Министерство образования и науки Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э. БАУМАНА

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления» (ИУ5)

ДИСЦИПЛИНА: «Технологии машинного обучения»

Отчет по лабораторной работе №4
«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

Выполнил:

Студент группы ИУ5-61Б

Кочетков М.Д.

Преподаватель:

Гапанюк Ю.Е.

Москва, 2020 г.

Цель лабораторной работы: изучение сложных способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
3. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
4. Постройте модель и оцените качество модели с использованием кросс-валидации.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.

Выполнение ЛР:

1. Загрузка и первичный анализ данных. Выберем dataframe для решения задачи классификации

In [1]:
```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
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, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

In [2]:
```
iris = load_iris()
```

In [3]:
```
# Наименования признаков
iris.feature_names
```

Out[3]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [4]:
```
# Размер выборки
iris.data.shape, iris.target.shape
```

Out[4]:
```
((150, 4), (150,))
```

In [5]:
```
# Сформируем DataFrame
iris_df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= iris['feature_names'] + ['target'])
```

In [6]:
```
# И выведем его статистические характеристики
iris_df.describe()
```

Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

2. *Разделим dataframe на тестовую и обучающую выборку*

In [7]:
```
iris_X_train, iris_X_test, iris_y_train, iris_y_test = train_test_split(
    iris.data, iris.target, test_size=0.3, random_state=1)
```

In [8]:
```
# Размер обучающей выборки
iris_X_train.shape, iris_y_train.shape
```

Out[8]:
```
((105, 4), (105,))
```

In [9]:
```
# Размер тестовой выборки
iris_X_test.shape, iris_y_test.shape
```

Out[9]:
```
((45, 4), (45,))
```

3. *Обучение модели ближайших соседей для произвольно заданного гиперпараметра K*

In [10]:
```
# 2 ближайших соседа
cl1_1 = KNeighborsClassifier(n_neighbors=2)
cl1_1.fit(iris_X_train, iris_y_train)
target1_1 = cl1_1.predict(iris_X_test)
len(target1_1), target1_1
```

Out[10]:
```
(45,
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
        1]))
```

In [11]:
```
# 5 ближайших соседей
cl1_2 = KNeighborsClassifier(n_neighbors=5)
cl1_2.fit(iris_X_train, iris_y_train)
target1_2 = cl1_2.predict(iris_X_test)
len(target1_2), target1_2
```

Out[11]:
```
(45,
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
        1]))
```

4. *Метрики качества классификации*

- Accuracy

In [12]:
```
# iris_y_test - эталонное значение классов из исходной (тестовой) выборки
# target* - предсказанное значение классов

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

Out[12]:
```
0.9777777777777777
```

In [13]:
```
# 5 ближайших соседей
accuracy_score(iris_y_test, target1_2)
```

Out[13]:
```
0.9777777777777777
```

In [14]:
```
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]))
```

In [15]:
```
# 2 ближайших соседа
print_accuracy_score_for_classes(iris_y_test, target1_1)
```

```
Метка    Accuracy
0        1.0
1        1.0
2        0.9230769230769231
```

In [16]:
```
# 7 ближайших соседей
print_accuracy_score_for_classes(iris_y_test, target1_2)
```

```
Метка    Accuracy
0        1.0
1        1.0
2        0.9230769230769231
```

- Матрица ошибок или Confusion Matrix

In [17]:
```
# Конвертация целевого признака в бинарный
def convert_target_to_binary(array:np.ndarray, target:int) -> np.ndarray:
    # Если целевой признак совпадает с указанным, то 1 иначе 0
    res = [1 if x==target else 0 for x in array]
    return res
```

In [18]:
```
# Если целевой признак ==2,
# то будем считать этот случай 1 в бинарном признаке
bin_iris_y_test = convert_target_to_binary(iris_y_test, 2)
# Конвертация предсказанных признаков
bin_target1_1 = convert_target_to_binary(target1_1, 2)
bin_target1_2 = convert_target_to_binary(target1_2, 2)
confusion_matrix(bin_iris_y_test, bin_target1_1, labels=[0, 1])
```

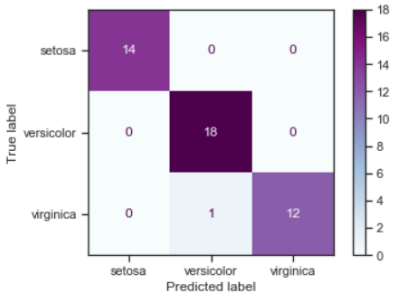
Out[18]:
```
array([[32,  0],
       [ 1, 12]])
```

In [19]:
```
tn, fp, fn, tp = confusion_matrix(bin_iris_y_test, bin_target1_1).ravel()
tn, fp, fn, tp
```

Out[19]:
```
(32, 0, 1, 12)
```

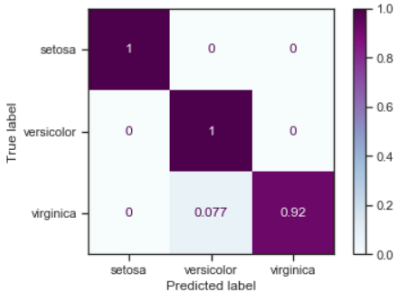
In [20]:
```
plot_confusion_matrix(cl1_1, iris_X_test, iris_y_test,
                      display_labels=iris.target_names, cmap=plt.cm.BuPu)
```

Out[20]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1174af6a0>



In [21]:
```
plot_confusion_matrix(cl1_1, iris_X_test, iris_y_test,
                      display_labels=iris.target_names,
                      cmap=plt.cm.BuPu, normalize='true')
```

Out[21]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1174af610>

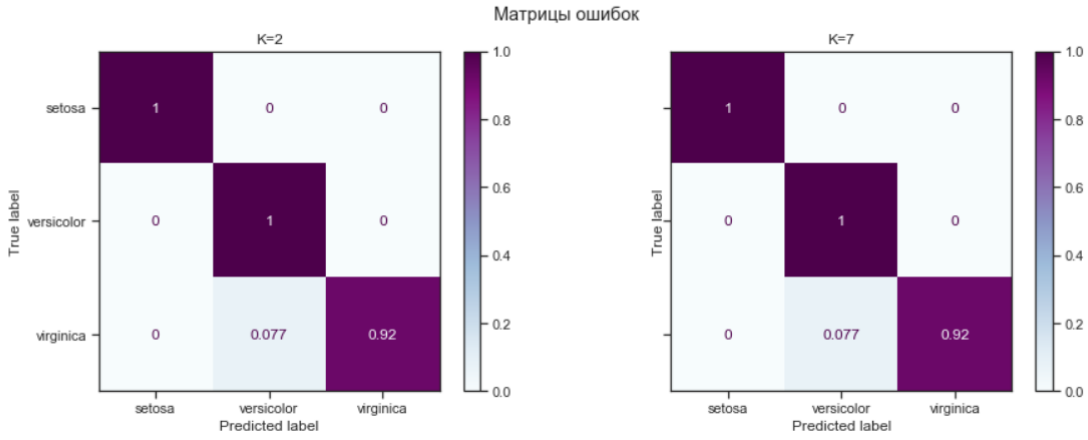


In [22]:
```
fig, ax = plt.subplots(1, 2, sharex='col', sharey='row', figsize=(15,5))

plot_confusion_matrix(cl1_1, iris_X_test, iris_y_test,
                      display_labels=iris.target_names,
                      cmap=plt.cm.BuPu, normalize='true', ax=ax[0])

plot_confusion_matrix(cl1_2, iris_X_test, iris_y_test,
                      display_labels=iris.target_names,
                      cmap=plt.cm.BuPu, normalize='true', ax=ax[1])

fig.suptitle('Матрицы ошибок')
ax[0].title.set_text('K=2')
ax[1].title.set_text('K=7')
```



- Precision, recall и F-мера

In [23]:
```
# Для 2 ближайших соседей
precision_score(bin_iris_y_test, bin_target1_1), recall_score(bin_iris_y_test, bin_target1_1)
```

Out[23]:
```
(1.0, 0.9230769230769231)
```

In [24]:
```
# Для 5 ближайших соседей
precision_score(bin_iris_y_test, bin_target1_2), recall_score(bin_iris_y_test, bin_target1_2)
```

Out[24]:
```
(1.0, 0.9230769230769231)
```

In [25]:
```
# Параметры TP, TN, FP, FN считаются как сумма по всем классам
precision_score(iris_y_test, target1_1, average='micro')
```

Out[25]:
```
0.9777777777777777
```

In [26]:
```
# Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется среднее значение, дисбаланс классов не учитывается.
precision_score(iris_y_test, target1_1, average='macro')
```

Out[26]:
```
0.9824561403508771
```

In [27]:
```
# 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(iris_y_test, target1_1, average='weighted')
```

Out[27]:
```
0.9789473684210527
```

In [28]:
```
# f-мера
```

In [29]:
```
f1_score(bin_iris_y_test, bin_target1_2)
```

Out[29]:
```
0.9600000000000001
```

In [30]:
```
f1_score(iris_y_test, target1_1, average='micro')
```

Out[30]:
```
0.9777777777777777
```

In [31]:
```
f1_score(iris_y_test, target1_1, average='macro')
```

Out[31]:
```
0.9776576576576578
```

In [32]:
```
f1_score(iris_y_test, target1_1, average='weighted')
```

Out[32]:
```
0.9776336336336338
```

In [33]:
```
classification_report(iris_y_test, target1_1,
                      target_names=iris.target_names, output_dict=True)
```

Out[33]:
```
{'setosa': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 14},
 'versicolor': {'precision': 0.9473684210526315,
  'recall': 1.0,
  'f1-score': 0.972972972972973,
  'support': 18},
 'virginica': {'precision': 1.0,
  'recall': 0.9230769230769231,
  'f1-score': 0.9600000000000001,
  'support': 13},
 'accuracy': 0.9777777777777777,
 'macro avg': {'precision': 0.9824561403508771,
  'recall': 0.9743589743589745,
  'f1-score': 0.9776576576576578,
  'support': 45},
 'weighted avg': {'precision': 0.9789473684210527,
  'recall': 0.9777777777777777,
  'f1-score': 0.9776336336336338,
  'support': 45}}
```

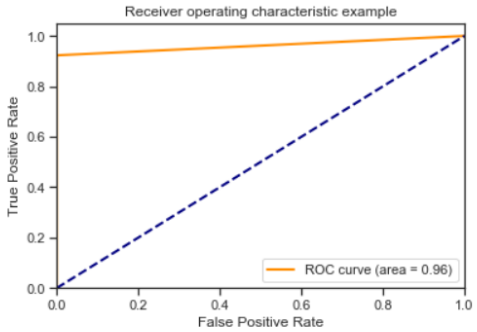
- ROC-кривая и ROC AUC

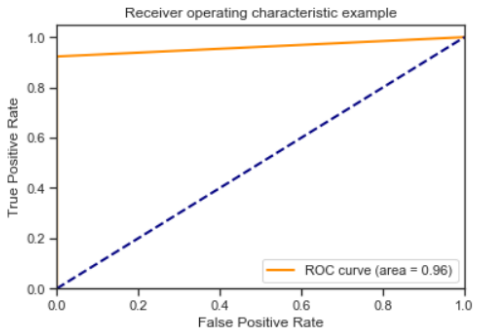
In [34]:
```
fpr, tpr, thresholds = roc_curve(bin_iris_y_test, bin_target1_1,
                                 pos_label=1)
fpr, tpr, thresholds
```

Out[34]:
```
(array([0., 0., 1.]),
 array([0.        , 0.92307692, 1.        ]),
 array([2, 1, 0]))
```

In [35]:
```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label, average):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

In [36]:
```
# Для 2 ближайших соседей
draw_roc_curve(bin_iris_y_test, bin_target1_1, pos_label=1, average='micro')
# Для 7 ближайших соседей
draw_roc_curve(bin_iris_y_test, bin_target1_2, pos_label=1, average='micro')
```





Проанализировав результаты полученных метрик качества классификации, можно судить о среднем качестве классификации.

5. *Построение модели с использованием кросс-валидации*

In [37]:
```
iris_cross = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                             iris.data, iris.target, cv=5)
iris_cross
```

Out[37]:
```
array([0.96666667, 0.93333333, 0.93333333, 0.9       , 1.        ])
```

In [38]:
```
np.mean(iris_cross)
```

Out[38]:
```
0.9466666666666665
```

In [39]:
```
scoring = {'precision': 'precision_weighted',
           'recall': 'recall_weighted',
           'f1': 'f1_weighted'}

iris_cross = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        iris.data, iris.target, scoring=scoring,
                        cv=5, return_train_score=True)
iris_cross
```

Out[39]:
```
{'fit_time': array([0.00065899, 0.00035   , 0.00038099, 0.00077415, 0.00048804]),
 'score_time': array([0.00364995, 0.01217699, 0.00354791, 0.00814104, 0.00469398]),
 'test_precision': array([0.96969697, 0.94444444, 0.94444444, 0.9023569 , 1.        ]),
 'train_precision': array([0.97674419, 0.98412698, 0.97674419, 0.98412698, 0.97674419]),
 'test_recall': array([0.96666667, 0.93333333, 0.93333333, 0.9       , 1.        ]),
 'train_recall': array([0.975     , 0.98333333, 0.975     , 0.98333333, 0.975     ]),
 'test_f1': array([0.96658312, 0.93265993, 0.93265993, 0.89974937, 1.        ]),
 'train_f1': array([0.97496479, 0.98332291, 0.97496479, 0.98332291, 0.97496479])}
```

6. *Нахождение наилучшего гиперпараметра K с использованием GridSearchCV и кросс-валидации*

In [40]:
```
n_range = np.array(range(5,40,3))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[40]:
```
[{'n_neighbors': array([ 5,  8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38])}]
```

In [41]:
```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
clf_gs.fit(iris_X_train, iris_y_train)
```

```
CPU times: user 121 ms, sys: 2.89 ms, total: 124 ms
Wall time: 136 ms
```

Out[41]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 5,  8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [42]:
```
clf_gs.cv_results_
```

Out[42]:
```
{'mean_fit_time': array([0.00049028, 0.00047402, 0.00053415, 0.00033312, 0.00063777,
        0.00039911, 0.00029883, 0.00030971, 0.00030766, 0.00032654,
        0.00029154, 0.00031986]),
 'std_fit_time': array([2.40463953e-04, 2.04236341e-04, 1.00431641e-04, 4.88578717e-05,
        2.89715717e-04, 1.47086217e-04, 1.50567326e-05, 4.78015873e-05,
        1.59106491e-05, 3.13719715e-05, 8.21683267e-06, 3.27051202e-05]),
 'mean_score_time': array([0.00146513, 0.0030148 , 0.00244069, 0.00117564, 0.00162005,
        0.00126657, 0.00121293, 0.00116339, 0.00120907, 0.00114422,
        0.00110011, 0.00115976]),
 'std_score_time': array([3.37036869e-04, 1.93199010e-03, 1.22310104e-03, 1.68393744e-04,
        1.91631297e-04, 1.84569159e-04, 1.42274704e-04, 1.25095809e-04,
        1.94427707e-04, 6.04840070e-05, 2.25450706e-05, 9.46503862e-05]),
 'param_n_neighbors': masked_array(data=[5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 5},
  {'n_neighbors': 8},
  {'n_neighbors': 11},
  {'n_neighbors': 14},
  {'n_neighbors': 17},
  {'n_neighbors': 20},
  {'n_neighbors': 23},
  {'n_neighbors': 26},
  {'n_neighbors': 29},
  {'n_neighbors': 32},
  {'n_neighbors': 35},
  {'n_neighbors': 38}],
 'split0_test_score': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]),
 'split1_test_score': array([0.95238095, 0.95238095, 0.95238095, 0.95238095, 0.95238095,
        0.95238095, 0.95238095, 0.9047619 , 0.9047619 , 0.95238095,
        0.9047619 , 0.95238095]),
 'split2_test_score': array([1.        , 0.95238095, 0.95238095, 0.95238095, 0.95238095,
        0.95238095, 0.95238095, 0.95238095, 0.9047619 , 0.95238095,
        0.9047619 , 0.9047619 ]),
 'split3_test_score': array([0.85714286, 0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 ,
        0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 ,
        0.95238095, 0.9047619 ]),
 'split4_test_score': array([1.        , 1.        , 0.95238095, 0.95238095, 0.9047619 ,
        0.95238095, 0.9047619 , 0.9047619 , 0.9047619 , 0.9047619 ,
        0.9047619 , 0.9047619 ]),
 ...

 'mean_test_score': array([0.96190476, 0.96190476, 0.95238095, 0.95238095, 0.94285714,
        0.95238095, 0.94285714, 0.93333333, 0.92380952, 0.94285714,
        0.93333333, 0.93333333]),
 'std_test_score': array([0.05553288, 0.03563483, 0.03011693, 0.03011693, 0.03563483,
        0.03011693, 0.03563483, 0.03809524, 0.03809524, 0.03563483,
        0.03809524, 0.03809524]),
 'rank_test_score': array([ 1,  2,  3,  3,  6,  3,  6, 11, 12,  6,  9,  9], dtype=int32)}
```

In [43]:
```
# Лучшая модель
clf_gs.best_estimator_
```

Out[43]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [43]:
```
# Лучшая модель
clf_gs.best_estimator_
```

Out[43]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [44]:
```
# Лучшее значение метрики
clf_gs.best_score_
```

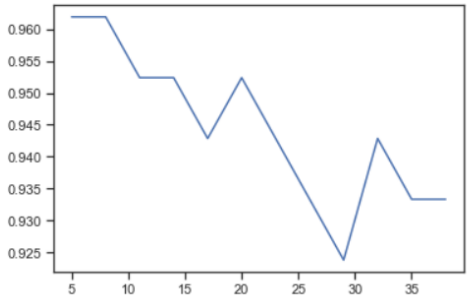
Out[44]:
```
0.961904761904762
```

In [45]:
```
# Лучшее значение параметров
clf_gs.best_params_
```

Out[45]:
```
{'n_neighbors': 5}
```

In [46]:
```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x117eb0880>]
```



Таким образом, лучшее найденное значение гиперпараметра = 5. При этом гиперпараметре получено наилучшее значение метрики = 0.962