**Московский государственный технический университет им. Н.Э. Баумана**

Кафедра «Системы обработки информации и управления»

**Лабораторная работа №4**

**по курсу «Методы машинного обучения»**

**на тему**

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

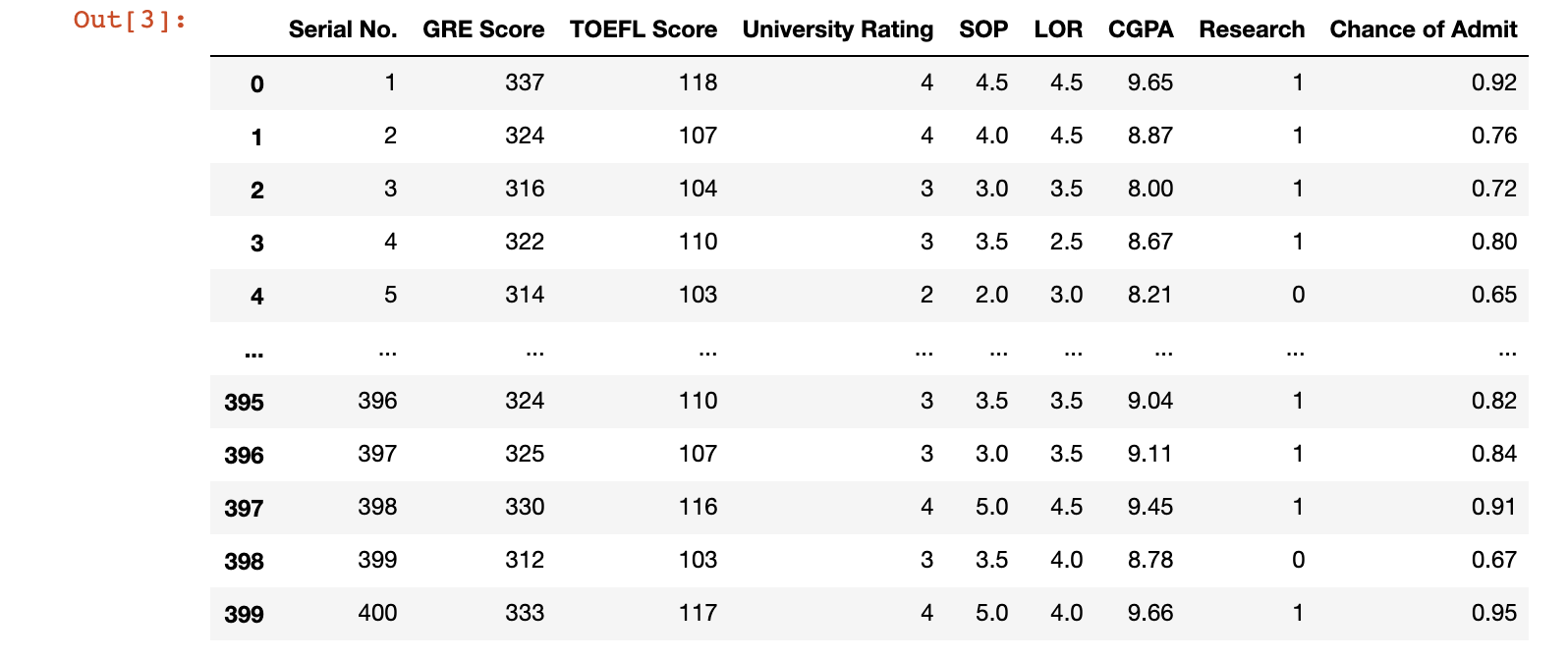
**Выполнил:**

**Хотин П.Ю.**

**ИУ5-24М**

Москва, 2020 год

from IPython.display import Image  
import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.datasets import load\_iris, load\_boston  
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, ShuffleSplit, StratifiedKFold  
from sklearn.metrics import accuracy\_score, balanced\_accuracy\_score  
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  
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")

data = pd.read\_csv("data/Admission\_Predict.csv")  
data

400 rows × 9 columns

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 400 entries, 0 to 399  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Serial No. 400 non-null int64   
 1 GRE Score 400 non-null int64   
 2 TOEFL Score 400 non-null int64   
 3 University Rating 400 non-null int64   
 4 SOP 400 non-null float64  
 5 LOR 400 non-null float64  
 6 CGPA 400 non-null float64  
 7 Research 400 non-null int64   
 8 Chance of Admit 400 non-null float64  
dtypes: float64(4), int64(5)  
memory usage: 28.2 KB

data.isnull().sum()

Serial No. 0  
GRE Score 0  
TOEFL Score 0  
University Rating 0  
SOP 0  
LOR 0  
CGPA 0  
Research 0  
Chance of Admit 0  
dtype: int64

data.shape

(400, 9)

data.loc[data['Chance of Admit '] < 0.65, 'isAdmit'] = 0  
data.loc[data['Chance of Admit '] >= 0.65, 'isAdmit'] = 1  
data.isAdmit

0 1.0  
1 1.0  
2 1.0  
3 1.0  
4 1.0  
 ...   
395 1.0  
396 1.0  
397 1.0  
398 1.0  
399 1.0  
Name: isAdmit, Length: 400, dtype: float64

np.unique(data.isAdmit)

array([0., 1.])

target = data.iloc[:, -1]  
new\_data = data.iloc[:, :-2]  
  
new\_data.shape, target.shape

((400, 8), (400,))

data\_X\_train, data\_X\_test, data\_y\_train, data\_y\_test = train\_test\_split(  
 new\_data, target, test\_size=0.6, random\_state=1  
)

data\_X\_train.shape, data\_X\_test.shape, data\_y\_train.shape, data\_y\_test.shape

((160, 8), (240, 8), (160,), (240,))

cl1\_1 = KNeighborsClassifier(n\_neighbors=50)  
cl1\_1.fit(data\_X\_train, data\_y\_train)  
target1\_0 = cl1\_1.predict(data\_X\_train)  
target1\_1 = cl1\_1.predict(data\_X\_test)  
accuracy\_score(data\_y\_train, target1\_0), accuracy\_score(data\_y\_test, target1\_1)

(0.775, 0.6791666666666667)

cl1\_2 = KNeighborsClassifier(n\_neighbors=15)  
cl1\_2.fit(data\_X\_train, data\_y\_train)  
target2\_0 = cl1\_2.predict(data\_X\_train)  
target2\_1 = cl1\_2.predict(data\_X\_test)  
accuracy\_score(data\_y\_train, target2\_0), accuracy\_score(data\_y\_test, target2\_1)

(0.8, 0.7416666666666667)

cl1\_3 = KNeighborsClassifier(n\_neighbors=3)  
cl1\_3.fit(data\_X\_train, data\_y\_train)  
target3\_0 = cl1\_3.predict(data\_X\_train)  
target3\_1 = cl1\_3.predict(data\_X\_test)  
accuracy\_score(data\_y\_train, target3\_0), accuracy\_score(data\_y\_test, target3\_1)

(0.925, 0.8416666666666667)

scores1 = cross\_val\_score(KNeighborsClassifier(n\_neighbors=15),   
 new\_data, target,   
 cv=5)  
scores1, np.mean(scores1)

(array([0.3125, 0.375 , 0.725 , 0.7125, 0.7125]), 0.5675)

scores2 = cross\_val\_score(KNeighborsClassifier(n\_neighbors=15),   
 new\_data, target,   
 cv=5, scoring='jaccard')  
scores2, np.mean(scores2)

(array([0.05172414, 0.32432432, 0.72151899, 0.7125 , 0.7125 ]),  
 0.5045134899194261)

scores3 = cross\_val\_score(KNeighborsClassifier(n\_neighbors=15),   
 new\_data, target,   
 cv=3, scoring='f1')  
scores3, np.mean(scores3)

(array([0.30088496, 0.83842795, 0.83333333]), 0.6575487455612663)

scoring = {  
 'accuracy': 'accuracy',  
 *'jaccard'*: 'jaccard',  
 *'f1'*: 'f1'  
}  
  
scores = cross\_validate(KNeighborsClassifier(n\_neighbors=15),   
 new\_data, target, scoring=scoring,   
 cv=5, return\_train\_score=True)  
scores

{'fit\_time': array([0.00293589, 0.00228405, 0.00289893, 0.00178099, 0.00226808]),  
 'score\_time': array([0.00762486, 0.00671482, 0.0059588 , 0.00602603, 0.00486207]),  
 'test\_accuracy': array([0.3125, 0.375 , 0.725 , 0.7125, 0.7125]),  
 'train\_accuracy': array([0.875 , 0.85 , 0.86875 , 0.896875, 0.846875]),  
 'test\_jaccard': array([0.05172414, 0.32432432, 0.72151899, 0.7125 , 0.7125 ]),  
 'train\_jaccard': array([0.84848485, 0.82222222, 0.84150943, 0.87007874, 0.82051282]),  
 'test\_f1': array([0.09836066, 0.48979592, 0.83823529, 0.83211679, 0.83211679]),  
 'train\_f1': array([0.91803279, 0.90243902, 0.91393443, 0.93052632, 0.90140845])}

%%time  
scores = cross\_val\_score(KNeighborsClassifier(n\_neighbors=15),   
 new\_data, target,   
 cv=LeaveOneOut())  
scores, np.mean(scores)

CPU times: user 1.29 s, sys: 6.52 ms, total: 1.29 s  
Wall time: 1.3 s  
  
  
  
  
  
(array([1., 1., 1., 1., 0., 1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 1., 1.,  
 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0.,  
 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1.,  
 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1.,  
 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 0., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1.,  
 1., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 1., 1., 1., 1., 0., 0., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1.,  
 1., 0., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,  
 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1.,  
 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,  
 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1.,  
 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1.]),  
 0.835)

kf = KFold(n\_splits=5)  
scores = cross\_val\_score(KNeighborsClassifier(n\_neighbors=10),   
 new\_data, target,   
 cv=kf)  
scores

array([0.625 , 0.8375, 0.85 , 0.825 , 0.675 ])

n\_range = np.array(range(5,55,5))  
tuned\_parameters = [{'n\_neighbors': n\_range}]  
tuned\_parameters

[{'n\_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]

%%time  
clf\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=5, scoring='accuracy')  
clf\_gs.fit(data\_X\_train, data\_y\_train)

CPU times: user 218 ms, sys: 3.27 ms, total: 222 ms  
Wall time: 221 ms  
  
  
  
  
  
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, 10, 15, 20, 25, 30, 35, 40, 45, 50])}],  
 pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,  
 scoring='accuracy', verbose=0)

clf\_gs.cv\_results\_

{'mean\_fit\_time': array([0.00224361, 0.00239301, 0.00187006, 0.00147581, 0.00148458,  
 0.0014236 , 0.00161104, 0.0015831 , 0.00143313, 0.00150428]),  
 'std\_fit\_time': array([2.95497784e-04, 3.89381702e-04, 3.07080709e-04, 5.95586599e-05,  
 4.83124293e-05, 4.01606171e-05, 1.48780330e-04, 1.96596150e-04,  
 5.30786373e-05, 8.59239654e-05]),  
 'mean\_score\_time': array([0.00285563, 0.00336461, 0.00227323, 0.00210838, 0.00206318,  
 0.00195217, 0.00198469, 0.0020319 , 0.00205312, 0.00211654]),  
 'std\_score\_time': array([0.00035558, 0.00097058, 0.00020221, 0.00022293, 0.00019277,  
 0.00014798, 0.00010986, 0.00014462, 0.00017963, 0.00022913]),  
 'param\_n\_neighbors': masked\_array(data=[5, 10, 15, 20, 25, 30, 35, 40, 45, 50],  
 mask=[False, False, False, False, False, False, False, False,  
 False, False],  
 fill\_value='?',  
 dtype=object),  
 'params': [{'n\_neighbors': 5},  
 {'n\_neighbors': 10},  
 {'n\_neighbors': 15},  
 {'n\_neighbors': 20},  
 {'n\_neighbors': 25},  
 {'n\_neighbors': 30},  
 {'n\_neighbors': 35},  
 {'n\_neighbors': 40},  
 {'n\_neighbors': 45},  
 {'n\_neighbors': 50}],  
 'split0\_test\_score': array([0.78125, 0.8125 , 0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,  
 0.75 , 0.75 , 0.75 ]),  
 'split1\_test\_score': array([0.84375, 0.78125, 0.8125 , 0.78125, 0.78125, 0.78125, 0.78125,  
 0.78125, 0.78125, 0.78125]),  
 'split2\_test\_score': array([0.75 , 0.84375, 0.71875, 0.8125 , 0.78125, 0.78125, 0.78125,  
 0.78125, 0.78125, 0.78125]),  
 'split3\_test\_score': array([0.5625 , 0.5625 , 0.59375, 0.78125, 0.78125, 0.78125, 0.78125,  
 0.78125, 0.78125, 0.78125]),  
 'split4\_test\_score': array([0.78125, 0.78125, 0.78125, 0.78125, 0.78125, 0.78125, 0.78125,  
 0.78125, 0.78125, 0.78125]),  
 'mean\_test\_score': array([0.74375, 0.75625, 0.73125, 0.78125, 0.775 , 0.775 , 0.775 ,  
 0.775 , 0.775 , 0.775 ]),  
 'std\_test\_score': array([0.09560662, 0.09960861, 0.07551904, 0.01976424, 0.0125 ,  
 0.0125 , 0.0125 , 0.0125 , 0.0125 , 0.0125 ]),  
 'rank\_test\_score': array([ 9, 8, 10, 1, 2, 2, 2, 2, 2, 2], dtype=int32)}

clf\_gs.best\_estimator\_

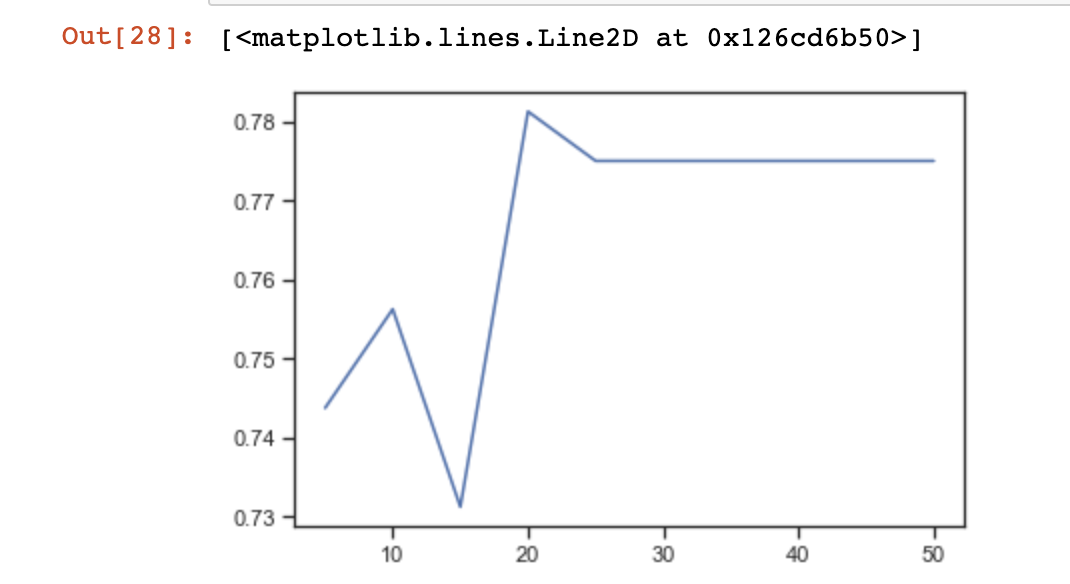
KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',  
 metric\_params=None, n\_jobs=None, n\_neighbors=20, p=2,  
 weights='uniform')

clf\_gs.best\_score\_

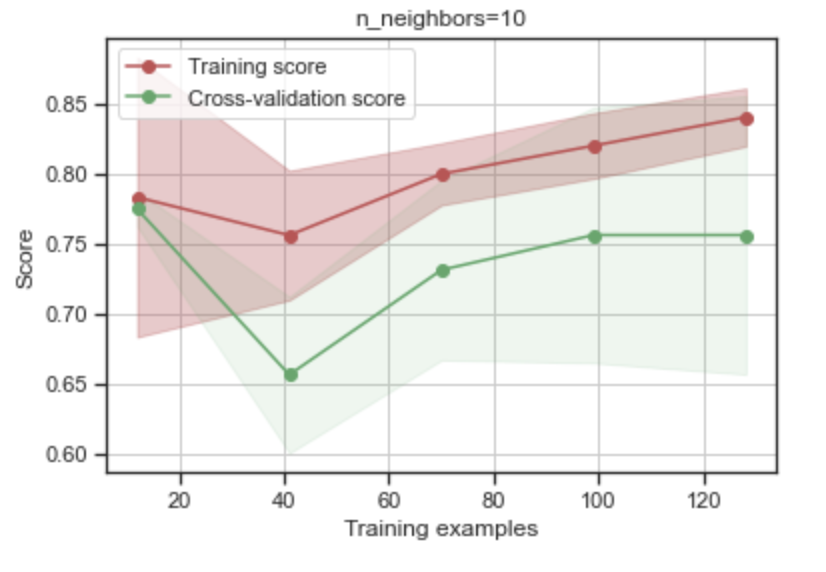
0.78125

clf\_gs.best\_params\_

{'n\_neighbors': 20}

plt.plot(n\_range, clf\_gs.cv\_results\_['mean\_test\_score'])

**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=0.3,  
 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=10), 'n\_neighbors=10',   
 data\_X\_train, data\_y\_train, cv=5)

**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',   
 data\_X\_train, data\_y\_train,   
 param\_name='n\_neighbors', param\_range=n\_range,   
 cv=3, scoring="accuracy")

<module 'matplotlib.pyplot'>