Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет**

**Лабораторная работа № 5**

**По курсу «Технологии машинного обучения»**

## «Линейные модели, SVM и деревья решений»

**ИСПОЛНИТЕЛЬ:**

Сергеев И.В.

Группа ИУ5-64Б

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

Москва 2020

**Описание задания**

**Цель лабораторной работы -** изучение линейных моделей, SVM и деревьев решений.

**Задание:**

* Выберите набор данных (датасет) для решения задачи классификации или регрессии.
* В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
* С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.
* Обучите следующие модели: одну из линейных моделей; SVM; дерево решений.
* Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

**Текст программы**

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

#%%  
  
from sklearn.model\_selection import train\_test\_split  
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  
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  
%matplotlib inline   
sns.set(style=**"ticks"**)  
import mglearn  
  
#%%  
  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
X, y = mglearn.datasets.make\_forge()  
X  
  
#%%  
  
fig, axes = plt.subplots(1, 2, figsize=(10, 3))  
  
for model, ax in zip([SVC(), LogisticRegression()], axes):  
 clf = model.fit(X, y)  
 mglearn.plots.plot\_2d\_separator(clf, X, fill=False, eps=0.5,  
 ax=ax, alpha=.7)  
 mglearn.discrete\_scatter(X[:, 0], X[:, 1], y, ax=ax)  
 ax.set\_title(**"{}"**.format(clf.\_\_class\_\_.\_\_name\_\_))  
 ax.set\_xlabel(**"Признак 0"**)  
 ax.set\_ylabel(**"Признак 1"**)  
axes[0].legend()  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)  
  
#%%  
  
logreg = LogisticRegression().fit(X\_train, y\_train)  
score\_train = logreg.score(X\_train, y\_train)  
score\_test = logreg.score(X\_test, y\_test)  
print(**"Правильность на обучающем наборе {}"**.format(score\_train))  
print(**"Правильность на тестовом наборе {}"**.format(score\_test))  
  
#%%  
  
decTree = DecisionTreeClassifier(max\_depth = 4).fit(X\_train, y\_train)  
score\_train\_tree = decTree.score(X\_train, y\_train)  
score\_test\_tree = decTree.score(X\_test, y\_test)  
print(**"Правильность на обучающем наборе {}"**.format(score\_train\_tree))  
print(**"Правильность на тестовом наборе {}"**.format(score\_test\_tree))  
  
#%%  
  
linSVM = SVC().fit(X\_train, y\_train)  
score\_train\_SVM = linSVM.score(X\_train, y\_train)  
score\_test\_SVM = linSVM.score(X\_test, y\_test)  
print(**"Правильность на обучающем наборе {}"**.format(score\_train\_SVM))  
print(**"Правильность на тестовом наборе {}"**.format(score\_test\_SVM))  
  
#%%  
  
print(**"Логистическая регрессия** \n**"**)  
for method in [accuracy\_score,confusion\_matrix, classification\_report]:  
 print(**"Метод {}"**.format(method.\_\_name\_\_))  
 print(method(y\_test,logreg.predict(X\_test)))  
  
#%%  
  
print(**"SVM** \n**"**)  
for method in [accuracy\_score,confusion\_matrix, classification\_report]:  
 print(**"Метод {}"**.format(method.\_\_name\_\_))  
 print(method(y\_test,linSVM.predict(X\_test)))  
  
#%%  
  
print(**"Tree** \n**"**)  
for method in [accuracy\_score,confusion\_matrix, classification\_report]:  
 print(**"Метод {}"**.format(method.\_\_name\_\_))  
 print(method(y\_test,decTree.predict(X\_test)))  
  
#%%  
  
CTP = [{**'C'**:[0.001,0.01,0.1,1,10,100,1000]}]  
GammaTP = [{**'gamma'**:[0.001,0.01,0.1,1,10,100,1000]}]  
DepthTP = [{**'max\_depth'**:[1,2,3,4,5,6,7,8,9,10]}]  
bestParams=[]  
  
for model,tuned\_parameters,koef\_name in zip([LogisticRegression(solver=**'liblinear'**),SVC(),DecisionTreeClassifier()],[CTP,GammaTP,DepthTP],[**'C'**,**'gamma'**,**'max\_depth'**]):  
 modelGS = GridSearchCV(model, tuned\_parameters, cv=LeaveOneOut(), scoring=**'accuracy'**)  
 modelGS.fit(X\_train,y\_train)  
 print(modelGS.best\_params\_.get(koef\_name))  
 bestParams.append(modelGS.best\_params\_.get(koef\_name))  
  
#%%  
  
print(**'Подобранные гиперпараметры:'**)  
print(**' Логистическая регрессия: параметр С: '**,bestParams[0])  
print(**' Метод Опорных Векторов: параметр gamma: '**,bestParams[1])  
print(**' Дерево решений: параметр глубина: '**,bestParams[2])  
  
#%%  
  
LogRegBest = LogisticRegression(C=bestParams[0], solver=**'liblinear'**)  
SVCBest = SVC(gamma = bestParams[1])  
DecTreeBest = DecisionTreeClassifier(max\_depth = bestParams[2])  
  
#%%  
  
for model in [LogRegBest,SVCBest,DecTreeBest]:  
 model.fit(X\_train,y\_train)  
  
#%%  
  
print(**"Логистическая регрессия** \n**"**)  
for method in [accuracy\_score,confusion\_matrix, classification\_report]:  
 print(**"Метод {}"**.format(method.\_\_name\_\_))  
 print(method(y\_test,LogRegBest.predict(X\_test)))  
  
#%%  
  
print(**"SVC** \n**"**)  
for method in [accuracy\_score,confusion\_matrix, classification\_report]:  
 print(**"Метод {}"**.format(method.\_\_name\_\_))  
 print(method(y\_test,SVCBest.predict(X\_test)))  
  
#%%  
  
print(**"Дерево решений** \n**"**)  
for method in [accuracy\_score,confusion\_matrix, classification\_report]:  
 print(**"Метод {}"**.format(method.\_\_name\_\_))  
 print(method(y\_test,DecTreeBest.predict(X\_test)))  
  
#%%

**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:

















