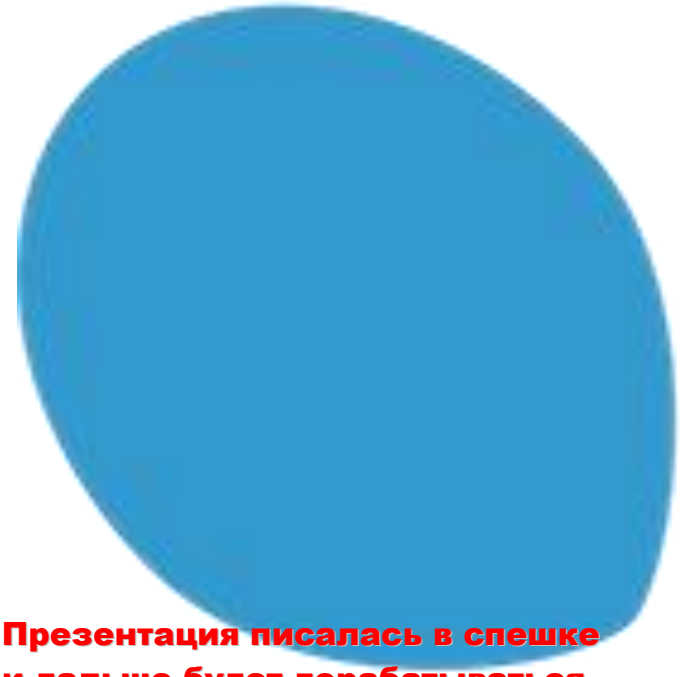
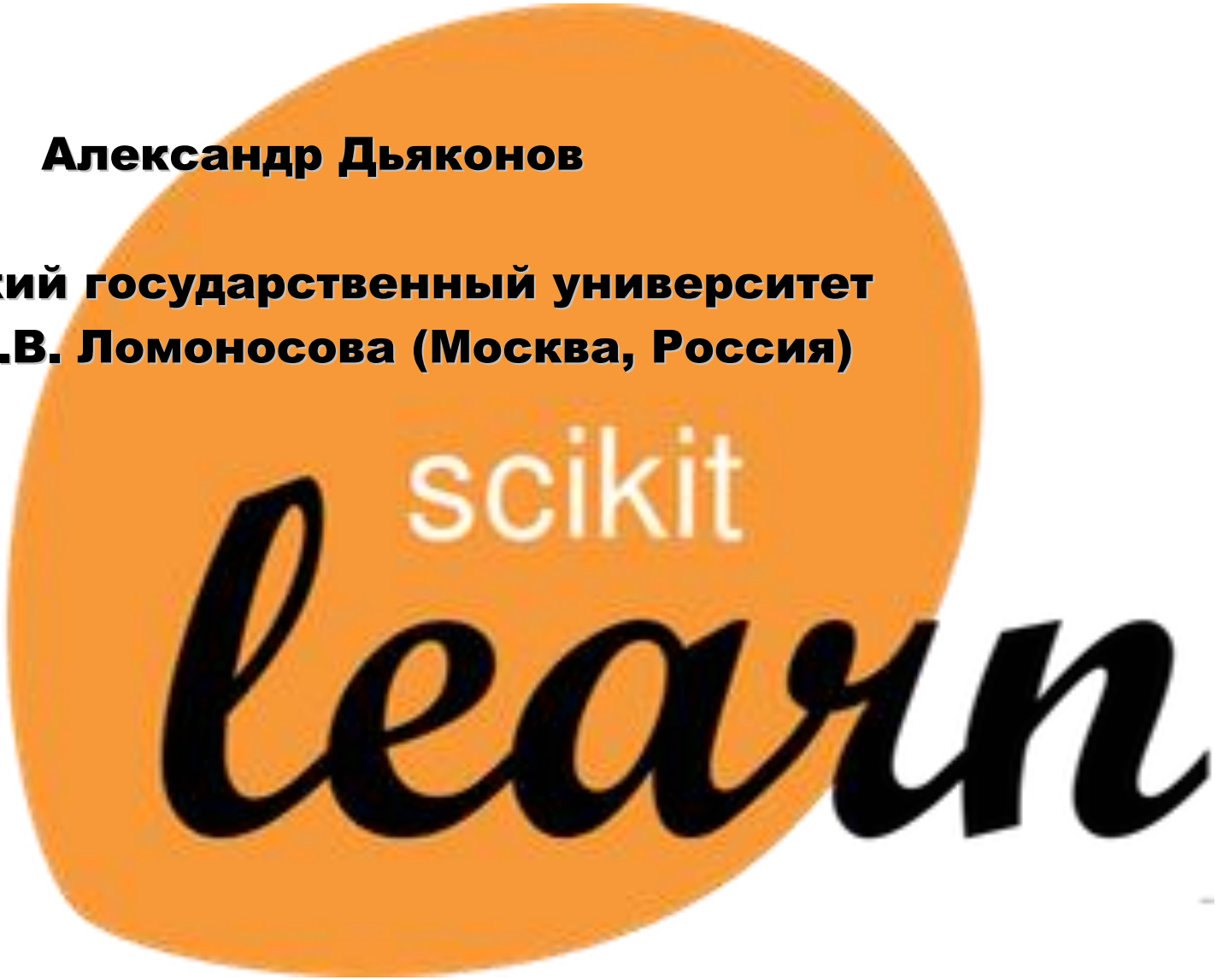
Введение в scikit-Learn

Александр Дьяконов

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





Презентация писалась в спешке
и дальше будет дорабатываться,
описаны не все возможности пакета.
Критику и замечания - присылать автору.

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

Как работать с моделями…

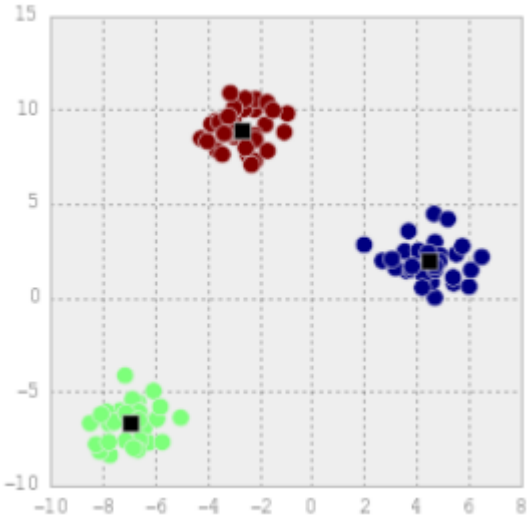
1. Загружаем/создаём выборку

```
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=42)
```

2. Предобрабатываем

3. Обучаем модель: fit()

4. Запускаем на тесте: predict()




```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=70)

mu = kmeans.cluster_centers_
plt.scatter(mu[:,0], mu[:,1], s=50, c='black', marker='s')
print(mu)
```

5. Оцениваем качество

```
from sklearn.metrics import confusion_matrix, accuracy_score
print(accuracy_score(y, labels))
print(confusion_matrix(y, labels))

from sklearn.metrics import adjusted_rand_score
adjusted_rand_score(y, labels)
```

```
0.0
[[ 0  0 34]
 [33  0  0]
 [ 0 33  0]]

1.0
```

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)**

`model.predict`	`model.transform`
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)

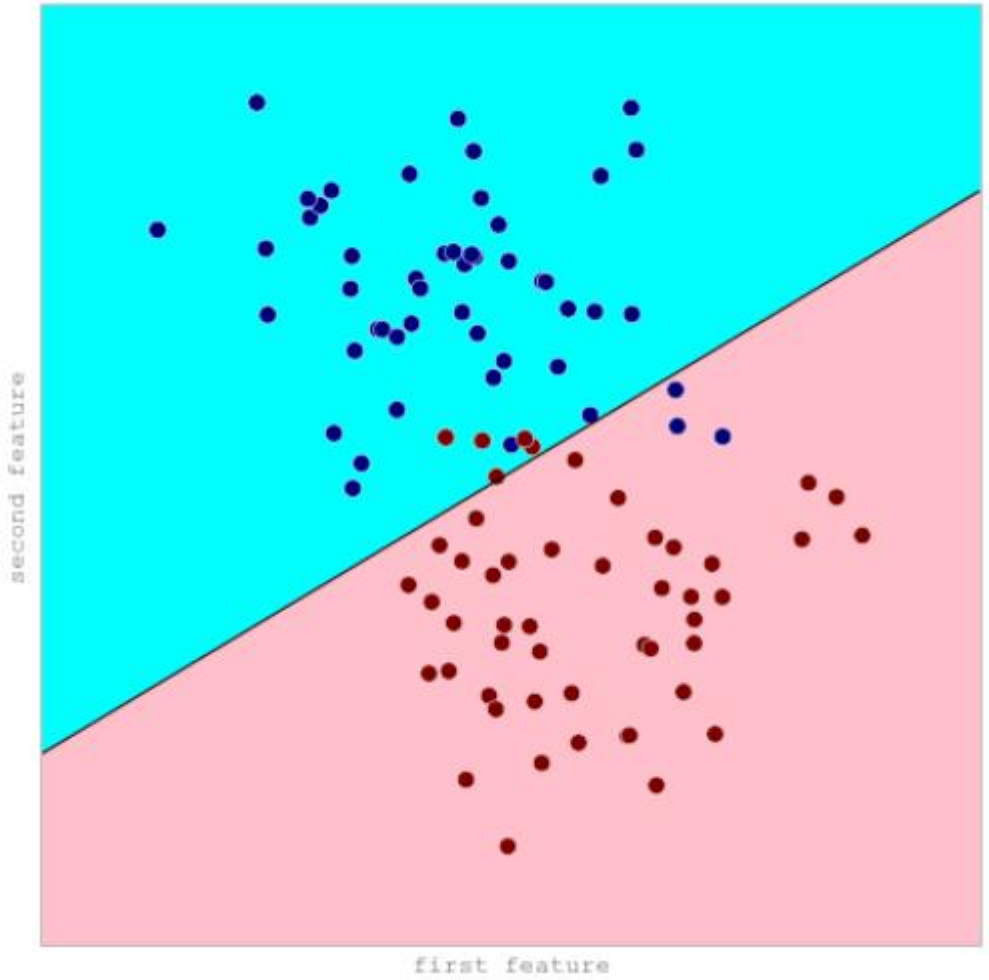
```
def plot_2d_separator(classifier, X, fill=False, line=True, ax=None, eps=None):
    if eps is None:
        eps = 1.0 #X.std() / 2.
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy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

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

# разбивка: обучение - контроль
from sklearn.cross_validation 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                                          0.92
print (np.mean(prediction == y_test))               0.92
print (classifier.score(X_test, y_test)) # более    0.96
удобная функция
print (classifier.score(X_train, y_train))

# визуализация
plt.xlabel("first feature")
plt.ylabel("second feature")
plot_2d_separator(classifier, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], c=y, s=70)

from sklearn.metrics import confusion_matrix        [[12  1]
print (confusion_matrix(y_test, prediction))         [ 4  8]]

from sklearn.metrics import classification_report                precision    recall  f1-score   support
print(classification_report(y_test, prediction))
                                                 0       0.75      0.92      0.83        13
                                                 1       0.89      0.67      0.76        12

                                         avg / total       0.82      0.80      0.80        25
```

Работа с моделями

Что получилось (логистическая регрессия)

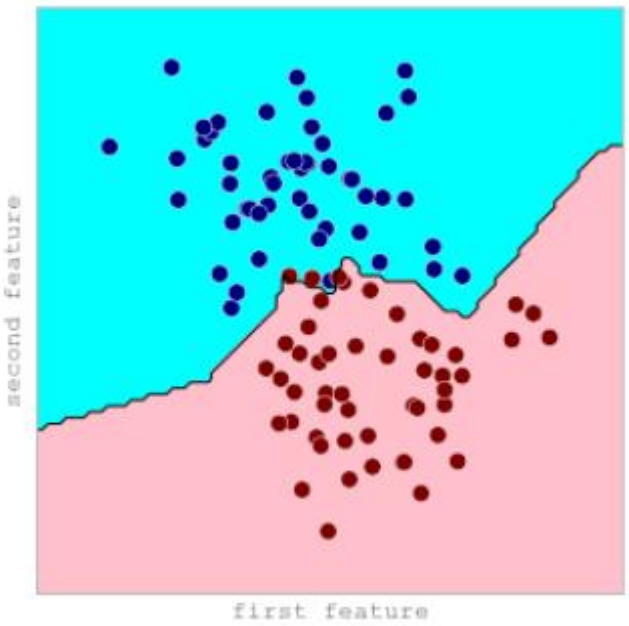


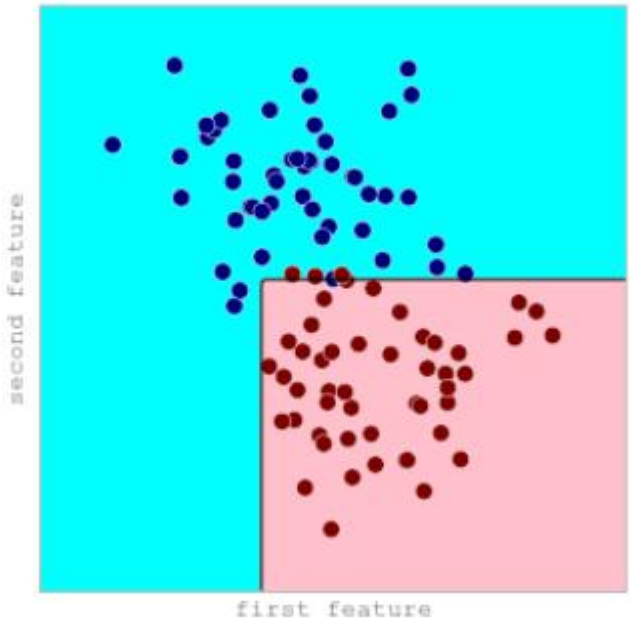
Чем отличаются алгоритмы

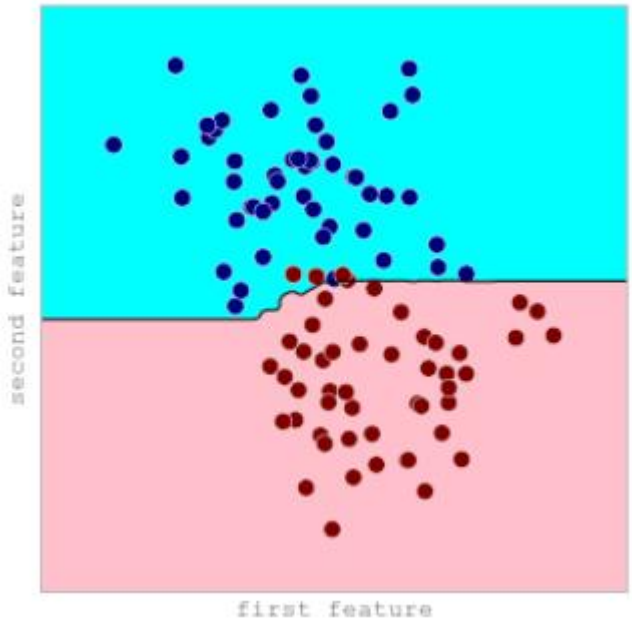
```
from sklearn.neighbors import
KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

print (knn)
print (knn.score(X_test, y_test))
KNeighborsClassifier(algorithm='auto',
leaf_size=30, metric='minkowski',
           metric_params=None,
n_neighbors=1, p=2, weights='uniform')
0.92
```

```
from sklearn.tree import
DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=10)
tree.fit(X_train, y_train)

print (tree)
print (tree.score(X_test, y_test))
DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=10,
            max_features=None,
max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2,
min_weight_fraction_leaf=0.0,
            random_state=None,
splitter='best')
0.88
```

```
from sklearn.ensemble import
RandomForestClassifier

rf =
RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None,
max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1,
min_samples_split=2,
            min_weight_fraction_leaf=0.0,
n_estimators=100, n_jobs=1,
            oob_score=False,
random_state=None, verbose=0,
            warm_start=False)
0.88
```







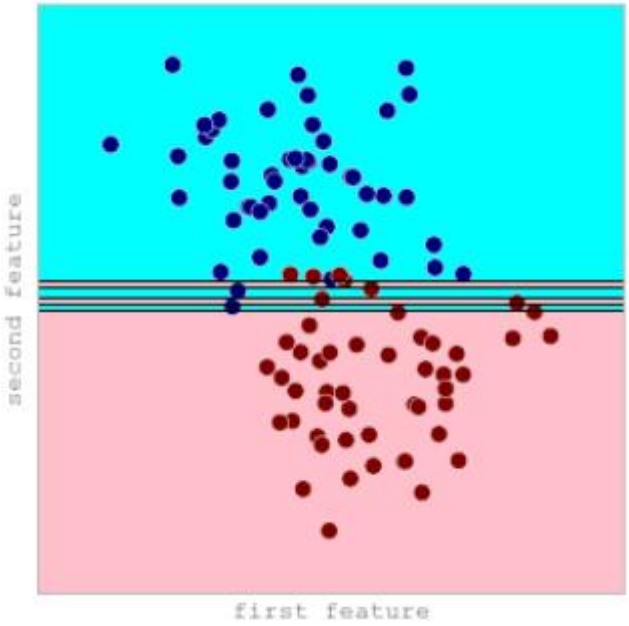
Вопрос: Что подозрительного в kNN?

Параметры алгоритмов

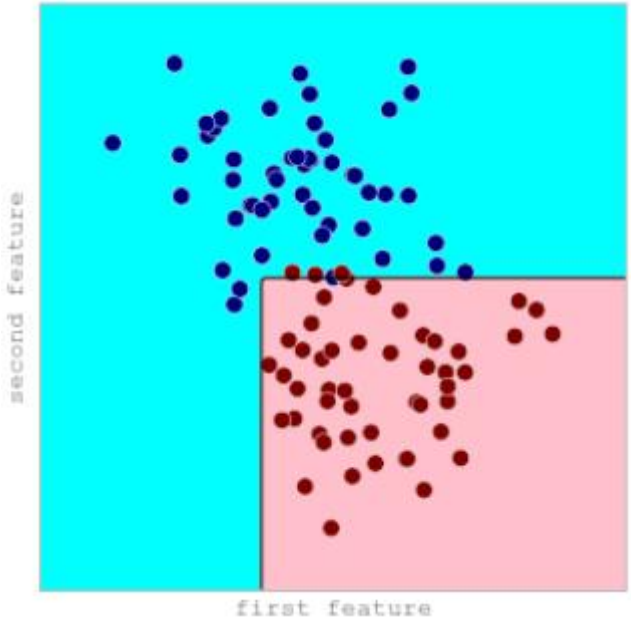
Каждый алгоритм имеет кучу параметров…

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

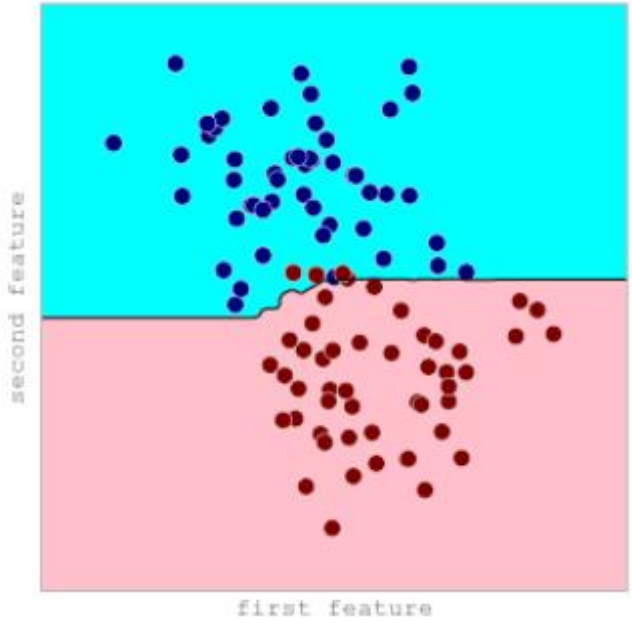
RandomForestClassifier(n_estimators=1)



RandomForestClassifier(n_estimators=5)



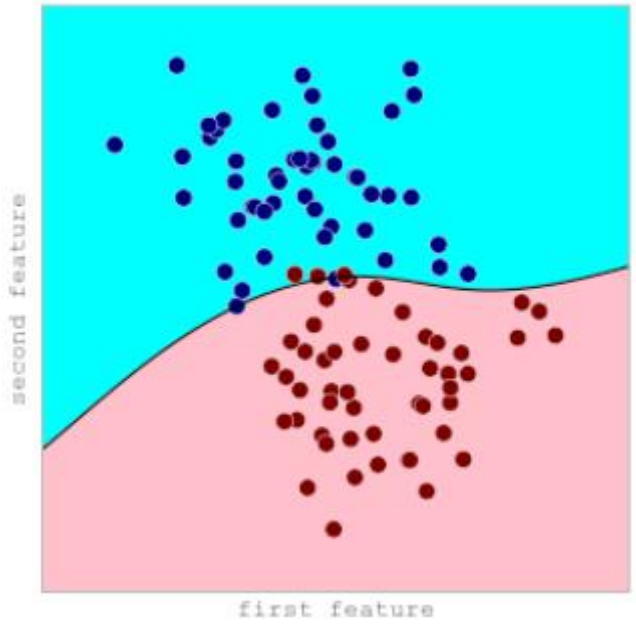
RandomForestClassifier(n_estimators=100)



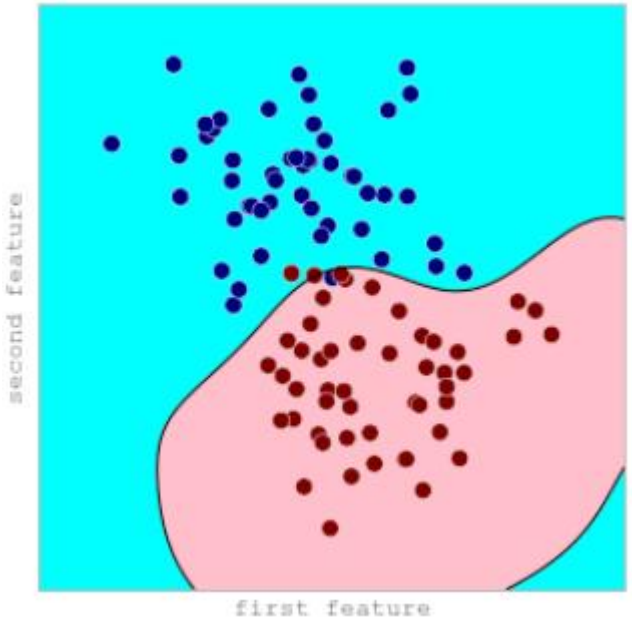
Параметры алгоритмов

Некоторые параметры очень существенные

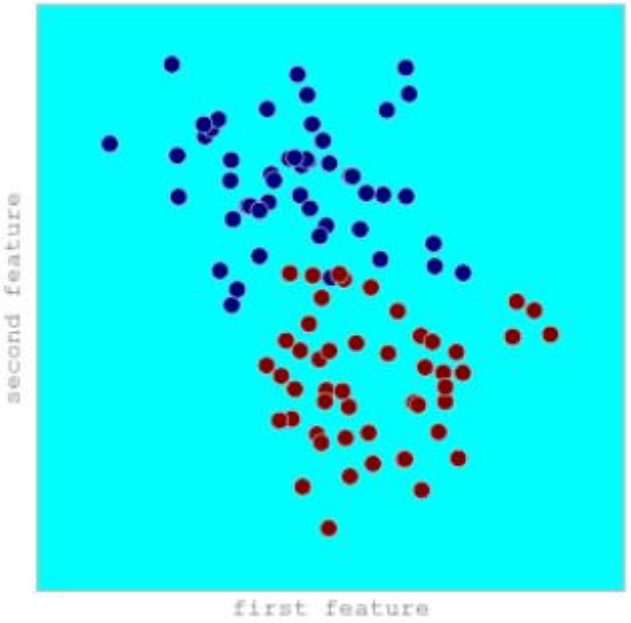
```
svm = SVC(kernel='poly')
```




```
svm = SVC(kernel='rbf')
```




```
svm = SVC(kernel='sigmoid')
```

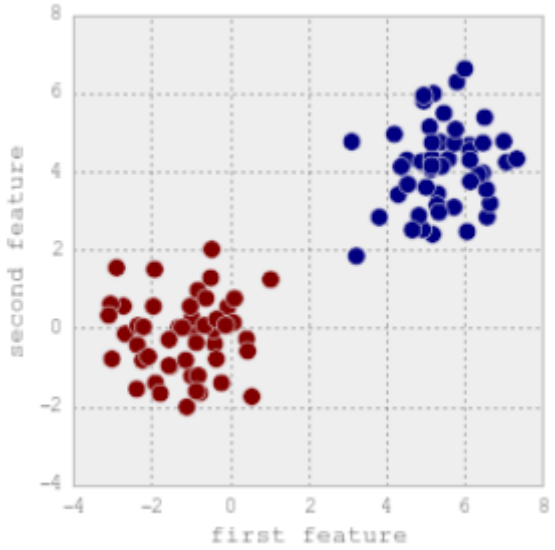


Какое ядро не изображено?

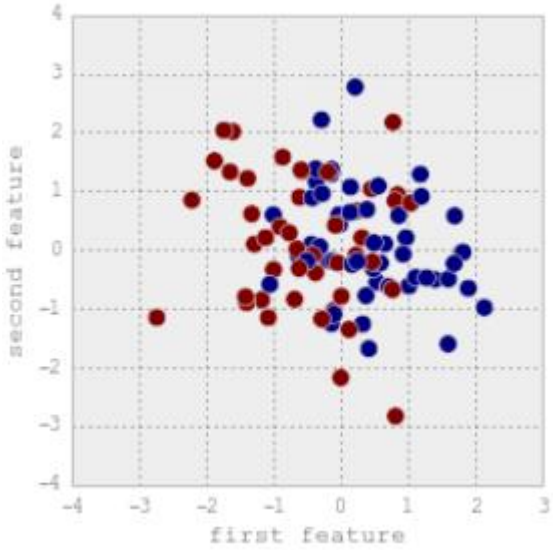
Как быть с сигмоидой?

Встроенные датасеты

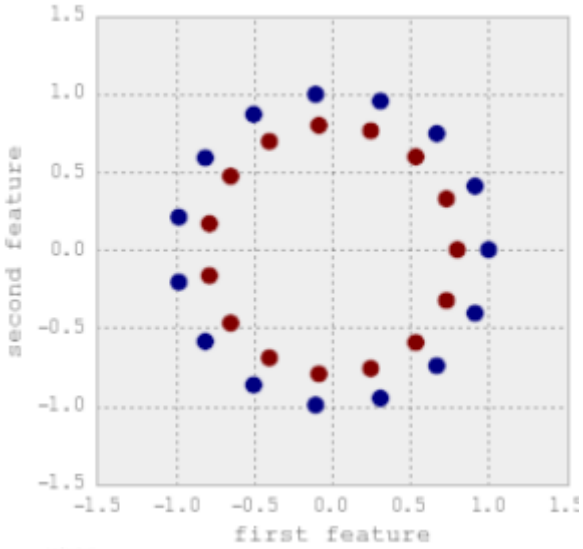
```
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=2)
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
plt.xlabel("first feature")
plt.ylabel("second feature")
```



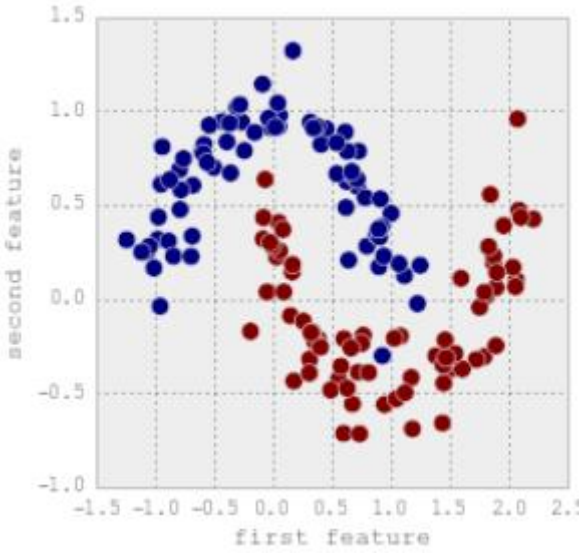

```
from sklearn.datasets import
make_classification

X, y = make_classification(100)
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
plt.xlabel("first feature")
plt.ylabel("second feature")
```



```
from sklearn.datasets import make_circles

X, y = make_circles(30)
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
plt.xlabel("first feature")
plt.ylabel("second feature")
```




```
from sklearn.datasets import make_moons
X, y = make_moons(150, noise=0.15)
plt.scatter(X[:, 0], X[:, 1], c=y, s=75)
plt.xlabel("first feature")
plt.ylabel("second feature")
```



Есть много и других (регрессия, sparse-матрицы и т.п.)

Разбиения выборок: cross_validation

Зачем нужны разбиения

```
from sklearn.cross_validation import
StratifiedKFold

# когда дисбаланс классов
y = np.array([1,1,2,2,2,2,3,3,3,3,3,3])
cv = StratifiedKFold(y, n_folds=2)
print(cv)

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

```
sklearn.cross_validation.StratifiedKFold(labels=[1
1 2 2 2 2 3 3 3 3 3 3], n_folds=2, shuffle=False,
random_state=None)
индексы теста = [0 2 3 6 7 8]
классы теста = [1 2 2 3 3 3]
индексы теста = [ 1  4  5  9 10 11]
классы теста = [1 2 2 3 3 3]
```

Какие бывают разбиения

```
from sklearn.cross_validation import KFold
for train, test in KFold(12,2):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[test]))
```

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

```
from sklearn.cross_validation import
LeaveOneOut # KFold(n, n_folds=n)
for train, test in LeaveOneOut(12):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[test]))
```

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

```
from sklearn.cross_validation import LeavePOut
for train, test in LeavePOut(12,3):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[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]
...
```

```
from sklearn.cross_validation import
ShuffleSplit
for train, test in ShuffleSplit(12, 3):
    print('индексы теста = ' + str(test))
    print('классы теста = ' + str(y[test]))
```

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

Ещё в cross_validation

cross_validation.train_test_split – от матрицы

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

cross_validation.cross_val_predict – формирование cv-мета-признаков

Оценка модели

```
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import ShuffleSplit

cv = ShuffleSplit(len(y), n_iter=5, test_size=.2)
cross_val_score(logreg, X, y, cv=cv)
```

```
array([0.85, 1. , 0.9 , 0.95,  1. ])
```

У этих функций много параметров…

Они (функции) «понимают» друг друга

Пока не указываем скорер – используется встроенный (в модель)

Скореры в оценке модели

```
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=5)
```

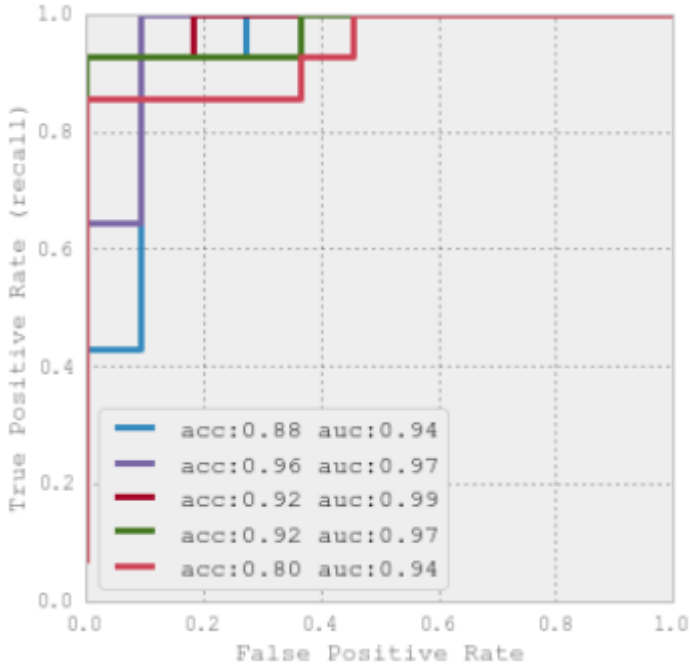
```
dict_keys(['mean_squared_error',
'log_loss', 'f1_macro', 'precision',
'precision_weighted',
'precision_samples',
'average_precision', 'recall_micro',
'f1_micro', 'accuracy',
'mean_absolute_error',
'recall_weighted', 'recall',
'f1_weighted', 'median_absolute_error',
'adjusted_rand_score', 'f1_samples',
'precision_macro', 'recall_samples',
'r2', 'f1', 'recall_macro',
'precision_micro', 'roc_auc'])

array([ 0.95,  0.9 ,  1.  ,  0.95,
0.95])
```

Есть много скореров

Можно написать свой

Качество: metrics

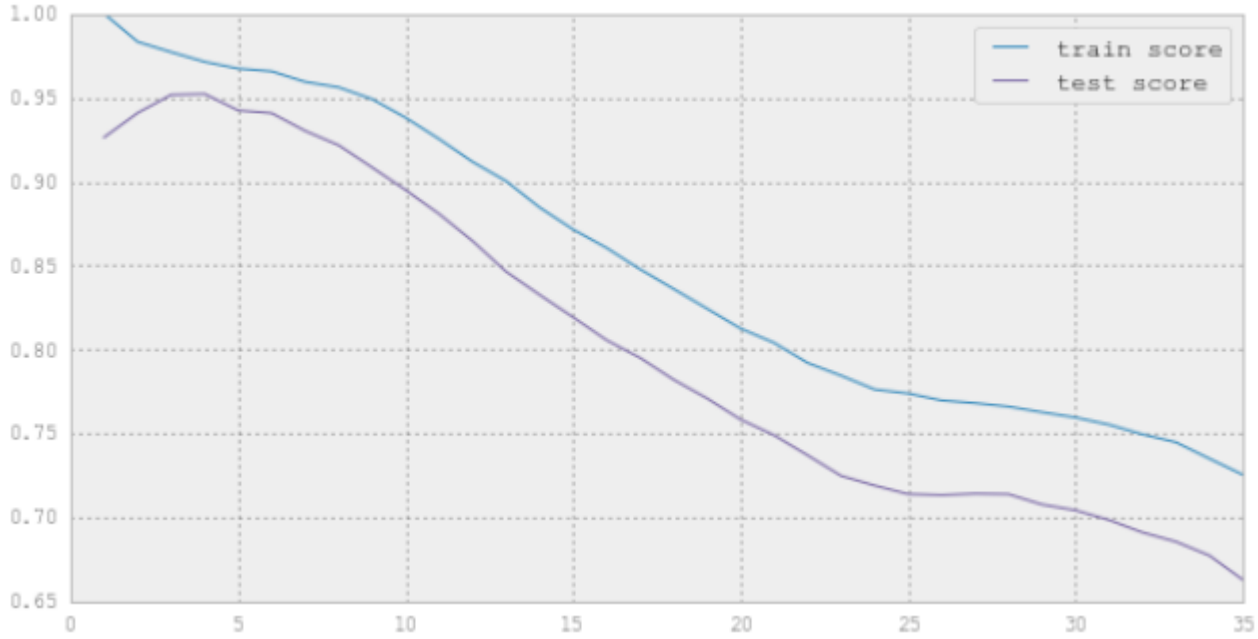
```
from sklearn.metrics import roc_curve, roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

for gamma in [.01, .05, 1, 2, 5]:
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate (recall)")
    svm = SVC(gamma=gamma).fit(X_train, y_train)
    decision_function = svm.decision_function(X_test)
    fpr, tpr, _ = roc_curve(y_test, decision_function)
    acc = svm.score(X_test, y_test)
    auc = roc_auc_score(y_test, svm.decision_function(X_test))
    plt.plot(fpr, tpr, label="acc:%.2f auc:%.2f" % (acc, auc), linewidth=3)
  plt.legend(loc="best")
```



Есть ещё много метрик…

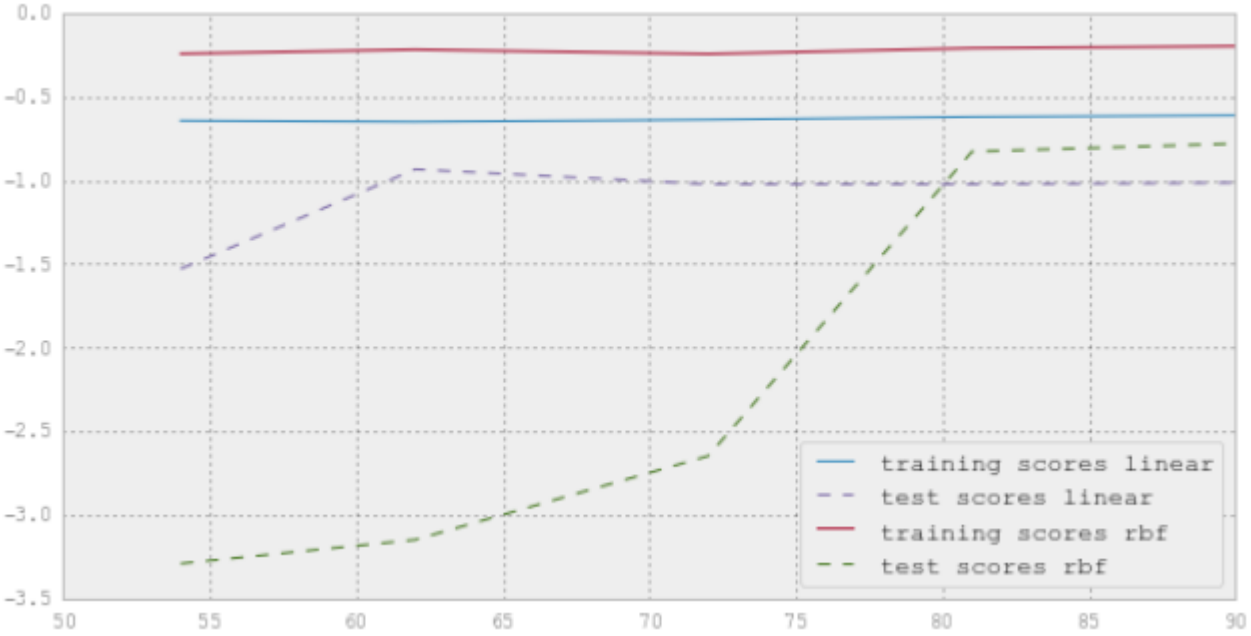
Выбор параметров модели: learning_curve.validation_curve

```
from sklearn.learning_curve import validation_curve
from sklearn.cross_validation import KFold
from sklearn.neighbors import KNeighborsRegressor

cv = KFold(n=len(x), shuffle=True)
n_neighbors = [i+1 for i in range(35)]
train_errors, test_errors = validation_curve(KNeighborsRegressor(), X, y,
param_name="n_neighbors",
                                             param_range=n_neighbors, cv=cv)
plt.plot(n_neighbors, train_errors.mean(axis=1), label="train score")
plt.plot(n_neighbors, test_errors.mean(axis=1), label="test score")
plt.legend(loc="best")
```



Выбор параметров модели: learning_curve.validation_curve

```
from sklearn.learning_curve import learning_curve
from sklearn.svm import SVR
# Правда отрицательные MSE (т.к. хотим максимизировать)
training_sizes, train_scores, test_scores = learning_curve(SVR(kernel='linear'), X, y, cv=10,
                                                           scoring="mean_squared_error",
                                                           train_sizes=[.6, .7, .8, .9, 1.])

training_sizes_2, train_scores_2, test_scores_2 = learning_curve(SVR(kernel='rbf'), X, y, cv=10,
                                                           scoring="mean_squared_error",
                                                           train_sizes=[.6, .7, .8, .9, 1.])
print(train_scores.mean(axis=1))
plt.plot(training_sizes, train_scores.mean(axis=1), label="training scores linear", c='red')
plt.plot(training_sizes, test_scores.mean(axis=1), label="test scores linear", ls='--', c='red')
plt.plot(training_sizes_2, train_scores_2.mean(axis=1), label="training scores rbf", c='blue')
plt.plot(training_sizes_2, test_scores_2.mean(axis=1), label="test scores rbf", c='blue', ls='--')
plt.legend(loc='best')
```



Выбор параметров модели: learning_curve.validation_curve

```
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVR

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1, 1]}
cv = KFold(n=len(X), n_folds=5, shuffle=True)

grid = GridSearchCV(SVR(), param_grid=param_grid, cv=cv, verbose=3)

grid.fit(X, y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] gamma=0.001, C=0.001 ............................................
[CV] .................. gamma=0.001, C=0.001, score=-0.076544 -   0.0s
[CV] gamma=0.001, C=0.001 ............................................
[CV] .................. gamma=0.001, C=0.001, score=-0.001319 -   0.0s
[CV] gamma=0.001, C=0.001 ............................................
...
```

```
print(grid.best_score_)
print(grid.best_params_)
print(grid.score(X_test, y_test))
```

```
0.958154154548
{'gamma': 1, 'C': 10}
0.963548256612
```

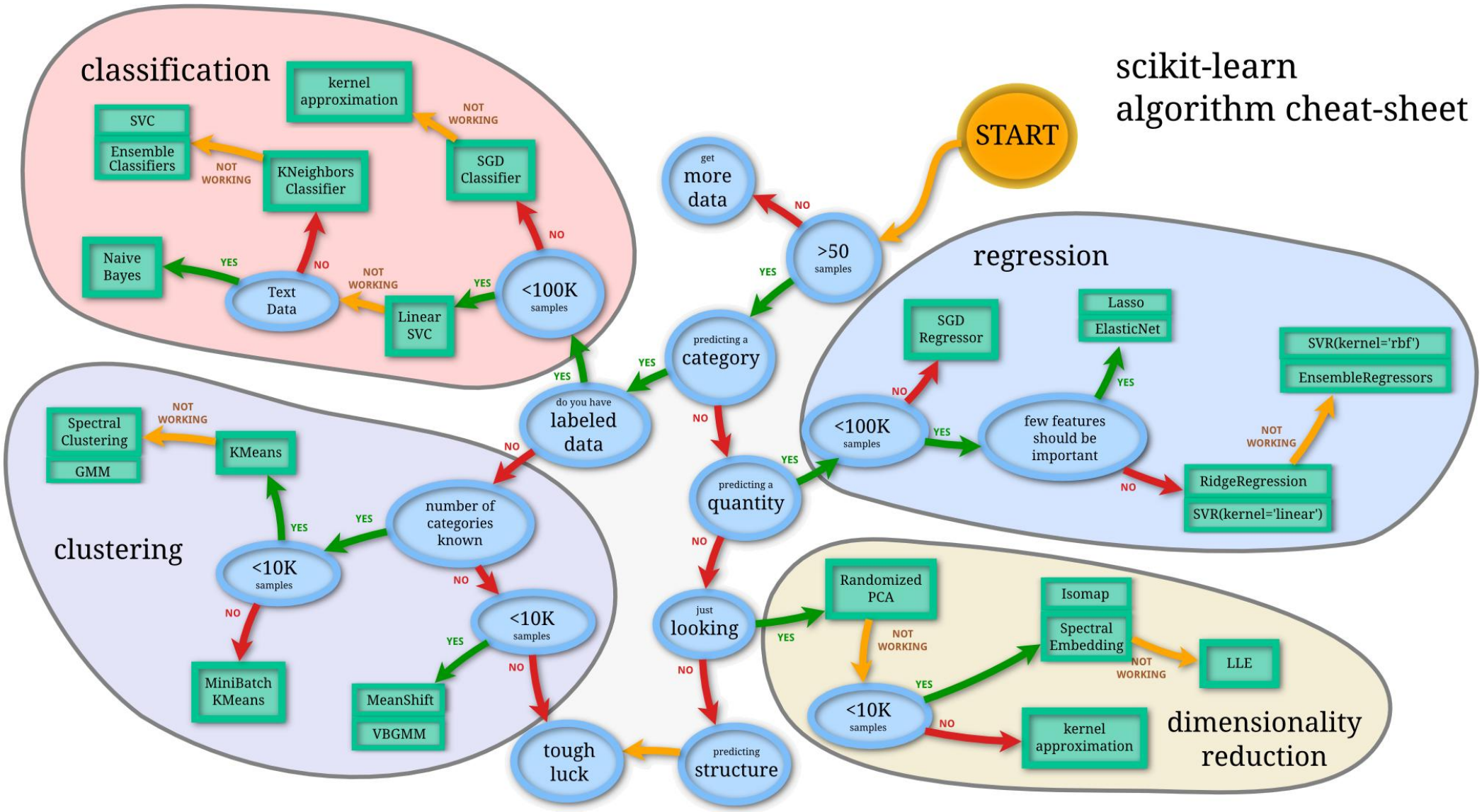
Последовательность операторов: pipeline

```
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.grid_search import GridSearchCV

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(text_train, y_train)
print(grid.best_params_)
grid.score(text_test, y_test)
```

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)
```

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).todense()
```

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

Полиномиальные признаки

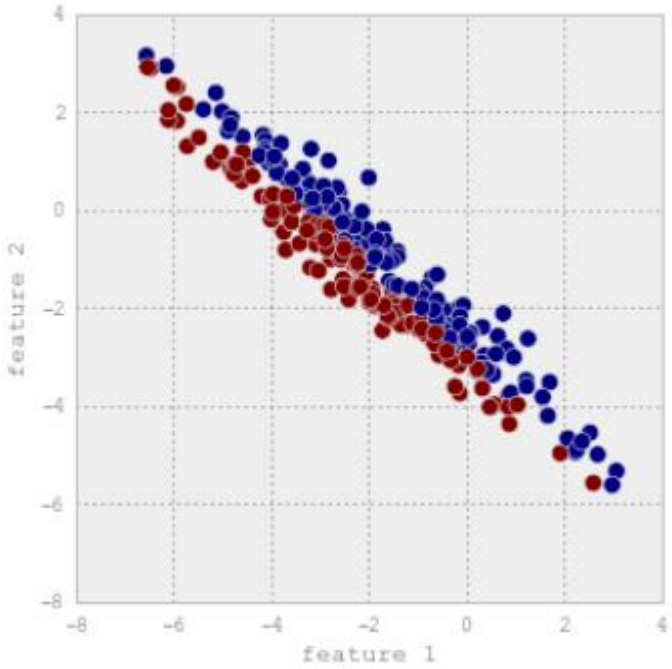
```
from sklearn.preprocessing import PolynomialFeatures
pf = PolynomialFeatures(degree=2)
f = [[1,0],[2,1],[3,2]]
pf.fit(f)
pf.transform(f)
```

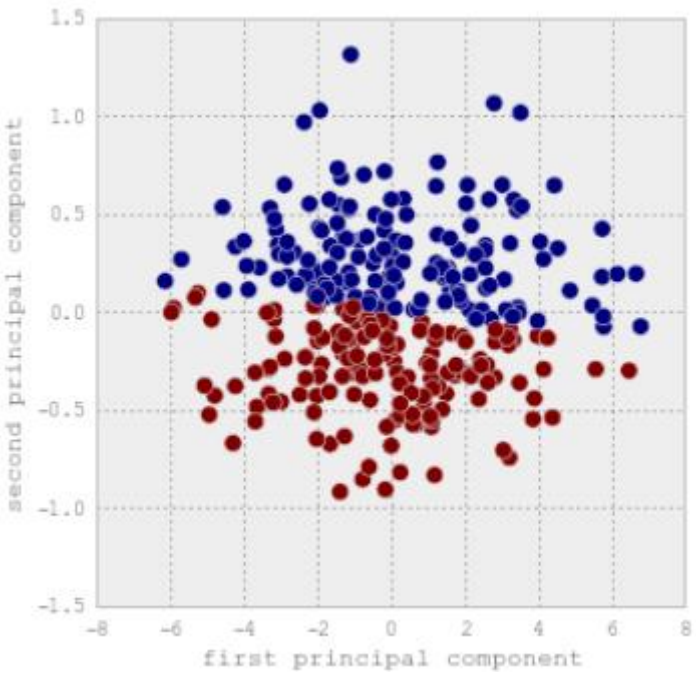
```
array([[1, 1, 0, 1, 0, 0],
       [1, 2, 1, 4, 2, 1],
       [1, 3, 2, 9, 6, 4]])
```

Декомпозиции матриц: decomposition

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

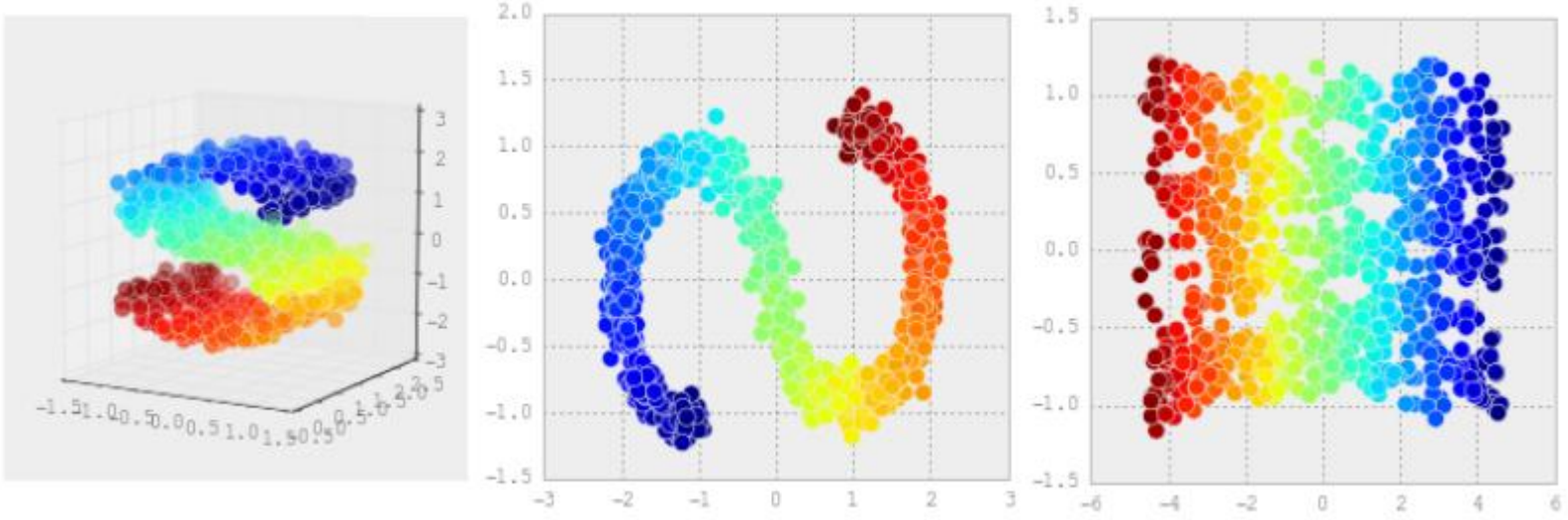
```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X_blob)
X_pca = pca.transform(X_blob)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, linewidths=0, s=70)
plt.xlabel("first principal component")
plt.ylabel("second principal component")
```





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)
```



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)

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.  ]]

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\\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).

Ссылки

В данной презентации много примеров взято из ноутбука

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

Спасибо Андреасу Мюллеру!

Инструкции по настройке конкретных моделей –
в слайдах курсов АМА и ПЗАД.

Ссылка на презентацию дана в блоге автора
https://alexanderdyakonov.wordpress.com/