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_____ _____ **Попов М.А.**
(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.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Технологии машинного обучения_____

Студент группы _____ИУ5-65Б_____

_____Попов Михаил Александрович_____
(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 _____ **_____Попов М.А.___**
(Подпись, дата) (И.О.Фамилия)

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

Содержание

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 был выбран набор данных, содержащий информацию о звездах и их принадлежности к пульсарам.

В датасете представлены следующие признаки:

- Mean of the integrated profile – среднее значение интегрированного профиля
- Standard deviation of the integrated profile – стандартное отклонение интегрированного профиля
- Excess kurtosis of the integrated profile – чрезмерный эксцесс интегрированного профиля
- Skewness of the integrated profile – перекос интегрированного профиля
- Mean of the DM-SNR curve – среднее значение кривой DM-SNR
- Standard deviation of the DM-SNR curve – стандартное отклонение кривой DM-SNR
- Excess kurtosis of the DM-SNR curve – избыточный эксцесс кривой DM-SNR
- Skewness of the DM-SNR curve – асимметрия кривой DM-SNR
- target_class – 0 (не является пульсаром), 1(является пульсаром)

В курсовой работе решалась задача классификации звезд, т.е. принадлежность к пульсарам или нет.

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 MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
import autosklearn.classification as ask
import autosklearn.metrics as askm


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('data/pulsar_stars.csv')
    return data


# 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)

    classMetricLogger.add('precision', model_name, precision)
    classMetricLogger.add('recall', model_name, recall)
    classMetricLogger.add('f1', model_name, f1)

    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'--------------------{model_name}--------------------\n')


data = load_data()

parts = np.split(data, [10], axis=1)
data = parts[0]

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)

# Первые пять строк датасета
st.subheader('Первые 5 значений')
st.write(data.head())

st.subheader('Размер датасета')
st.write(data.shape)

st.subheader('Количество нулевых элементов')
st.write(data.isnull().sum())

st.subheader('Колонки и их типы данных')
st.write(data.dtypes)

st.subheader('Статистические данные')
st.write(data.describe())

# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
st.subheader('Целевой признак содержит только 0 и 1')
st.write(data['target_class'].unique())

st.subheader('Корреляционная матрица')
fig1, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(data.corr(), annot=True, fmt='.2f')
st.pyplot(fig1)

# разделение выборки на обучающую и тестовую
# X_train, X_test, Y_train, Y_test, X, Y = preprocess_data(data)
X = data.drop("target_class", axis=1)
Y = data["target_class"]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)

# Числовые колонки для масштабирования
scale_cols = [' Mean of the integrated profile',
              ' Standard deviation of the integrated profile',
              ' Excess kurtosis of the integrated profile',
              ' Skewness of the integrated profile',
              ' Mean of the DM-SNR curve',
              ' Standard deviation of the DM-SNR curve',
              ' Excess kurtosis of the DM-SNR curve',
              ' Skewness of the DM-SNR curve']
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
```

```
    new_col_name = col + '_scaled'
    data[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=(10, 5))
    ax[0].hist(data[col], 50)
    ax[1].hist(data[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')
scale_cols_postfix = [x + '_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['target_class']
fig1, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
st.pyplot(fig1)

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)
```

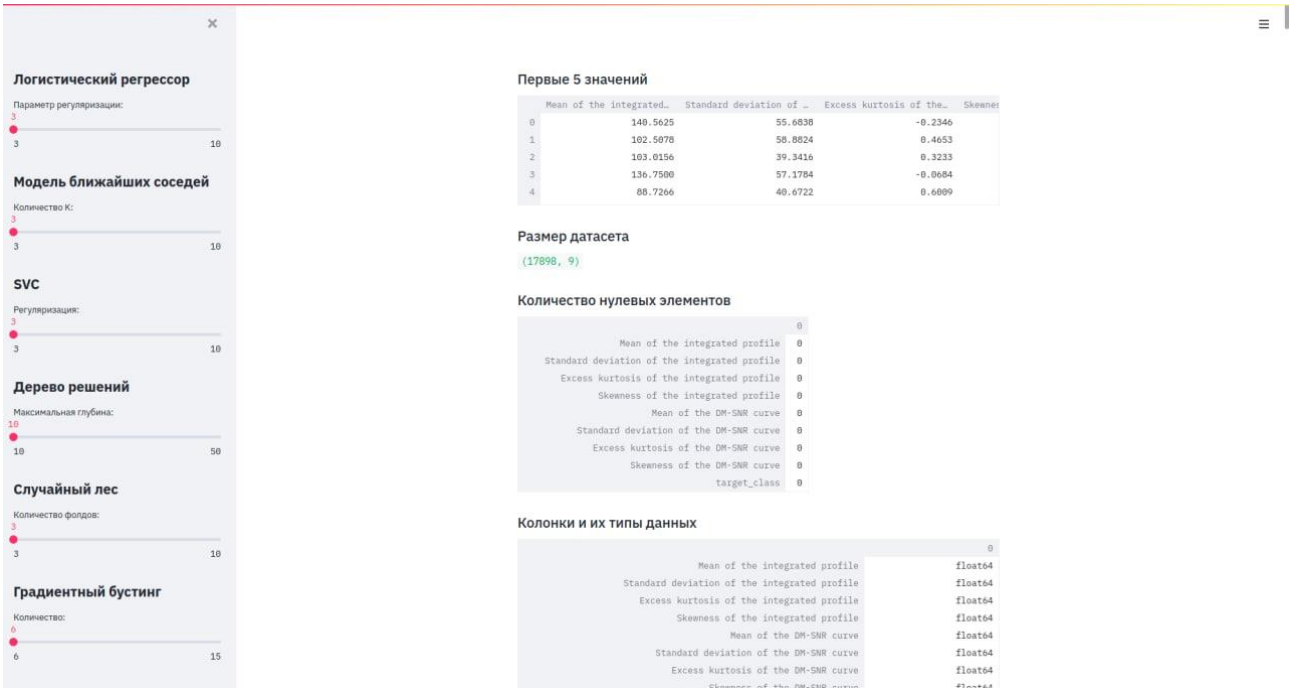
```
grid_svc.fit(X_train, Y_train)
st.write(grid_svc.best_params_)

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)
```

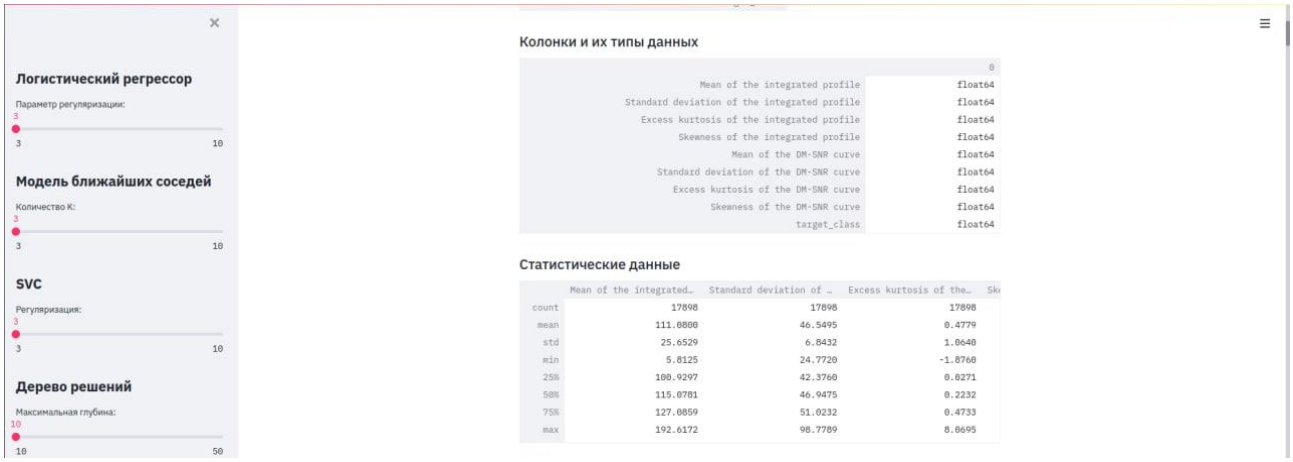
```
grid_rfc.fit(X_train, Y_train)
st.write(grid_rfc.best_params_)

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 auto-sklearn')
data2 = pd.read_csv('data/pulsar_stars.csv')
X = data2.drop("target_class", axis=1)
Y = data2["target_class"]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)
# define the model
TIME_BUDGET = 500
automl = ask.AutoSklearnClassifier(time_left_for_this_task=TIME_BUDGET, metric=askm.f1)
# train the model
automl.fit(X_train, Y_train)
# predict
Y_pred = automl.predict(X_test)
# score
precision = precision_score(Y_test, Y_pred)
recall = recall_score(Y_test, Y_pred)
f1 = f1_score(Y_test, Y_pred)
st.write(f"precision_score: {precision}")
st.write(f"recall_score: {recall}")
st.write(f"f1_score: {f1}")
# show all models
show_models_str = automl.show_models()
st.write(show_models_str)
sprint_statistics_str = automl.sprint_statistics()
st.write(sprint_statistics_str)
```

<u>Скрины веб-приложения:</u>

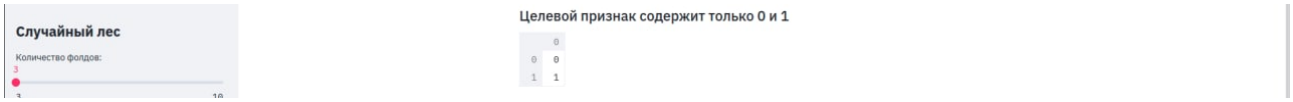
Просматриваем характеристики датасета



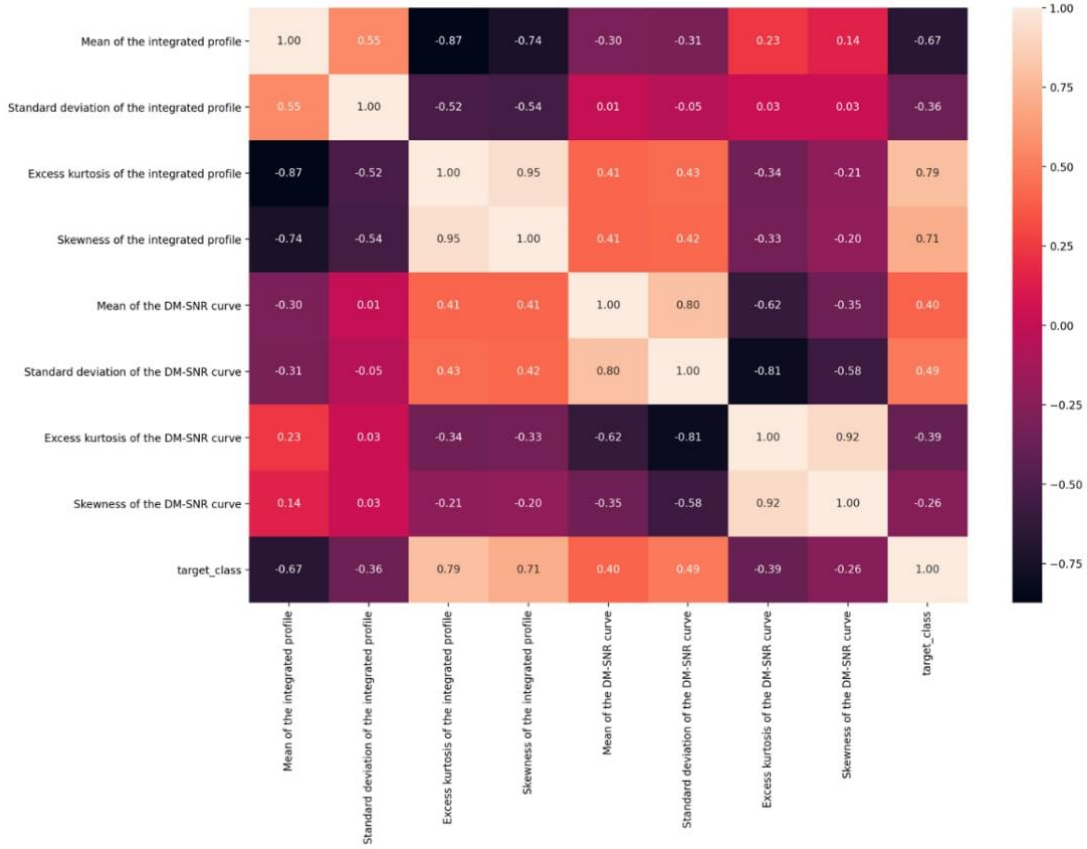


Все значения датасета числовые (float64). Пропусков в данных нет.

Проверим, что целевой признак бинарный.

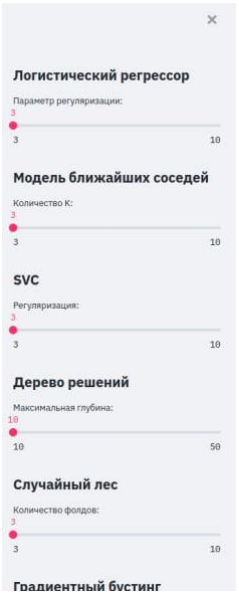


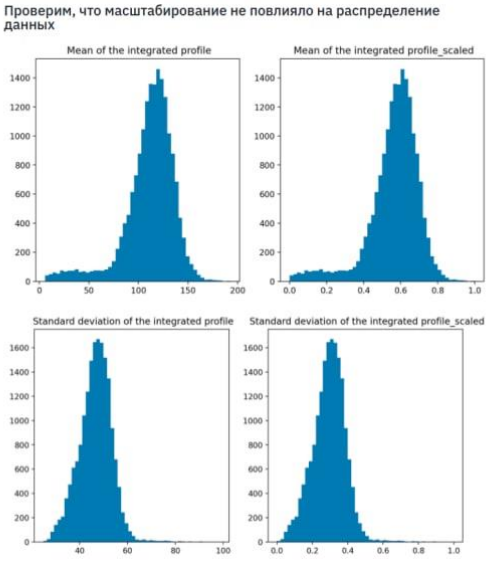
Построим корреляционную матрицу

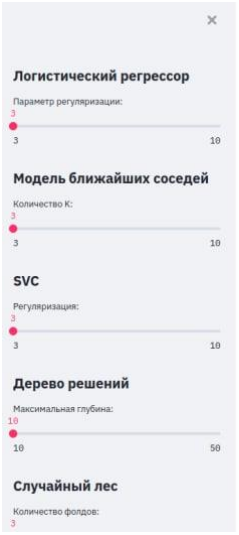


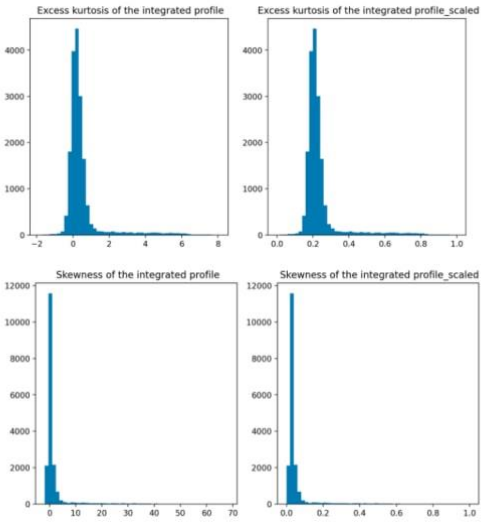
Наиболее сильно с целевым признаком коррелируют Excess kurtosis of the integrated profile (0.79) и Skewness of the integrated profile (0.71). Остальные признаки коррелируют на достаточном уровне, поэтому оставим все признаки в модели.

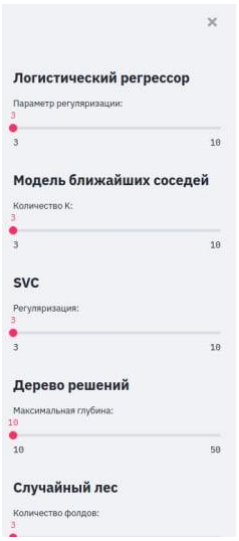
Проведем масштабирование данных

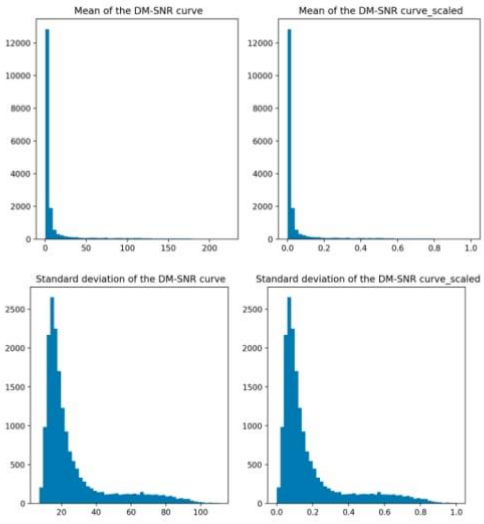


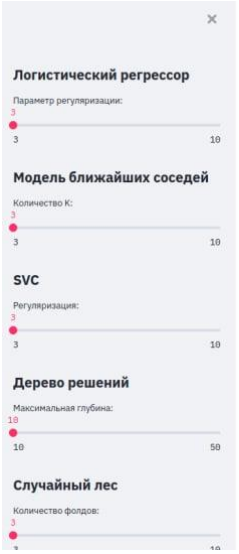


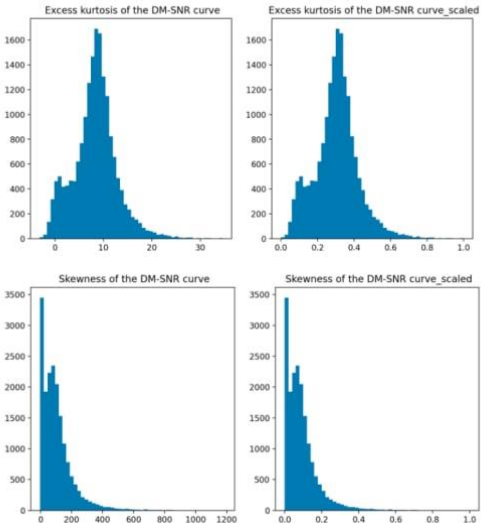










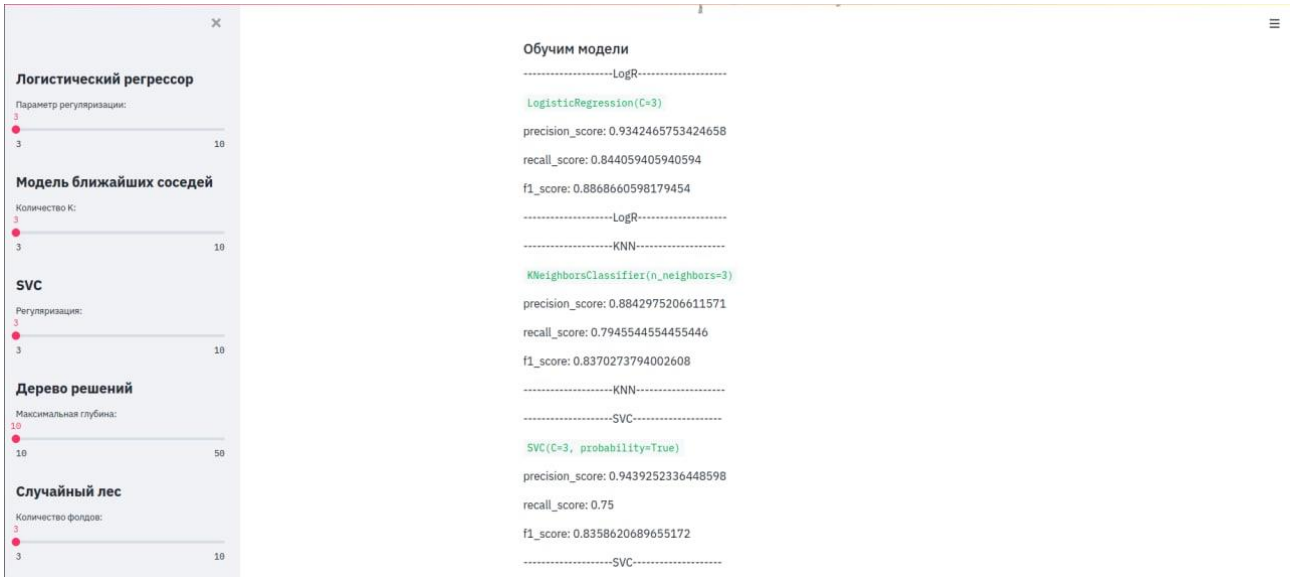


Как видно масштабирование не повлияло на распределение данных.

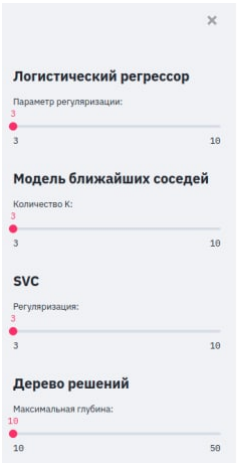
Проверим, что масштабирование не повлияло на корреляцию признаков

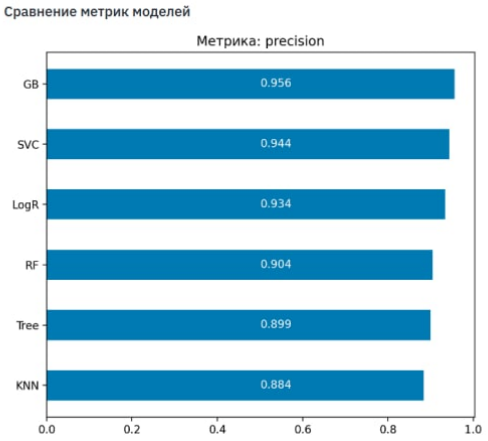
Как видно масштабирование не повлияло на корреляцию признаков.

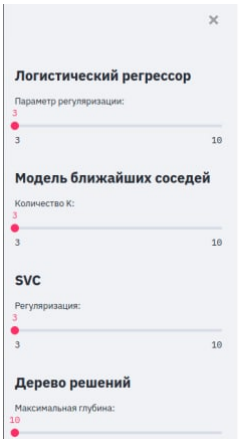
Разделим выборку на обучающую и тестовую и обучим модели с параметрами, заданными слайдерами в левой части интерфейса.

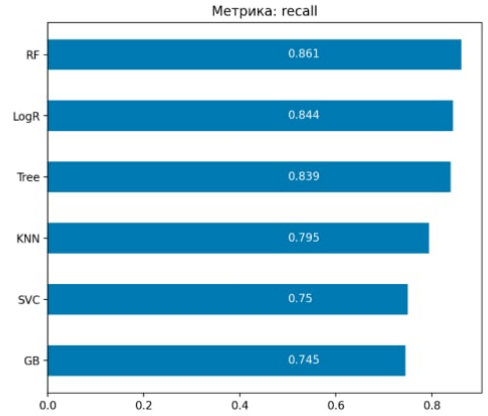


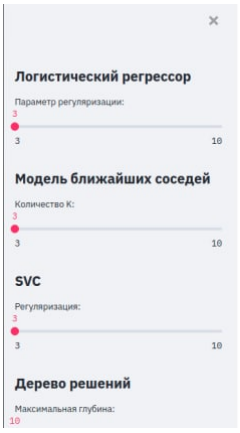
Выведем значение метрик в виде графиков

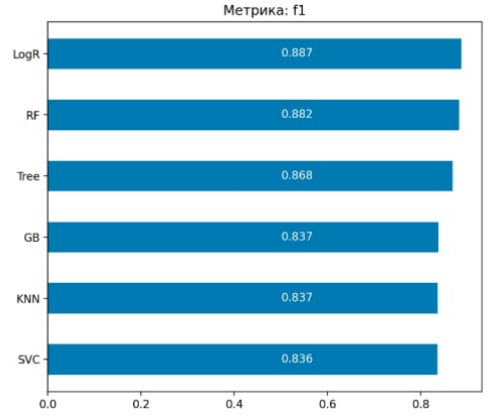






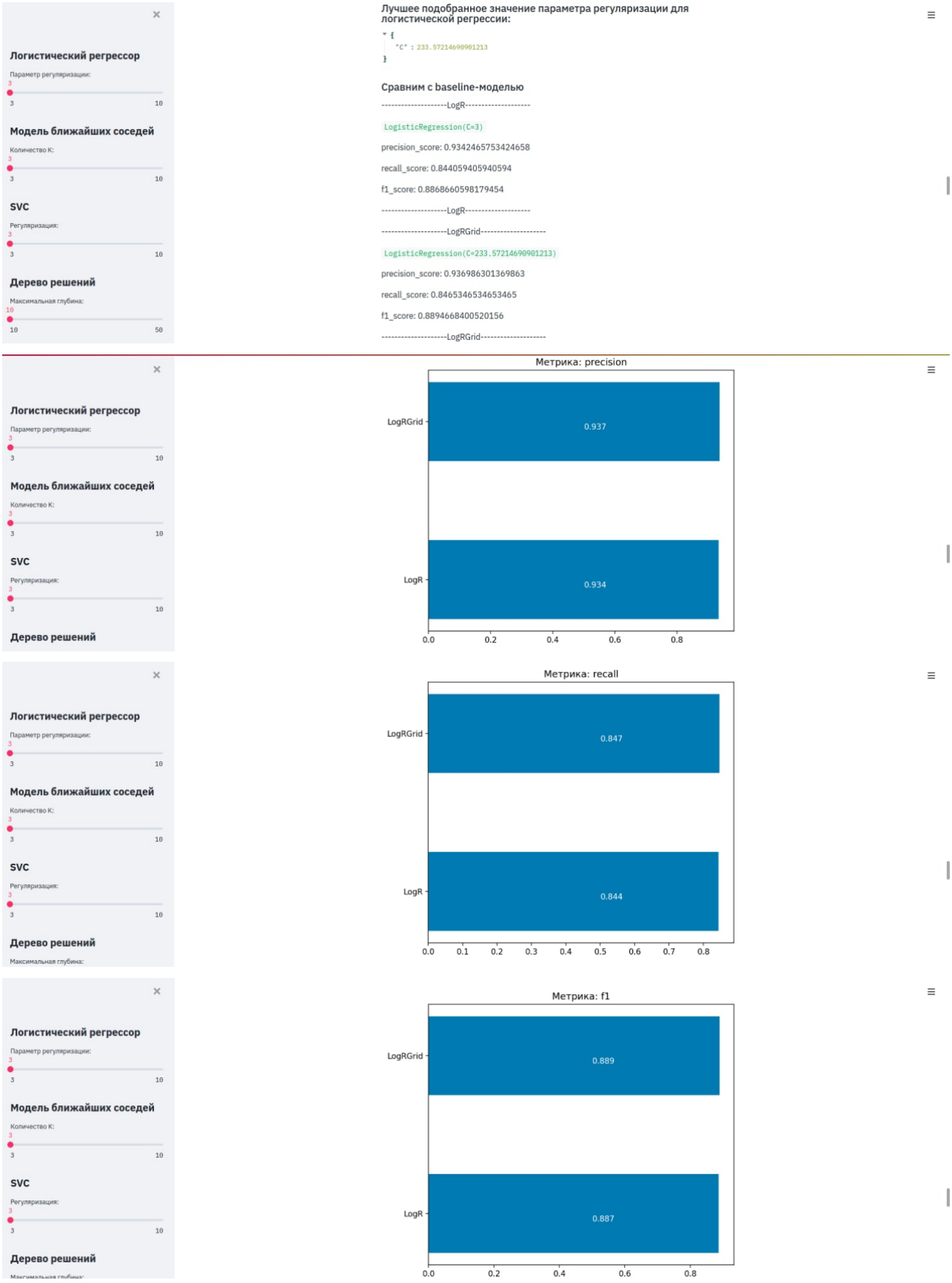






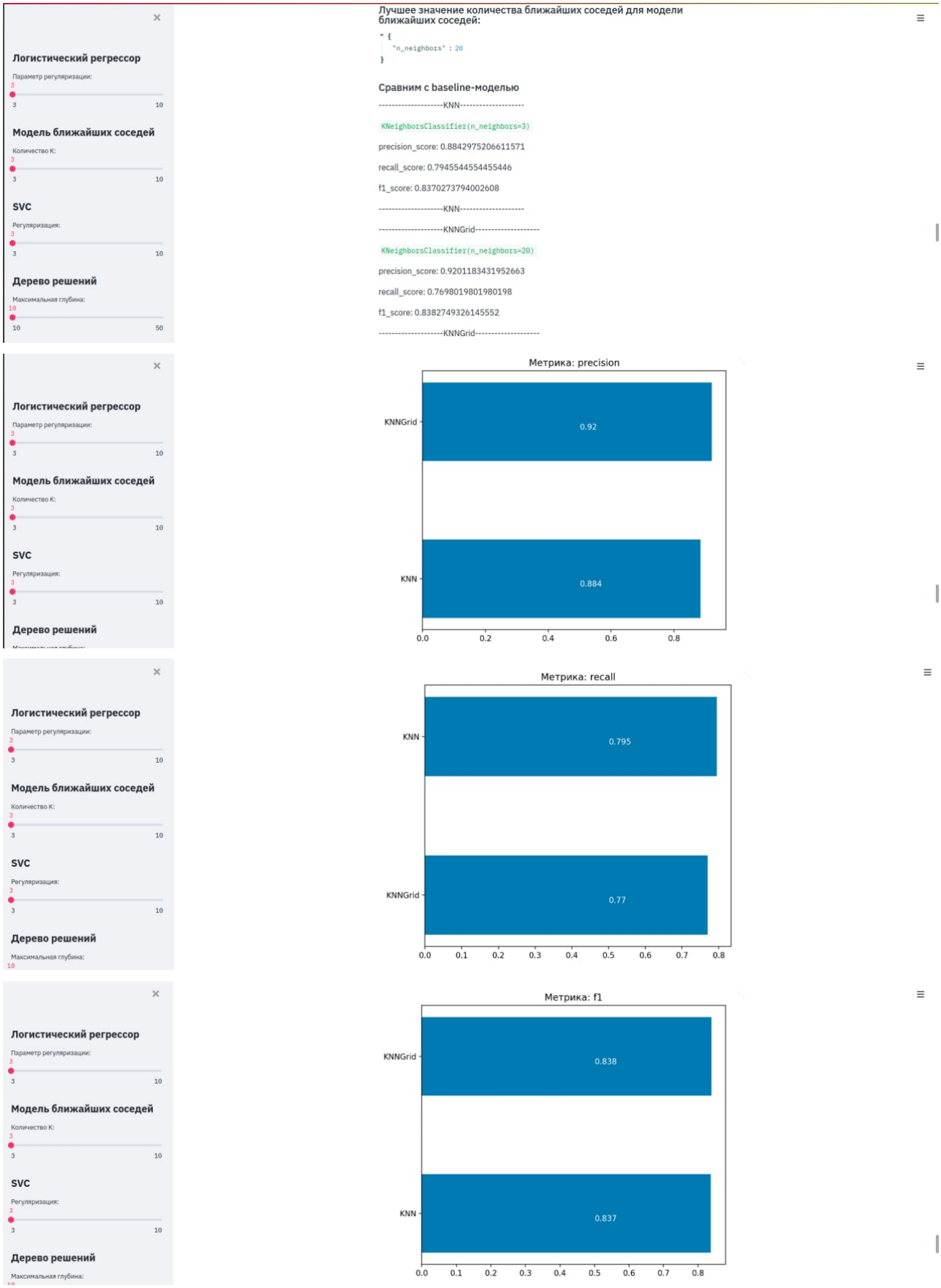
Как видно из графиков, все модели показали себя хорошо. Уровень моделей примерно одинаковый. Однако, т.к. F1-мера лучше всех у модели логистической регрессии, можно сделать вывод, что логистическая регрессия показала себя лучше всех.

Подберем гиперпараметры для логистической регрессии



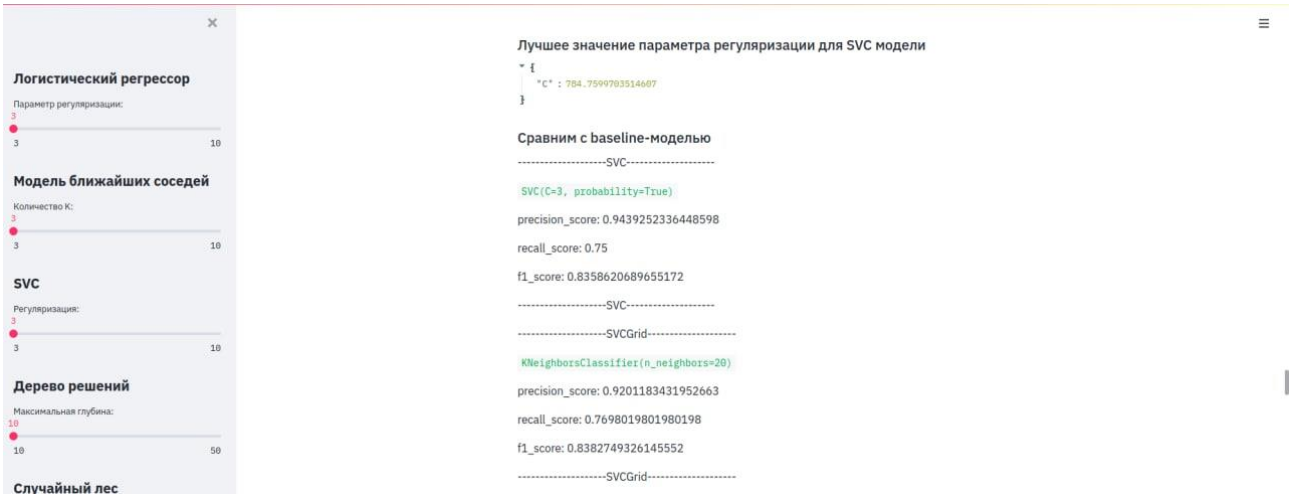
Как видно с подобранным параметром регуляризации логистическая регрессия показала себя лучше.

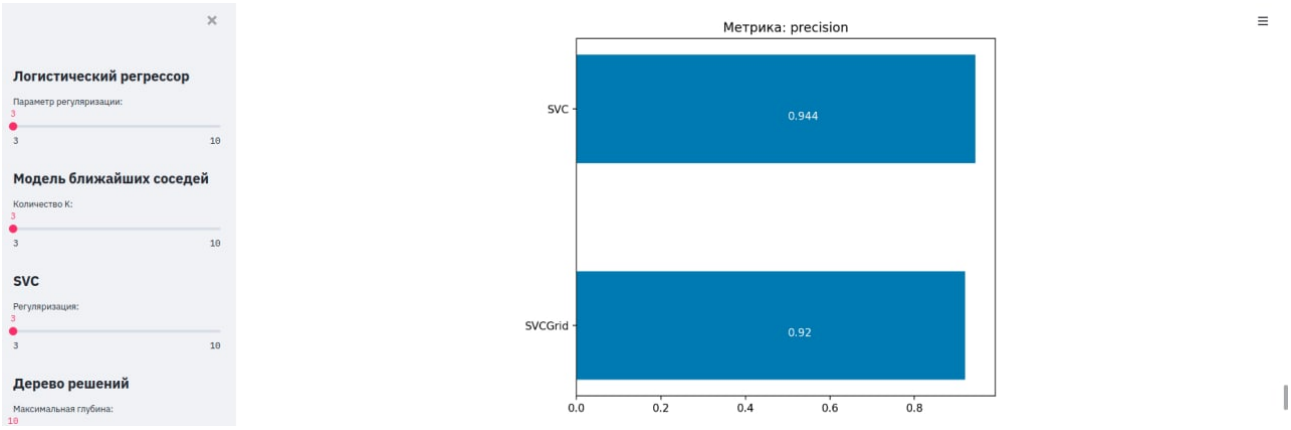
Подберем гиперпараметры для модели ближайших соседей

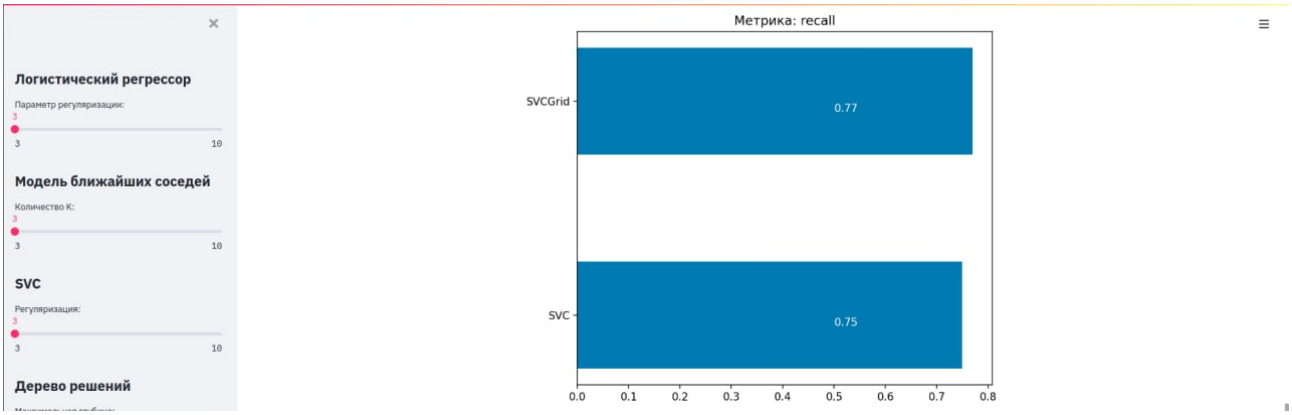


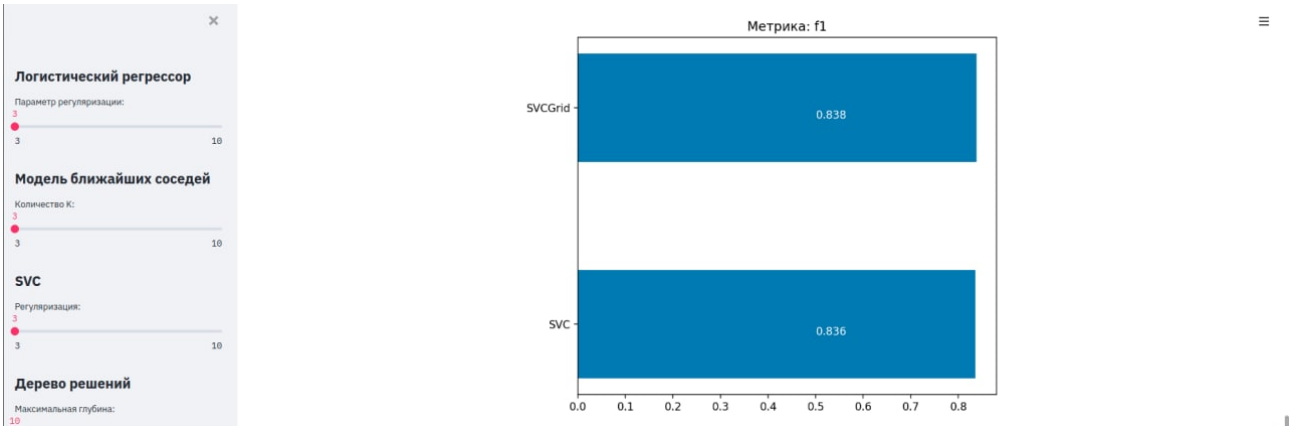
Как видно изначальная модель ближайших соседей, при количестве соседей, равном 3, показала себя лучше, чем с подобранным количеством, равным 20.

Подберем гиперпараметры для SVC модели



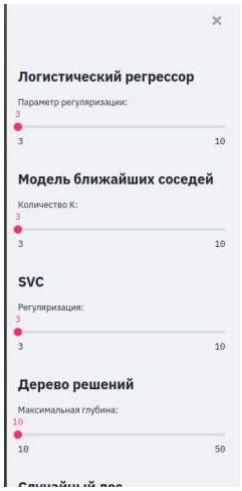






Как видно с подобранным параметром регуляризации SVM модель показала себя лучше, чем изначальная.

Подберем гиперпараметры для дерева решений



Лучшее значение максимальной глубины для дерева решений:

```
{
  "max_depth" : 5
}
```

Сравним с baseline-моделью

----------Tree----------

`DecisionTreeClassifier(max_depth=10, random_state=10)`

precision_score: 0.8992042440318302

recall_score: 0.8391089108910891

f1_score: 0.8681177976952625

----------Tree----------

----------TreeGrid----------

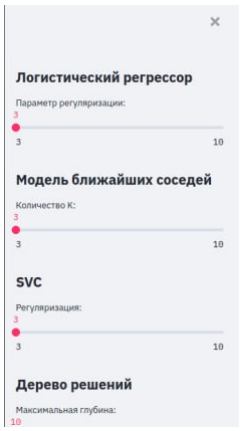
`KNeighborsClassifier(n_neighbors=20)`

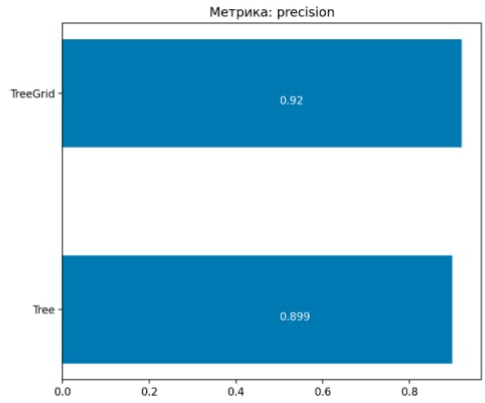
precision_score: 0.9201183431952663

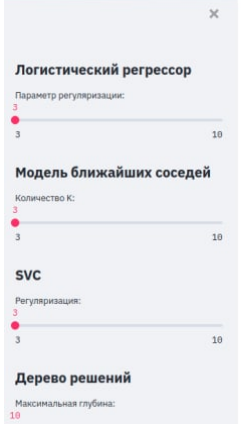
recall_score: 0.7698019801980198

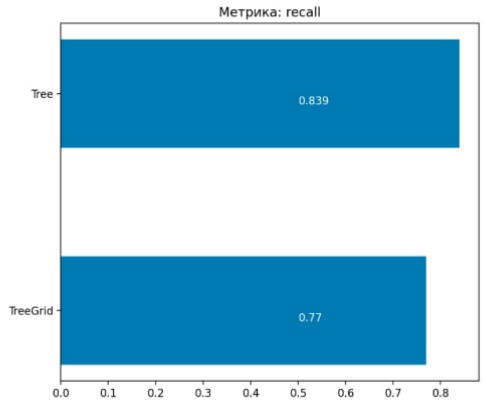
f1_score: 0.8382749326145552

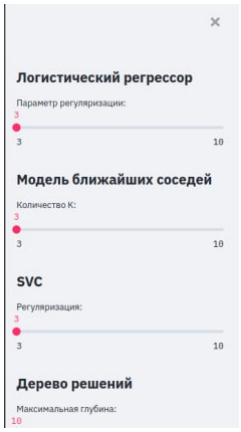
----------TreeGrid----------

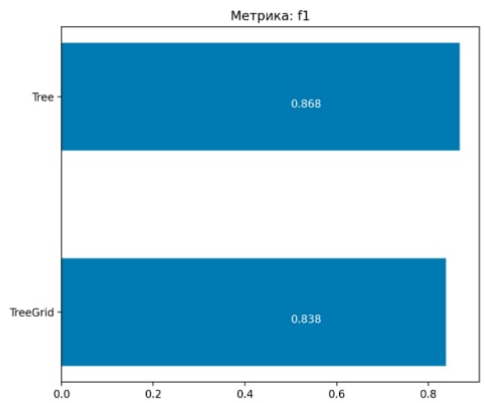






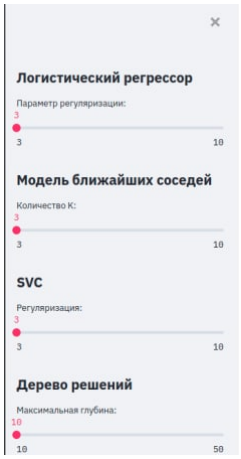


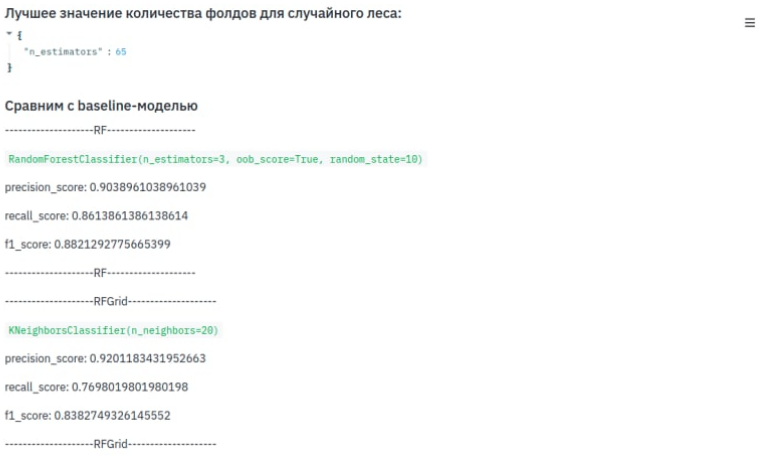


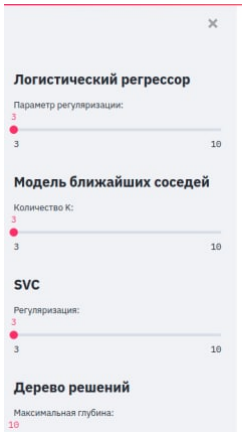


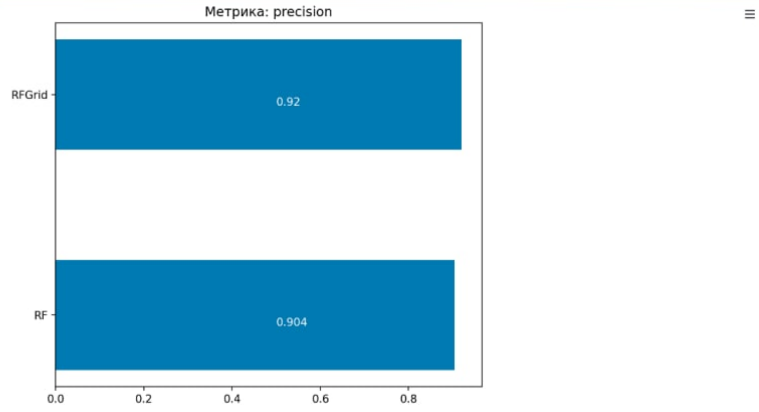
Как видно изначальная модель с глубиной дерева, равной 10, показала себя лучше, чем с подобранной глубиной 5.

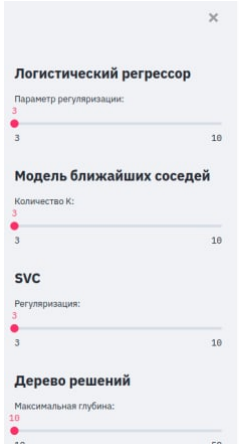
Подберем гиперпараметры для случайного леса

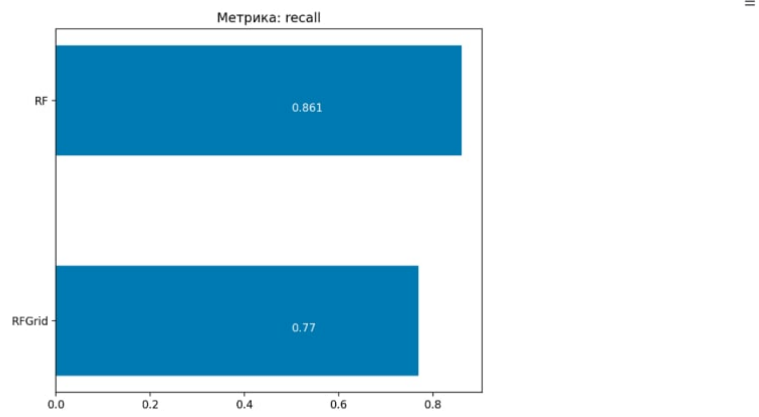


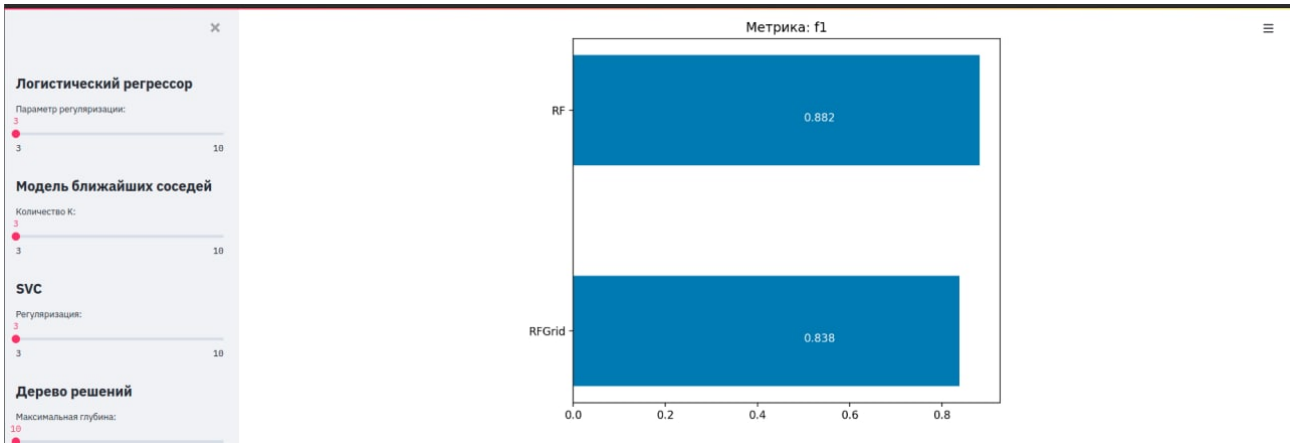






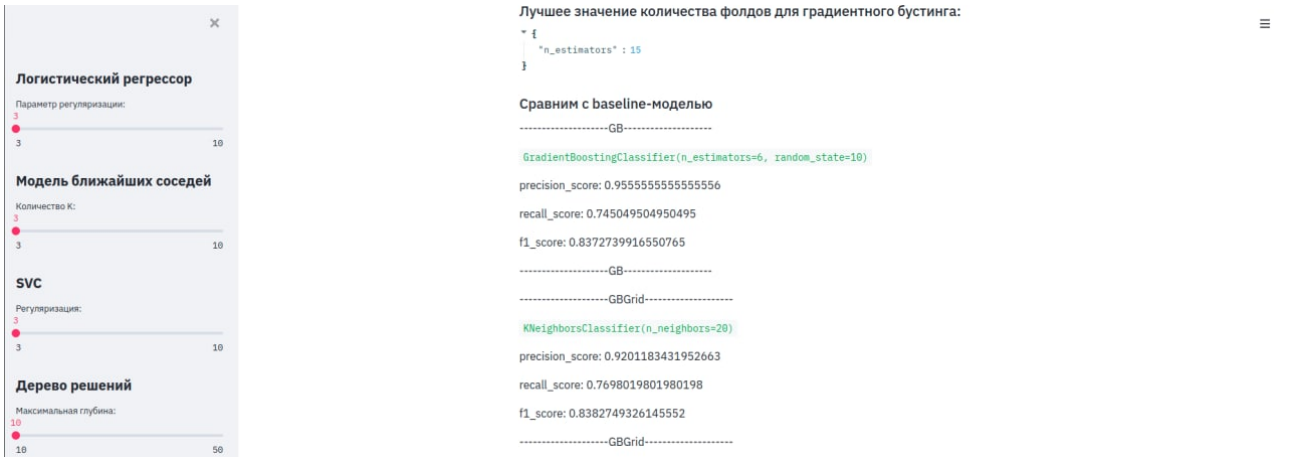


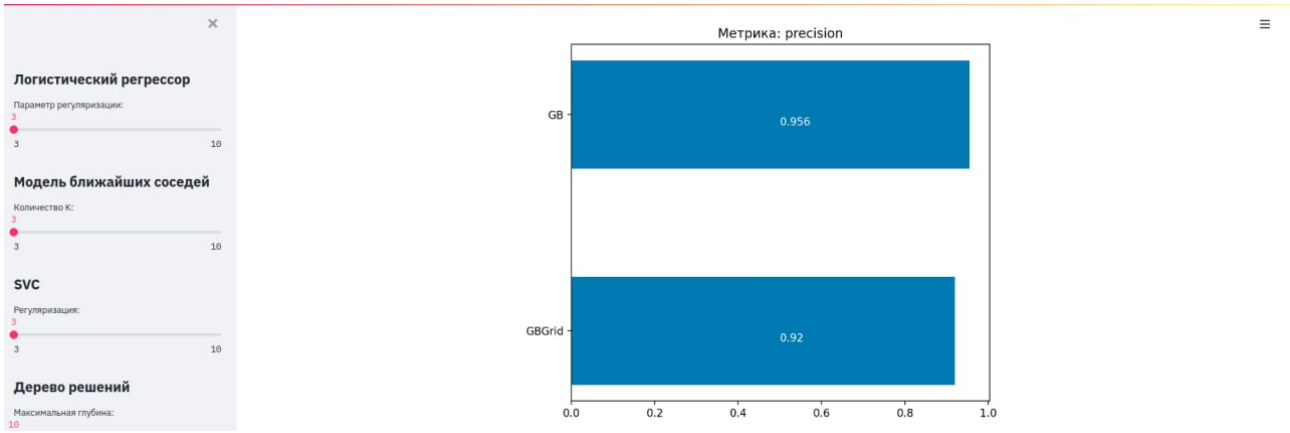


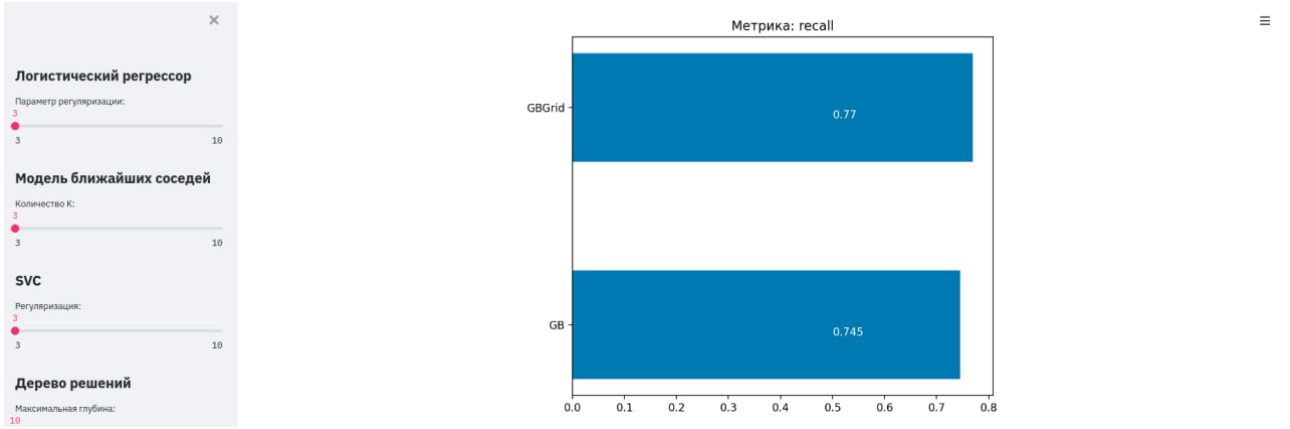


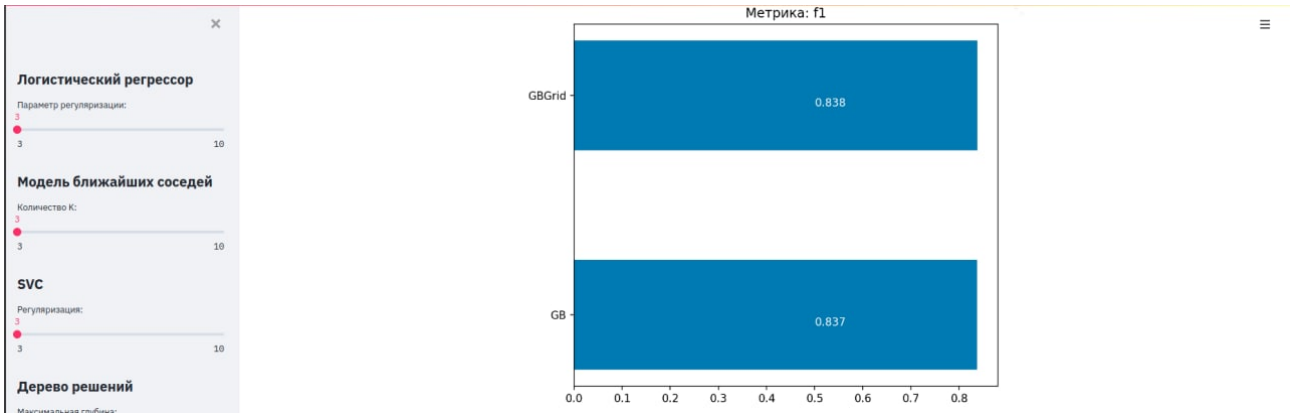
Как видно изначальная модель случайного леса показала себя лучше.

Подберем гиперпараметры для градиентного бустинга



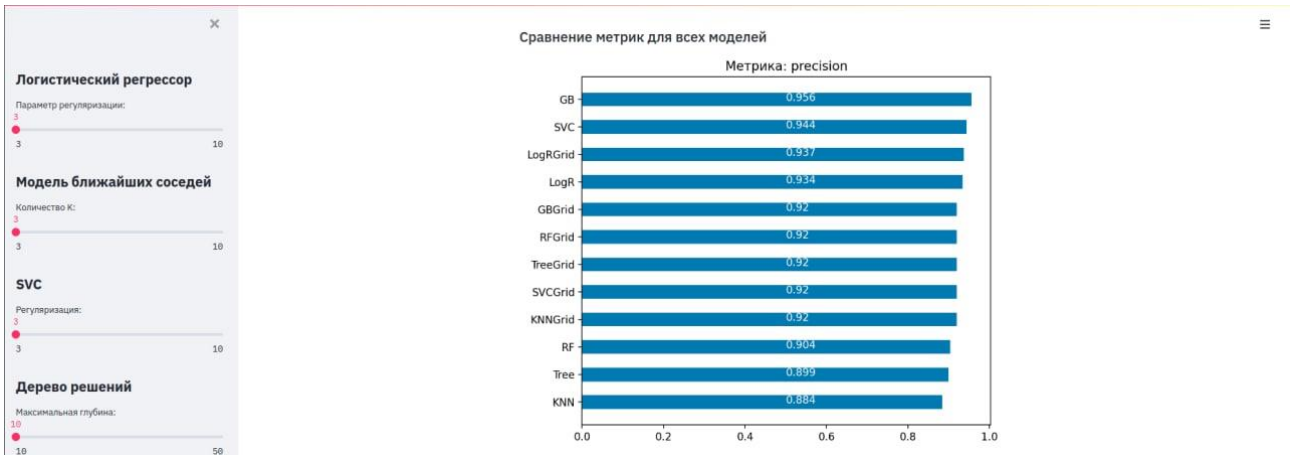


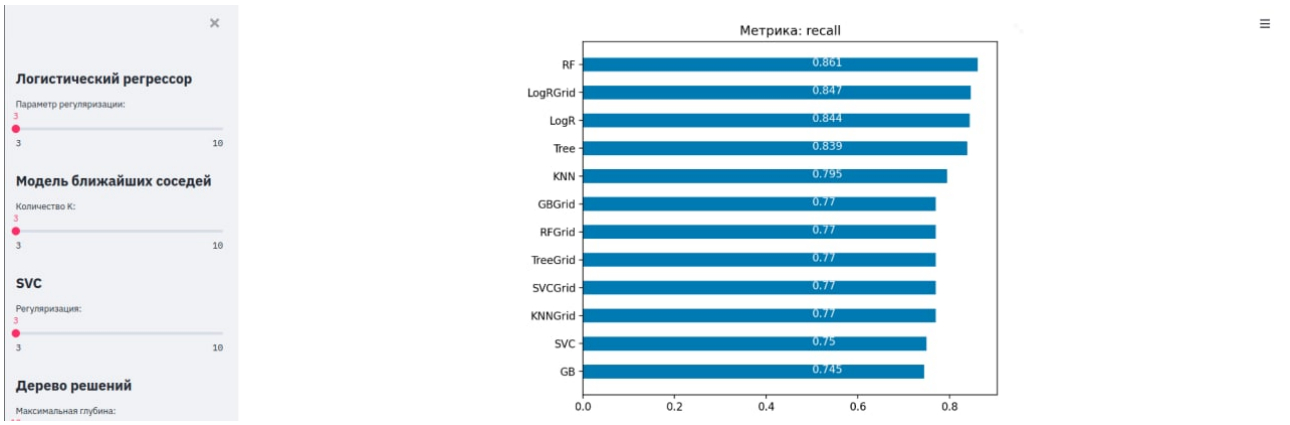


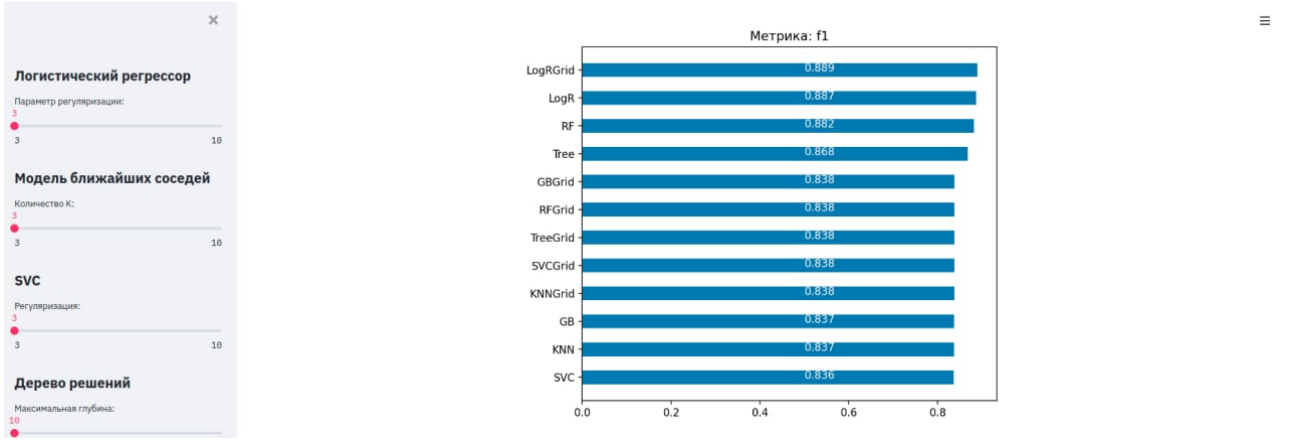


Как видно модель с подобранным количеством фолдов, равным 15, показала себя лучше изначальной.

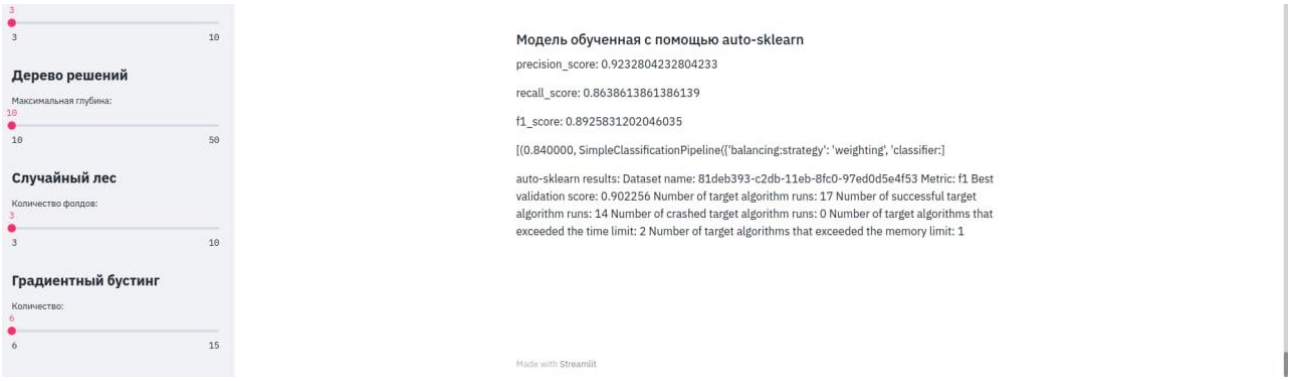
Итоговые результаты для всех моделей







Обучим модель с помощью auto-sklearn



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

В результате работы по метрике precision лучше всего себя показала модель градиентного бустинга, по метрике recall лучше всего себя показала модель случайного леса, а по F1-метрике лучше всего себя показала модель логистической регрессии. Исходя из этого, а также того, что F1-мера является средним гармоническим precision и recall, можно сделать вывод, что лучшей моделью оказалась логистическая регрессия. Это можно связать с тем, что распределение признаков нормальное. Также целевой признак является бинарным, а логистическая регрессия хорошо показывает себя в задачах бинарной классификации.

Также была построена модель с помощью библиотеки auto-sklearn. Качество этой модели по F1-мере (0.893) оказалось немного выше качества логистической регрессии (0.889).

Источники

1. https://www.kaggle.com/shivam1901/pulsar-star
2. https://github.com/ugapanyuk/ml_course_2021/wiki/COURSE_TMO
3. https://www.sklearn.org
4. https://automl.github.io/auto-sklearn/master/
5. https://docs.streamlit.io/en/stable/