`rbf`» или
любое из `sklearn.metrics.pairwise_kernels`

`n_neighbors` – **число соседей**, нужно для 1-nn, если он используется

`eigen_tol` – **порог сходимости** при декомпозиции матрицы Лапласа

`assign_labels` – **стратегия для простановки меток** после лапласового
вложения: «`kmeans`» / «`discretize`»

`degree` – **степень полинома**

`coef0` – **коэффициент полиномиального или сигмоидального ядра**

`kernel_params` – **параметры для самописных ядер**

`n_jobs`

Иерархическая кластеризация

Агломеративная кластеризация

`sklearn.cluster.AgglomerativeClustering`

`n_clusters` – число кластеров

`affinity` – «euclidean», «l1», «l2», «manhattan», «cosine»,
«precomputed», функция

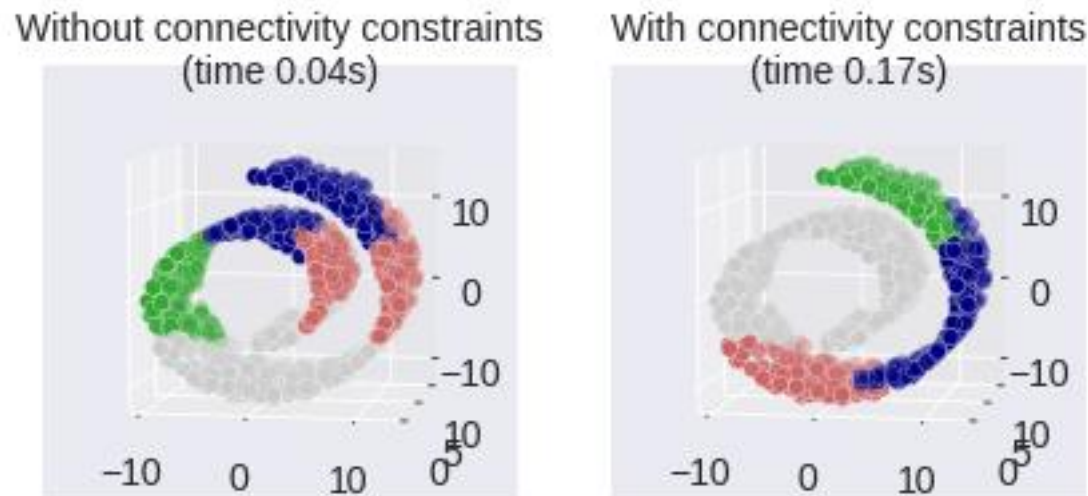
`memory` – параметр связанный с кэшированием дерева

`connectivity` – матрица связности (матрица или функция)
чтобы только соседние кластеры объединялись
по умолчанию нет

`compute_full_tree` – строить ли всё дерево (обычно до `n_clusters`)

`linkage` – критерий слияния: «ward», «complete», «average», «single»,

Использование матрицы связности

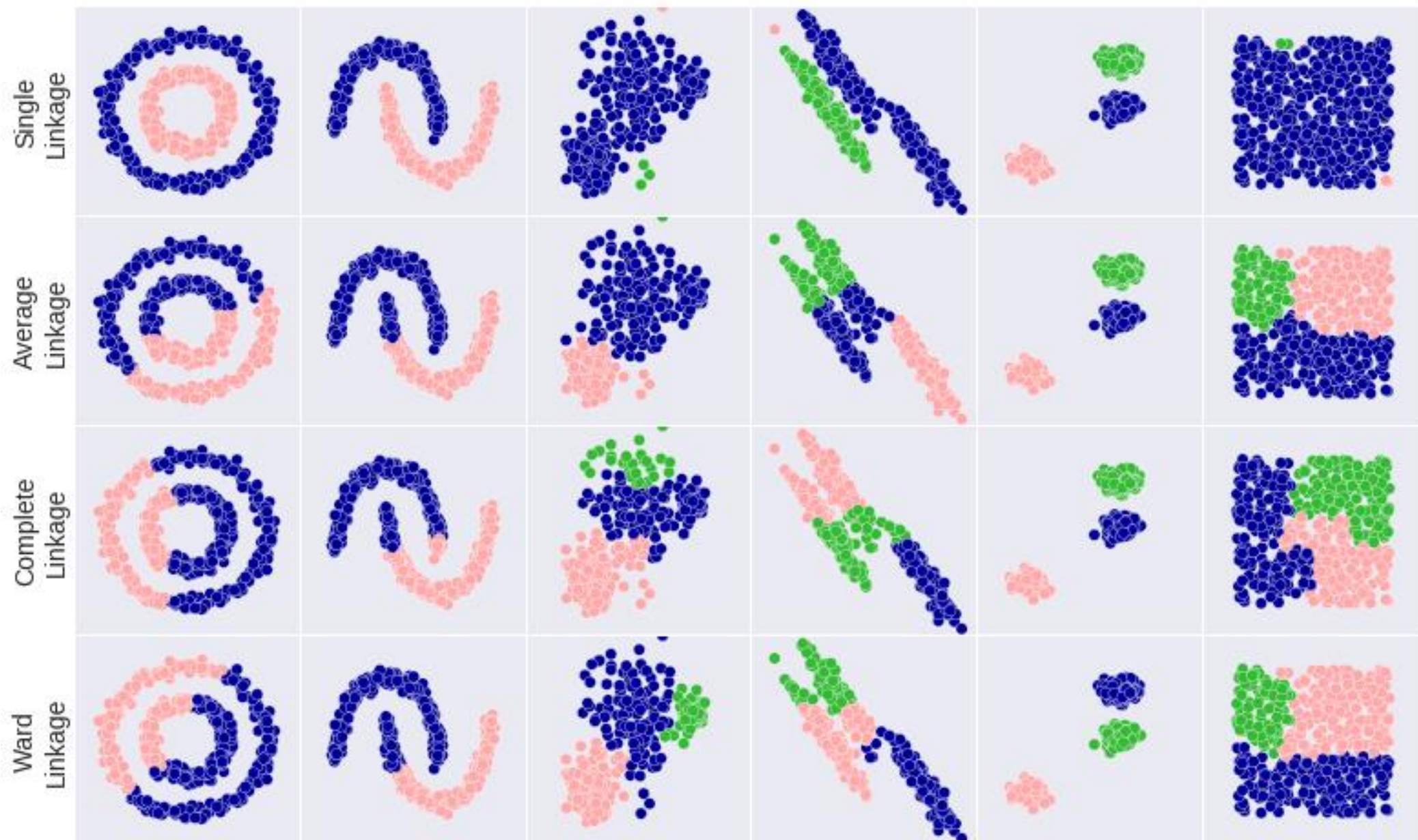


матрица k-соседства

Агломеративная кластеризация для сокращения признакового пространства

```
from sklearn import cluster
agglo = cluster.FeatureAgglomeration(n_clusters=32)
agglo.fit(X)
X_reduced = agglo.transform(X)
X.shape, X_reduced.shape
(1797, 64), (1797, 32)
```


Разные виды связности



DBSCAN = Density-Based Spatial Clustering of Applications with Noise

`sklearn.cluster.DBSCAN`

eps – порог близости

min_samples – число объектов в окрестности, чтобы считать точку основной

metric –пользовательская или ...(?)

metric_params – параметры метрики

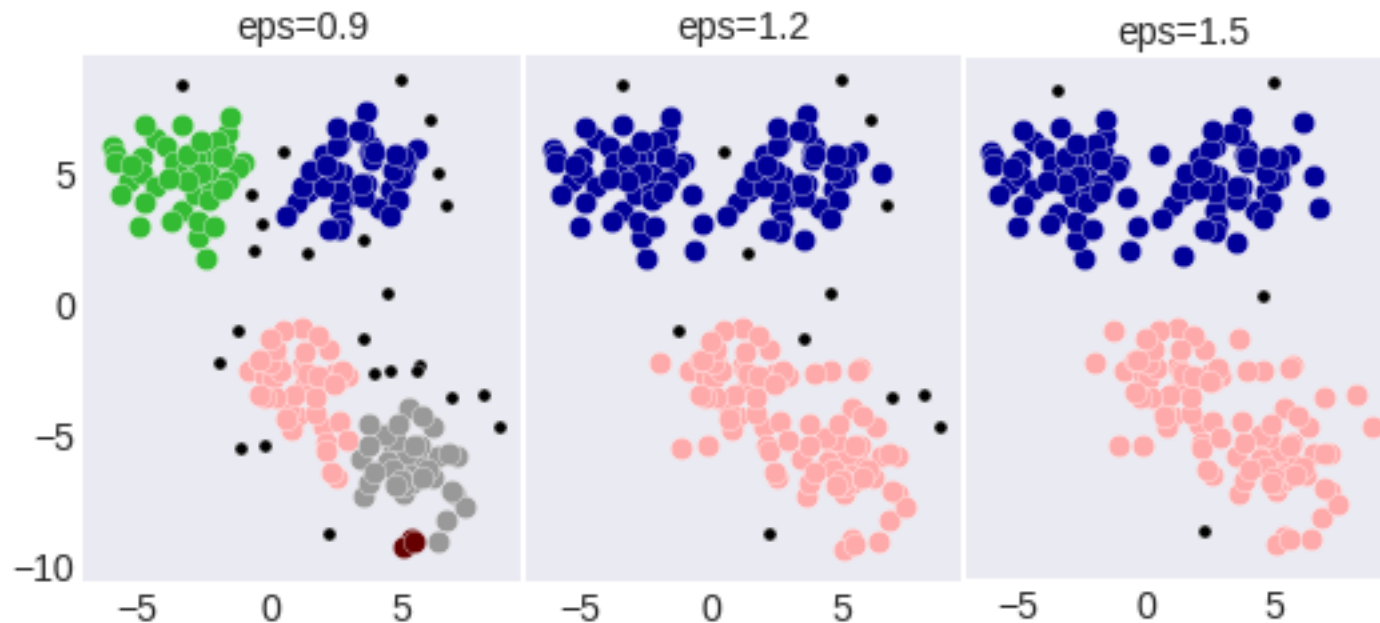
algorithm – алгоритм для нахождения ближайших соседей: «auto», «ball_tree», «kd_tree», «brute»

leaf_size – размер листьев в BallTree / KDTree

p – степень в метрике Минковского

n_jobs

DBSCAN = Density-Based Spatial Clustering of Applications with Noise



```
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=0.9, min_samples=3).fit(X)
labels = clustering.labels_
```

BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

`sklearn.cluster.Birch`

`threshold` – порог для организации нового подкластера

`branching_factor` – максимальное число подкластеров в каждом узле

(если превышает, то узел расщепляется на два)

`n_clusters` – число кластеров

если число, то делаем кластеризацию с помощью `AgglomerativeClustering`

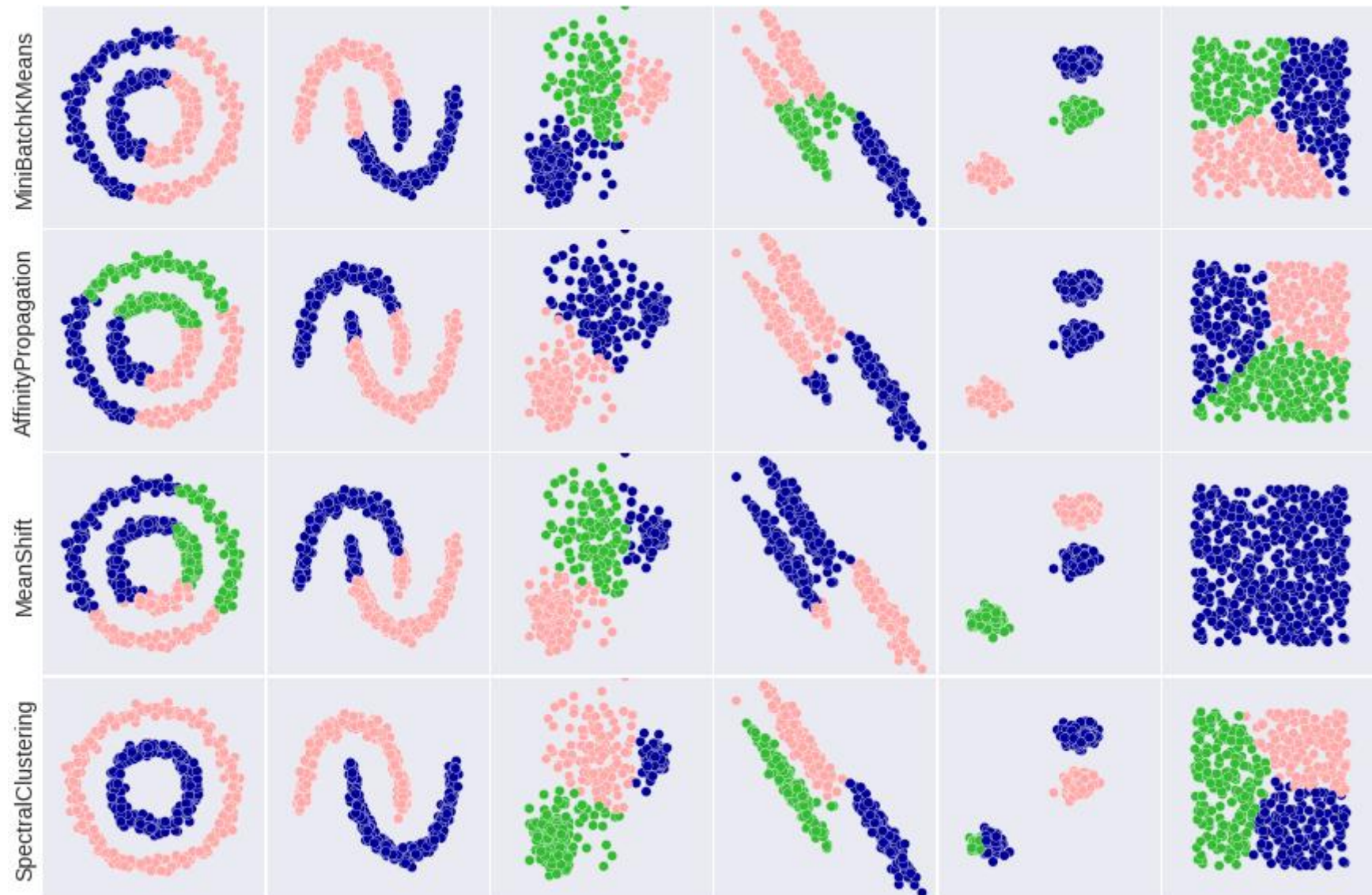
если `None` не делаем формально окончательную кластеризацию

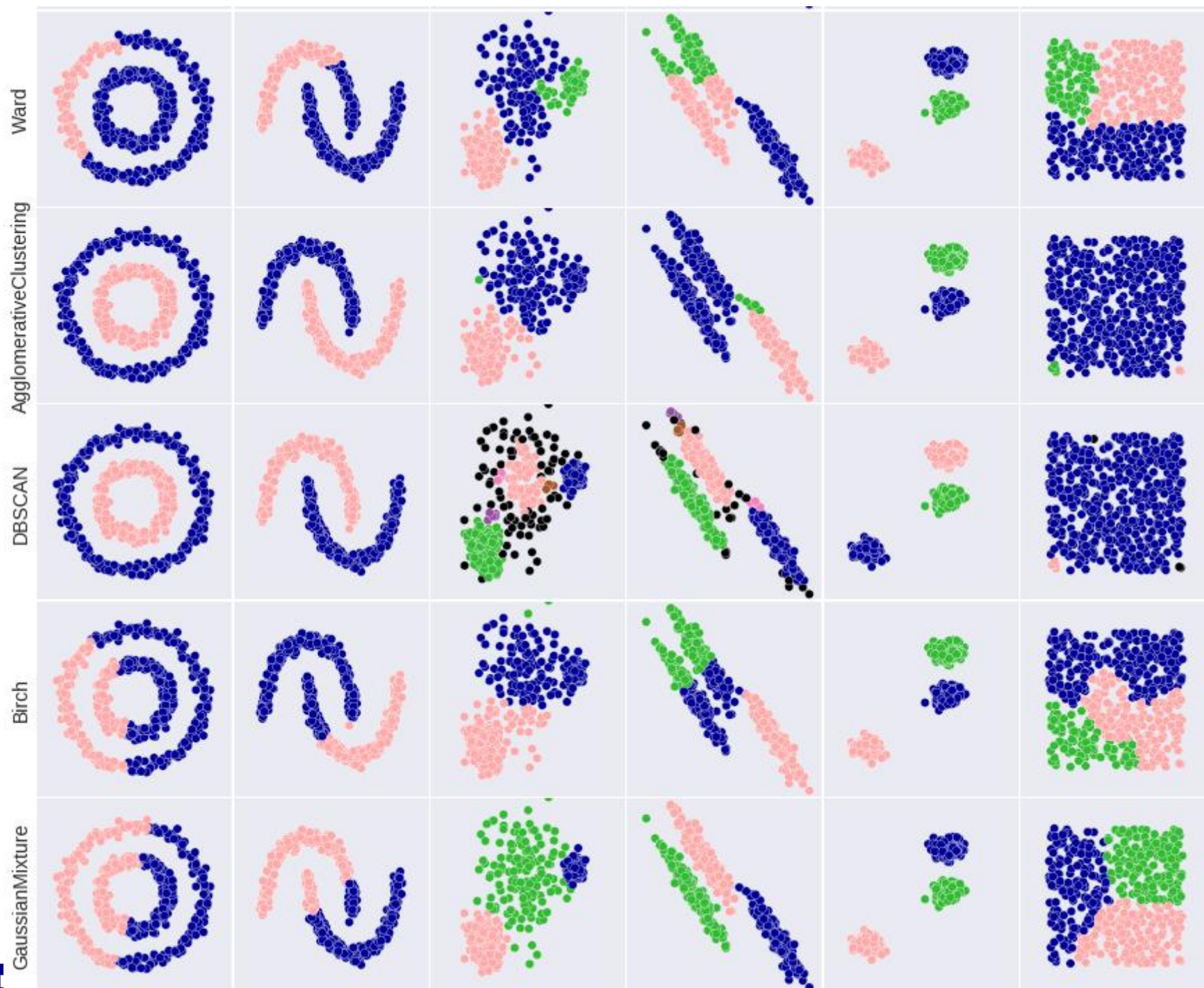
если `sklearn.cluster.Estimtor`, то делаем им кластеризацию подкластеров

`compute_labels` – вычислять ли метки

сору

Сравнение алгоритмов кластеризации





Своя модель

```
from sklearn.base import BaseEstimator, ClassifierMixin

class MeanClassifier(BaseEstimator, ClassifierMixin):
    """пример модели"""

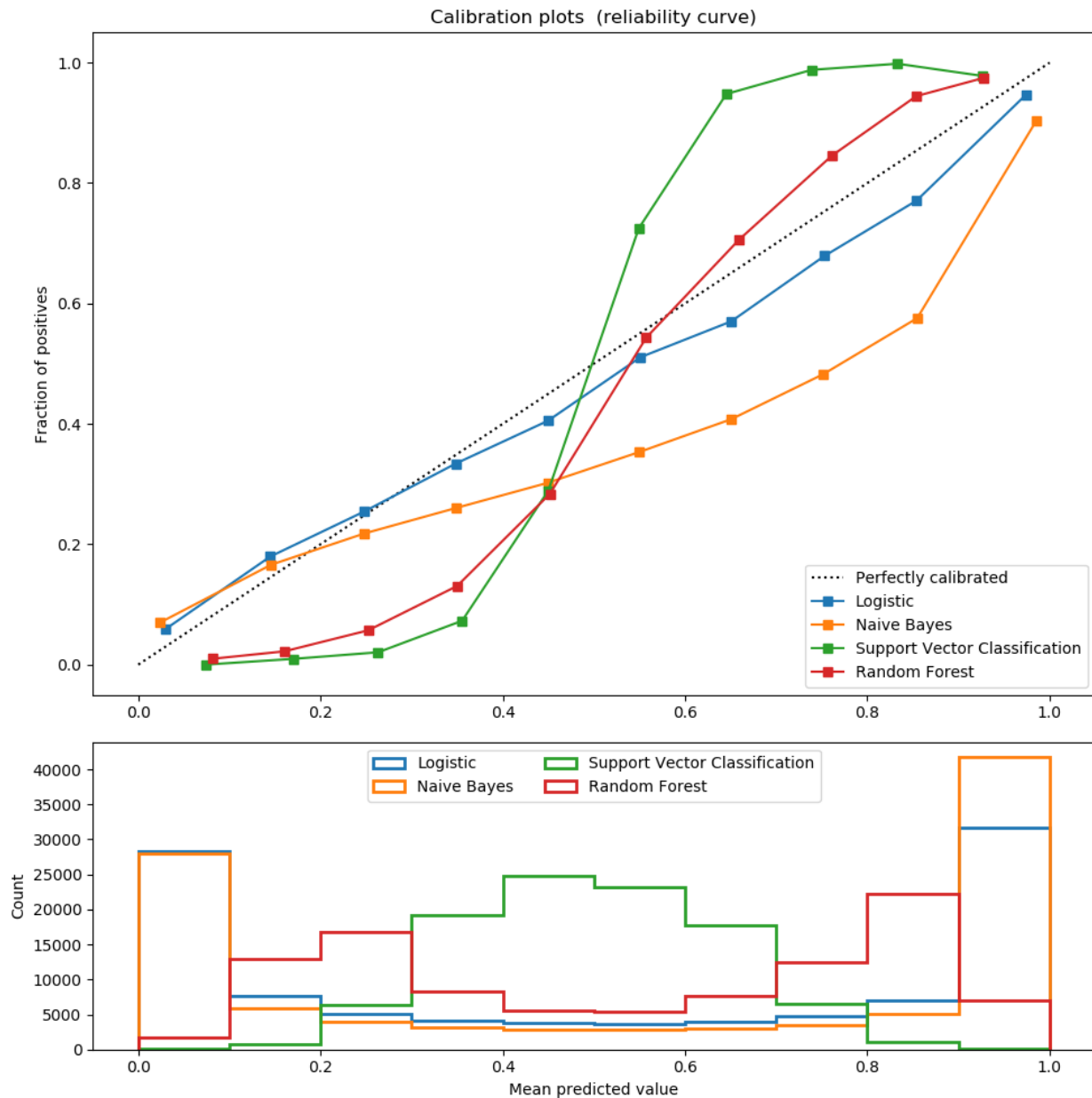
    def __init__(self, intValue=0):
        """
        инициализация
        """
        self.intValue = intValue
        # ...

    def fit(self, X, y=None):
        """
        настройка
        """
        assert (type(self.intValue) == int), "intValue parameter must be integer"
        # ...
        return self

    def predict(self, X, y=None):
        """
        работа
        """
        # ...
        return(a)

    def score(self, X, y=None):
        # оценка
        return(np.mean(self.predict(X) == y))
```

Калибровка классификаторов: `sklearn.calibration`



Сохранение моделей

первый способ

```
import pickle  
s = pickle.dumps(clf)  
clf2 = pickle.loads(s)
```

второй способ

```
from sklearn.externals import joblib  
joblib.dump(clf, 'filename.joblib')  
clf2 = joblib.load('filename.joblib')
```

Ссылки

В данной презентации много примеров взято из ноутбука

https://github.com/amueller/scipy_2015_sklearn_tutorial/tree/master/notebooks

Спасибо Андреасу Мюллеру!

См. API Reference

<http://scikit-learn.org/stable/modules/classes.html>