



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  
(Группа)

\_\_\_\_\_  
(Подпись, дата) Старых Ф.А.  
(И.О.Фамилия)

Руководитель курсового проекта

\_\_\_\_\_  
(Подпись, дата) Гапанюк Ю.Е.  
(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

\_\_\_\_\_ Старых Фёдор Артемович  
(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**

\_\_\_\_\_ Старых Ф.А.  
(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.....	24
Источники .....	24

## 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 – перекося 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

```

```

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 масштабирования')
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)))

```

## Скрины веб-приложения:

### Просматриваем характеристики датасета

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество К:

3

3

10

SVC

Регуляризация:

3

3

10

Дерево решений

Максимальная глубина:

10

10

50

Случайный лес

Количество фолдов:

3

3

10

Градиентный бустинг

Количество:

6

6

15

Первые 5 значений

	Mean of the integrated...	Standard deviation of ...	Excess kurtosis of the...	Skewne...
0	149.5625	55.6838	-0.2346	
1	102.5878	58.8824	0.4653	
2	103.0156	39.3416	0.3233	
3	136.7500	57.1784	-0.0684	
4	88.7266	48.6722	0.6889	

Размер датасета

(17898, 9)

Количество нулевых элементов

Mean of the integrated profile	0
Standard deviation of the integrated profile	0
Excess kurtosis of the integrated profile	0
Skewness of the integrated profile	0
Mean of the DM-SNR curve	0
Standard deviation of the DM-SNR curve	0
Excess kurtosis of the DM-SNR curve	0
Skewness of the DM-SNR curve	0
target_class	0

Колонки и их типы данных

Mean of the integrated profile	0
Standard deviation of the integrated profile	float64
Excess kurtosis of the integrated profile	float64
Skewness of the integrated profile	float64
Mean of the DM-SNR curve	float64
Standard deviation of the DM-SNR curve	float64
Excess kurtosis of the DM-SNR curve	float64
Skewness of the DM-SNR curve	float64
target_class	float64

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество К:

3

3

10

SVC

Регуляризация:

3

3

10

Дерево решений

Максимальная глубина:

10

10

50

Колонки и их типы данных

Mean of the integrated profile	0
Standard deviation of the integrated profile	float64
Excess kurtosis of the integrated profile	float64
Skewness of the integrated profile	float64
Mean of the DM-SNR curve	float64
Standard deviation of the DM-SNR curve	float64
Excess kurtosis of the DM-SNR curve	float64
Skewness of the DM-SNR curve	float64
target_class	float64

Статистические данные

	Mean of the integrated...	Standard deviation of ...	Excess kurtosis of the...	Sk...
count	17898	17898	17898	
mean	111.0800	46.5495	0.4779	
std	25.6529	6.8432	1.8640	
min	5.8125	24.7720	-1.8760	
25%	100.9297	42.3760	0.8271	
50%	115.0781	46.9475	0.2232	
75%	127.0859	51.0232	0.4733	
max	192.6172	98.7789	8.8695	

Целевой признак содержит только 0 и 1

Случайный лес

Количество фолдов:

3

3

10

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

✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

10

Дерево решений

Максимальная глубина:

10

10

50

Случайный лес

Количество фолдов:

3

3

10

Градиентный бустинг

✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

10

Дерево решений

Максимальная глубина:

10

10

50

Случайный лес

Количество фолдов:

3

3

10

✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

10

Дерево решений

Максимальная глубина:

10

10

50

Случайный лес

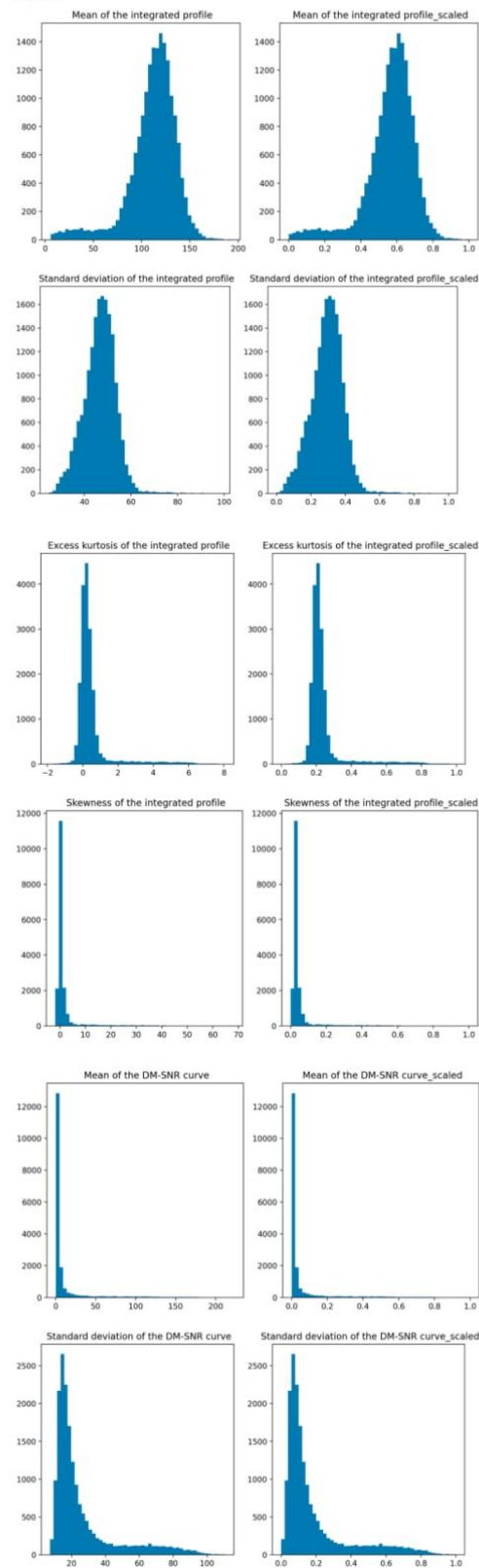
Количество фолдов:

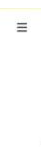
3

3

10

Проверим, что масштабирование не повлияло на распределение данных





```

.....Tree.....
DecisionTreeClassifier(max_depth=10, random_state=10)

precision_score: 0.8992042440318302

recall_score: 0.8391089108910891

f1_score: 0.868117796952625

.....Tree.....
.....RF.....
RandomForestClassifier(n_estimators=3, oob_score=True, random_state=10)

precision_score: 0.9038961038961039

recall_score: 0.8613861386138614

f1_score: 0.8821292775665399

.....RF.....
.....GB.....
GradientBoostingClassifier(n_estimators=6, random_state=10)

precision_score: 0.9555555555555556

recall_score: 0.745049504950495

f1_score: 0.8372739916550765

.....GB.....

```

## Выведем значение метрик в виде графиков

Сравнение метрик моделей

Метрика: precision

Модель	precision
GB	0.956
SVC	0.944
LogR	0.934
RF	0.904
Tree	0.899
KNN	0.884

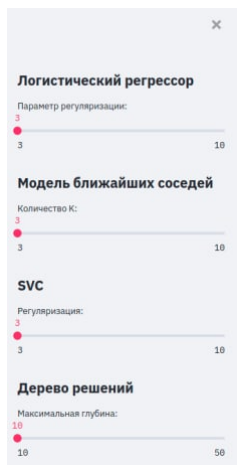
Модель	Метрика: recall
RF	0.861
LogR	0.844
Tree	0.839
KNN	0.795
SVC	0.75
GB	0.745

Модель	Метрика: f1
LogR	0.887
RF	0.882
Tree	0.868
GB	0.837
KNN	0.837
SVC	0.836



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



Лучшее подобранное значение параметра регуляризации для логистической регрессии:

```
{
  "C": 233.57214699901213
}
```

Сравним с baseline-моделью

-----LogR-----

```
LogisticRegression(C=3)
```

precision\_score: 0.9342465753424658

recall\_score: 0.844059405940594

f1\_score: 0.8868660598179454

-----LogR-----

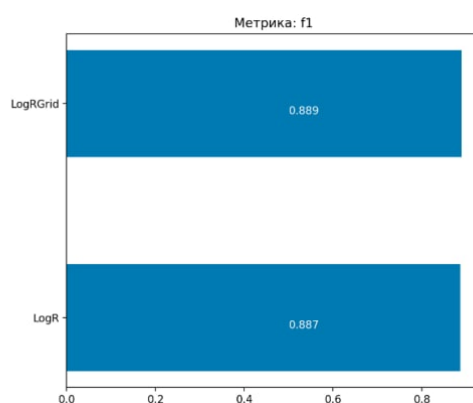
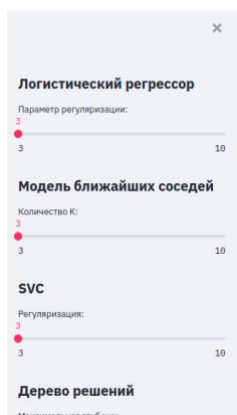
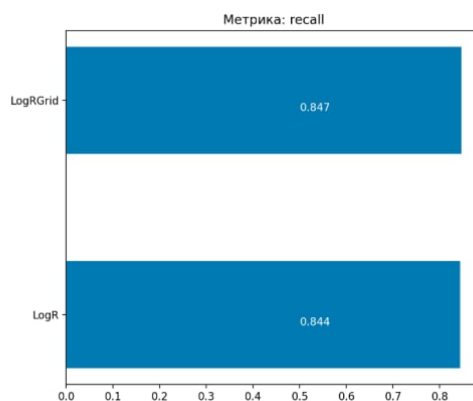
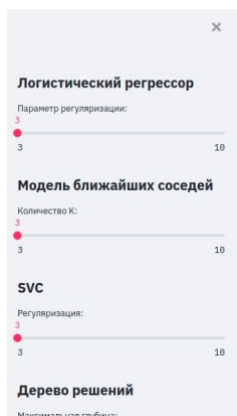
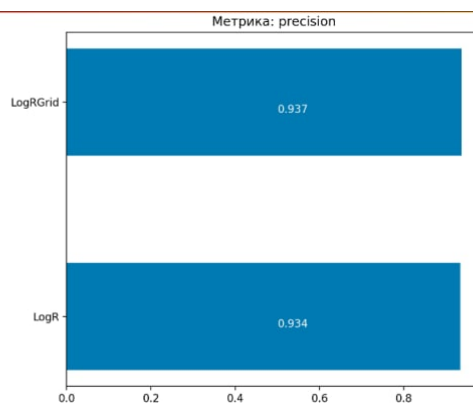
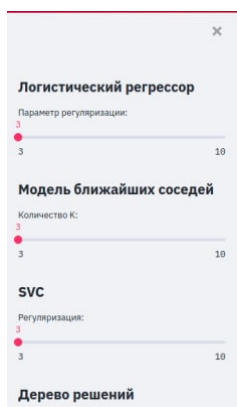
```
LogisticRegression(C=233.57214699901213)
```

precision\_score: 0.936986301369863

recall\_score: 0.8465346534653465

f1\_score: 0.8894668400520156

-----LogRGrid-----



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

Подберем гиперпараметры для модели ближайших соседей

Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

3

3 10

Дерево решений

Максимальная глубина:

10

10 50

Лучшее значение количества ближайших соседей для модели ближайших соседей:

```
{
  "n_neighbors": 20
}
```

Сравним с baseline-моделью

-----KNN-----

`KNeighborsClassifier(n_neighbors=3)`

precision\_score: 0.8842975206611571

recall\_score: 0.7945544554455446

f1\_score: 0.8370273794002608

-----KNN-----

-----KNNGrid-----

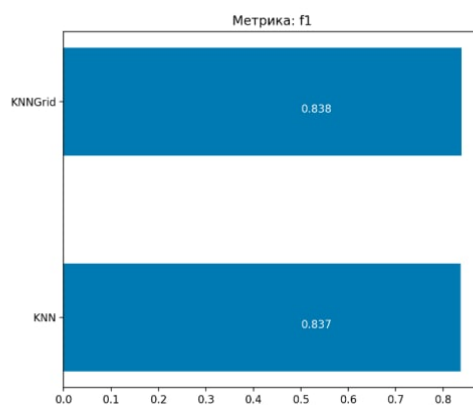
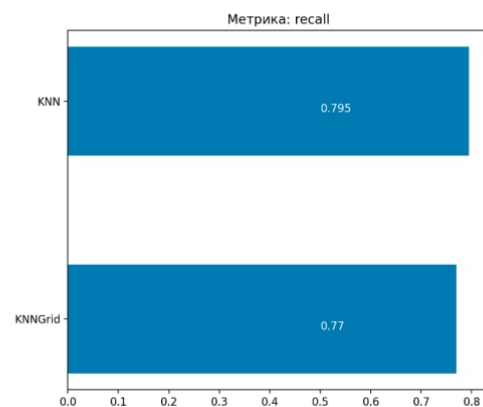
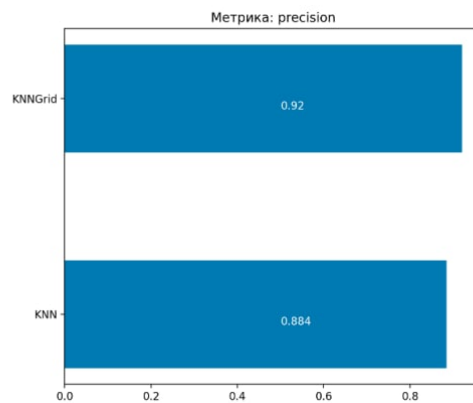
`KNeighborsClassifier(n_neighbors=20)`

precision\_score: 0.9201183431952663

recall\_score: 0.7698019801980198

f1\_score: 0.8382749326145552

-----KNNGrid-----



Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

3

3 10

Дерево решений

Максимальная глубина:

10

Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

3

3 10

Дерево решений

Максимальная глубина:

10

Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

3

3 10

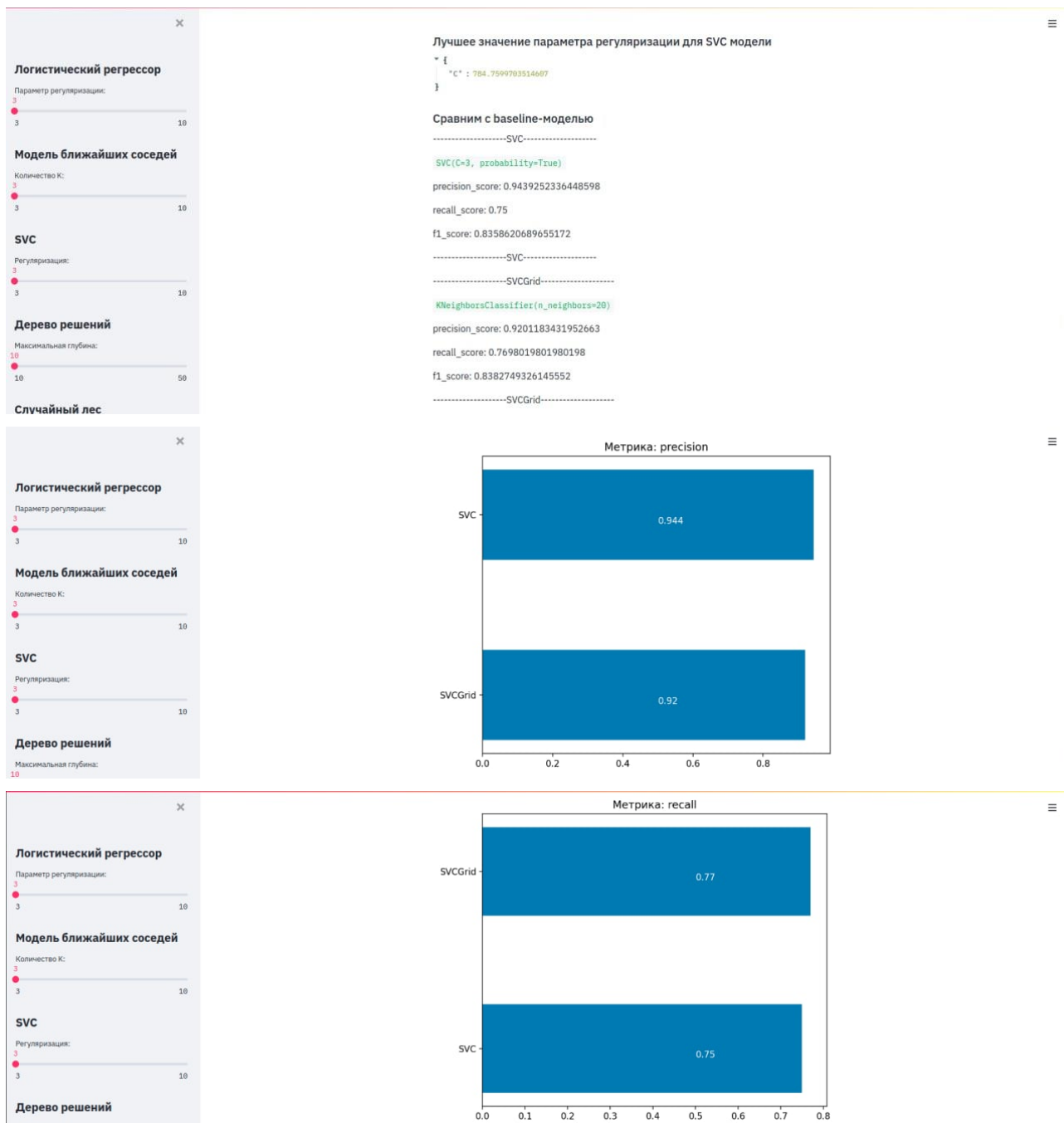
Дерево решений

Максимальная глубина:

10

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

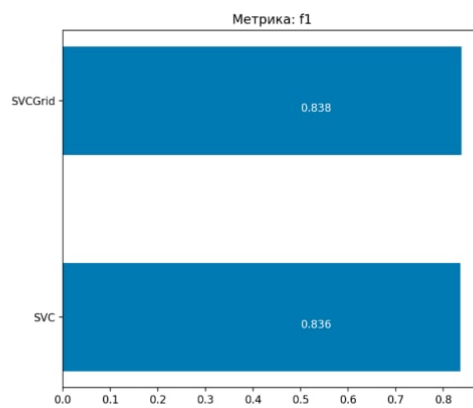


**Логистический регрессор**  
 Параметр регуляризации: 3

**Модель ближайших соседей**  
 Количество K: 3

**SVC**  
 Регуляризация: 3

**Дерево решений**  
 Максимальная глубина: 10



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

Подберем гиперпараметры для дерева решений

**Логистический регрессор**  
 Параметр регуляризации: 3

**Модель ближайших соседей**  
 Количество K: 3

**SVC**  
 Регуляризация: 3

**Дерево решений**  
 Максимальная глубина: 10

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

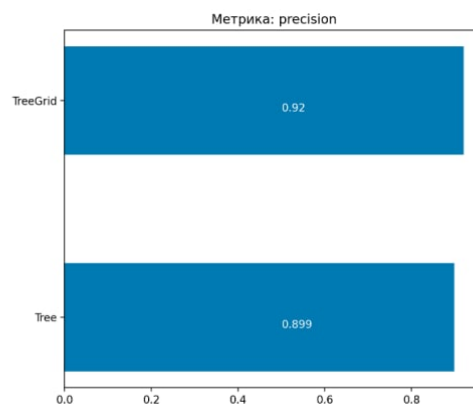
-----TreeGrid-----

**Логистический регрессор**  
 Параметр регуляризации: 3

**Модель ближайших соседей**  
 Количество K: 3

**SVC**  
 Регуляризация: 3

**Дерево решений**  
 Максимальная глубина: 10



Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

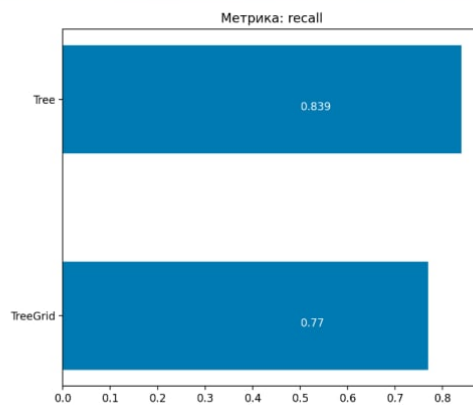
3

3 10

Дерево решений

Максимальная глубина:

10



Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

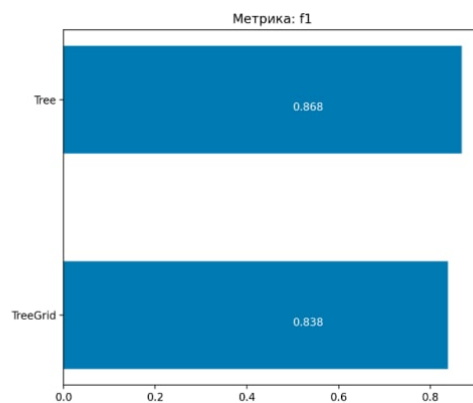
3

3 10

Дерево решений

Максимальная глубина:

10



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

Подберем гиперпараметры для случайного леса

Логистический регрессор

Параметр регуляризации:

3

3 10

Модель ближайших соседей

Количество K:

3

3 10

SVC

Регуляризация:

3

3 10

Дерево решений

Максимальная глубина:

10

10 50

Лучшее значение количества фолдов для случайного леса:

```
{
  "n_estimators": 65
}
```

Сравним с baseline-моделью

-----RF-----

```
RandomForestClassifier(n_estimators=3, oob_score=True, random_state=10)
```

precision\_score: 0.9038961038961039

recall\_score: 0.8613861386138614

f1\_score: 0.8821292775665399

-----RF-----

-----RFGrid-----

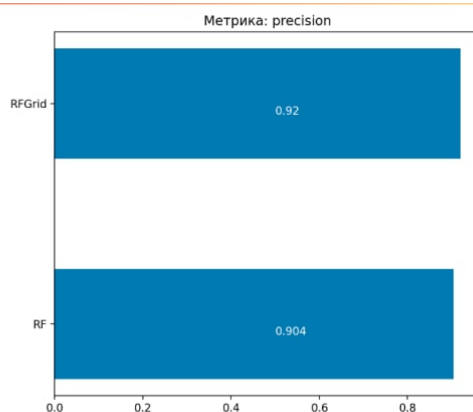
```
KNeighborsClassifier(n_neighbors=20)
```

precision\_score: 0.9201183431952663

recall\_score: 0.7698019801980198

f1\_score: 0.8382749326145552

-----RFGrid-----



Логистический регрессор

Параметр регуляризации: 3

Модель ближайших соседей

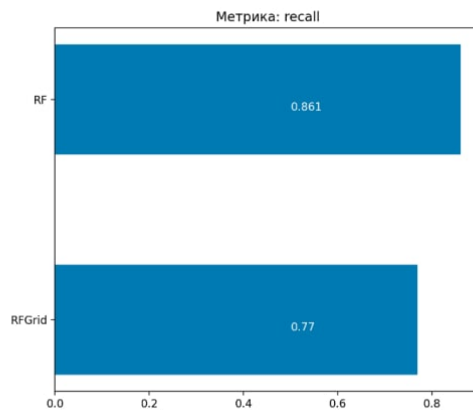
Количество K: 3

SVC

Регуляризация: 3

Дерево решений

Максимальная глубина: 10



Логистический регрессор

Параметр регуляризации: 3

Модель ближайших соседей

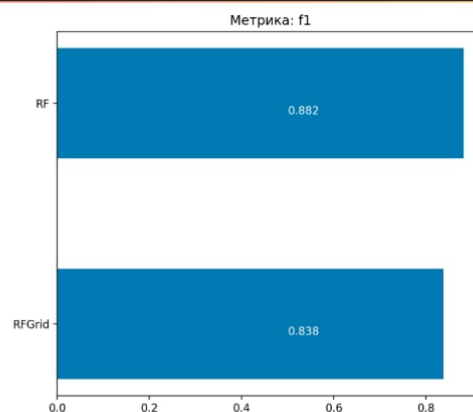
Количество K: 3

SVC

Регуляризация: 3

Дерево решений

Максимальная глубина: 10



Как видно изначальная модель случайного леса показала себя лучше.

Подберем гиперпараметры для градиентного бустинга

Логистический регрессор

Параметр регуляризации: 3

Модель ближайших соседей

Количество K: 3

SVC

Регуляризация: 3

Дерево решений

Максимальная глубина: 10

Лучшее значение количества фолдов для градиентного бустинга:

```
{
  "n_estimators": 10
}
```

Сравним с baseline-моделью

-----GB-----

```
GradientBoostingClassifier(n_estimators=6, random_state=10)
```

precision\_score: 0.9555555555555556

recall\_score: 0.745049504950495

f1\_score: 0.8372739916550765

-----GB-----

-----GBGrid-----

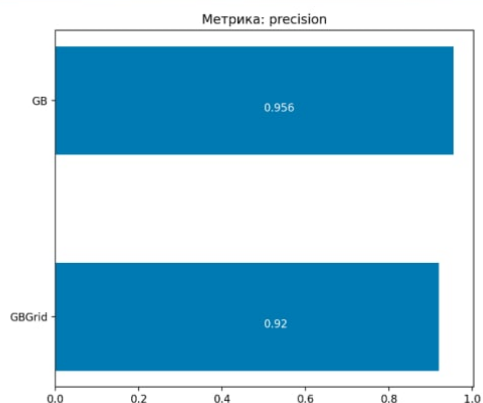
```
KNeighborsClassifier(n_neighbors=20)
```

precision\_score: 0.9201183431952663

recall\_score: 0.7698019801980198

f1\_score: 0.8382749326145552

-----GBGrid-----



✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

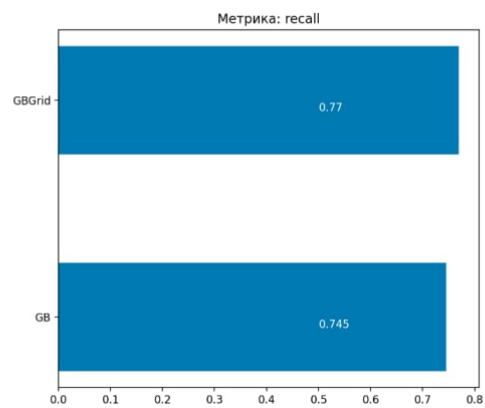
10

Дерево решений

Максимальная глубина:

10

10



✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

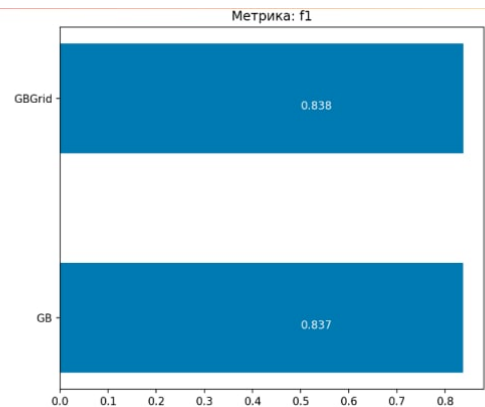
10

Дерево решений

Максимальная глубина:

10

10



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

Итоговые результаты для всех моделей

✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

10

Дерево решений

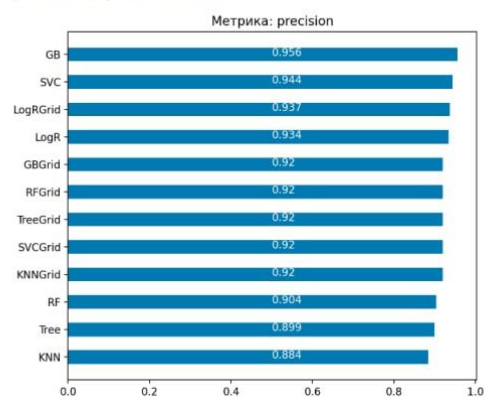
Максимальная глубина:

10

10

50

Сравнение метрик для всех моделей



✕

Логистический регрессор

Параметр регуляризации:

3

3

10

Модель ближайших соседей

Количество K:

3

3

10

SVC

Регуляризация:

3

3

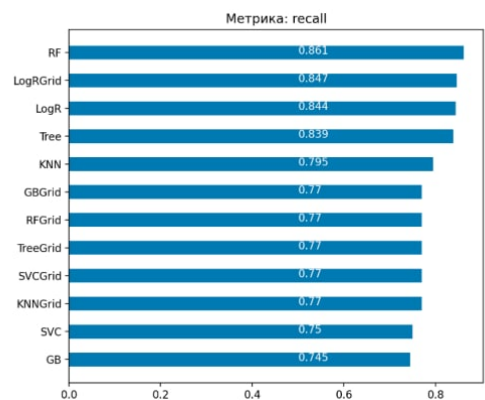
10

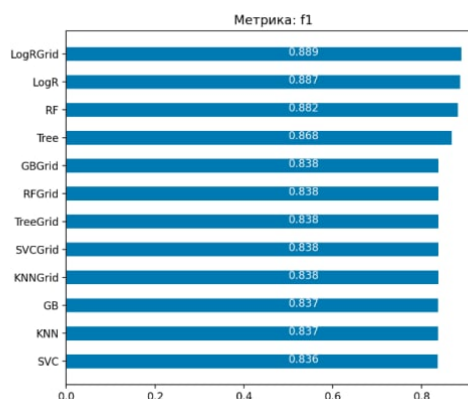
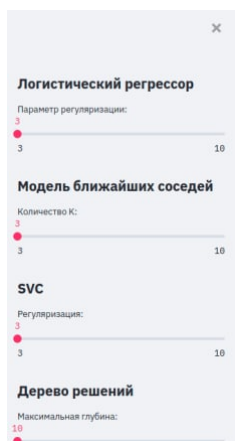
Дерево решений

Максимальная глубина:

10

10





## 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.

## Источники

1. <https://www.kaggle.com/shivam1901/pulsar-star>
2. [https://github.com/ugapanyuk/ml\\_course\\_2021/wiki/COURSE\\_TMO](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/>