



Отчет
Лабораторная работа № 4
По курсу «Технологии машинного обучения»
**«Подготовка обучающей и тестовой выборки, кросс-валидация
и подбор гиперпараметров на примере метода ближайших
соседей»**

ИСПОЛНИТЕЛЬ:

Сергеев И.В.
Группа ИУ5-64Б

"__" _____ 2020 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

"__" _____ 2020 г.

Описание задания

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

Задание:

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

Текст программы

Программа разрабатывалась в IDE PyCharm. Ниже приведён полный листинг программы:

```
###

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, cross_validate, KFold,
LeaveOneOut, GridSearchCV, learning_curve, validation_curve
import time
%matplotlib inline
sns.set(style="ticks")

###

data = pd.read_csv('dataset.txt', sep=',')
data.head()

###

ClassCoded = data[['class']]
LaEnc = LabelEncoder()
```

```

data[['class']] = pd.DataFrame(LaEnc.fit_transform(ClassCoded), columns=['class'])
data.head()

###

# Разделим датасет на обучающий (train) и тестовый (test) набор
trainX, testX, trainY, testY =
train_test_split(data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']],

data['class'], test_size=0.3, random_state = 1)

###

print('Число строк обучающего и тестового наборов:')
print('  Обучающий: {0} ({1} %).format(trainX.shape[0],
round(trainX.shape[0]/data.shape[0]*100,4)))
print('  Тестовый: {0} ({1} %).format(testX.shape[0],
round(testX.shape[0]/data.shape[0]*100,4)))

###

# Проведём обучение 3 моделей с различным числом соседей
classifier3N = KNeighborsClassifier(n_neighbors = 3)
classifier3N.fit(trainX, trainY)

###

# Оценка качества обучения модели.
Nb3Score = round(classifier3N.score(testX, testY)*100, 4)
print('Классификатор с 3 соседями: {} %'.format(Nb3Score))

###

Results3N = classifier3N.predict(testX)
accuracy_score(testY, Results3N)

###

# Рассмотрим результаты для каждого класса отдельно
resultDF = {
    'Prediction': Results3N,
    'Answer': testY
}
ResultData = pd.DataFrame(resultDF, columns = ['Prediction', 'Answer'])
Classes = np.unique(testY)
for c in Classes:
    temp_data_flt = ResultData[ResultData['Answer']==c]
    temp_acc = accuracy_score(
        temp_data_flt['Answer'].values,
        temp_data_flt['Prediction'].values)
    print('Результат для класса {0}: {1} %'.format(c, round(temp_acc*100, 4)))

###

ClassReport3N = classification_report(testY, Results3N, output_dict = True)
for ReportName, ReportResult in ClassReport3N.items():
    print('\nClass :', ReportName)
    for result0, result1 in ReportResult.items():
        print('    {0} : \t{1}'.format(result0, result1))

###

```

```

# Кросс-валидация
# Стратифицированная K-Fold перекрёстная проверка
Scores1 = cross_val_score(classifier3N,
data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], data['class'], cv =
3)
print('Результаты нестратифицированной перекрёстной проверки:')
for Sc in Scores1:
    print('    {} %'.format(round(Sc*100,4)))
print('Средний результат нестратифицированной перекрёстной проверки: {}
%\n'.format(round(Scores1.mean()*100,4)))

###

# Перекрёстная проверка с исключением по одному
loo = LeaveOneOut()
LOOStartTime = time.time()
Scores3 = cross_val_score(classifier3N,
data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], data['class'], cv
= loo)
LOOStopTime = time.time()
print('Средний результат перекрёстной проверки с исключением по одному: {}
%'.format(round(Scores3.mean()*100,4)))
print('Время выполнения перекрёстной проверки с исключением по одному: {}
c'.format(round(LOOStopTime-LOOStartTime,4)))

###

# Оптимизация гиперпараметров
NeighborsArr = np.array(range(1,16))
tuned_parameters = [{'n_neighbors':NeighborsArr}]
ClassifierGS = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=loo,
scoring='accuracy')
ClassifierGS.fit(data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']],
data['class'])

###

GSBP = ClassifierGS.best_params_.get('n_neighbors')
GSBP

###

ClassifierGSBP = KNeighborsClassifier(n_neighbors = GSBP)
ClassifierGSBP.fit(trainX, trainY)
ResultsGSBP = ClassifierGSBP.predict(testX)
ClassReport5N = classification_report(testY, ResultsGSBP, output_dict = True)
for ReportName, ReportResult in ClassReport5N.items():
    print('\nClass : ', ReportName)
    for result0, result1 in ReportResult.items():
        print('    {} : \t{1}'.format(result0, result1))

###

# Кривые обучения и валидации
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.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=0.1,
                 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={}'.format(5),
                   trainX, trainY, cv=20)

#%%
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)

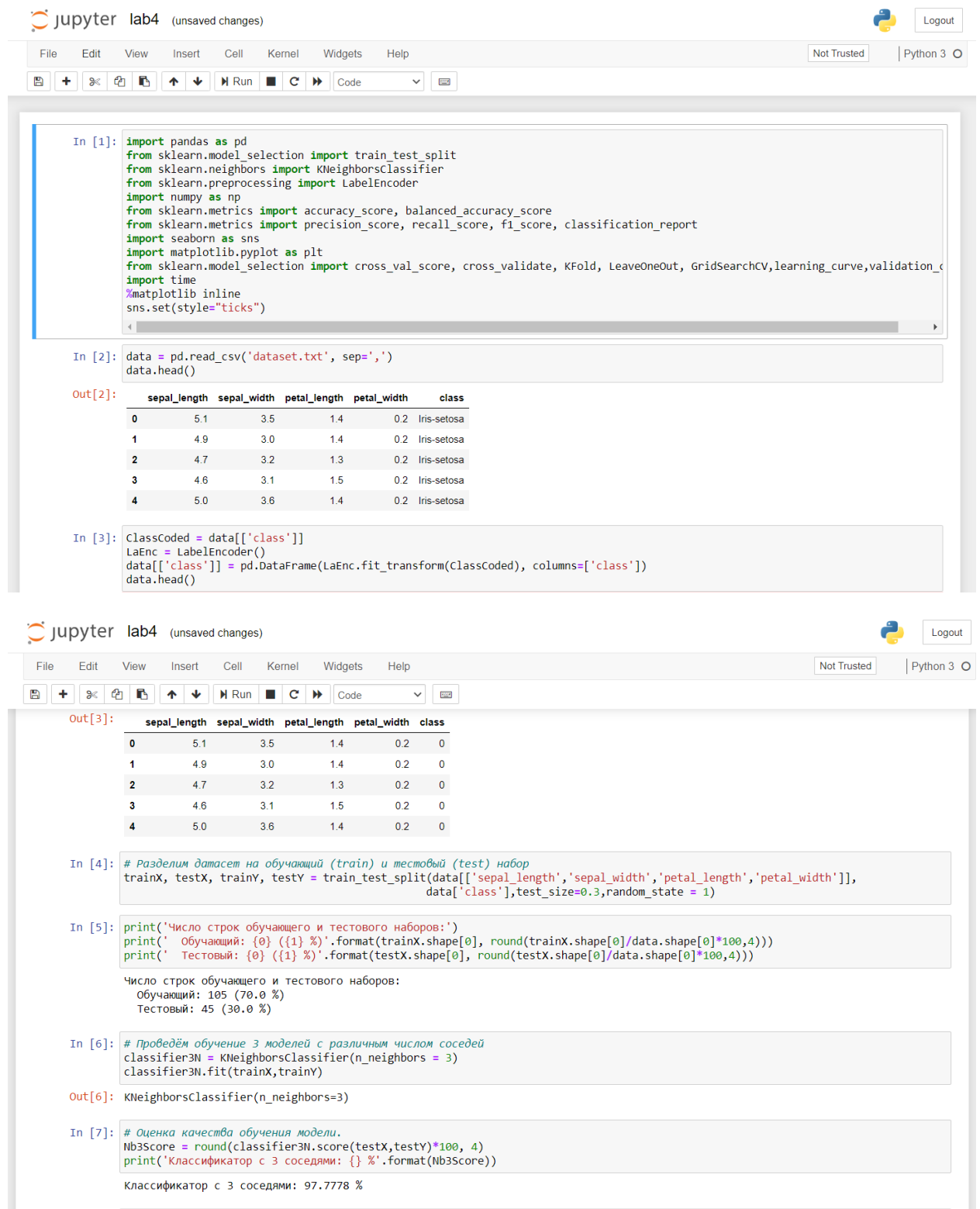
    plt.title(title)
    plt.xlabel(param_name)
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(param_range, train_scores_mean, label="Training score",
            color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.2,
                    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',
                    trainX, trainY,
                    param_name='n_neighbors', param_range=NeighborsArr,
                    cv=20, scoring="accuracy")

```

Примеры выполнения программы

Выполнение программы, а также наглядная демонстрация входных и выходных данных (таблиц, графиков и тд) осуществлялась на базе Jupyter Notebook, сервер которого запускался из-под PyCharm. Ниже приведены скриншоты, отражающие работу программы:



The first screenshot shows the Jupyter Notebook interface with the following code cells:

```
In [1]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, cross_validate, KFold, LeaveOneOut, GridSearchCV, learning_curve, validation_curve
import time
%matplotlib inline
sns.set(style="ticks")
```

```
In [2]: data = pd.read_csv('dataset.txt', sep=',')
data.head()
```

The output of In [2] is a table with 5 columns: sepal_length, sepal_width, petal_length, petal_width, and class. The first 5 rows are shown:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [3]: ClassCoded = data[['class']]
LaEnc = LabelEncoder()
data[['class']] = pd.DataFrame(LaEnc.fit_transform(ClassCoded), columns=['class'])
data.head()
```

The second screenshot shows the continuation of the code:

```
Out[3]:
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
In [4]: # Разделим датасет на обучающий (train) и тестовый (test) набор
trainX, testX, trainY, testY = train_test_split(data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']],
data['class'], test_size=0.3, random_state = 1)
```

```
In [5]: print('Число строк обучающего и тестового наборов:')
print('  Обучающий: {} ({} %)'.format(trainX.shape[0], round(trainX.shape[0]/data.shape[0]*100,4)))
print('  Тестовый: {} ({} %)'.format(testX.shape[0], round(testX.shape[0]/data.shape[0]*100,4)))
```

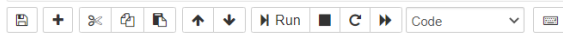
Число строк обучающего и тестового наборов:
Обучающий: 105 (70.0 %)
Тестовый: 45 (30.0 %)

```
In [6]: # Проведём обучение 3 моделей с различным числом соседей
classifier3N = KNeighborsClassifier(n_neighbors = 3)
classifier3N.fit(trainX, trainY)
```

```
Out[6]: KNeighborsClassifier(n_neighbors=3)
```

```
In [7]: # Оценка качества обучения модели.
Nb3Score = round(classifier3N.score(testX, testY)*100, 4)
print('Классификатор с 3 соседями: {} %'.format(Nb3Score))
```

Классификатор с 3 соседями: 97.7778 %



```
In [8]: Results3N = classifier3N.predict(testX)
accuracy_score(testY, Results3N)
```

```
Out[8]: 0.9777777777777777
```

```
In [9]: # Рассмотрим результаты для каждого класса отдельно
resultDF = {
    'Prediction': Results3N,
    'Answer': testY
}
ResultData = pd.DataFrame(resultDF, columns = ['Prediction', 'Answer'])
Classes = np.unique(testY)
for c in Classes:
    temp_data_flt = ResultData[ResultData['Answer']==c]
    temp_acc = accuracy_score(
        temp_data_flt['Answer'].values,
        temp_data_flt['Prediction'].values)
    print('Результат для класса {0}: {1} %'.format(c, round(temp_acc*100, 4)))
```

```
Результат для класса 0: 100.0 %
Результат для класса 1: 100.0 %
Результат для класса 2: 92.3077 %
```

```
In [10]: ClassReport3N = classification_report(testY, Results3N, output_dict = True)
for ReportName, ReportResult in ClassReport3N.items():
    print('\nClass : ', ReportName)
    for result0, result1 in ReportResult.items():
        print('    {0} : {1}'.format(result0, result1))
```

```
Class : 0
precision : 1.0
recall : 1.0
f1-score : 1.0
support : 14
```



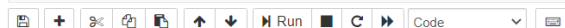
```
In [9]: # Рассмотрим результаты для каждого класса отдельно
resultDF = {
    'Prediction': Results3N,
    'Answer': testY
}
ResultData = pd.DataFrame(resultDF, columns = ['Prediction', 'Answer'])
Classes = np.unique(testY)
for c in Classes:
    temp_data_flt = ResultData[ResultData['Answer']==c]
    temp_acc = accuracy_score(
        temp_data_flt['Answer'].values,
        temp_data_flt['Prediction'].values)
    print('Результат для класса {0}: {1} %'.format(c, round(temp_acc*100, 4)))
```

```
Результат для класса 0: 100.0 %
Результат для класса 1: 100.0 %
Результат для класса 2: 92.3077 %
```

```
In [10]: ClassReport3N = classification_report(testY, Results3N, output_dict = True)
for ReportName, ReportResult in ClassReport3N.items():
    print('\nClass : ', ReportName)
    for result0, result1 in ReportResult.items():
        print('    {0} : {1}'.format(result0, result1))
```

```
Class : 0
precision : 1.0
recall : 1.0
f1-score : 1.0
support : 14

Class : 1
precision : 0.9473684210526315
recall : 1.0
f1-score : 0.972972972972973
support : 18
```



```
In [11]: # Кросс-валидация
# Стратифицированная K-Fold перекрёстная проверка
Scores1 = cross_val_score(classifier3N, data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], data['class'], cv = 3)
print('Результаты нестратифицированной перекрёстной проверки:')
for Sc in Scores1:
    print('    {} %'.format(round(Sc*100,4)))
print('Средний результат нестратифицированной перекрёстной проверки: {} %\n'.format(round(Scores1.mean()*100,4)))
```

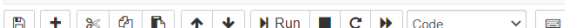
```
Результаты нестратифицированной перекрёстной проверки:
98.0 %
96.0 %
98.0 %
Средний результат нестратифицированной перекрёстной проверки: 97.3333 %
```

```
In [12]: # Перекрёстная проверка с исключением по одному
loo = LeaveOneOut()
LOOStartTime = time.time()
Scores3 = cross_val_score(classifier3N, data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], data['class'], cv = 10)
LOOStopTime = time.time()
print('Средний результат перекрёстной проверки с исключением по одному: {} %'.format(round(Scores3.mean()*100,4)))
print('Время выполнения перекрёстной проверки с исключением по одному: {} c'.format(round(LOOStopTime-LOOStartTime,4)))
```

```
Средний результат перекрёстной проверки с исключением по одному: 96.0 %
Время выполнения перекрёстной проверки с исключением по одному: 1.105 c
```

```
In [13]: # Оптимизация гиперпараметров
NeighborsArr = np.array(range(1,16))
tuned_parameters = [{'n_neighbors': NeighborsArr}]
ClassifierGS = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=loo, scoring='accuracy')
ClassifierGS.fit(data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']], data['class'])
```

```
Out[13]: GridSearchCV(cv=LeaveOneOut(), estimator=KNeighborsClassifier(),
param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])}],
```



```
In [14]: GSBP = ClassifierGS.best_params_.get('n_neighbors')
GSBP
```

```
Out[14]: 10
```

```
In [15]: ClassifierGSBP = KNeighborsClassifier(n_neighbors = GSBP)
ClassifierGSBP.fit(trainX, trainY)
ResultsGSBP = ClassifierGSBP.predict(testX)
ClassReport5N = classification_report(testY, ResultsGSBP, output_dict = True)
for ReportName, ReportResult in ClassReport5N.items():
    print('\nClass : ', ReportName)
    for result0, result1 in ReportResult.items():
        print('    {0} : \t{1}'.format(result0, result1))
```

```
Class : 0
precision : 1.0
recall : 1.0
f1-score : 1.0
support : 14

Class : 1
precision : 1.0
recall : 0.9444444444444444
f1-score : 0.9714285714285714
support : 18

Class : 2
precision : 0.9285714285714286
recall : 1.0
f1-score : 0.962962962962963
support : 13
```



```
In [16]: # Кривые обучения и валидации
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.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=0.1,
                     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")

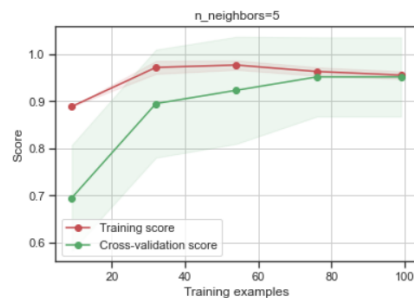
    plt.legend(loc="best")
    return plt
```

```
In [17]: plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors={}'.format(5),
                             trainX, trainY, cv=20)
```

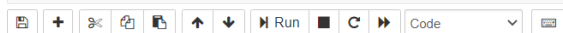
```
Out[17]: <module 'matplotlib.pyplot' from 'c:\users\is-st\appdata\local\programs\python\python37-32\lib\site-packages\matplotlib1
ib\pyplot.py'>
```

```
In [17]: plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors={}'.format(5),
                             trainX, trainY, cv=20)
```

```
Out[17]: <module 'matplotlib.pyplot' from 'c:\users\is-st\appdata\local\programs\python\python37-32\lib\site-packages\matplotlib1
ib\pyplot.py'>
```

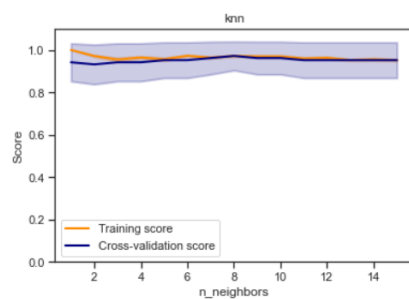


```
In [18]: def plot_validation_curve(estimator, title, X, y,
                                   param_name, param_range, cv,
                                   scoring="accuracy"):
    train_scores, test_scores = validation_curve(
        estimator, X, y, param_name=param_name, param_range=param_range,
        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)
```



```
In [19]: plot_validation_curve(KNeighborsClassifier(), 'knn',  
                             trainX, trainY,  
                             param_name='n_neighbors', param_range=NeighborsArr,  
                             cv=20, scoring="accuracy")
```

```
Out[19]: <module 'matplotlib.pyplot' from 'c:\\users\\is-st\\appdata\\local\\programs\\python\\python37-32\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



In []: