Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Отчет по лабораторной работе № 4

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

по курсу

“Методы машинного обучения”

Выполнил:
Али Диб А.Ж.
Студент группы ИУ5-22М

Москва, 2020

```
import  numpy  as  np
import  pandas  as  pd
from  typing  import  Dict ,  Tuple
from  scipy  import  stats
from  sklearn.datasets  import  *
from  sklearn.model_selection  import  train_test_split
from  sklearn.neighbors  import  KNeighborsRegressor ,  KNeighborsClassifier
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut, S
from  sklearn.metrics  import  accuracy_score ,  balanced_accuracy_score
from sklearn.metrics  import  plot_confusion_matrix
from  sklearn.metrics  import  precision_score ,  recall_score ,  f1_score ,  class
from  sklearn.metrics  import  confusion_matrix
from  sklearn.metrics  import  mean_absolute_error ,  mean_squared_error ,  mean_sq
from  sklearn.metrics  import  roc_curve ,  roc_auc_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import learning_curve, validation_curve
import  seaborn  as  sns
import  matplotlib.pyplot  as  plt
% matplotlib inline
sns . set ( style = "ticks" )
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Future
  import pandas.util.testing as tm
```

Изучение качества классификации

Решение задачи классификации - это предсказание значений качественного (категориалы

Подготовка данных и построение базовых моделей для оценки

Будем использовать набор данных "Breast cancer wisconsin".

```
#https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_can
cancer = load_breast_cancer()
```

```
# Наименования признаков
cancer.feature_names
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
# Значения признаков
cancer.data[:3]
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, 1.326e+03, 8.474e-02, 7.864e-02,
        8.690e-02, 7.017e-02, 1.812e-01, 5.667e-02, 5.435e-01, 7.339e-01,
        3.398e+00, 7.408e+01, 5.225e-03, 1.308e-02, 1.860e-02, 1.340e-02,
        1.389e-02, 3.532e-03, 2.499e+01, 2.341e+01, 1.588e+02, 1.956e+03,
        1.238e-01, 1.866e-01, 2.416e-01, 1.860e-01, 2.750e-01, 8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02]])
```

```
type(cancer.data)
```

```
numpy.ndarray
```

```
# Значения целевого признака
np.unique(cancer.target)
```

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

```
# Наименования значений целевого признака
cancer.target_names
```

```
array(['malignant', 'benign'], dtype='<U9')
```

```
list(zip(np.unique(cancer.target), cancer.target_names))
```

```
[(0, 'malignant'), (1, 'benign')]
```

```
# Значения целевого признака
cancer.target[:20]
```

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

```
# Размер выборки
cancer.data.shape, cancer.target.shape
```

```
((569, 30), (569,))
```

```
# Сформируем DataFrame
cancer_df = pd.DataFrame(data= np.c_[cancer['data'], cancer['target']],
                         columns= list(cancer['feature_names']) + ['target'])
```

```
# И выведем его статистические характеристики
cancer_df.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	conc
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.

Разделение выборки на обучающую и тестовую

Для разделения выборки на обучающую и тестовую используется функция train_test_split.

```
cancer_X_train, cancer_X_test, cancer_y_train, cancer_y_test = train_test_split(
    cancer.data, cancer.target, test_size=0.5, random_state=1)
```

Как правило, параметр test_size устанавливают в 20% или 30%. Здесь используется test_size
"ухудшить" результат на хорошем наборе данных и показать различные возможности испол

Параметр random_state позволяет задавать базовое значение для генератора случайных чи
Если задается параметр random_state то результаты разбиения будут одинаковыми при раз
параметр удобно использовать для создания "устойчивых" учебных примеров, которые выд
различных запусках.

```
# Размер обучающей выборки
cancer_X_train.shape, cancer_y_train.shape
```

```
((284, 30), (284,))
```

```
# Размер тестовой выборки
cancer_X_test.shape, cancer_y_test.shape
```

```
((285, 30), (285,))
```

Функция train_test_split разделила исходную выборку таким образом, чтобы в обучающей и

```
np.unique(cancer_y_train)
```

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

```
np.unique(cancer_y_test)
```

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

```
def class_proportions(array: np.ndarray) -> Dict[int, Tuple[int, float]]:
    """
    Вычисляет пропорции классов
    array - массив, содержащий метки классов
    """
    # Получение меток классов и количества меток каждого класса
    labels, counts = np.unique(array, return_counts=True)
    # Превращаем количество меток в процент их встречаемости
    # делим количество меток каждого класса на общее количество меток
    counts_perc = counts/array.size
    # Теперь sum(counts_perc)==1.0
    # Создаем результирующий словарь,
    # ключом словаря является метка класса,
    # а значением словаря процент встречаемости метки
    res = dict()
    for label, count2 in zip(labels, zip(counts, counts_perc)):
        res[label] = count2
    return res

def print_class_proportions(array: np.ndarray):
    """
    Вывод пропорций классов
    """
    proportions = class_proportions(array)
    if len(proportions)>0:
        print('Метка \t Количество \t Процент встречаемости')
    for i in proportions:
        val, val_perc = proportions[i]
        val_perc_100 = round(val_perc * 100, 2)
        print('{} \t {} \t \t {}%'.format(i, val, val_perc_100))


# В исходной выборке нет явного дисбаланса классов для целевого признака
print_class_proportions(cancer.target)

# Функция train_test_split разделила исходную выборку таким образом,
# чтобы в обучающей и тестовой частях сохранились пропорции классов.
```

```
Метка    Количество      Процент встречаемости
0        212             37.26%
1        357             62.74%
```

```
# Для обучающей выборки
print_class_proportions(cancer_y_train)
```

```
Метка      Количество      Процент встречаемости
0          109             38.38%
1          175             61.62%
```

```
# Для тестовой выборки
print_class_proportions(cancer_y_test)
```

```
Метка      Количество      Процент встречаемости
0          103             36.14%
1          182             63.86%
```

Построим базовые модели на основе метода ближайших соседей

2 cells hidden

Метрики качества классификации

1) Accuracy

Метрика вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классо

```
# cancer_y_test - эталонное значение классов из исходной (тестовой) выборки
# target* - предсказанное значение классов

# 2 ближайших соседа
accuracy_score(cancer_y_test, target1_1)
```

```
0.8842105263157894
```

```
# 10 ближайших соседей
accuracy_score(cancer_y_test, target1_2)
```

```
0.9157894736842105
```

Точность в случае 10 ближайших соседей составляет более 91%, а точность в случае 2 ближ

Метрика "Accuracy" показывает точность по всем классам, но точность может быть различ

Это очень серьезная проблема, которая часто возникает на несбалансированных выборках

```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
```

```
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))


# 2 ближайших соседа
print_accuracy_score_for_classes(cancer_y_test, target1_1)
```

```
Метка     Accuracy
0         0.883495145631068
1         0.8846153846153846
```

Accuracy для классов 0 и 1 составляет 88%.

```
# 10 ближайших соседей
print_accuracy_score_for_classes(cancer_y_test, target1_2)
```

```
Метка     Accuracy
0         0.8446601941747572
1         0.9560439560439561
```

Accuracy для класса 0 составляет 88%, но для классов 1 95%.

Вывод.

Метрика Accuracy интуитивно понятна и часто используется на практике. Но если количест
лучше всего вычислять Accuracy отдельно для каждого класса.

2) Матрица ошибок или Confusion Matrix

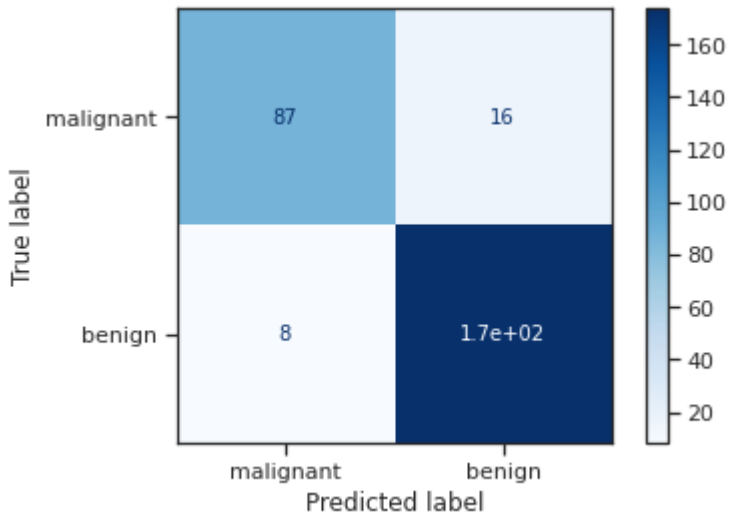
Количество верно и ошибочно классифицированных данных, представленное в виде матри

```
tn, fp, fn, tp = confusion_matrix(cancer_y_test, target1_2).ravel()
tn, fp, fn, tp
```

```
(87, 16, 8, 174)
```

```
plot_confusion_matrix(cl1_2, cancer_X_test, cancer_y_test,
                      display_labels=cancer.target_names, cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fae0d643f
```




```
plot_confusion_matrix(cl1_2, cancer_X_test, cancer_y_test,
                      display_labels=cancer.target_names,
                      cmap=plt.cm.Blues, normalize='true')
```

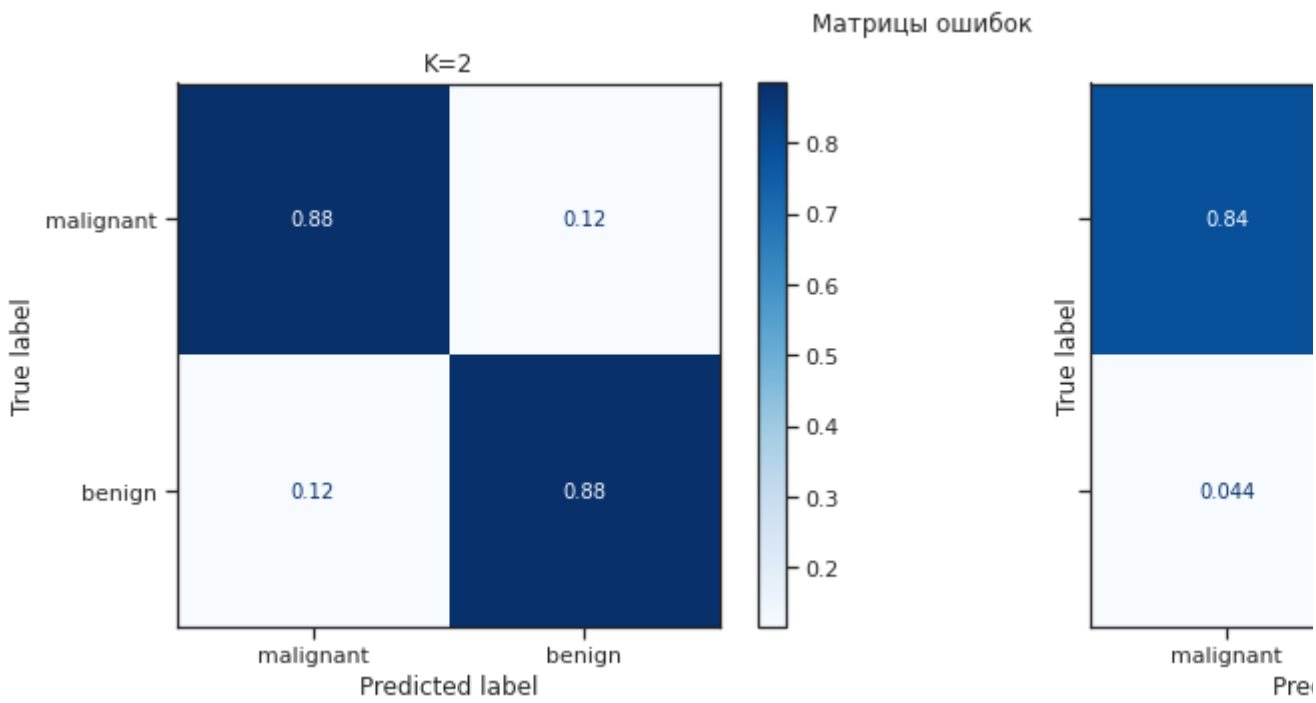
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fae0d54e5
```

```
fig, ax = plt.subplots(1, 2, sharex='col', sharey='row', figsize=(15,5))

plot_confusion_matrix(cl1_1, cancer_X_test, cancer_y_test,
                      display_labels=cancer.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[0])

plot_confusion_matrix(cl1_2, cancer_X_test, cancer_y_test,
                      display_labels=cancer.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[1])

fig.suptitle('Матрицы ошибок')
ax[0].title.set_text('K=2')
ax[1].title.set_text('K=10')
```



3) Precision, recall и F-мера

```
# По умолчанию метрики считаются для 1 класса бинарной классификации
# Для 2 ближайших соседей
precision_score(cancer_y_test, target1_1), recall_score(cancer_y_test, target1_1)
```

```
(0.930635838150289, 0.8846153846153846)
```

```
# Для 10 ближайших соседей
precision_score(cancer_y_test, target1_2), recall_score(cancer_y_test, target1_2)
```

```
(0.9157894736842105, 0.9560439560439561)
```

```
# Параметры TP, TN, FP, FN считаются как сумма по всем классам
```

```
precision_score(cancer_y_test, target1_1, average='micro')
```

```
0.8842105263157894
```

```
# Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется среднее значение, дисбаланс классов не учитывается.
precision_score(cancer_y_test, target1_1, average='macro')
```

```
0.8715679190751445
```

```
# Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется средневзвешенное значение, дисбаланс классов учитывается
# в виде веса классов (вес - количество истинных значений каждого класса).
precision_score(cancer_y_test, target1_1, average='weighted')
```

```
0.8879411317310617
```

F-мера

```
f1_score(cancer_y_test, target1_2)
```

```
0.935483870967742
```

```
f1_score(cancer_y_test, target1_1, average='micro')
```

```
0.8842105263157894
```

```
f1_score(cancer_y_test, target1_1, average='macro')
```

```
0.8767769407140518
```

```
f1_score(cancer_y_test, target1_1, average='weighted')
```

```
0.8851662730360129
```

Функция classification_report позволяет выводить значения точности, полноты и F-меры для

```
classification_report(cancer_y_test, target1_1,
                      target_names=cancer.target_names, output_dict=True)
```

```
{'accuracy': 0.8842105263157894,
 'benign': {'f1-score': 0.9070422535211268,
  'precision': 0.930635838150289,
  'recall': 0.8846153846153846,
  'support': 182},
 'macro avg': {'f1-score': 0.8767769407140518,
  'precision': 0.8715679190751445,
  'recall': 0.8840552651232263,
```

Стратегии кросс-валидации

```
  'support': 103}
```

```
scoring = {'precision': 'precision_weighted',
           'recall': 'recall_weighted',
           'f1': 'f1_weighted'}
```

K-fold

```
kf = KFold(n_splits=5)
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        cancer.data, cancer.target, scoring=scoring,
                        cv=kf, return_train_score=True)
scores
```

```
{'fit_time': array([0.00461268, 0.00116181, 0.00116491, 0.00106263, 0.001091
 'score_time': array([0.01139617, 0.00795913, 0.00789094, 0.00695252, 0.006875
 'test_f1': array([0.87833964, 0.9122807 , 0.94764978, 0.90663256, 0.89111744]
 'test_precision': array([0.89296517, 0.9122807 , 0.94855194, 0.9172852 , 0.91
 'test_recall': array([0.87719298, 0.9122807 , 0.94736842, 0.90350877, 0.88495
 'train_f1': array([0.97605126, 0.96732351, 0.96727296, 0.9650388 , 0.97162092
 'train_precision': array([0.97753988, 0.96981109, 0.96967738, 0.96766249, 0.9
 'train_recall': array([0.97582418, 0.96703297, 0.96703297, 0.96483516, 0.9714
```

Repeated K-Fold

```
kf = RepeatedKFold(n_splits=5, n_repeats=2)
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        cancer.data, cancer.target, scoring=scoring,
                        cv=kf, return_train_score=False)
scores, np.mean(scores['test_precision'])
```

```
({'fit_time': array([0.00244737, 0.0010705 , 0.00112605, 0.00106359, 0.0010447
         0.00108576, 0.00103593, 0.00109339, 0.00103092, 0.00102997]),
  'score_time': array([0.00895882, 0.00707459, 0.00692487, 0.00701952, 0.00689
         0.00706792, 0.00712609, 0.00691986, 0.00688624, 0.00931263]),
  'test_f1': array([0.91479967, 0.95642996, 0.89698771, 0.94716501, 0.89304339
         0.90574567, 0.90239212, 0.90580438, 0.91260832, 0.93803367]),
  'test_precision': array([0.93177388, 0.95809814, 0.90829731, 0.94757348, 0.8
         0.91951025, 0.90271543, 0.9153053 , 0.9151893 , 0.93816143]),
  'test_recall': array([0.9122807 , 0.95614035, 0.89473684, 0.94736842, 0.8938
         0.90350877, 0.90350877, 0.90350877, 0.9122807 , 0.9380531 ])},
 0.9234099446595081)
```

ShuffleSplit

Генерируется N случайных перемешиваний данных, в каждом перемешивании заданная до

```
kf = ShuffleSplit(n_splits=5, test_size=0.25)
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        cancer.data, cancer.target, scoring=scoring,
                        cv=kf, return_train_score=False)
scores
```

```
{'fit_time': array([0.00182581, 0.00102854, 0.00102663, 0.00116014, 0.00100994
 'score_time': array([0.01241708, 0.00808048, 0.00803518, 0.00846529, 0.008014
 'test_f1': array([0.87412587, 0.92318645, 0.91635089, 0.89531463, 0.9020979 ]
 'test_precision': array([0.87412587, 0.92338898, 0.91698836, 0.89561654, 0.90
 'test_recall': array([0.87412587, 0.92307692, 0.91608392, 0.8951049 , 0.90209
```

Оптимизация гиперпараметров

Grid Search

```
n_range = np.array(range(1,15,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accu
clf_gs.fit(cancer_X_train, cancer_y_train)
```

```
CPU times: user 284 ms, sys: 0 ns, total: 284 ms
Wall time: 286 ms
```

```
clf_gs.cv_results_
```

```
{'mean_fit_time': array([0.00091767, 0.00068064, 0.00072966, 0.0006598 , 0.000
       0.00063553, 0.00064073, 0.00064344, 0.00064263, 0.0006258 ,
       0.00078754, 0.00063276, 0.0006557 , 0.00062914]),
 'mean_score_time': array([0.0035708 , 0.00283265, 0.00283136, 0.00281153, 0.0
       0.00284863, 0.00284986, 0.00286865, 0.00285835, 0.0029449 ,
       0.00350924, 0.00289102, 0.00313835, 0.0030261 ]),
 'mean_test_score': array([0.92957393, 0.91898496, 0.94367168, 0.92957393, 0.9
       0.92957393, 0.93652882, 0.93652882, 0.94003759, 0.94003759,
       0.93646617, 0.92938596, 0.93646617, 0.93295739]),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
  {'n_neighbors': 12},
  {'n_neighbors': 13},
  {'n_neighbors': 14}],
 'rank_test_score': array([10, 14,  1, 10,  2, 10,  5,  5,  3,  3,  7, 13,  7,
       dtype=int32),
 'split0_test_score': array([0.94736842, 0.92982456, 0.92982456, 0.9122807 , 0
       0.89473684, 0.9122807 , 0.89473684, 0.92982456, 0.9122807 ,
       0.92982456, 0.9122807 , 0.92982456, 0.89473684]),
 'split1_test_score': array([0.85964912, 0.89473684, 0.92982456, 0.89473684, 0
       0.9122807 , 0.92982456, 0.94736842, 0.94736842, 0.96491228,
       0.96491228, 0.96491228, 0.96491228, 0.96491228]),
 'split2_test_score': array([0.94736842, 0.9122807 , 0.92982456, 0.92982456, 0
       0.92982456, 0.94736842, 0.94736842, 0.94736842, 0.94736842,
       0.92982456, 0.92982456, 0.92982456, 0.94736842]),
 'split3_test_score': array([0.96491228, 0.94736842, 0.98245614, 0.98245614, 0
       0.98245614, 0.98245614, 0.98245614, 0.96491228, 0.96491228,
       0.96491228, 0.96491228, 0.96491228, 0.96491228]),
 'split4_test_score': array([0.92857143, 0.91071429, 0.94642857, 0.92857143, 0
       0.92857143, 0.91071429, 0.91071429, 0.91071429, 0.91071429,
       0.89285714, 0.875     , 0.89285714, 0.89285714]),
```

```
# Лучшая модель
clf_gs.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

```
clf_gs.best_score_
```

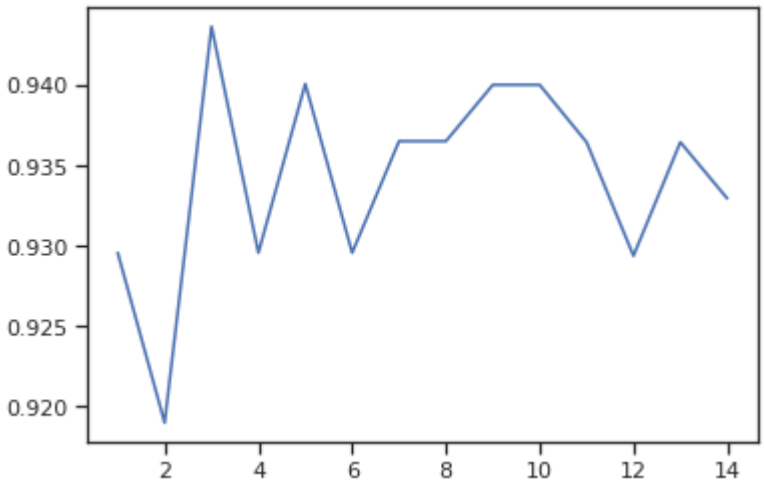
```
0.9436716791979951
```

```
# Лучшее значение параметров
clf_gs.best_params_
```

```
{'n_neighbors': 3}
```

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fae0ceb89e8>]
```



Grid search с стратегией KFold кросс-валидации

```
kf = KFold(n_splits=10)
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=kf, scoring='acc
clf_gs.fit(cancer_X_train, cancer_y_train)
```

```
CPU times: user 417 ms, sys: 731 µs, total: 418 ms
Wall time: 422 ms
```

```
clf_gs.cv_results_
```

```
                dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
  {'n_neighbors': 12},
  {'n_neighbors': 13},
  {'n_neighbors': 14}],
 'rank_test_score': array([ 3, 14,  7,  7,  1,  9,  2,  6,  3,  3, 10, 11, 13,
       dtype=int32),
 'split0_test_score': array([0.96551724, 0.96551724, 0.96551724, 0.93103448, 0
        0.96551724, 0.96551724, 0.96551724, 0.96551724, 0.96551724,
        0.96551724, 0.96551724, 0.96551724, 0.96551724]),
 'split1_test_score': array([0.93103448, 0.89655172, 0.82758621, 0.86206897, 0
        0.79310345, 0.86206897, 0.79310345, 0.86206897, 0.86206897,
        0.86206897, 0.86206897, 0.86206897, 0.86206897]),
 'split2_test_score': array([0.93103448, 0.93103448, 1.        , 1.        , 0
        0.96551724, 0.96551724, 0.96551724, 0.93103448, 0.93103448,
        0.93103448, 0.96551724, 0.93103448, 0.96551724]),
 'split3_test_score': array([0.93103448, 0.89655172, 0.96551724, 0.96551724, 0
        1.        , 0.96551724, 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ]),
 'split4_test_score': array([0.92857143, 0.89285714, 0.92857143, 0.92857143, 0
        0.92857143, 0.92857143, 0.92857143, 0.92857143, 0.92857143,
        0.92857143, 0.92857143, 0.92857143, 0.92857143, 0.92857143]),
 'split5_test_score': array([0.96428571, 0.92857143, 0.92857143, 0.92857143, 0
        0.92857143, 0.96428571, 0.96428571, 0.96428571, 0.96428571,
        0.92857143, 0.92857143, 0.92857143, 0.92857143, 0.92857143]),
 'split6_test_score': array([0.96428571, 1.        , 1.        , 1.        , 1
        1.        , 1.        , 1.        , 1.        , 1.        ,
        0.96428571, 0.96428571, 0.96428571, 0.96428571, 0.96428571]),
 'split7_test_score': array([0.96428571, 0.96428571, 0.96428571, 0.96428571, 0
        0.96428571, 0.96428571, 0.96428571, 0.96428571, 0.96428571,
        0.96428571, 0.96428571, 0.96428571, 0.96428571, 0.96428571]),
 'split8_test_score': array([0.89285714, 0.89285714, 0.89285714, 0.92857143, 0
        0.92857143, 0.92857143, 0.92857143, 0.92857143, 0.92857143,
        0.92857143, 0.92857143, 0.92857143, 0.89285714, 0.89285714]),
 'split9_test_score': array([0.96428571, 0.89285714, 0.92857143, 0.89285714, 0
        0.89285714, 0.89285714, 0.89285714, 0.89285714, 0.89285714,
        0.89285714, 0.89285714, 0.85714286, 0.85714286, 0.85714286]),
 'std_fit_time': array([3.22154265e-04, 6.49448884e-06, 1.27236247e-05, 3.0463
        1.63417903e-04, 5.95168468e-05, 1.48803757e-05, 9.58264747e-06,
        1.77362464e-04, 1.43622507e-04, 7.30205700e-05, 2.97646228e-04,
        7.60436572e-05, 2.16525028e-05]),
 'std_score_time': array([7.05335495e-04, 1.88612034e-04, 1.00456010e-04, 7.60
        1.97849069e-04, 1.82424455e-04, 2.96341538e-05, 3.34536271e-05,
```

```
# Лучшая модель
clf_gs.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
clf_gs.best_score_
```

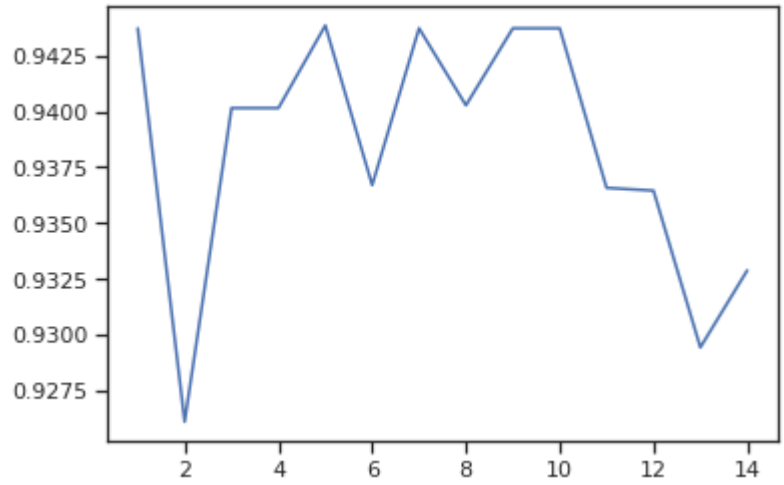
```
0.9438423645320198
```

```
# Лучшее значение параметров
clf_gs.best_params_
```

```
{'n_neighbors': 5}
```

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fae0ce2f2b0>]
```



сравнивая качество модели между K = 5, K = 10

```
# 10 ближайших соседей
accuracy_score(cancer_y_test, target1_2)
```

```
0.9157894736842105
```

```
# 4 ближайших соседей
cl1_3 = KNeighborsClassifier(n_neighbors=5)
cl1_3.fit(cancer_X_train, cancer_y_train)
target1_3 = cl1_3.predict(cancer_X_test)
accuracy_score(cancer_y_test, target1_3)
```

```
0.9052631578947369
```

В этом тестовом наборе данных мы отмечаем, что K = 10 лучше, чем K = 5, хотя при поиске г
Это происходит из-за выбора конкретного набора тестовых данных, но в целом K = 5 будет

Построение кривых обучения и валидации

Построение кривых обучения - learning_curve

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable, optional
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:
          - None, to use the default 3-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, optional (default=None)
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like, shape (n_ticks,), dtype float or int
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the dtype is float, it is regarded as a
        fraction of the maximum size of the training set (that is determined
        by the selected validation method), i.e. it has to be within (0, 1].
        Otherwise it is interpreted as absolute sizes of the training sets.
        Note that for classification the number of samples usually have to
```

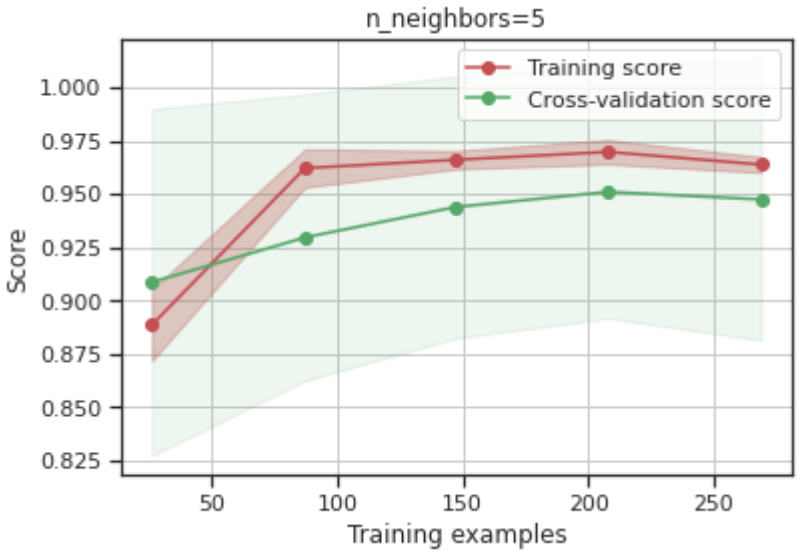
```
        be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.3,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt


plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors=5',
                    cancer_X_train, cancer_y_train, cv=20)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-packages/matpl
```



Построение кривой валидации - validation_curve

```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring="accuracy"):

    train_scores, test_scores = validation_curve(
        estimator, X, y, param_name=param_name, param_range=param_range,
        cv=cv, scoring=scoring, n_jobs=1)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.title(title)
    plt.xlabel(param_name)
    plt.ylabel(str(scoring))
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(param_range, train_scores_mean, label="Training score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.4,
                     color="darkorange", lw=lw)
    plt.plot(param_range, test_scores_mean, label="Cross-validation score",
                 color="navy", lw=lw)
    plt.fill_between(param_range, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")
    return plt


plot_validation_curve(KNeighborsClassifier(), 'knn',
                      cancer_X_train, cancer_y_train,
                      param_name='n_neighbors', param_range=n_range,
                      cv=20, scoring="accuracy")
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-packages/matpl
```