|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Системы обработки информации и управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***Решение комплексной задачи машинного обучения\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_\_\_\_ИУ5-65Б\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Мащенко Е.И.\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Гапанюк Ю.Е.\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2021 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

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

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_Технологии машинного обучения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_\_\_ИУ5-65Б\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_Мащенко Елена Игоревна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсового проекта \_\_\_\_\_\_Решение комплексной задачи машинного обучения\_\_\_\_\_\_\_\_\_

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

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

Направленность КП (учебный, исследовательский, практический, производственный, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_учебная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_\_\_\_\_\_кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения проекта: 25% к \_\_\_ нед., 50% к \_\_\_ нед., 75% к \_\_ нед., 100% к \_\_\_ нед.

***Задание\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Провести типовое исследование, направленное на решение комплексной задачи машинного обучения на основе материалов дисциплины\_\_\_\_\_\_\_\_\_\_***

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

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

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

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

***Оформление курсового проекта:***

Расчетно-пояснительная записка на \_\_25\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

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

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

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

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

Дата выдачи задания « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Гапанюк Ю.Е.\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_Мащенко Е.И.\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

## Содержание

Введение 4

Описание задачи 4

Последовательность выполнения работы 4

Вывод 25

Источники 25

## Введение

Курсовой проект – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описание моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работы по дисциплине.

## Описание задачи

Решение задачи машинного обучения на основе материалов дисциплины.

В качестве датасета (https://www.kaggle.com/sakshigoyal7/credit-card-customers) был выбран набор данных, содержащий информацию о клиентах банка, указывающих свой возраст, зарплату, семейный статус, лимит кредитной карты, категорию кредитной карты и т.д.

В датасете представлены следующие признаки:

* CLIENTNUM – Номер клиента. Уникальный идентификатор клиента, владеющего счетом.
* Attrition\_Flag – целевой признак, *Existing Customer* - ещё клиент, *Attrited Custome* - ушедший клиент.
* Customer\_Age – возраст клиента в годах.
* Gender– пол, M = мужской, F = женский.
* Education\_Level - уровень образования владельца счета (пример: средняя школа, выпускник и тд.).

…

В курсовой работе решалась задача классификации для предсказания оттока клиентов в банке, выдающем кредитные карты. Бизнес-менеджер портфеля потребительских кредитных карт сталкивается с проблемой ухода клиентов. Необходимо проанализировать данные, чтобы выяснить причину этого, и использовать их для прогнозирования клиентов, которые, вероятно, откажутся.

Также необходимо было разработать макет веб-приложения, предназначенного для анализа данных. Макет должен позволять задавать гиперпараметры моделей, производить обучение и осуществлять просмотр результатов обучения, в том числе в виде графиков.

В качестве дополнительного задания: применение любой библиотеки autoML и сравнение качества моделей, построенных вручную и с помощью библиотеки autoML.

## Последовательность выполнения работы

В качестве фреймворка для веб-приложения был выбран streamlit.

Код программы:

import streamlit as st

import seaborn as sns

import pandas as pd

import numpy as np

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import f1\_score, precision\_score, recall\_score

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import plot\_confusion\_matrix

from sklearn.metrics import roc\_curve, roc\_auc\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from tpot import TPOTClassifier

class MetricLogger:

    def \_\_init\_\_(self):

        self.df = pd.DataFrame(

            {'metric': pd.Series([], dtype='str'),

             'alg': pd.Series([], dtype='str'),

             'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):

        """

        Добавление значения

        """

        # Удаление значения если оно уже было ранее добавлено

        self.df.drop(self.df[(self.df['metric'] == metric) & (self.df['alg'] == alg)].index, inplace=True)

        # Добавление нового значения

        temp = [{'metric': metric, 'alg': alg, 'value': value}]

        self.df = self.df.append(temp, ignore\_index=True)

    def get\_data\_for\_metric(self, metric, ascending=True):

        """

        Формирование данных с фильтром по метрике

        """

        temp\_data = self.df[self.df['metric'] == metric]

        temp\_data\_2 = temp\_data.sort\_values(by='value', ascending=ascending)

        return temp\_data\_2['alg'].values, temp\_data\_2['value'].values

    def plot(self, str\_header, metric, ascending=True, figsize=(5, 5)):

        """

        Вывод графика

        """

        array\_labels, array\_metric = self.get\_data\_for\_metric(metric, ascending)

        fig, ax1 = plt.subplots(figsize=figsize)

        pos = np.arange(len(array\_metric))

        rects = ax1.barh(pos, array\_metric,

                         align='center',

                         height=0.5,

                         tick\_label=array\_labels)

        ax1.set\_title(str\_header)

        for a, b in zip(pos, array\_metric):

            plt.text(0.5, a - 0.05, str(round(b, 3)), color='white')

        plt.show()

def load\_data():

    # Загрузка данных

    data = pd.read\_csv('BankChurners.csv')

    return data

# 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")

# функции для обучения моделей

def train\_model(model\_name, model, classMetricLogger, is\_print=1):

    model.fit(X\_train, Y\_train)

    # Предсказание значений

    Y\_pred = model.predict(X\_test)

    precision = precision\_score(Y\_test.values, Y\_pred)

    recall = recall\_score(Y\_test.values, Y\_pred)

    f1 = f1\_score(Y\_test.values, Y\_pred)

    roc\_auc = roc\_auc\_score(Y\_test.values, Y\_pred)

    classMetricLogger.add('precision', model\_name, precision)

    classMetricLogger.add('recall', model\_name, recall)

    classMetricLogger.add('f1', model\_name, f1)

    classMetricLogger.add('roc\_auc', model\_name, roc\_auc)

    fig, ax = plt.subplots(ncols=2, figsize=(10,5))

    draw\_roc\_curve(Y\_test.values, Y\_pred, 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)

    if is\_print == 1:

        st.pyplot(fig)

    if is\_print == 1:

        st.write(f'--------------------{model\_name}--------------------')

        st.write(model)

        st.write(f"precision\_score: {precision}")

        st.write(f"recall\_score: {recall}")

        st.write(f"f1\_score: {f1}")

        st.write(f"roc\_auc: {roc\_auc}")

        st.write(f'--------------------{model\_name}--------------------\n')

data = load\_data()

#Производим удаление последних двух столбцов, как указано в описании к данному датасету

data.drop(columns=data.columns[[data.shape[1]-2, data.shape[1]-1]], inplace=True)

st.sidebar.header('Логистический регрессор')

cs\_1 = st.sidebar.slider('Параметр регуляризации:', min\_value=3, max\_value=10, value=3, step=1)

st.sidebar.header('Модель ближайших соседей')

n\_estimators\_2 = st.sidebar.slider('Количество K:', min\_value=3, max\_value=10, value=3, step=1)

st.sidebar.header('SVC')

cs\_3 = st.sidebar.slider('Регуляризация:', min\_value=3, max\_value=10, value=3, step=1)

st.sidebar.header('Дерево решений')

max\_depth\_4 = st.sidebar.slider('Максимальная глубина:', min\_value=10, max\_value=50, value=10, step=1)

st.sidebar.header('Случайный лес')

n\_estimators\_5 = st.sidebar.slider('Количество фолдов:', min\_value=3, max\_value=10, value=3, step=1)

st.sidebar.header('Градиентный бустинг')

n\_estimators\_6 = st.sidebar.slider('Количество:', min\_value=6, max\_value=15, value=6, step=1)

st.header('Курсовой проект по анализу данных')

st.write('В качестве набора данных мы будем использовать данные об оттоке клиентов в банке, выдающем кредитные карты - https://www.kaggle.com/sakshigoyal7/credit-card-customers.')

st.write('В рассматриваемом примере будем решать задачу классификации. Для решения задачи классификации в качестве целевого признака будем использовать "Attrition\_Flag ". Поскольку признак содержит только значения 2 значения: Existing Customer (1) и Attrited Custome (0), то это задача бинарной классификации.')

# Первые пять строк датасета

st.subheader('Первые 5 значений')

st.write(data.head())

st.subheader('Размер датасета')

st.write(data.shape)

st.subheader('Колонки и их типы данных')

st.write(data.dtypes)

st.subheader('Наличие пустых значений')

st.write(data.isnull().sum())

st.write('Вывод. Представленный набор данных не содержит пропусков.')

st.subheader('Статистические данные')

st.write(data.describe())

st.subheader('Уникальные значения для категориальных признаков')

col\_obj = data.dtypes[data.dtypes==object].index.values.tolist()

for i in enumerate(col\_obj):

    uniq\_obj = data[i[1]].unique()

    st.write(f'{i[0]+1}. {i[1]}: {uniq\_obj} | КОЛ-ВО: {len(uniq\_obj)}')

st.subheader('Оценка дисбаланса классов для Attrition\_Flag')

fig1 = plt.figure(figsize=(3,3))

ax = plt.hist(data['Attrition\_Flag'])

st.pyplot(fig1)

# посчитаем дисбаланс классов

total = data.shape[0]

class\_0, class\_1 = data['Attrition\_Flag'].value\_counts()

st.write('Класс Existing Customer составляет {}%, а класс Attrited Custome составляет {}%.'

      .format(round(class\_0 / total, 4)\*100, round(class\_1 / total, 4)\*100))

st.write('Вывод. Дисбаланс классов присутствует.')

st.subheader('Кодирование категориальных признаков')

st.write('Закодируем признак Gender и целевой признак Attrition\_Flag на основе подхода LabelEncoding. Для остальных категориальных признаков используем метод One-hot Encoding')

le = LabelEncoder()

data['Gender'] = le.fit\_transform(data['Gender'])

mapping = {'Existing Customer' : 0, 'Attrited Customer' : 1}

data['Attrition\_Flag'] = data['Attrition\_Flag'].apply(lambda x: mapping[x])

#Скопируем датасет для применения к его категориальным признакам метода 'Label encoding' с целью удобства дальнейшего представления корреляционной матрицы, а также последующего применения при использовании классификатора дерева решений

data\_copy = data

data\_copy = data\_copy.apply(LabelEncoder().fit\_transform)

data = pd.get\_dummies(data, drop\_first=True)

st.write('Убедимся, что целевой признак для задачи бинарной классификации содержит только 0 и 1.')

st.write(data['Attrition\_Flag'].unique())

st.subheader('Корреляционная матрица')

fig1, ax = plt.subplots(figsize=(20, 15))

sns.heatmap(data\_copy.corr())

st.pyplot(fig1)

data\_all=data.copy()

st.subheader('Масштабирование данных')

st.write('Масштабирование выполняем на основе Z-оценки.')

scale\_cols = data.dtypes[data.dtypes!=object].index.values.tolist()

scale\_cols.remove('Attrition\_Flag')

scale\_cols.remove('Gender')

se = StandardScaler()

sc1\_data = se.fit\_transform(data[scale\_cols])

data[scale\_cols] = se.fit\_transform(data[scale\_cols])

scaled\_cols = data\_copy.dtypes[data\_copy.dtypes!=object].index.values.tolist()

scaled\_cols.remove('Attrition\_Flag')

scaled\_cols.remove('Gender')

data\_copy[scaled\_cols] = se.fit\_transform(data\_copy[scaled\_cols])

# Добавим масштабированные данные в набор данных

for i in range(len(scale\_cols)):

    col = scale\_cols[i]

    new\_col\_name = col + '\_scaled'

    data\_all[new\_col\_name] = sc1\_data[:, i]

st.subheader('Проверим, что масштабирование не повлияло на распределение данных')

for col in scale\_cols:

    col\_scaled = col + '\_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8, 3))

    ax[0].hist(data\_all[col], 50)

    ax[1].hist(data\_all[col\_scaled], 50)

    ax[0].title.set\_text(col)

    ax[1].title.set\_text(col\_scaled)

    st.pyplot(fig)

st.subheader('Корреляционная матрица после масштабирование')

fig1, ax = plt.subplots(figsize=(20, 15))

sns.heatmap(data\_copy.corr())

st.pyplot(fig1)

st.subheader('Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.')

st.write('На основе корреляционной матрицы можно сделать следующие выводы:')

st.write('1. Корреляционные матрицы для исходных и масштабированных данных совпадают.')

st.write('2. Среди двух сильно коррелирующих признаков "Credit\_Limit"  и "Avg\_Open\_To\_Buy" необходимо удалить признак "Avg\_Open\_To\_Buy", так как он сильнее коррелирует с другими признаками объектов и слабее коррелирует с целевым признаком')

st.write('3. Среди двух сильно коррелирующих признаков "Total\_Trans\_Amt"  и "Total\_Trans\_Ct" необходимо удалить признак "Total\_Trans\_Amt", так как он сильнее коррелирует с другими признаками объектов')

data\_copy.drop(columns=['Avg\_Open\_To\_Buy', 'Total\_Trans\_Amt'], inplace=True)

data.drop(columns=['Avg\_Open\_To\_Buy', 'Total\_Trans\_Amt'], inplace=True)

st.write(data.head())

# разделение выборки на обучающую и тестовую

# X\_train, X\_test, Y\_train, Y\_test, X, Y = preprocess\_data(data)

X = data.drop("Attrition\_Flag", axis=1)

Y = data["Attrition\_Flag"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=1)

st.subheader('Выбор метрик для последующей оценки качества моделей.')

st.write('В качестве метрик для решения задачи классификации будем использовать:')

st.write('1. Метрика precision')

st.write('2. Метрика recall (полнота)')

st.write('3. Метрика F1-мера')

st.write('4. Метрика ROC AUC')

st.subheader('Выбор метрик для последующей оценки качества моделей.')

st.write('В качестве метрик для решения задачи классификации будем использовать:')

st.write('1. Логистическая регрессия')

st.write('2. Метод ближайших соседей')

st.write('3. Машина опорных векторов')

st.write('4. Решающее дерево')

st.write('5. Случайный лес')

st.write('6. Градиентный бустинг')

st.subheader('Обучим модели')

# Модели

models = {'LogR': LogisticRegression(C=cs\_1),

          'KNN': KNeighborsClassifier(n\_neighbors=n\_estimators\_2),

          'SVC': SVC(C=cs\_3, probability=True),

          'Tree': DecisionTreeClassifier(max\_depth=max\_depth\_4, random\_state=10),

          'RF': RandomForestClassifier(n\_estimators=n\_estimators\_5, oob\_score=True, random\_state=10),

          'GB': GradientBoostingClassifier(n\_estimators=n\_estimators\_6, random\_state=10)}

# Сохранение метрик

classMetricLogger = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLogger)

st.set\_option('deprecation.showPyplotGlobalUse', False)

st.subheader('Сравнение метрик моделей')

# Метрики качества модели

metrics = classMetricLogger.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Лучшее подобранное значение параметра регуляризации для логистической регрессии:')

params = {'C': np.logspace(1, 3, 20)}

grid\_lr = GridSearchCV(estimator=LogisticRegression(),

                       param\_grid=params,

                       cv=3,

                       n\_jobs=-1)

grid\_lr.fit(X\_train, Y\_train)

st.write(grid\_lr.best\_params\_)

st.subheader('Сравним с baseline-моделью')

# Модели

models = {'LogR': LogisticRegression(C=cs\_1),

          'LogRGrid': grid\_lr.best\_estimator\_}

# Сохранение метрик

classMetricLoggerLogR = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLoggerLogR)

    train\_model(model\_name, model, classMetricLogger, 0)

# Метрики качества модели

metrics = classMetricLoggerLogR.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLoggerLogR.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Лучшее значение количества ближайших соседей для модели ближайших соседей:')

params = {'n\_neighbors': list(range(5, 100, 5))}

grid\_knn = GridSearchCV(estimator=KNeighborsClassifier(),

                        param\_grid=params,

                        cv=3,

                        n\_jobs=-1)

grid\_knn.fit(X\_train, Y\_train)

st.write(grid\_knn.best\_params\_)

st.subheader('Сравним с baseline-моделью')

# Модели

models = {'KNN': KNeighborsClassifier(n\_neighbors=n\_estimators\_2),

          'KNNGrid': grid\_knn.best\_estimator\_}

# Сохранение метрик

classMetricLoggerKNN = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLoggerKNN)

    train\_model(model\_name, model, classMetricLogger, 0)

# Метрики качества модели

metrics = classMetricLoggerKNN.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLoggerKNN.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Лучшее значение параметра регуляризации для SVC модели')

params = {'C': np.logspace(1, 3, 20)}

grid\_svc = GridSearchCV(estimator=SVC(),

                        param\_grid=params,

                        cv=3,

                        n\_jobs=-1)

grid\_svc.fit(X\_train, Y\_train)

st.write(grid\_svc.best\_params\_)

st.subheader('Сравним с baseline-моделью')

# Модели

models = {'SVC': SVC(C=cs\_3, probability=True),

          'SVCGrid': grid\_knn.best\_estimator\_}

# Сохранение метрик

classMetricLoggerSVC = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLoggerSVC)

    train\_model(model\_name, model, classMetricLogger, 0)

# Метрики качества модели

metrics = classMetricLoggerSVC.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLoggerSVC.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Лучшее значение максимальной глубины для дерева решений:')

params = {'max\_depth': list(range(5, 500, 10))}

grid\_dtc = GridSearchCV(estimator=DecisionTreeClassifier(),

                        param\_grid=params,

                        cv=3,

                        n\_jobs=-1)

grid\_dtc.fit(X\_train, Y\_train)

st.write(grid\_dtc.best\_params\_)

st.subheader('Сравним с baseline-моделью')

# Модели

models = {'Tree': DecisionTreeClassifier(max\_depth=max\_depth\_4, random\_state=10),

          'TreeGrid': grid\_knn.best\_estimator\_}

# Сохранение метрик

classMetricLoggerTree = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLoggerTree)

    train\_model(model\_name, model, classMetricLogger, 0)

# Метрики качества модели

metrics = classMetricLoggerTree.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLoggerTree.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Лучшее значение количества фолдов для случайного леса:')

params = {'n\_estimators': list(range(5, 200, 10))}

grid\_rfc = GridSearchCV(estimator=RandomForestClassifier(),

                        param\_grid=params,

                        cv=3,

                        n\_jobs=-1)

grid\_rfc.fit(X\_train, Y\_train)

st.write(grid\_rfc.best\_params\_)

st.subheader('Сравним с baseline-моделью')

# Модели

models = {'RF': RandomForestClassifier(n\_estimators=n\_estimators\_5, oob\_score=True, random\_state=10),

          'RFGrid': grid\_knn.best\_estimator\_}

# Сохранение метрик

classMetricLoggerRF = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLoggerRF)

    train\_model(model\_name, model, classMetricLogger, 0)

# Метрики качества модели

metrics = classMetricLoggerRF.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLoggerRF.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Лучшее значение количества фолдов для градиентного бустинга:')

params = {'n\_estimators': list(range(5, 200, 10))}

grid\_gbc = GridSearchCV(estimator=GradientBoostingClassifier(),

                        param\_grid=params,

                        cv=3,

                        n\_jobs=-1)

grid\_gbc.fit(X\_train, Y\_train)

st.write(grid\_gbc.best\_params\_)

st.subheader('Сравним с baseline-моделью')

# Модели

models = {'GB': GradientBoostingClassifier(n\_estimators=n\_estimators\_6, random\_state=10),

          'GBGrid': grid\_knn.best\_estimator\_}

# Сохранение метрик

classMetricLoggerGB = MetricLogger()

for model\_name, model in models.items():

    train\_model(model\_name, model, classMetricLoggerGB)

    train\_model(model\_name, model, classMetricLogger, 0)

# Метрики качества модели

metrics = classMetricLoggerGB.df['metric'].unique()

# Построим графики метрик качества модели

for metric in metrics:

    st.pyplot(classMetricLoggerGB.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

st.subheader('Сравнение метрик для всех моделей')

metrics = classMetricLoggerGB.df['metric'].unique()

# Построим графики метрик качества всех моделей

for metric in metrics:

    st.pyplot(classMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6)))

# ---------------------------------------------------------------------------------------------------------------------

# ---------------------------------------------------------------------------------------------------------------------

st.subheader('Модель обученная с помощью TPOT')

tpot = TPOTClassifier(generations=5, population\_size=50,cv=5, verbosity=2, random\_state=1)

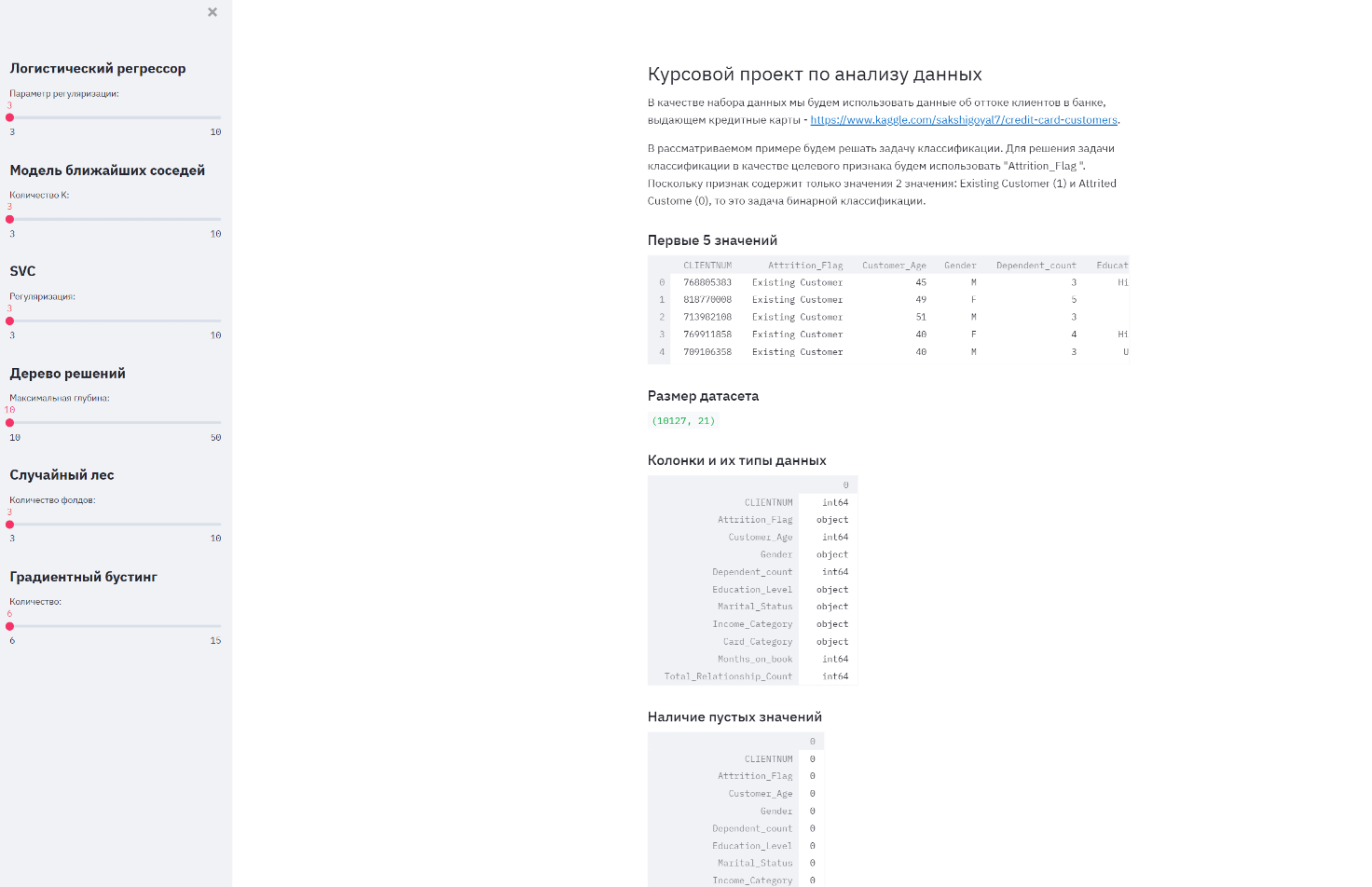
tpot.fit(X\_train, Y\_train)

st.write(tpot.score(X\_test, Y\_test))

tpot.export('tpot\_digits\_pipeline.py')

Скрины веб-приложения:

Просматриваем характеристики датасета



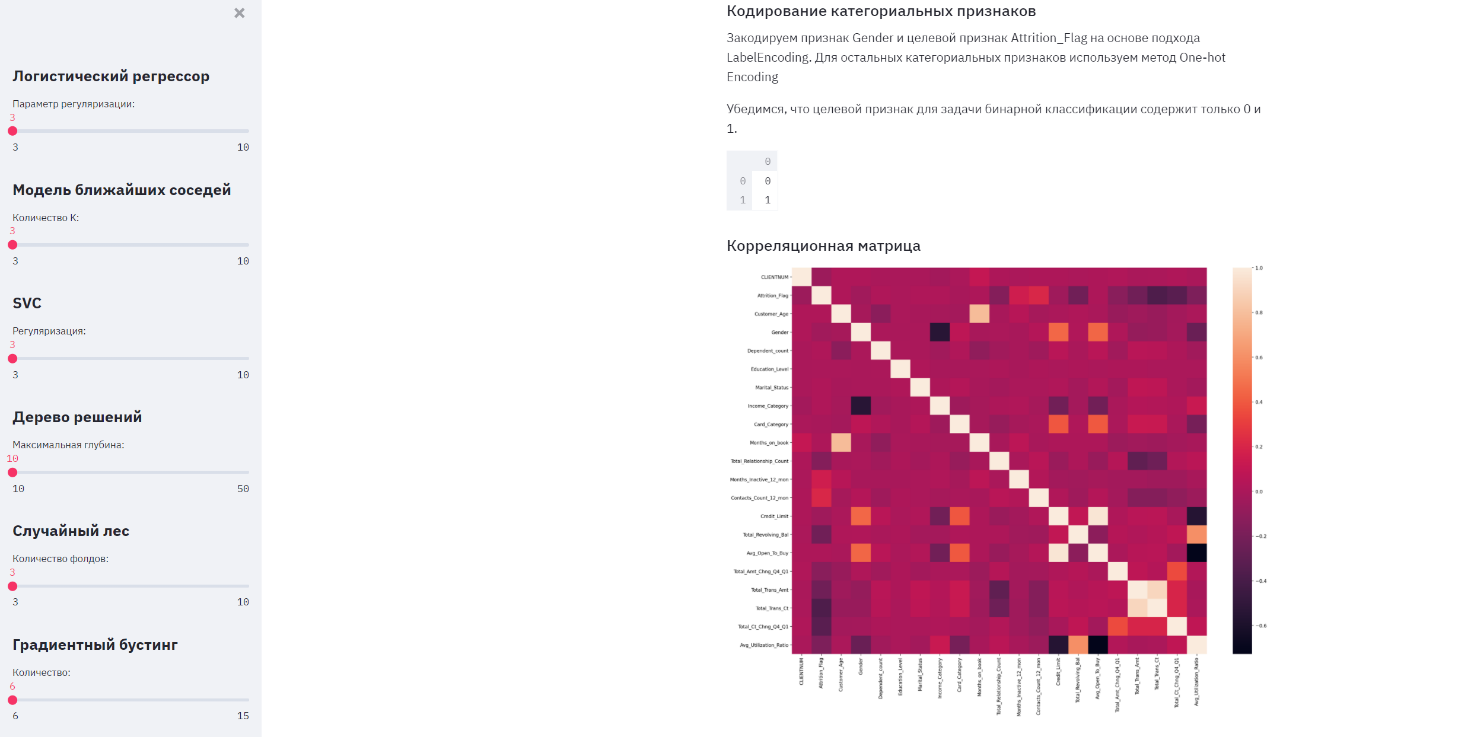


Пропусков в данных нет.

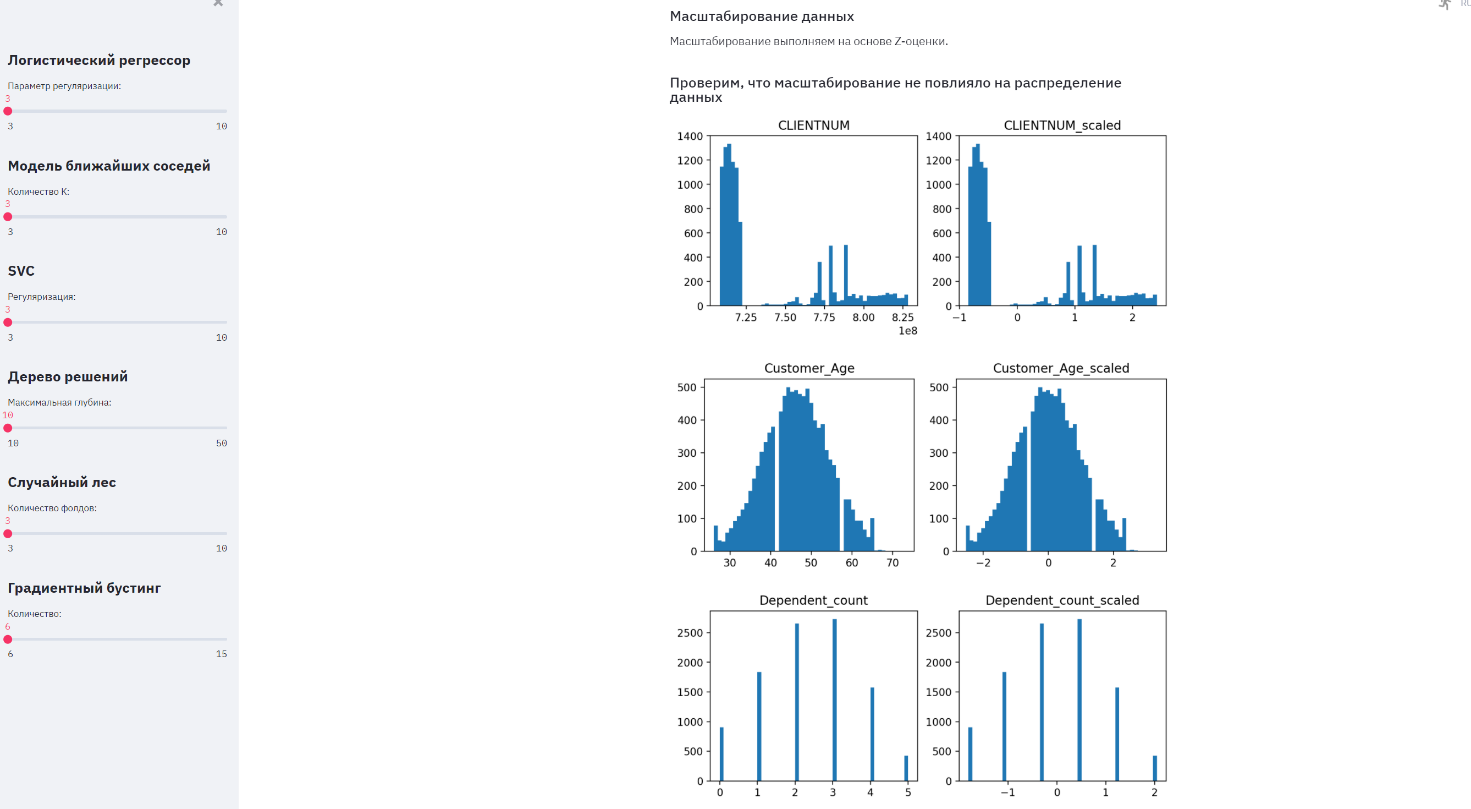
Класс Existing Customer составляет 83.93%, а класс Attrited Custome составляет 16.07%.

Дисбаланс классов присутствует.

Выполним кодирование категориальных признаков и построим корреляционную матрицу.



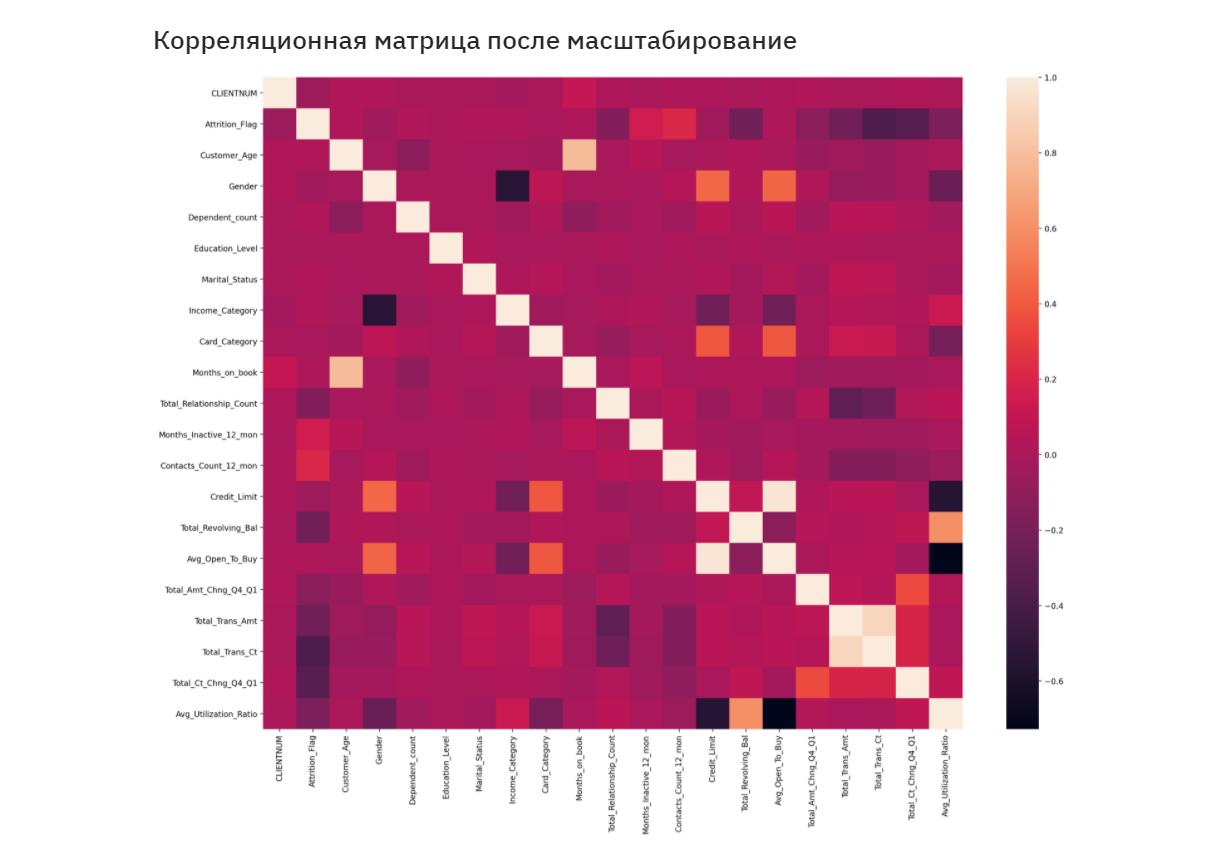
Проведем масштабирование данных



…

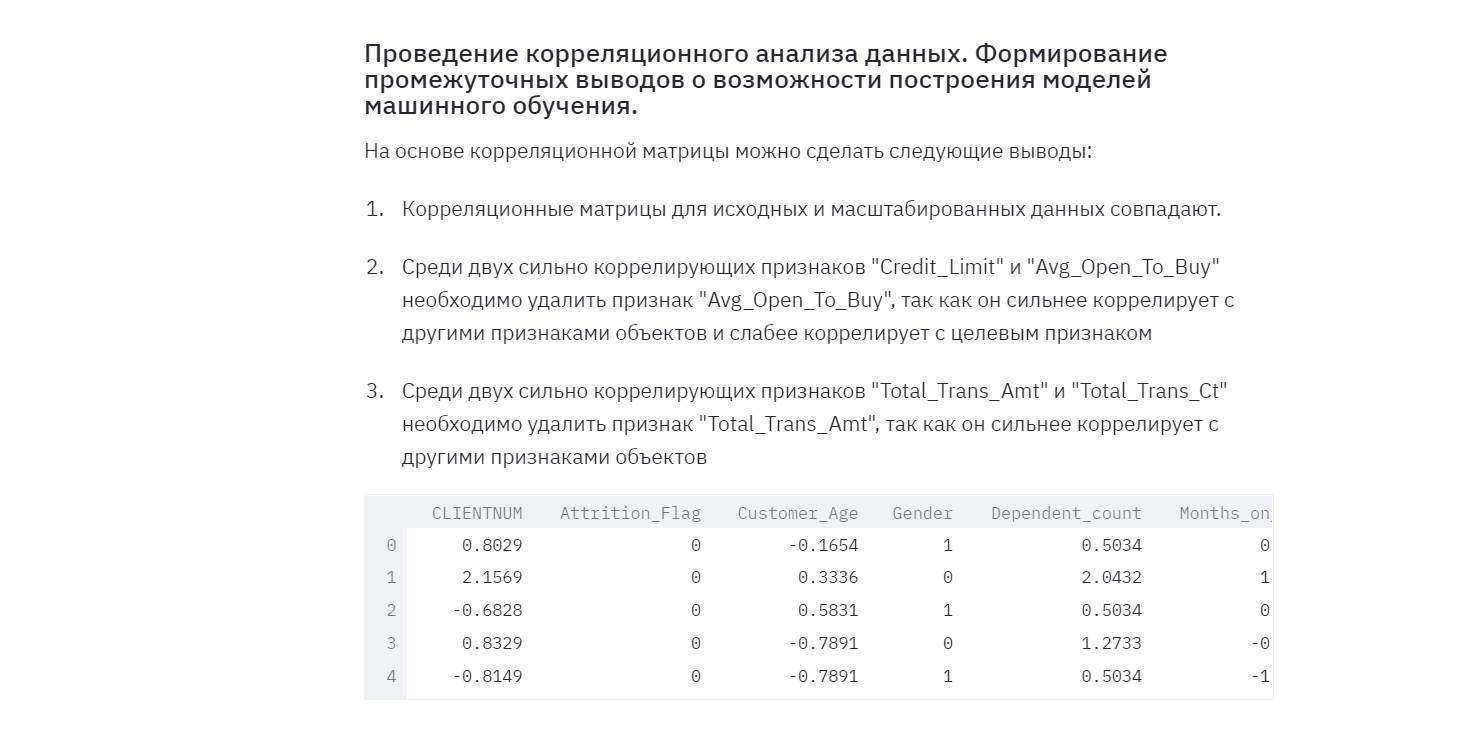
Как видно масштабирование не повлияло на распределение данных.

Проверим, что масштабирование не повлияло на корреляцию признаков.



Как видно масштабирование не повлияло на корреляцию признаков.

Проведем анализ корреляционной матрицы и удалим лишние признаки.



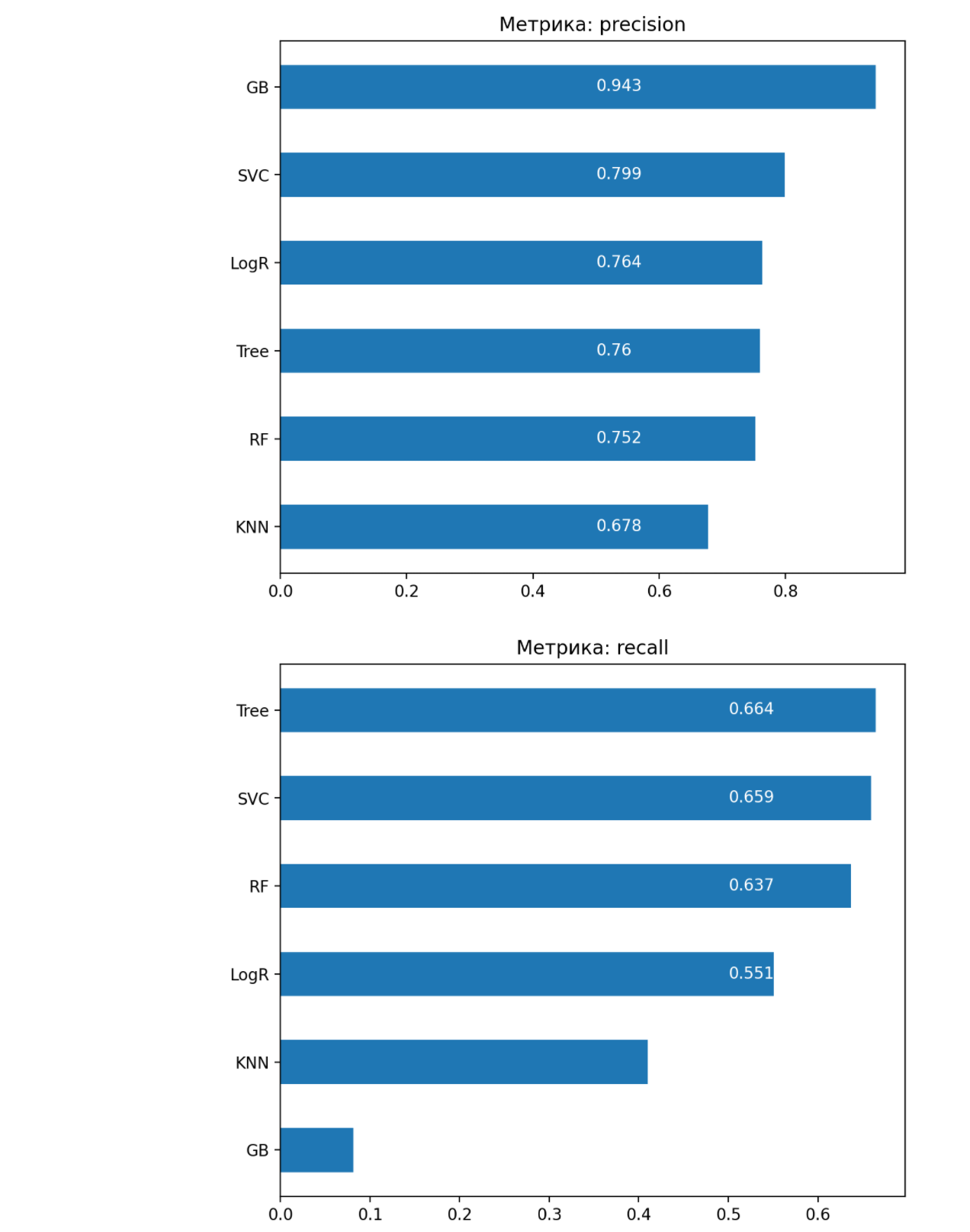
Разделим выборку на обучающую и тестовую и обучим модели с параметрами, заданными слайдерами в левой части интерфейса.

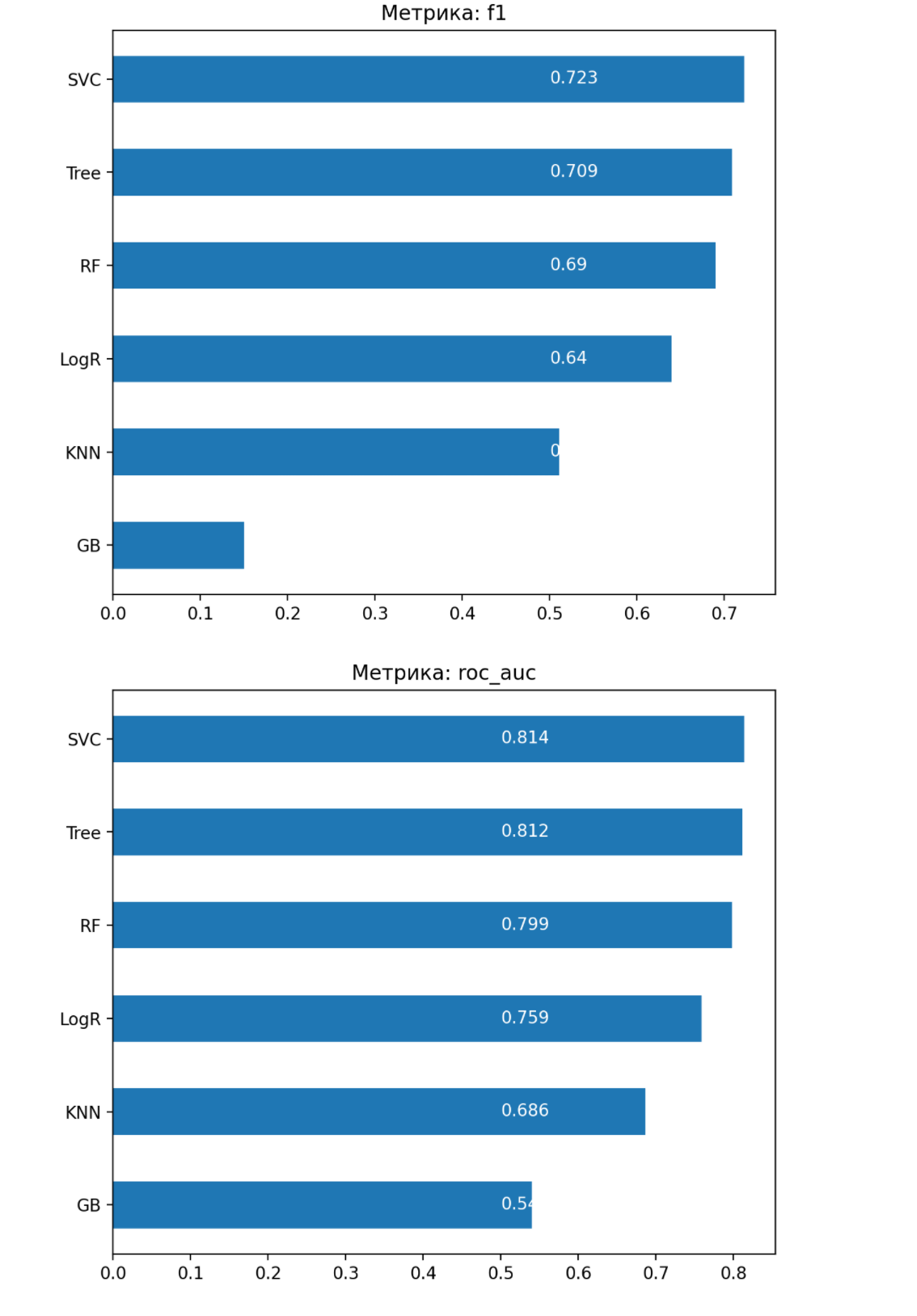






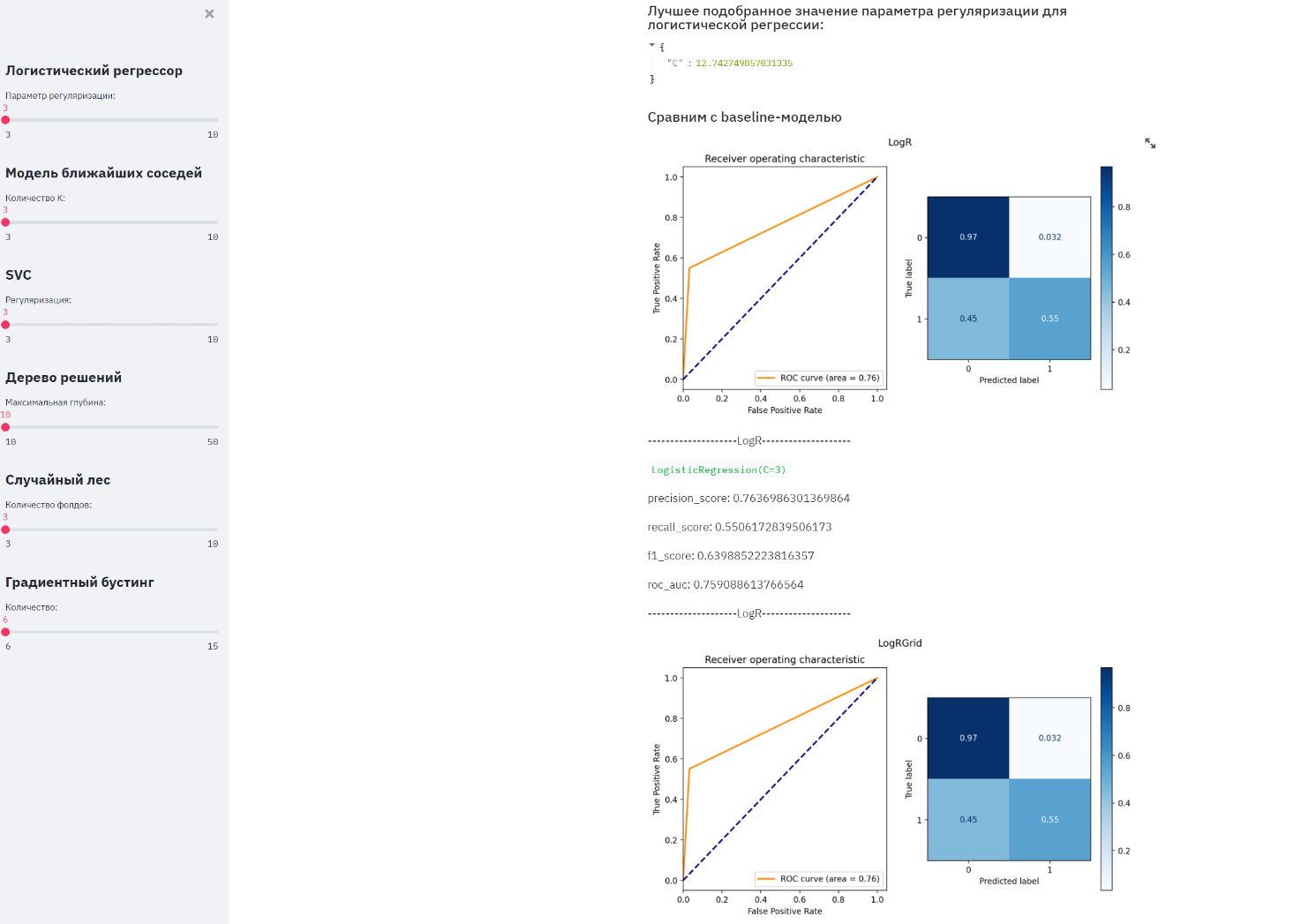
Выведем значение метрик в виде графиков.

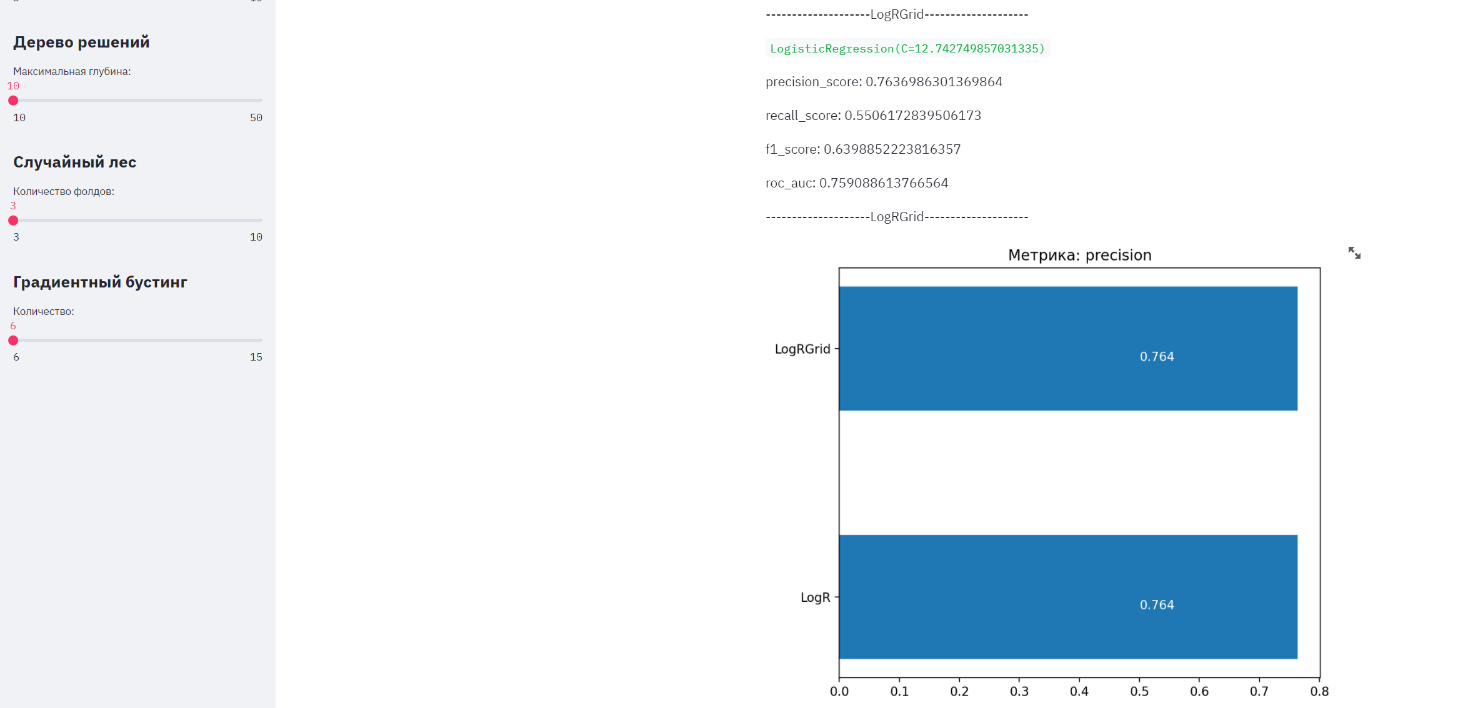


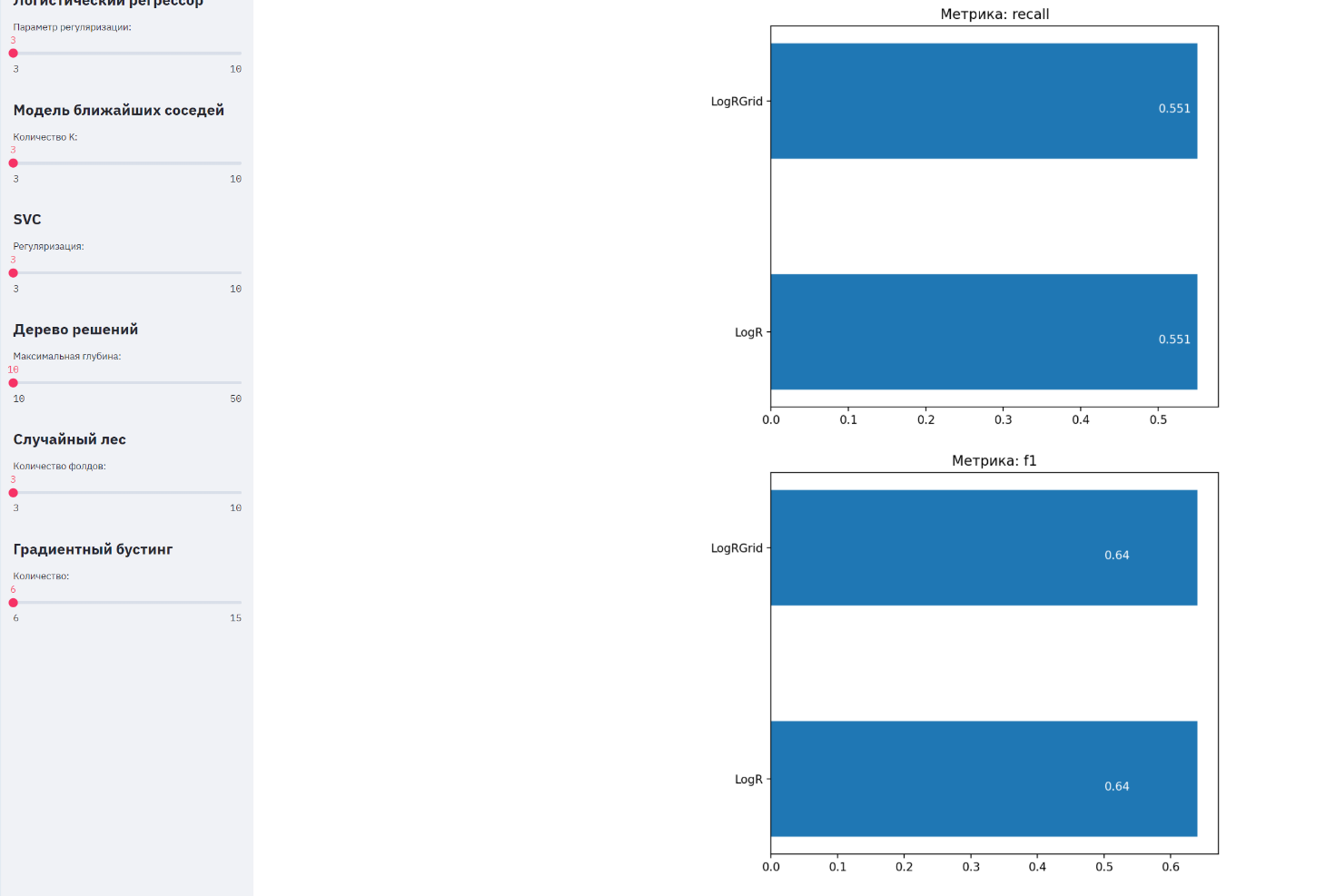


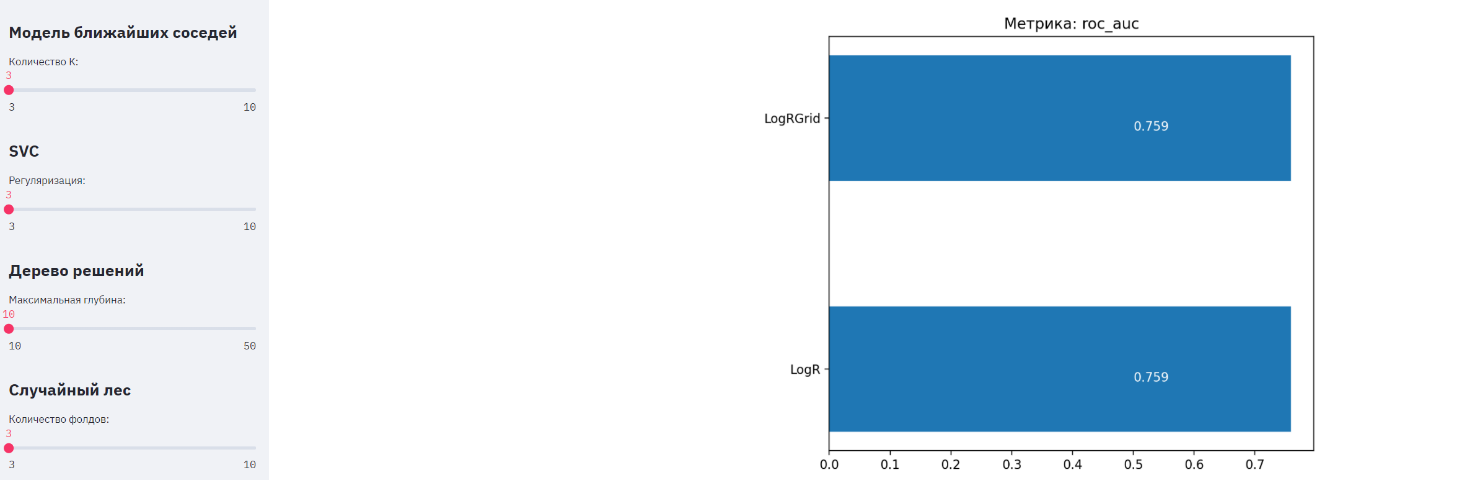
Т.к. F1-мера и ROC\_AUC лучше всех у модели SVC, можно сделать вывод, что метод опорных векторов показал себя лучше всех.

Подберем гиперпараметры для логистической регрессии.





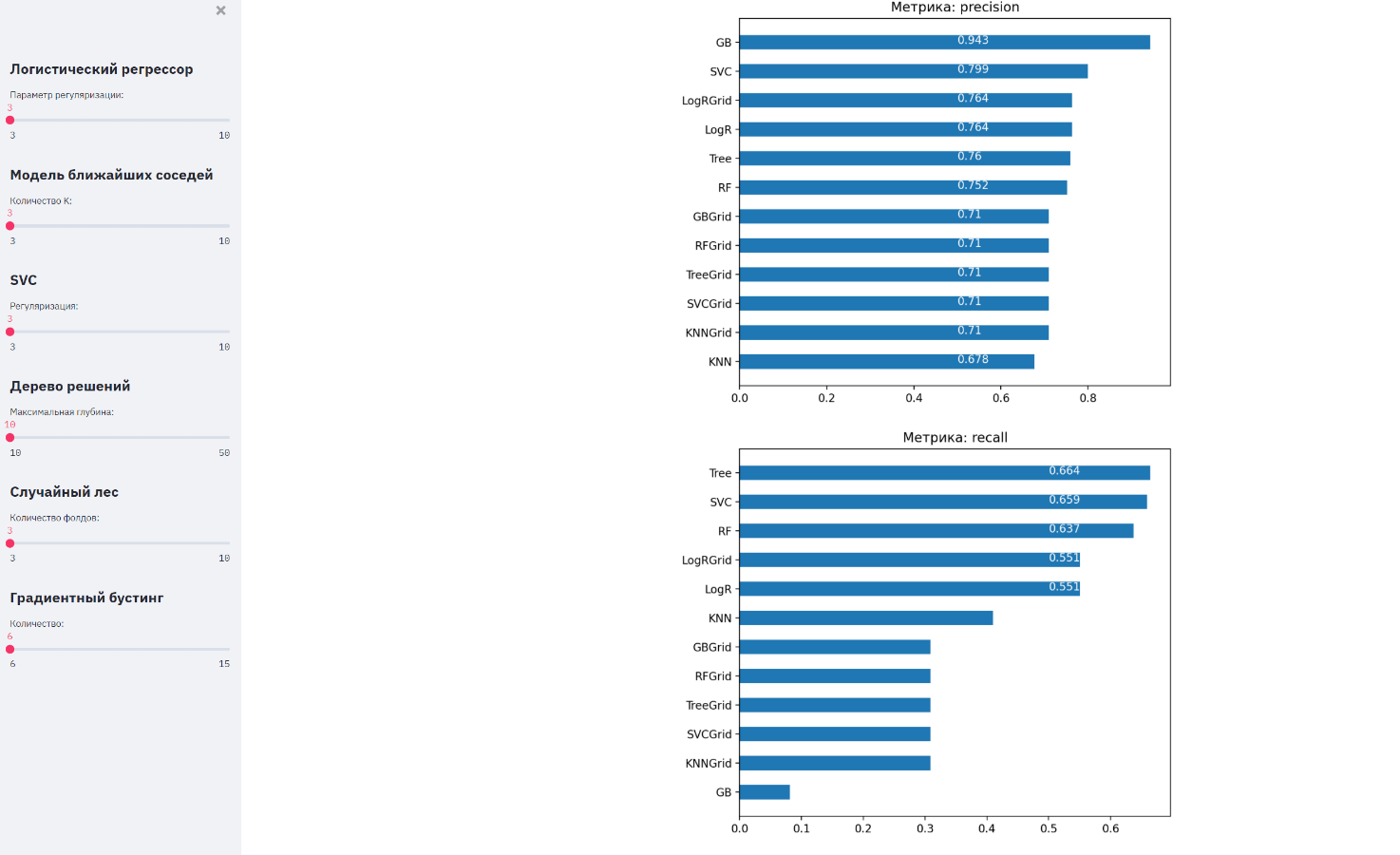


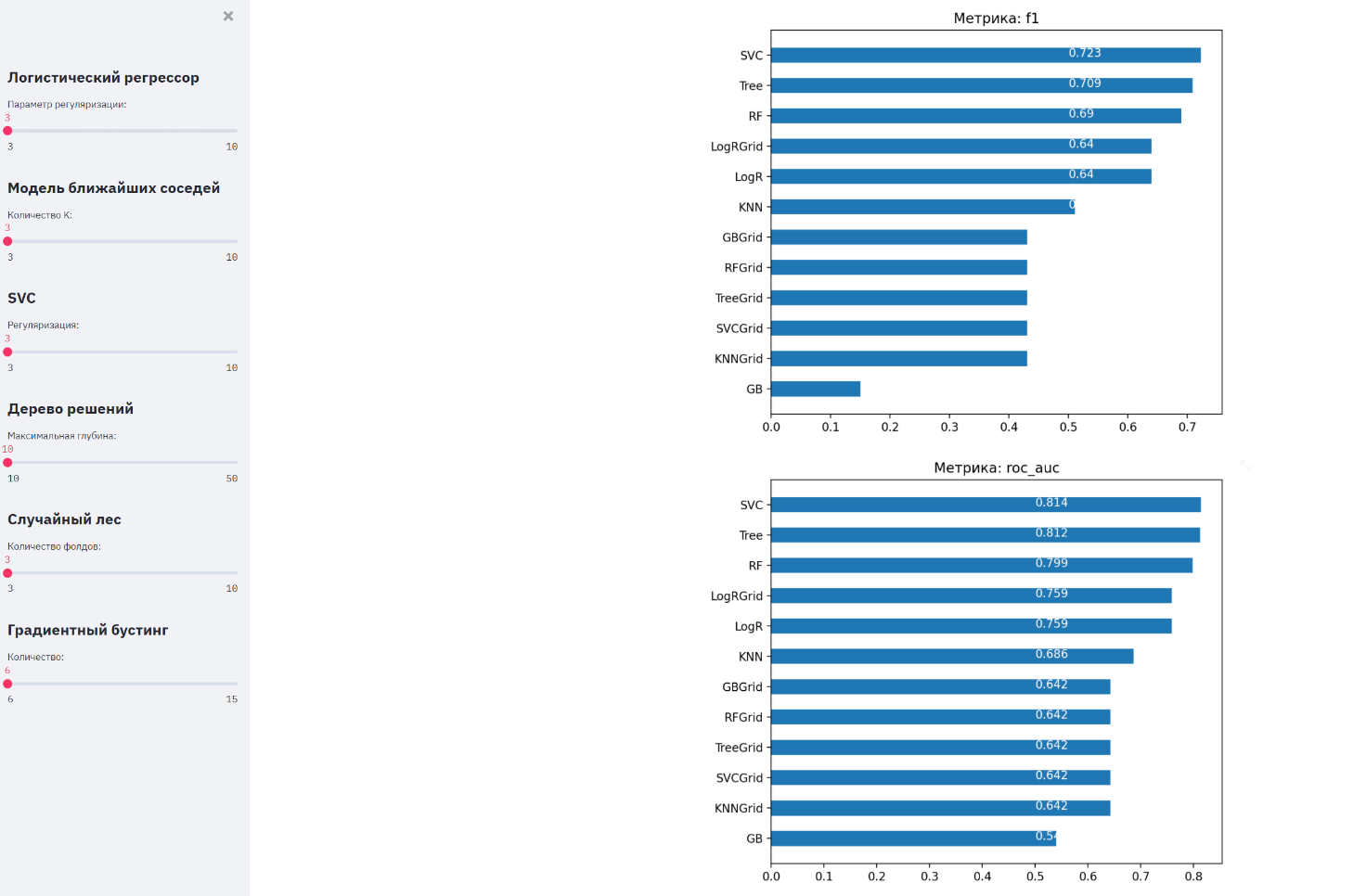


Как видно с подобранным параметром регуляризации логистическая регрессия не изменилась.

Аналогичные действия проводим для остальных моделей.

Итоговые результаты для всех моделей





Обучим модель с помощью TPOT.

## Вывод

В качестве метрик были выбраны следующие:

* precision – точность, т.е. доля объектов, названных классификатором положительными и при этом действительно являющиеся положительными
* recall – полнота, т.е. доля объектов положительного класса, которую нашел классификатор из всех объектов положительного класса
* F1-мера – среднее гармоническое precision и recall
* Roc\_auc - Используется для оценки качества бинарной классификации. Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В результате работы по метрике precision лучше всего себя показала модель градиентного бустинга, по метрике recall лучше всего себя показала модель решающего дерева, по F1-метрике лучше всего себя показала модель методов опорных векторов и по метрике roc\_auc она же. Исходя из этого, можно сделать вывод, что лучшей моделью оказалась SVC.

## Источники

1. <https://github.com/ugapanyuk/ml_course_2021/wiki/COURSE_TMO>
2. <https://www.sklearn.org>
3. <http://epistasislab.github.io/tpot/>
4. <https://www.kaggle.com/sakshigoyal7/credit-card-customers>
5. https://docs.streamlit.io/en/stable/