Московский государственный технический университет имени 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:**

Студент ИУ5-24М

Черната Н. С.

Москва, 2020

### Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. Проведите эксперименты с тремя различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните качество полученной модели с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

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\_Ver1.1.csv")  
data

Serial No.

GRE Score

TOEFL Score

University Rating

SOP

LOR

CGPA

Research

Chance of Admit

0

1

337

118

4

4.5

4.5

9.65

1

0.92

1

2

324

107

4

4.0

4.5

...

...

...

...

0.73

499

500

327

113

4

4.5

4.5

9.04

0

0.84

500 rows × 9 columns

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 500 entries, 0 to 499  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Serial No. 500 non-null int64   
 1 GRE Score 500 non-null int64   
 2 TOEFL Score 500 non-null int64   
 3 University Rating 500 non-null int64   
 4 SOP 500 non-null float64  
 5 LOR 500 non-null float64  
 6 CGPA 500 non-null float64  
 7 Research 500 non-null int64   
 8 Chance of Admit 500 non-null float64  
dtypes: float64(4), int64(5)  
memory usage: 35.3 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

(500, 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  
 ...   
495 1.0  
496 1.0  
497 1.0  
498 1.0  
499 1.0  
Name: isAdmit, Length: 500, dtype: float64

np.unique(data.isAdmit)

array([0., 1.])

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

((500, 8), (500,))

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

((200, 8), (300, 8), (200,), (300,))

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)

(0.79, 0.6533333333333333)

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)

(0.83, 0.6966666666666667)

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, 0.82)

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

(array([0.35, 0.52, 0.71, 0.6 , 0.77]), 0.5900000000000001)

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

(array([0.08450704, 0.52 , 0.71 , 0.43661972, 0.69333333]),  
 0.4888920187793427)

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

(array([0.31428571, 0.83985765, 0.12698413]), 0.427042497505131)

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([0.00699782, 0.00499678, 0.00399804, 0.00399756, 0.00499845]),  
 'score\_time': array([0.01299381, 0.0109942 , 0.00799537, 0.00799584, 0.00799394]),  
 'test\_accuracy': array([0.35, 0.52, 0.71, 0.6 , 0.77]),  
 'train\_accuracy': array([0.865 , 0.8625, 0.8975, 0.855 , 0.8625]),  
 'test\_jaccard': array([0.08450704, 0.52 , 0.71 , 0.43661972, 0.69333333]),  
 'train\_jaccard': array([0.83685801, 0.83333333, 0.86900958, 0.82477341, 0.83333333]),  
 'test\_f1': array([0.15584416, 0.68421053, 0.83040936, 0.60784314, 0.81889764]),  
 'train\_f1': array([0.91118421, 0.90909091, 0.92991453, 0.90397351, 0.90909091])}

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

Wall time: 3.89 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., 1., 1., 1., 0.,  
 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 0., 1.,  
 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 1.,  
 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 1., 0., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.,  
 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0.,  
 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,  
 0., 1., 1., 1., 1., 1., 1.]),  
 0.824)

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

array([0.76, 0.77, 0.81, 0.62, 0.74])

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)

Wall time: 503 ms  
  
  
  
  
  
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
 param\_grid=[{'n\_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}],  
 scoring='accuracy')

clf\_gs.cv\_results\_

{'mean\_fit\_time': array([0.00539775, 0.00439777, 0.00360117, 0.00439715, 0.00319815,  
 0.00319843, 0.00399475, 0.00579686, 0.00499306, 0.00459676]),  
 'std\_fit\_time': array([0.00048992, 0.00101884, 0.00048532, 0.00135525, 0.00039971,  
 0.00040043, 0.00062511, 0.0020421 , 0.00063718, 0.00079986]),  
 'mean\_score\_time': array([0.00659523, 0.00419726, 0.00499401, 0.00479727, 0.0043973 ,  
 0.00400066, 0.00439377, 0.00579667, 0.00620103, 0.0057972 ]),  
 'std\_score\_time': array([1.01912224e-03, 4.00114443e-04, 6.31988295e-04, 1.16610586e-03,  
 4.89375878e-04, 7.27704123e-06, 5.00204961e-04, 9.79326166e-04,  
 7.52164723e-04, 1.16565645e-03]),  
 '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([0.8 , 0.825, 0.825, 0.8 , 0.8 , 0.8 , 0.8 , 0.8 , 0.8 ,  
 0.8 ]),  
 'split1\_test\_score': array([0.875, 0.9 , 0.825, 0.8 , 0.8 , 0.8 , 0.8 , 0.8 , 0.8 ,  
 0.8 ]),  
 'split2\_test\_score': array([0.875, 0.85 , 0.8 , 0.825, 0.8 , 0.8 , 0.8 , 0.8 , 0.8 ,  
 0.8 ]),  
 'split3\_test\_score': array([0.725, 0.725, 0.75 , 0.775, 0.775, 0.775, 0.775, 0.775, 0.775,  
 0.775]),  
 'split4\_test\_score': array([0.775, 0.75 , 0.775, 0.75 , 0.775, 0.775, 0.775, 0.775, 0.775,  
 0.775]),  
 'mean\_test\_score': array([0.81 , 0.81 , 0.795, 0.79 , 0.79 , 0.79 , 0.79 , 0.79 , 0.79 ,  
 0.79 ]),  
 'std\_test\_score': array([0.05830952, 0.06442049, 0.02915476, 0.0254951 , 0.01224745,  
 0.01224745, 0.01224745, 0.01224745, 0.01224745, 0.01224745]),  
 'rank\_test\_score': array([ 2, 1, 3, 10, 4, 4, 4, 4, 4, 4])}

clf\_gs.best\_estimator\_

KNeighborsClassifier(n\_neighbors=10)

clf\_gs.best\_score\_

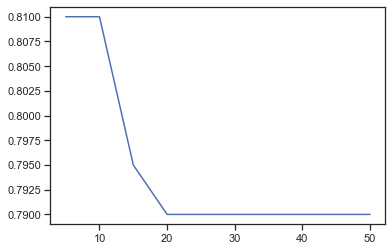
0.8100000000000002

clf\_gs.best\_params\_

{'n\_neighbors': 10}

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

[<matplotlib.lines.Line2D at 0x29560f902b0>]

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)

<module 'matplotlib.pyplot' from 'c:\\users\\ncher\\appdata\\local\\programs\\python\\python36\\lib\\site-packages\\matplotlib\\pyplot.py'>

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' from 'c:\\users\\ncher\\appdata\\local\\programs\\python\\python36\\lib\\site-packages\\matplotlib\\pyplot.py'>

