Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет**

**Лабораторная работа № 6**

**По курсу «Технологии машинного обучения»**

## «Ансамбли моделей машинного обучения»

**ИСПОЛНИТЕЛЬ:**

Сергеев И.В.

Группа ИУ5-64Б

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

"\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

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

"\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

Москва 2020

**Описание задания**

**Цель лабораторной работы -** изучение ансамблей моделей машинного обучения

**Задание:**

* Выберите набор данных (датасет) для решения задачи классификации или регресии.
* 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 две ансамблевые модели. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.

**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.model\_selection import GridSearchCV  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import classification\_report  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
import seaborn as sns  
%matplotlib inline   
sns.set(style=**"ticks"**)  
  
#%%  
  
data = pd.read\_csv(**'dataset.csv'**,sep=**","**)  
data.columns  
  
#%%  
  
data.shape  
  
#%%  
  
data[**' shares'**].mean()  
  
#%%  
  
data.head()  
  
#%%  
  
df = data[:-35000]  
df.shape  
  
#%%  
  
answers = []  
for row in range(df.shape[0]):  
 if df[**' shares'**][row]<100:   
 answers.append(0)  
 if 100 <= df[**' shares'**][row] < 1000:  
 answers.append(1)  
 if 1000 <= df[**' shares'**][row] < 10000:  
 answers.append(2)  
 if df[**' shares'**][row]>=10000:  
 answers.append(3)  
y = pd.DataFrame(data=answers)  
y.head()  
  
#%%  
  
X = df[[**' n\_non\_stop\_words'**, **' n\_non\_stop\_unique\_tokens'**,  
 **' num\_hrefs'**, **' num\_self\_hrefs'**, **' num\_imgs'**, **' num\_videos'**,  
 **' average\_token\_length'**, **' num\_keywords'**, **' data\_channel\_is\_lifestyle'**,  
 **' data\_channel\_is\_entertainment'**, **' data\_channel\_is\_bus'**,  
 **' data\_channel\_is\_socmed'**, **' data\_channel\_is\_tech'**,  
 **' data\_channel\_is\_world'**, **' kw\_min\_min'**, **' kw\_max\_min'**, **' kw\_avg\_min'**,  
 **' kw\_min\_max'**, **' kw\_max\_max'**, **' kw\_avg\_max'**, **' kw\_min\_avg'**,  
 **' kw\_max\_avg'**, **' kw\_avg\_avg'**, **' self\_reference\_min\_shares'**,  
 **' self\_reference\_max\_shares'**, **' self\_reference\_avg\_sharess'**,  
 **' weekday\_is\_monday'**, **' weekday\_is\_tuesday'**, **' weekday\_is\_wednesday'**,  
 **' weekday\_is\_thursday'**, **' weekday\_is\_friday'**, **' weekday\_is\_saturday'**,  
 **' weekday\_is\_sunday'**, **' is\_weekend'**, **' LDA\_00'**, **' LDA\_01'**, **' LDA\_02'**,  
 **' LDA\_03'**, **' LDA\_04'**, **' global\_subjectivity'**,  
 **' global\_sentiment\_polarity'**, **' global\_rate\_positive\_words'**,  
 **' global\_rate\_negative\_words'**, **' rate\_positive\_words'**,  
 **' rate\_negative\_words'**, **' avg\_positive\_polarity'**,  
 **' min\_positive\_polarity'**, **' max\_positive\_polarity'**,  
 **' avg\_negative\_polarity'**, **' min\_negative\_polarity'**,  
 **' max\_negative\_polarity'**, **' title\_subjectivity'**,  
 **' title\_sentiment\_polarity'**, **' abs\_title\_subjectivity'**,  
 **' abs\_title\_sentiment\_polarity'**]]  
X.head()  
  
#%%  
  
df.isnull().sum()  
  
#%%  
  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y, test\_size=0.3, random\_state = 1)  
  
#%%  
  
forest\_model = RandomForestClassifier()  
forest\_model.fit(X\_train,y\_train)  
forest\_prediction = forest\_model.predict(X\_test)  
print(**"Accuracy :**\t**"**,accuracy\_score(y\_test, forest\_prediction))  
print(**"Classification report:**\n**"**,classification\_report(y\_test, forest\_prediction))  
  
#%%  
  
parameters = {  
 **'n\_estimators'** : [10,20,50,100,200,500],  
 **'max\_depth'** : [8, 9, 10, 11, 12],  
 **'random\_state'** : [0],  
}  
  
clf = GridSearchCV(RandomForestClassifier(), parameters, cv=10, n\_jobs=-1)  
clf.fit(X\_train, y\_train)  
  
print(clf.score(X\_train, y\_train))  
print(clf.best\_params\_)  
  
  
#%%  
  
forest\_best\_model = RandomForestClassifier(n\_estimators=50,max\_depth=9,random\_state=0)  
forest\_best\_model.fit(X\_train,y\_train)  
forest\_best\_prediction = forest\_best\_model.predict(X\_test)  
print(**"Accuracy :**\t**"**,accuracy\_score(y\_test, forest\_best\_prediction))  
print(**"Классификации отчет:**\n**"**,classification\_report(y\_test, forest\_best\_prediction))  
  
#%%  
  
gb\_model = GradientBoostingClassifier()  
gb\_model.fit(X\_train,y\_train)  
gb\_prediction = gb\_model.predict(X\_test)  
print(**"Accuracy :**\t**"**,accuracy\_score(y\_test, gb\_prediction))  
print(**"Classification report:**\n**"**,classification\_report(y\_test, gb\_prediction))  
  
#%%  
  
parameters = {  
 **"learning\_rate"** : [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],  
 **"subsample"**:[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],  
 **'n\_estimators'** : [10],  
 **'max\_depth'** : [3],  
 **'random\_state'** : [0],  
 }  
  
clf\_gb = GridSearchCV(GradientBoostingClassifier(), parameters, cv=10, n\_jobs=-1)  
clf\_gb.fit(X\_train, y\_train)  
  
print(clf\_gb.score(X\_train, y\_train))  
print(clf\_gb.best\_params\_)  
  
  
#%%  
  
gb\_best\_model = GradientBoostingClassifier(learning\_rate=0.15,subsample=0.618,n\_estimators=10,max\_depth=3,random\_state=0)  
gb\_best\_model.fit(X\_train,y\_train)  
gb\_best\_prediction = gb\_best\_model.predict(X\_test)  
print(**"Accuracy :**\t**"**,accuracy\_score(y\_test, gb\_best\_prediction))  
print(**"Classification report:**\n**"**,classification\_report(y\_test, gb\_best\_prediction))  
  
#%%

**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:







