**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет "Информатика и системы управления"

Кафедра ИУ5 "Системы обработки информации и управления"

Технологии машинного обучения

Отчет по лабораторной работе №6

Выполнила:

Желтова Александра,

группа ИУ5-63Б

Москва, 2021

**Задание**

Разработайте макет веб-приложения, предназначенного для анализа данных.

Вариант 1. Макет должен быть реализован для одной модели машинного обучения. Макет должен позволять:

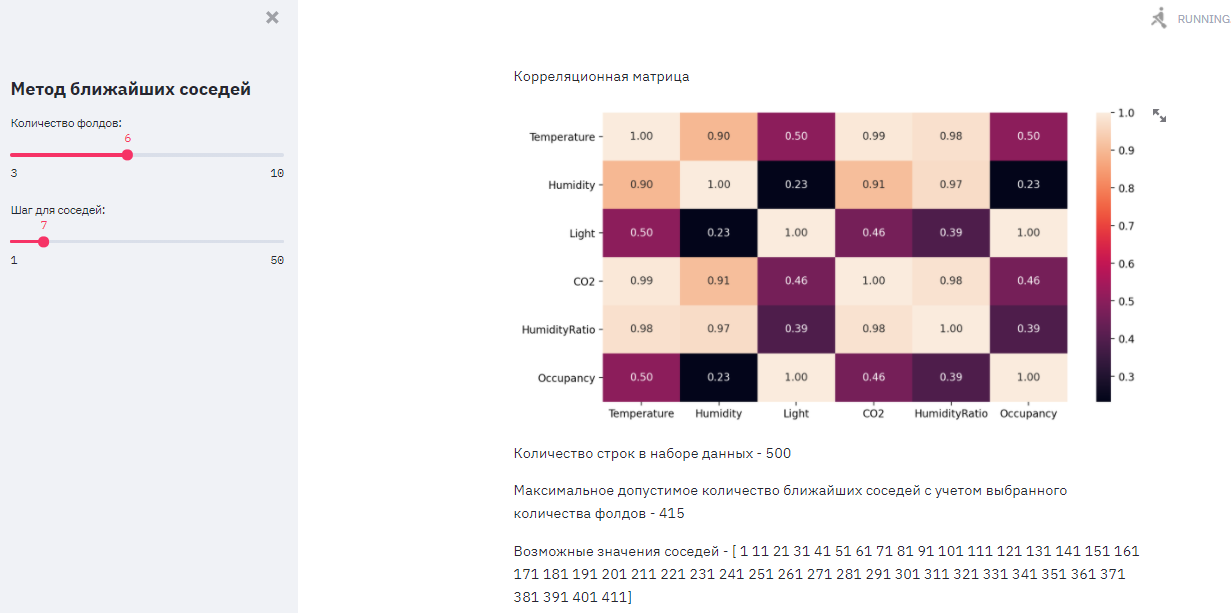
* задавать гиперпараметры алгоритма,
* производить обучение,
* осуществлять просмотр результатов обучения, в том числе в виде графиков.

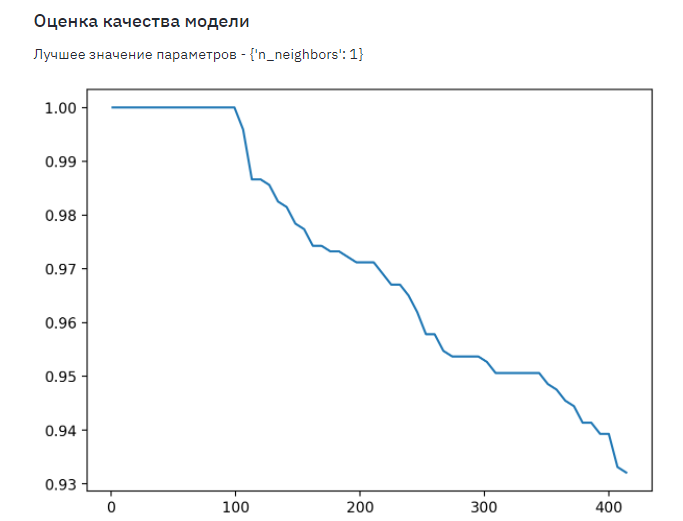
Вариант 2. Макет должен быть реализован для нескольких моделей машинного обучения. Макет должен позволять:

* выбирать модели для обучения,
* производить обучение,
* осуществлять просмотр результатов обучения, в том числе в виде графиков.

**Выполнение лабораторной работы**

Вариант 1

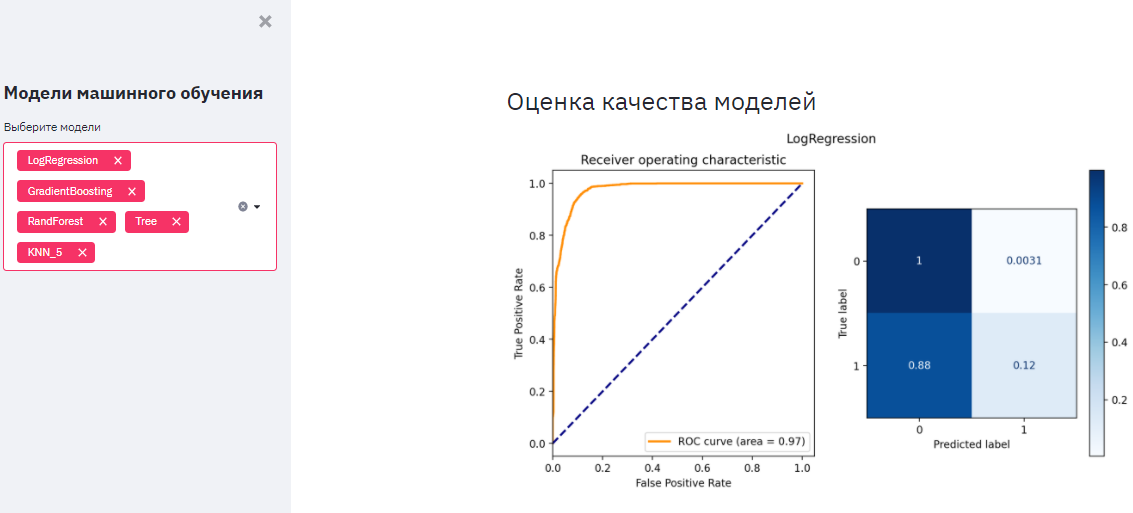
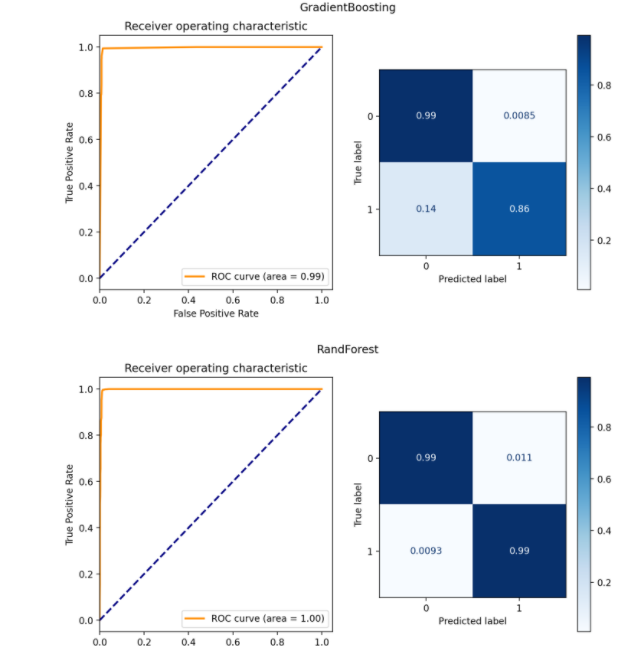


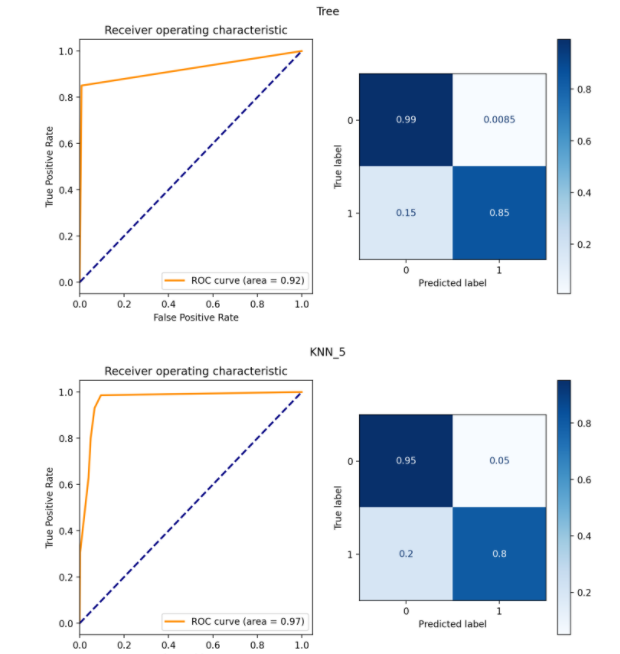


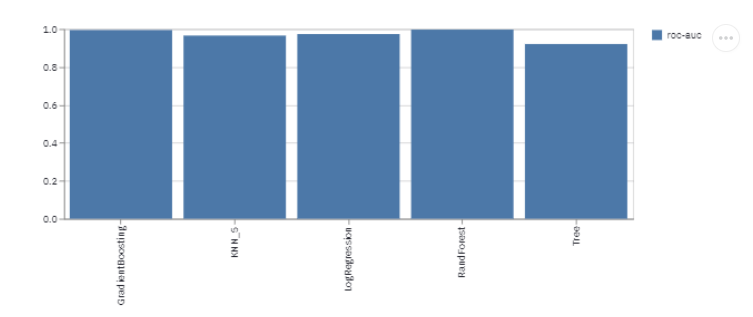
Текст программы

import streamlit as st  
import seaborn as sns  
import pandas as pd  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import GridSearchCV  
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier  
import matplotlib.pyplot as plt  
  
@st.cache  
def load\_data():  
 *'''  
 Загрузка данных  
 '''* data = pd.read\_csv(**'data/occupancy\_datatraining.txt'**, sep=**","**, nrows=500)  
 return data  
  
  
@st.cache  
def preprocess\_data(data\_in):  
 *'''  
 Масштабирование признаков, функция возвращает X и y для кросс-валидации  
 '''* data\_out = data\_in.copy()  
 *# Числовые колонки для масштабирования* scale\_cols = [**'Temperature'**, **'Humidity'**, **'Light'**, **'CO2'**]  
 new\_cols = []  
 sc1 = MinMaxScaler()  
 sc1\_data = sc1.fit\_transform(data\_out[scale\_cols])  
 for i in range(len(scale\_cols)):  
 col = scale\_cols[i]  
 new\_col\_name = col + **'\_scaled'** new\_cols.append(new\_col\_name)  
 data\_out[new\_col\_name] = sc1\_data[:,i]  
 return data\_out[new\_cols], data\_out[**'Occupancy'**]  
  
st.sidebar.header(**'Метод ближайших соседей'**)  
data = load\_data()  
cv\_slider = st.sidebar.slider(**'Количество фолдов:'**, min\_value=3, max\_value=10, value=3, step=1)  
step\_slider = st.sidebar.slider(**'Шаг для соседей:'**, min\_value=1, max\_value=50, value=10, step=1)  
  
st.write(**'Корреляционная матрица'**)  
fig1, ax = plt.subplots(figsize=(10, 5))  
sns.heatmap(data.corr(), annot=True, fmt=**'.2f'**)  
st.pyplot(fig1)  
  
*#Количество записей*data\_len = data.shape[0]  
*#Вычислим количество возможных ближайших соседей*rows\_in\_one\_fold = int(data\_len / cv\_slider)  
allowed\_knn = int(rows\_in\_one\_fold \* (cv\_slider-1))  
st.write(**'Количество строк в наборе данных - {}'**.format(data\_len))  
st.write(**'Максимальное допустимое количество ближайших соседей с учетом выбранного количества фолдов - {}'**.format(allowed\_knn))  
  
*# Подбор гиперпараметра*n\_range\_list = list(range(1, allowed\_knn, step\_slider))  
n\_range = np.array(n\_range\_list)  
st.write(**'Возможные значения соседей - {}'**.format(n\_range))  
tuned\_parameters = [{**'n\_neighbors'**: n\_range}]  
  
data\_X, data\_y = preprocess\_data(data)  
clf\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=cv\_slider, scoring=**'roc\_auc'**)  
clf\_gs.fit(data\_X, data\_y)  
  
st.subheader(**'Оценка качества модели'**)  
  
st.write(**'Лучшее значение параметров - {}'**.format(clf\_gs.best\_params\_))  
  
*# Изменение качества на тестовой выборке в зависимости от К-соседей*fig1 = plt.figure(figsize=(7,5))  
ax = plt.plot(n\_range, clf\_gs.cv\_results\_[**'mean\_test\_score'**])  
st.pyplot(fig1)

Вариант 2





Текст программы

import streamlit as st  
import pandas as pd  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.metrics import roc\_curve, roc\_auc\_score  
from sklearn.metrics import plot\_confusion\_matrix  
import matplotlib.pyplot as plt  
  
  
@st.cache  
def load\_data():  
 *'''  
 Загрузка данных  
 '''* data = pd.read\_csv(**'data/occupancy\_datatraining.txt'**, sep=**","**)  
 return data  
  
  
@st.cache  
def preprocess\_data(data\_in):  
 *'''  
 Масштабирование признаков, функция возвращает X и y для кросс-валидации  
 '''* data\_out = data\_in.copy()  
 *# Числовые колонки для масштабирования* scale\_cols = [**'Temperature'**, **'Humidity'**, **'Light'**, **'CO2'**]  
 new\_cols = []  
 sc1 = MinMaxScaler()  
 sc1\_data = sc1.fit\_transform(data\_out[scale\_cols])  
 for i in range(len(scale\_cols)):  
 col = scale\_cols[i]  
 new\_col\_name = col + **'\_scaled'** new\_cols.append(new\_col\_name)  
 data\_out[new\_col\_name] = sc1\_data[:, i]  
  
 temp\_X = data\_out[new\_cols]  
 temp\_y = data\_out[**'Occupancy'**]  
 *# Чтобы в тесте получилось низкое качество используем только 0,5% данных для обучения* X\_train, X\_test, y\_train, y\_test = train\_test\_split(temp\_X, temp\_y, train\_size=0.005, random\_state=1)  
 return X\_train, X\_test, y\_train, y\_test  
  
  
*# Отрисовка ROC-кривой*def draw\_roc\_curve(y\_true, y\_score, ax, pos\_label=1, average=**'micro'**):  
 fpr, tpr, thresholds = roc\_curve(y\_true, y\_score,  
 pos\_label=pos\_label)  
 roc\_auc\_value = roc\_auc\_score(y\_true, y\_score, average=average)  
 *# plt.figure()* lw = 2  
 ax.plot(fpr, tpr, color=**'darkorange'**,  
 lw=lw, label=**'ROC curve (area = %0.2f)'** % roc\_auc\_value)  
 ax.plot([0, 1], [0, 1], color=**'navy'**, lw=lw, linestyle=**'--'**)  
 ax.set\_xlim([0.0, 1.0])  
 ax.set\_xlim([0.0, 1.05])  
 ax.set\_xlabel(**'False Positive Rate'**)  
 ax.set\_ylabel(**'True Positive Rate'**)  
 ax.set\_title(**'Receiver operating characteristic'**)  
 ax.legend(loc=**"lower right"**)  
  
  
*# Модели*models\_list = [**'LogRegression'**, **'KNN\_5'**, **'Tree'**, **'RandForest'**, **'GradientBoosting'**]  
clas\_models = {**'LogRegression'**: LogisticRegression(),  
 **'KNN\_5'**: KNeighborsClassifier(n\_neighbors=5),  
 **'Tree'**: DecisionTreeClassifier(),  
 **'RandForest'**: RandomForestClassifier(),  
 **'GradientBoosting'**: GradientBoostingClassifier()}  
  
  
@st.cache(suppress\_st\_warning=True)  
def print\_models(models\_select, X\_train, X\_test, y\_train, y\_test):  
 current\_models\_list = []  
 roc\_auc\_list = []  
 for model\_name in models\_select:  
 model = clas\_models[model\_name]  
 model.fit(X\_train, y\_train)  
 *# Предсказание значений* Y\_pred = model.predict(X\_test)  
 *# Предсказание вероятности класса "1" для roc auc* Y\_pred\_proba\_temp = model.predict\_proba(X\_test)  
 Y\_pred\_proba = Y\_pred\_proba\_temp[:, 1]  
  
 roc\_auc = roc\_auc\_score(y\_test.values, Y\_pred\_proba)  
 current\_models\_list.append(model\_name)  
 roc\_auc\_list.append(roc\_auc)  
  
 *# Отрисовка ROC-кривых* fig, ax = plt.subplots(ncols=2, figsize=(10, 5))  
 draw\_roc\_curve(y\_test.values, Y\_pred\_proba, ax[0])  
 plot\_confusion\_matrix(model, X\_test, y\_test.values, ax=ax[1],  
 display\_labels=[**'0'**, **'1'**],  
 cmap=plt.cm.Blues, normalize=**'true'**)  
 fig.suptitle(model\_name)  
 st.pyplot(fig)  
  
 if len(roc\_auc\_list) > 0:  
 temp\_d = {**'roc-auc'**: roc\_auc\_list}  
 temp\_df = pd.DataFrame(data=temp\_d, index=current\_models\_list)  
 st.bar\_chart(temp\_df)  
  
  
st.sidebar.header(**'Модели машинного обучения'**)  
models\_select = st.sidebar.multiselect(**'Выберите модели'**, models\_list)  
  
data = load\_data()  
X\_train, X\_test, y\_train, y\_test = preprocess\_data(data)  
  
st.header(**'Оценка качества моделей'**)  
print\_models(models\_select, X\_train, X\_test, y\_train, y\_test)