**Московский государственный технический университет**

**им. Н.Э. Баумана**

**(МГТУ им. Н.Э. Баумана)**

**Радиотехнический факультет (РТ)**

Отчёт по лабораторной работе №4

По курсу «Технологии машинного обучения»

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

|  |  |
| --- | --- |
| Проверила: | Выполнил: |
| Преподаватель кафедры ИУ-5 | студент группы РТ5-61Б |
| Гапанюк Ю.Е. | Абраменков Г.Г. |
| Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_ | Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 202\_ г. | «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 202\_ г. |

Москва, 2020

**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(**'**\n**Class :'**,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:{} с'**.format(round(LOOStopTime-LOOStartTime,4)))  
  
#%%  
  
# Оптимизаия 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(**'**\n**Class :'**,ReportName)  
 for result0, result1 in ReportResult.items():  
 print(**' {0} :** \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:

















