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 задачи машинного обучения

Студент ___ИУ5-62Б___ _____ **М.И. Брусникина**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Консультант _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

2020 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 ___ИУ5___
(Индекс)
_____ В.М. Черненький
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Технологии машинного обучения_____

Студент группы ____ИУ5-62Б_____

_____Брусникина Мария Игоревна_____
(Фамилия, имя, отчество)

Тема курсового проекта ___Решение задачи машинного обучения_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к _5_ нед., 50% к _9_ нед., 75% к _13_ нед., 100% к _16_ нед.

Задание _Решение задачи машинного обучения на основе материалов дисциплины._____
_Результатом курсового проекта является отчет, содержащий описания моделей,_____
_тексты программ и результаты экспериментов._____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _35__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « _7_ » ___фев._____ 2020 г.

Руководитель курсового проекта _____ Ю.Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

Студент _____ М.И. Брусникина
(Подпись, дата) (И.О.Фамилия)

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

Содержание

ВВЕДЕНИЕ

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

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

В рамках курсового проекта возможно проведение типового или нетипового исследования.

- Типовое исследование - решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
- Нетиповое исследование - решение нестандартной задачи. Тема должна быть согласована с преподавателем. Как правило, такая работа выполняется группой студентов.

Курсовой проект по ТМО

Задание

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Приведенная схема исследования является рекомендуемой. В зависимости от решаемой задачи возможны модификации.

Ход выполнения курсового проекта

1) Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.

Будем использовать набор данных, который описывает выборку кандидатов на пульсары, собранную во время исследования Вселенной с высоким временным разрешением. Пульсары - это редкий тип нейтронных звезд, которые производят радиоизлучение, обнаруживаемое на Земле. Они представляют значительный научный интерес как исследования пространства-времени, межзвездной среды и состояний материи.

Каждый кандидат описывается 8 непрерывными переменными и одной переменной класса. Первые четыре являются простыми статистическими данными, полученными из интегрального импульсного профиля (сложенного профиля). Это массив непрерывных переменных, описывающих разрешенную по долготе версию сигнала, которая была усреднена как по времени, так и по частоте. Остальные четыре переменные аналогично получены из кривой DM-SNR.

Таким образом, набор данных содержит следующие колонки:

1. Среднее значение интегрального профиля
2. Стандартное отклонение интегрального профиля
3. Избыточный коэффициент эксцесса интегрального профиля
4. Асимметрия интегрального профиля
5. Среднее значение кривой DM-SNR
6. Стандартное отклонение кривой DM-SNR
7. Избыточный коэффициент эксцесса кривой DM-SNR
8. Асимметрия кривой DM-SNR
9. Класс

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score, confusion_matrix, plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier
%matplotlib inline
sns.set(style="ticks")
```

In [2]:

```
data = pd.read_csv('pulsar_stars.csv')
```

2) Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

In [3]:

```
data.head()
```

Out[3]:

	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	target_class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

In [4]:

```
data.shape
```

Out[4]:

```
(17898, 9)
```

In [5]:

```
data.columns
```

Out[5]:

```
Index([' 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',
       'target_class'],
      dtype='object')
```

In [6]:

```
data.dtypes
```

Out[6]:

```
 Mean of the integrated profile                  float64
 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                                       int64
dtype: object
```

In [7]:

```
data.isnull().sum()
```

Out[7]:

```
 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
dtype: int64
```

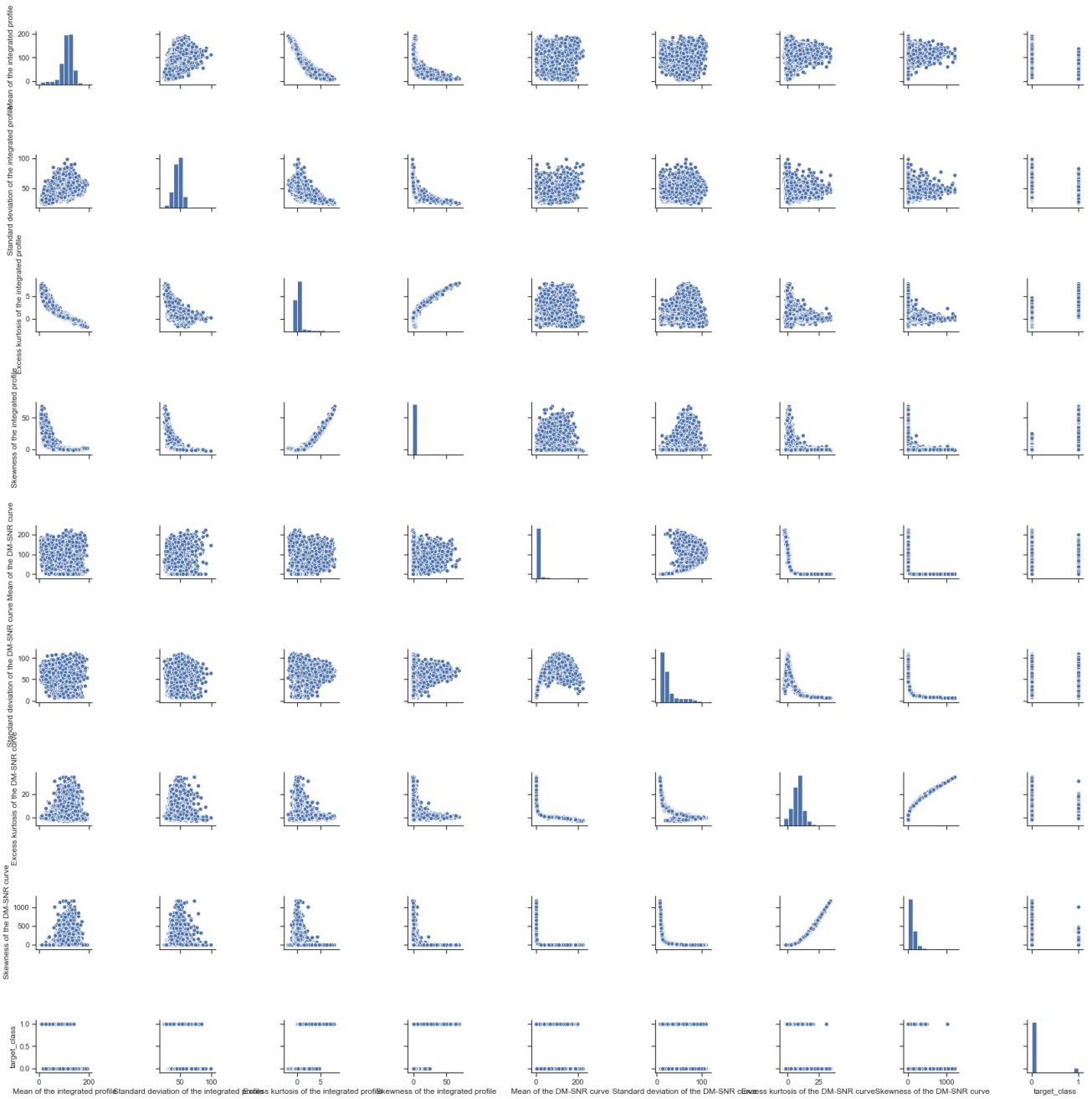
Таким образом, представленный набор данных не содержит пропусков.

In [8]:

```
sns.pairplot(data)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x5571230>
```

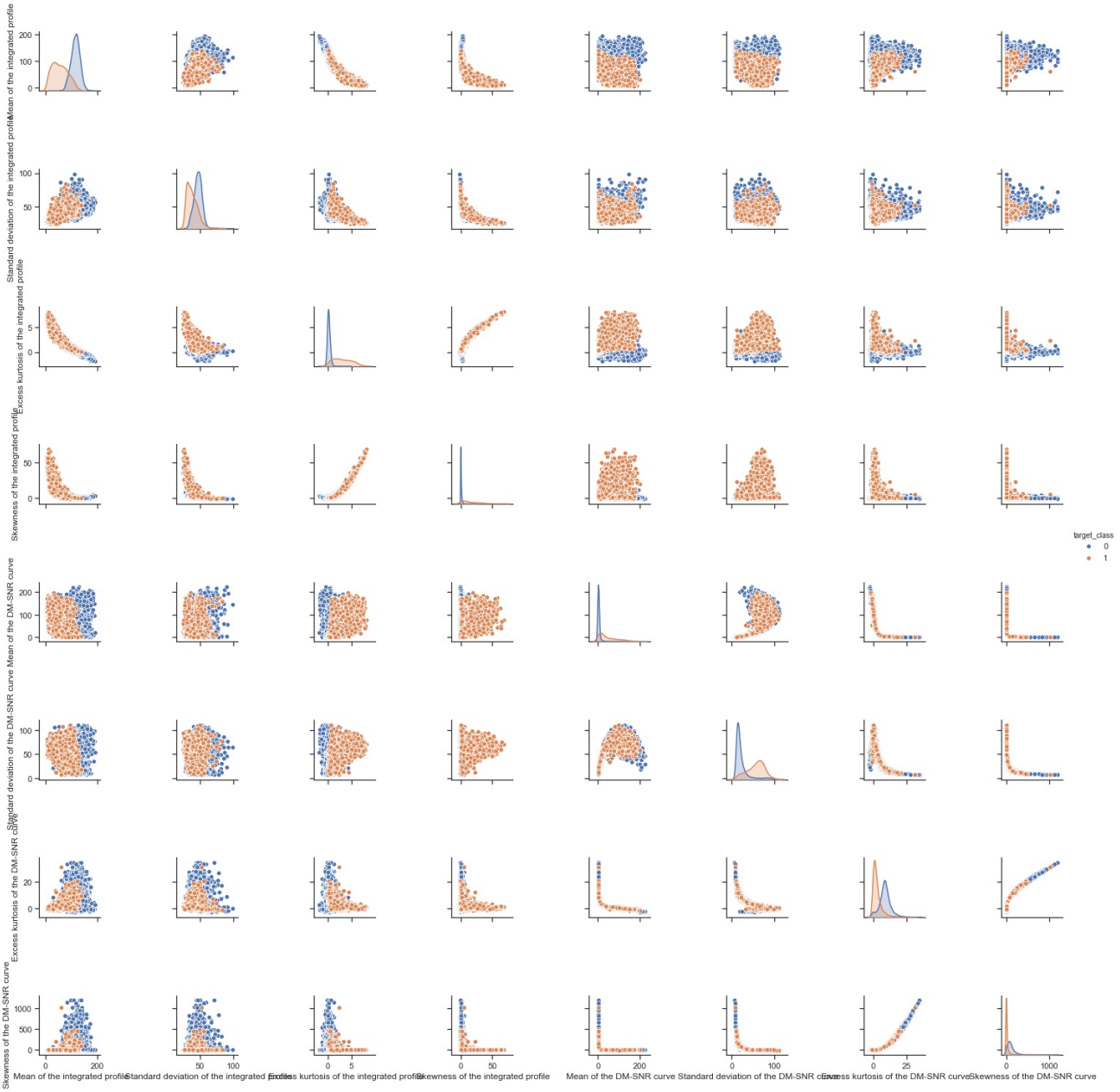


In [9]:

```
sns.pairplot(data, hue="target_class")
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x15645250>
```



In [10]:

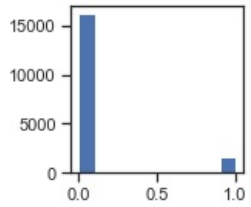
```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
data['target_class'].unique()
```

Out[10]:

```
array([0, 1], dtype=int64)
```

In [11]:

```
# Оценим дисбаланс классов для target_class
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['target_class'])
plt.show()
```



In [12]:

```
data['target_class'].value_counts()
```

Out[12]:

```
0    16259
1     1639
Name: target_class, dtype: int64
```

In [13]:

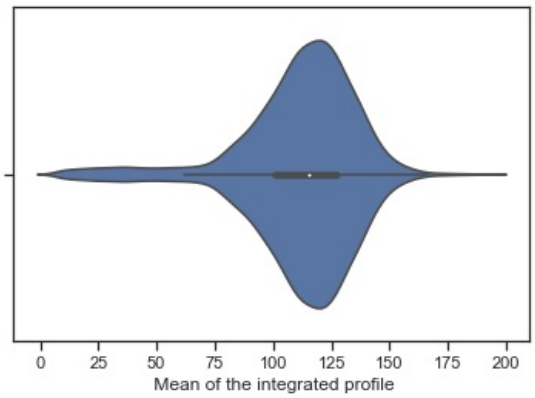
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['target_class'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

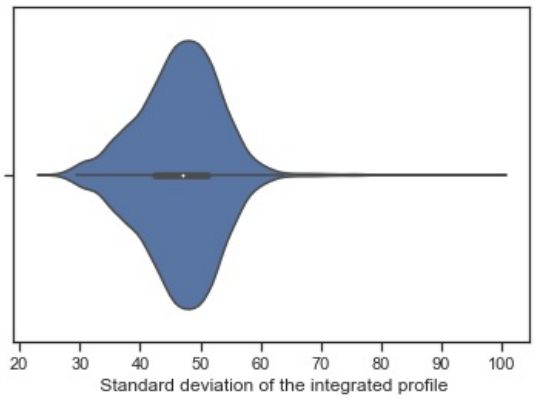
```
Класс 0 составляет 90.84%, а класс 1 составляет 9.16%.
```

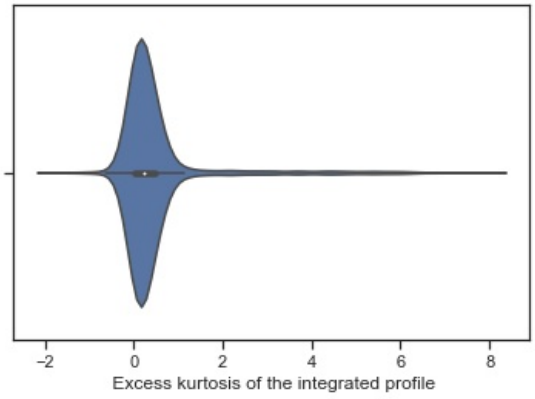
Таким образом, дисбаланс классов присутствует, но является приемлемым.

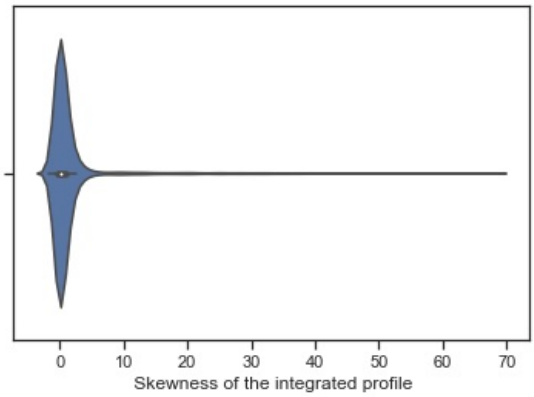
In [14]:

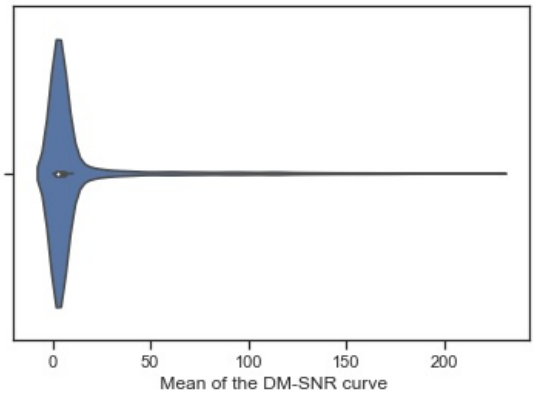
```
# Скрипичные диаграммы для числовых колонок
for col in [' Mean of the integrated profile', ' Standard deviation of the integrated profile',
            ' Excess kurtosis of the integrated profile', ' Skewness of the integrated profile',
            ' Mean of the DM-SNR curve', ' Standard deviation of the DM-SNR curve',
           ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve']:
    sns.violinplot(x=data[col])
    plt.show()
```

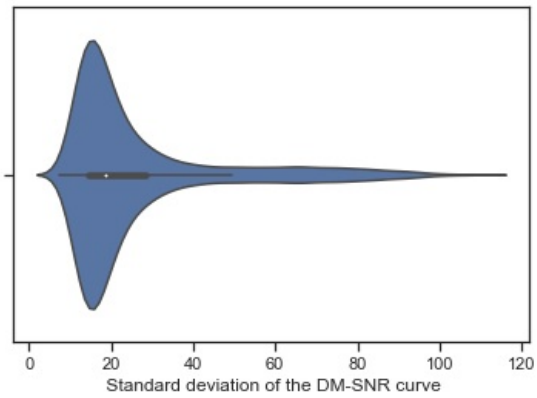


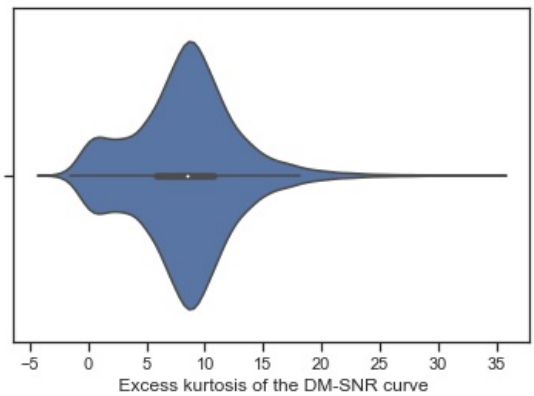


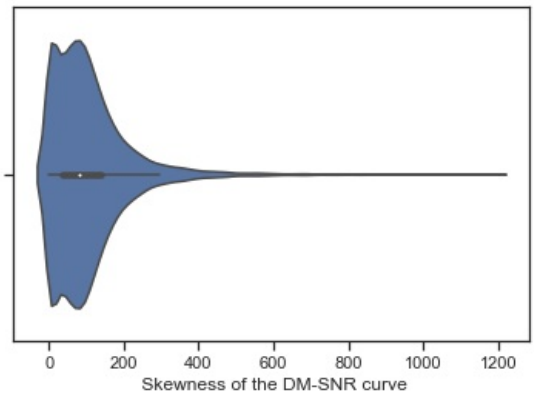



0
10
20
30
40
50
60
70
Skewness of the integrated profile


0
50
100
150
200
Mean of the DM-SNR curve


0
20
40
60
80
100
120
Standard deviation of the DM-SNR curve


−5
0
5
10
15
20
25
30
35
Excess kurtosis of the DM-SNR curve


0
200
400
600
800
1000
1200
Skewness of the DM-SNR curve

3) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирование не требуется.

Выполним масштабирование данных.

In [15]:

```
# 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']
```

In [16]:

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

In [17]:

```
# 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]
```

In [18]:

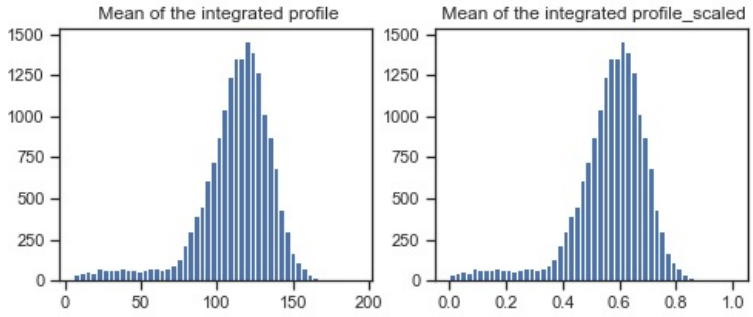
```
data.head()
```

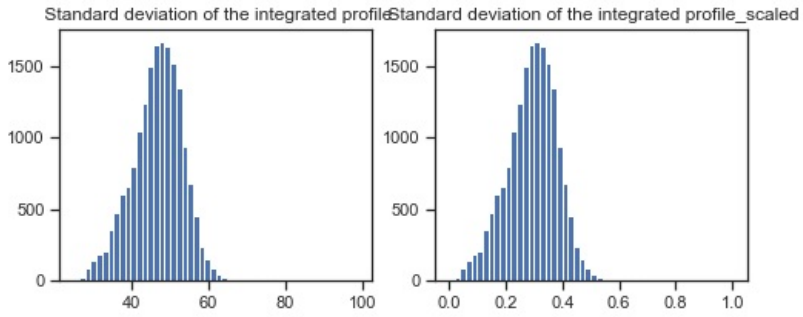
Out[18]:

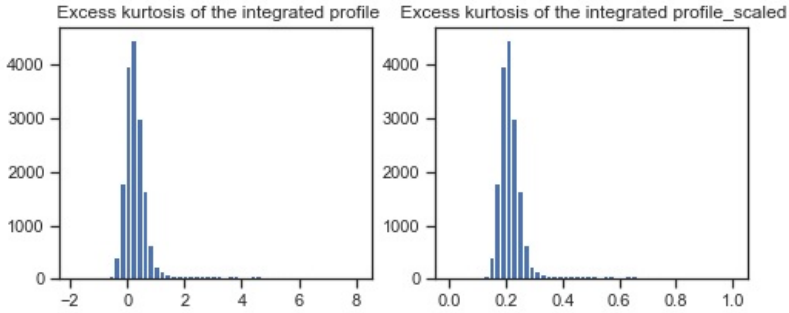
	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class	Mean of the integrated profile_scaled	Stand deviatio the integra profile_sca
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0	0.721342	0.417
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0	0.517628	0.460
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0	0.520346	0.196
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0	0.700933	0.437
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0	0.443854	0.214

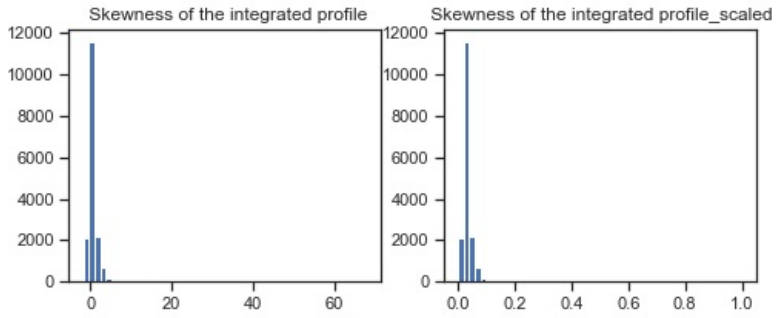
In [19]:

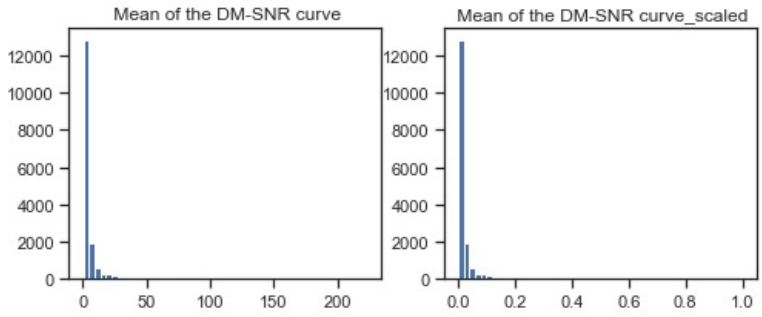
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data[col], 50)
    ax[1].hist(data[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

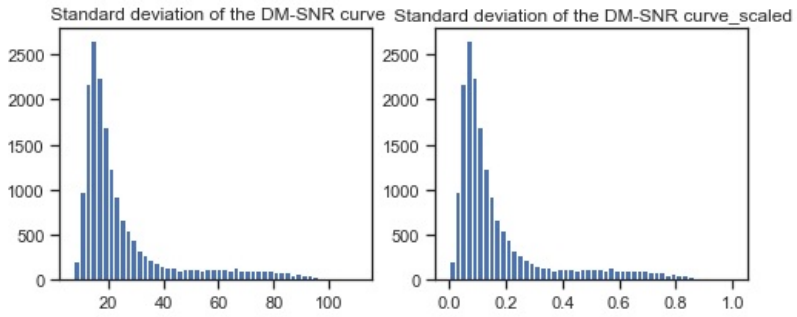


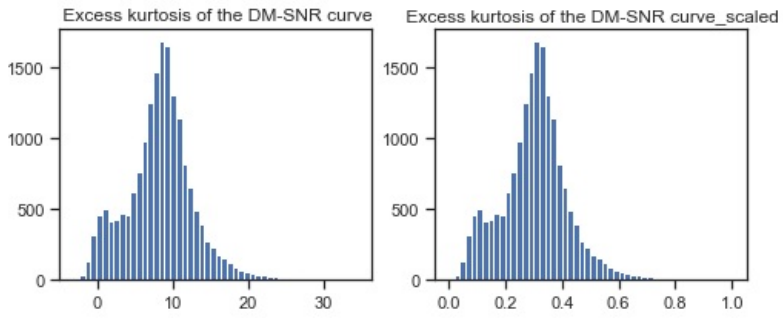

Standard deviation of the integrated profile
Standard deviation of the integrated profile_scaled

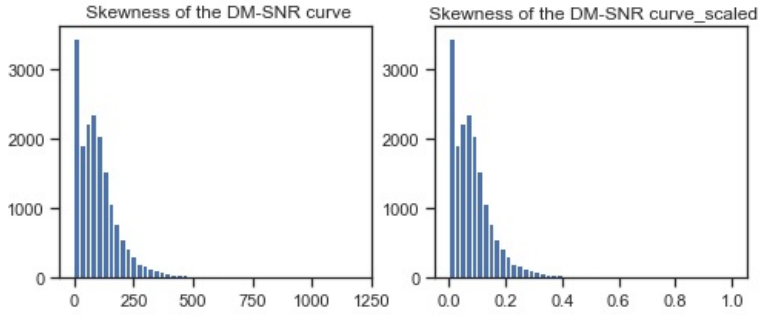

Excess kurtosis of the integrated profile
Excess kurtosis of the integrated profile_scaled


Skewness of the integrated profile
Skewness of the integrated profile_scaled


Mean of the DM-SNR curve
Mean of the DM-SNR curve_scaled


Standard deviation of the DM-SNR curve
Standard deviation of the DM-SNR curve_scaled


Excess kurtosis of the DM-SNR curve
Excess kurtosis of the DM-SNR curve_scaled



4) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения

In [20]:

```
corr_cols_1 = scale_cols + ['target_class']
corr_cols_1
```

Out[20]:

```
[' 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',
 'target_class']
```

In [21]:

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['target_class']
corr_cols_2
```

Out[21]:

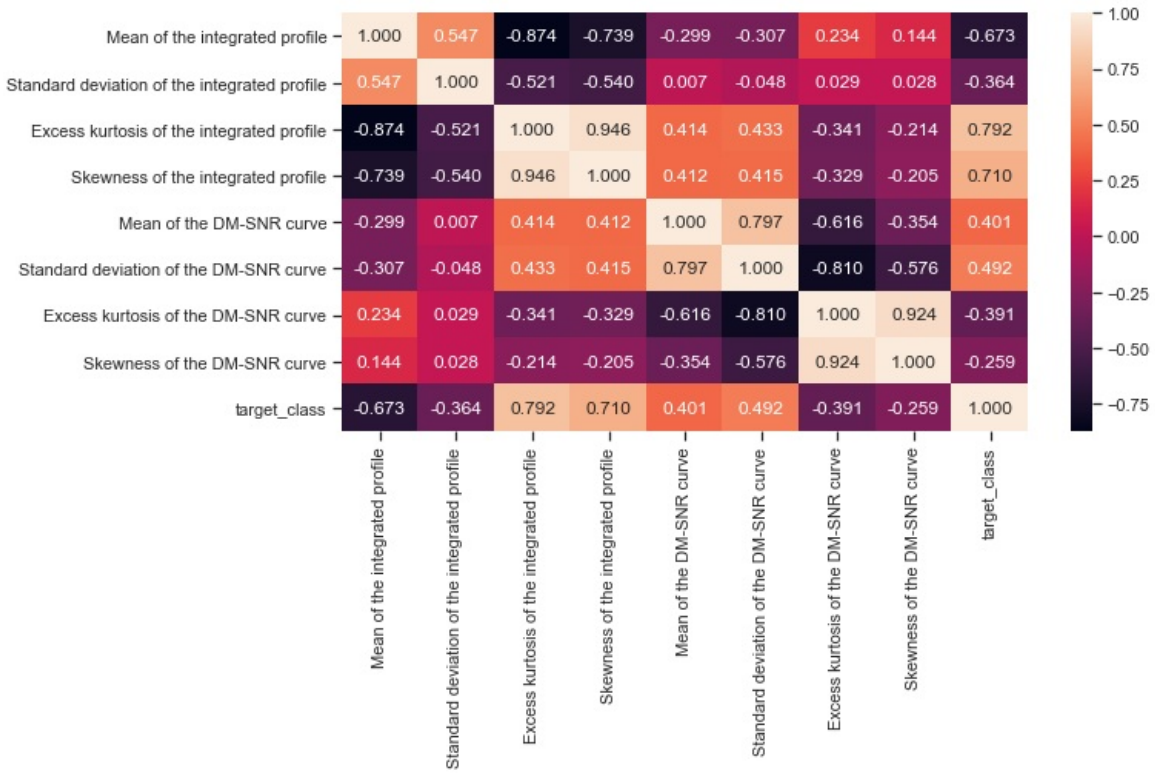
```
[' Mean of the integrated profile_scaled',
 ' Standard deviation of the integrated profile_scaled',
 ' Excess kurtosis of the integrated profile_scaled',
 ' Skewness of the integrated profile_scaled',
 ' Mean of the DM-SNR curve_scaled',
 ' Standard deviation of the DM-SNR curve_scaled',
 ' Excess kurtosis of the DM-SNR curve_scaled',
 ' Skewness of the DM-SNR curve_scaled',
 'target_class']
```

In [22]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.3f')
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d7c9b10>
```

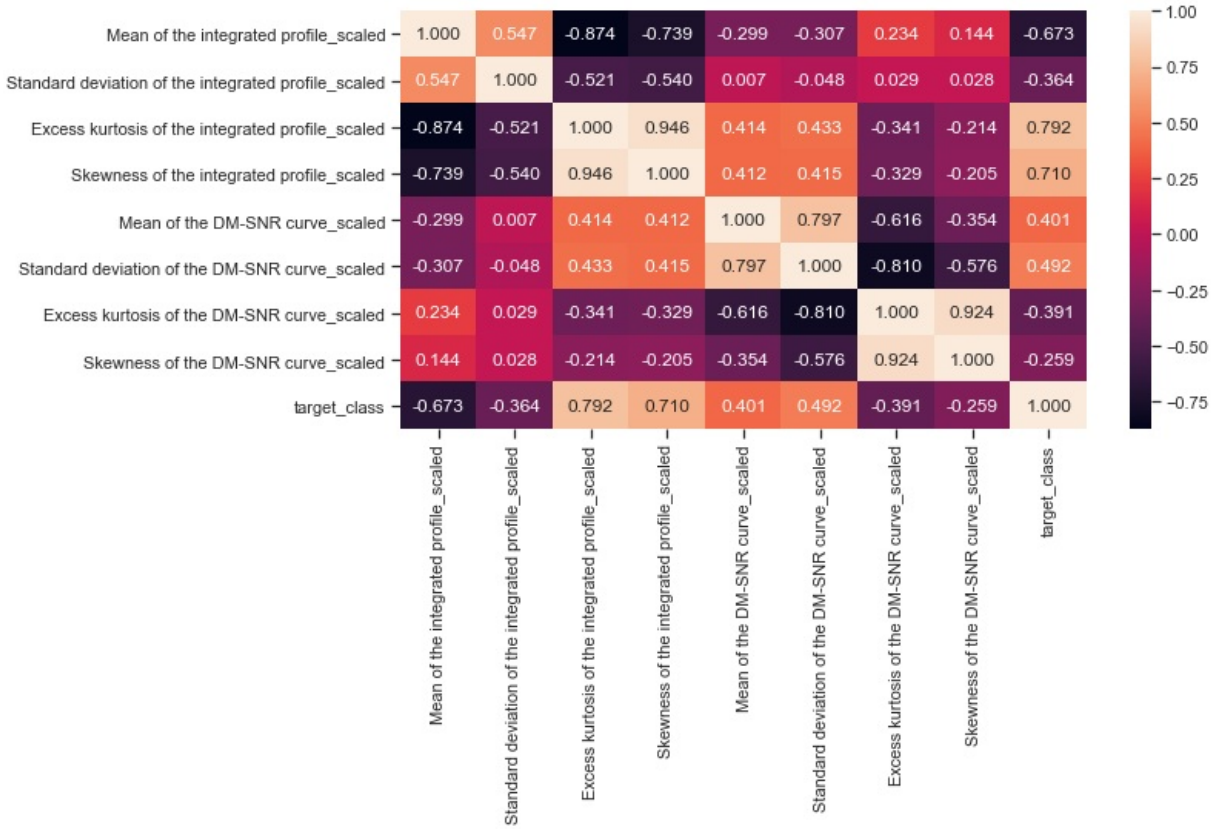


In [23]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.3f')
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bcd59b0>
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак классификации "target_class" наиболее сильно коррелирует с "Excess kurtosis of the integrated profile" (0.792) и "Skewness of the integrated profile" (0.710). Эти признаки обязательно следует оставить в модели классификации.
- Признаки "Excess kurtosis of the DM-SNR curve" и "Skewness of the DM-SNR curve" имеют корреляцию, близкую по модулю к 1, поэтому оба признака не следует включать в модель. Будем использовать признак "Excess kurtosis of the DM-SNR curve".
- Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5) Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи классификации будем использовать:

- Precision - доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.
- Recall - доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.
- ROC AUC. Основана на вычислении следующих характеристик:
  - True Positive Rate, откладывается по оси ординат. Совпадает с recall.
  - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.
- F1-мера - вычисляется как среднее гармоническое от precision и recall.
- Accuracy - вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов. Метрика "Accuracy" показывает точность по всем классам, но точность может быть различной для различных классов. Так как в данном наборе данных присутствует дисбаланс классов, будем использовать balanced_accuracy_score.

In [24]:

```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, 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
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества.

In [25]:

```
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()
```

6) Выбор наиболее подходящих моделей для решения задачи классификации или регрессии

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия
- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

7) Формирование обучающей и тестовой выборок на основе исходного набора данных

In [26]:

```
# Признаки для задачи классификации
class_cols = [' Mean of the integrated profile_scaled',
              ' Standard deviation of the integrated profile_scaled',
              ' Excess kurtosis of the integrated profile_scaled',
              ' Skewness of the integrated profile_scaled',
              ' Mean of the DM-SNR curve_scaled',
              ' Standard deviation of the DM-SNR curve_scaled',
              ' Excess kurtosis of the DM-SNR curve_scaled']
```

In [27]:

```
X = data[class_cols]
Y = data['target_class']
X.shape
```

Out[27]:

```
(17898, 7)
```

In [28]:

```
# С использованием метода train_test_split разделим выборку на обучающую и тестовую
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)
```

In [29]:

```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[29]:

```
((13423, 7), (4475, 7), (13423,), (4475,))
```

8) Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки

In [30]:

```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_4':KNeighborsClassifier(n_neighbors=4),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

In [31]:

```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

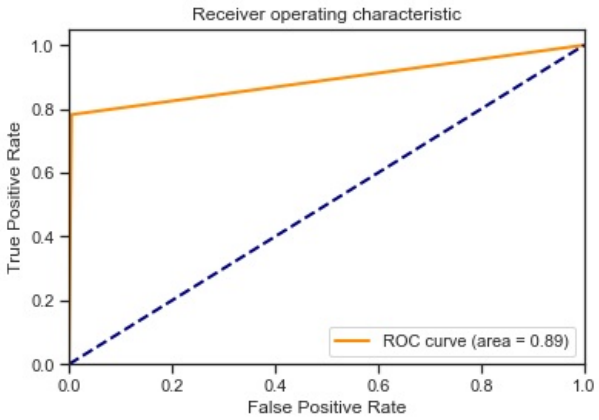
In [32]:

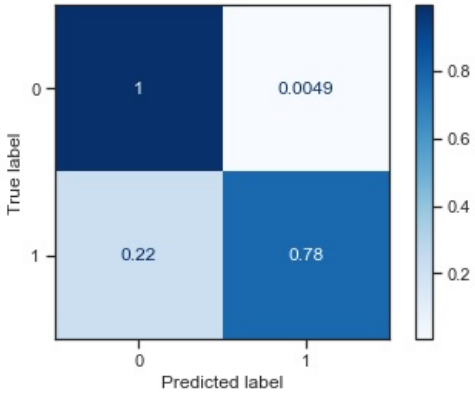
```
def train_model(model_name, model, MetricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    precision = precision_score(Y_test.values, Y_pred)
    recall = recall_score(Y_test.values, Y_pred)
    roc_auc = roc_auc_score(Y_test.values, Y_pred)
    f1 = f1_score(Y_test.values, Y_pred)
    bal_accuracy = balanced_accuracy_score(Y_test.values, Y_pred)

    MetricLogger.add('precision', model_name, precision)
    MetricLogger.add('recall', model_name, recall)
    MetricLogger.add('roc_auc', model_name, roc_auc)
    MetricLogger.add('f1', model_name, f1)
    MetricLogger.add('bal_accuracy', model_name, bal_accuracy)

    print('**************************************************')
    print(model)
    print('**************************************************')
    draw_roc_curve(Y_test.values, Y_pred)

    plot_confusion_matrix(model, X_test, Y_test.values,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

In [33]:

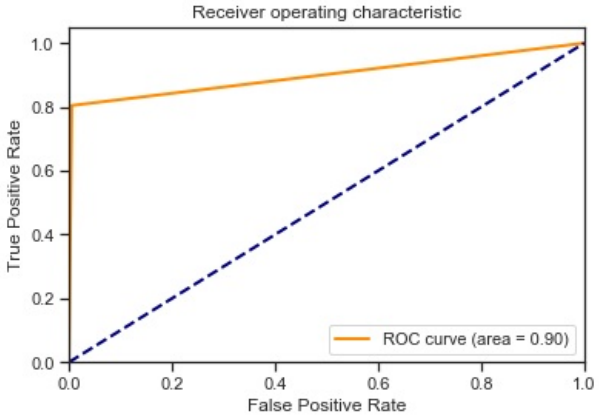
```
for model_name, model in clas_models.items():
    train_model(model_name, model, clasMetricLogger)
```

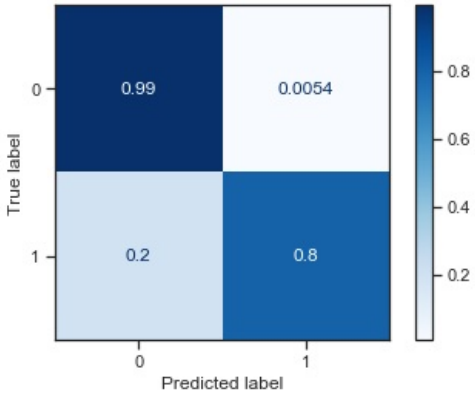
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



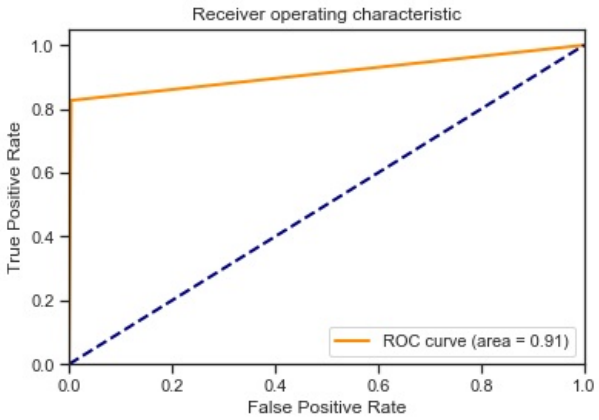


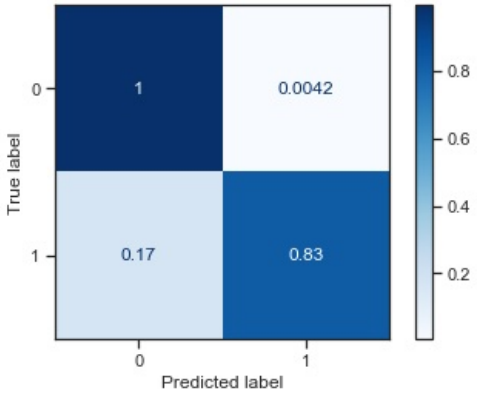

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
**************************************************
```



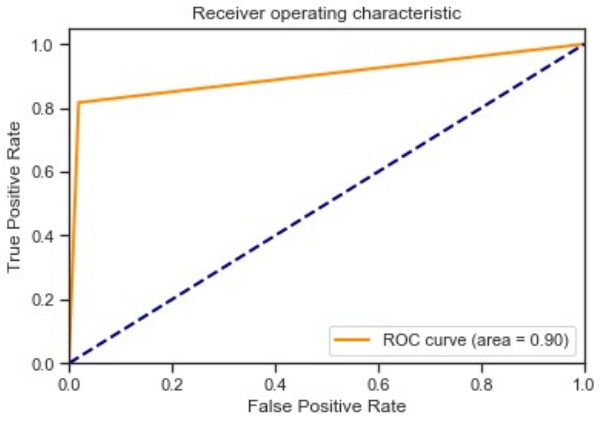


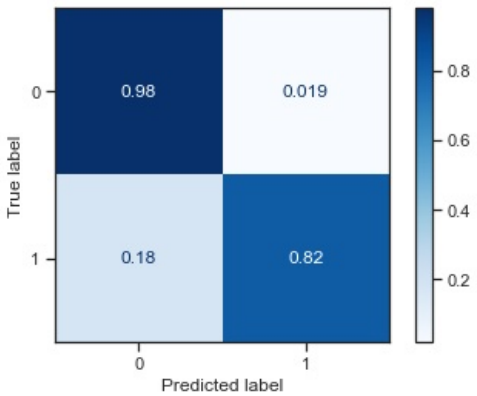

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**************************************************
```



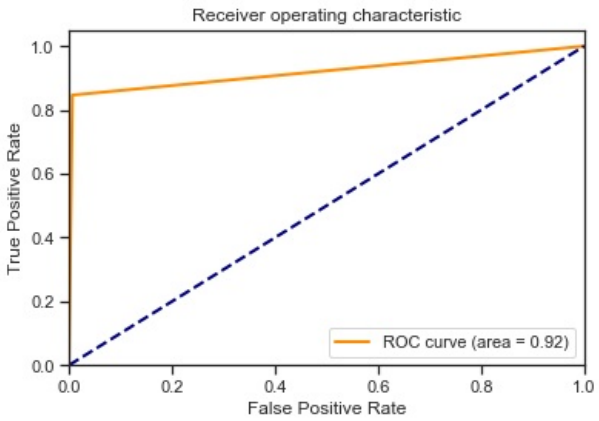


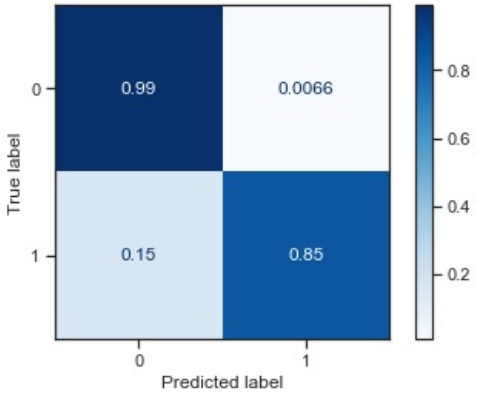

```
******************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
******************************************************
```



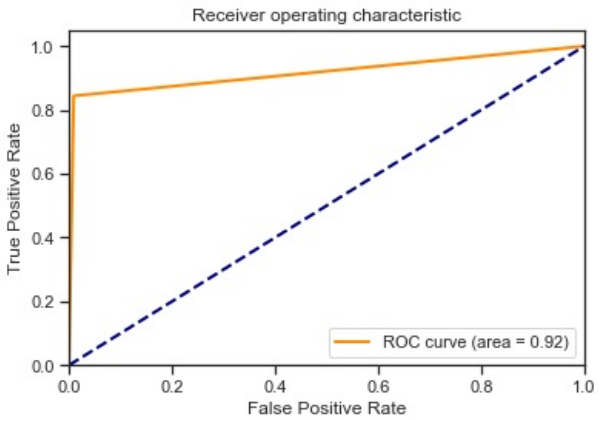


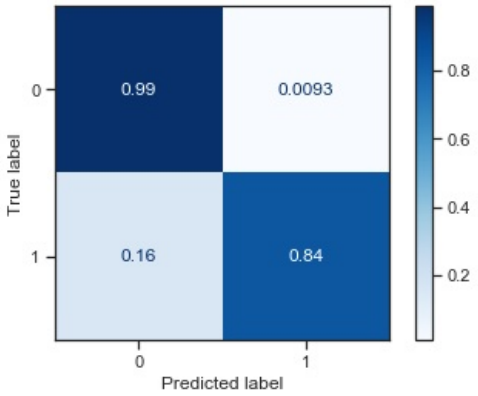
```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
**************************************************
```






```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```





9) Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы

Метод ближайших соседей

In [34]:

```
n_range = np.array(range(1,500,50))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[34]:

```
[{'n_neighbors': array([  1,  51, 101, 151, 201, 251, 301, 351, 401, 451])}]
```

In [35]:

```
gs_N = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='balanced_accuracy')
gs_N.fit(X_train, Y_train)
```

Out[35]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([  1,  51, 101, 151, 201, 251, 301, 351, 401, 451])}]
,
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

In [36]:

```
# Лучшая модель
gs_N.best_estimator_
```

Out[36]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [37]:

```
# Лучшее значение параметров
gs_N.best_params_
```

Out[37]:

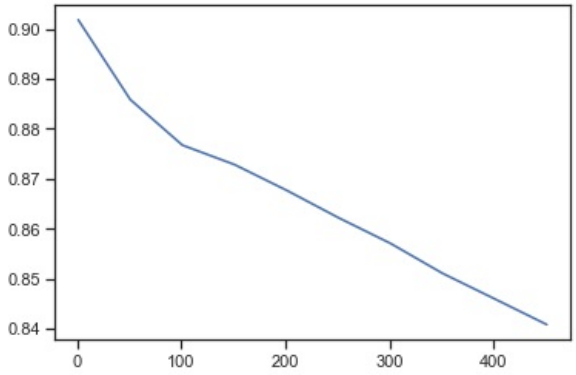
```
{'n_neighbors': 1}
```

In [38]:

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, gs_N.cv_results_['mean_test_score'])
```

Out[38]:

```
[<matplotlib.lines.Line2D at 0x1ba6e5b0>]
```



Логистическая регрессия

In [39]:

```
grid={"C":np.logspace(-3,3,7)}
gs_LR = GridSearchCV(LogisticRegression(), grid, cv=5, scoring='balanced_accuracy')
gs_LR.fit(X_train, Y_train)
```

Out[39]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

In [40]:

```
# Лучшая модель
gs_LR.best_estimator_
```

Out[40]:

```
LogisticRegression(C=1000.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [41]:

```
# Лучшее значение параметров
gs_LR.best_params_
```

Out[41]:

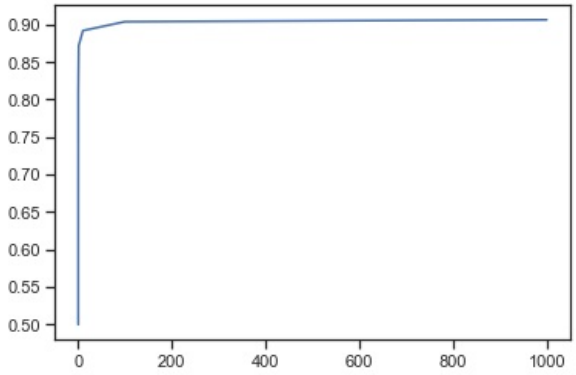
```
{'C': 1000.0}
```

In [42]:

```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,3,7), gs_LR.cv_results_['mean_test_score'])
```

Out[42]:

```
[<matplotlib.lines.Line2D at 0x1b82b550>]
```



Машина опорных векторов

In [43]:

```
SVC_grid={"C":np.logspace(-3,3,7)}
gs_SVC = GridSearchCV(SVC(), SVC_grid, cv=5, scoring='balanced_accuracy')
gs_SVC.fit(X_train, Y_train)
```

Out[43]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

In [44]:

```
# Лучшая модель
gs_SVC.best_estimator_
```

Out[44]:

```
SVC(C=1000.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [45]:

```
# Лучшее значение параметров
gs_SVC.best_params_
```

Out[45]:

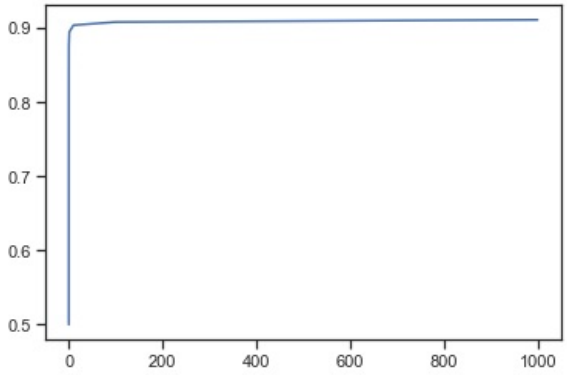
```
{'C': 1000.0}
```

In [46]:

```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,3,7), gs_SVC.cv_results_['mean_test_score'])
```

Out[46]:

```
[<matplotlib.lines.Line2D at 0x1bc02690>]
```



Решающее дерево

In [47]:

```
tree_params={"max_depth":range(1,11), "max_features":range(1,8)}
gs_Tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=5, scoring='balanced_accuracy')
gs_Tree.fit(X_train, Y_train)
```

Out[47]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': range(1, 11),
                         'max_features': range(1, 8)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

In [48]:

```
# Лучшая модель
gs_Tree.best_estimator_
```

Out[48]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=5, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [49]:

```
# Лучшее значение параметров
gs_Tree.best_params_
```

Out[49]:

```
{'max_depth': 4, 'max_features': 5}
```

Случайный лес

Будем использовать случайный поиск, т.к. полный решетчатый поиск работает слишком долго.

In [50]:

```
RF_params={"max_depth":range(1,11), "n_estimators":range(2,400)}
gs_RF = RandomizedSearchCV(RandomForestClassifier(), RF_params, cv=5, scoring='balanced_accuracy')
gs_RF.fit(X_train, Y_train)
```

Out[50]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=None,
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': range(1, 11),
                                        'n_estimators': range(2, 400)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='balanced_accuracy',
                   verbose=0)
```

In [51]:

```
# Лучшая модель
gs_RF.best_estimator_
```

Out[51]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=318,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [52]:

```
# Лучшее значение параметров
gs_RF.best_params_
```

Out[52]:

```
{'n_estimators': 318, 'max_depth': 8}
```

Градиентный бустинг

In [53]:

```
GB_params={"max_features":range(1,8), "max_leaf_nodes":range(2,16)}
gs_GB = RandomizedSearchCV(GradientBoostingClassifier(), GB_params, cv=5, scoring='balanced_accuracy')
gs_GB.fit(X_train, Y_train)
```

Out[53]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                        criterion='friedman_mse',
                                                        init=None,
                                                        learning_rate=0.1,
                                                        loss='deviance',
                                                        max_depth=3,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=2,
                                                        min_weight_fraction_leaf=0.0,
                                                        n_estimators=100,
                                                        n_ite...e,
                                                        presort='deprecated',
                                                        random_state=None,
                                                        subsample=1.0,
                                                        tol=0.0001,
                                                        validation_fraction=0.1,
                                                        verbose=0,
                                                        warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_features': range(1, 8),
                                        'max_leaf_nodes': range(2, 16)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='balanced_accuracy',
                   verbose=0)
```

In [54]:

```
# Лучшая модель
gs_GB.best_estimator_
```

Out[54]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=5, max_leaf_nodes=15,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [55]:

```
# Лучшее значение параметров
gs_GB.best_params_
```

Out[55]:

```
{'max_leaf_nodes': 15, 'max_features': 5}
```

10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей

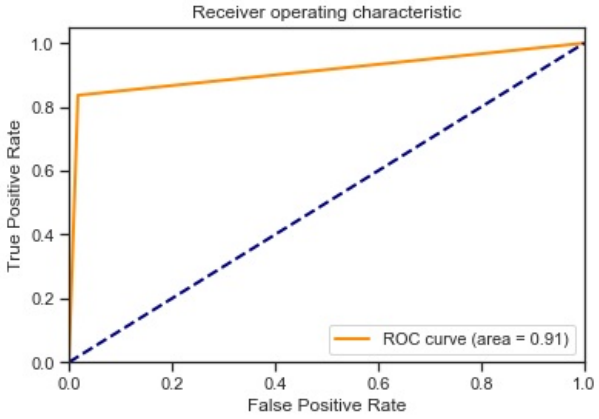
In [56]:

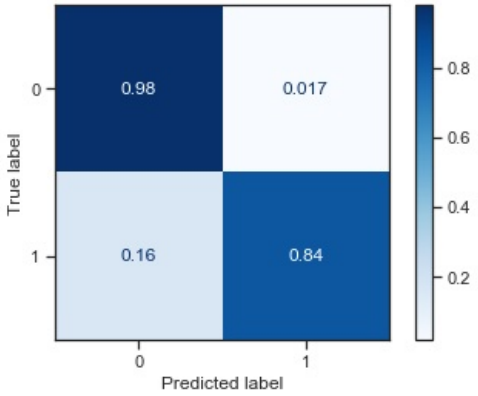
```
models_grid = {'KNN_new':gs_N.best_estimator_,
               'LogR_new':gs_LR.best_estimator_,
               'SVC_new':gs_SVC.best_estimator_,
               'Tree_new':gs_Tree.best_estimator_,
               'RF_new':gs_RF.best_estimator_,
               'GB_new':gs_GB.best_estimator_}
```

In [57]:

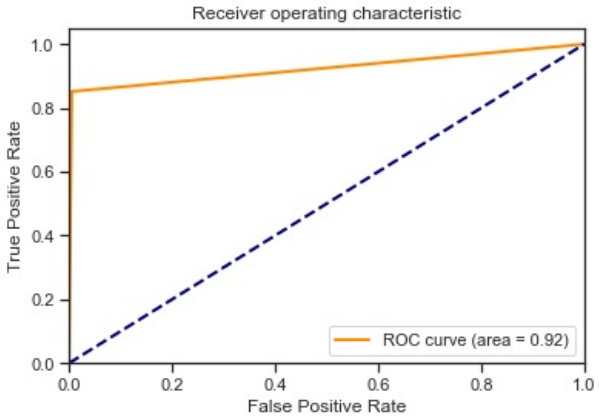
```
for model_name, model in models_grid.items():
    train_model(model_name, model, clasMetricLogger)
```

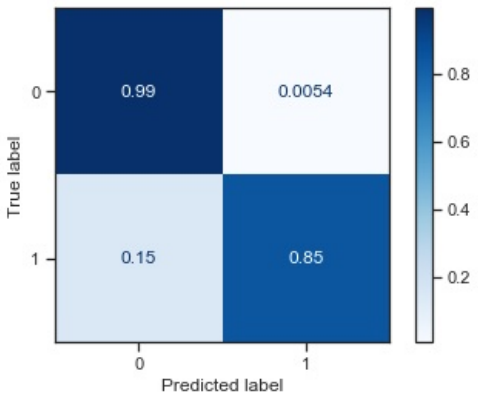
```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
**************************************************
```



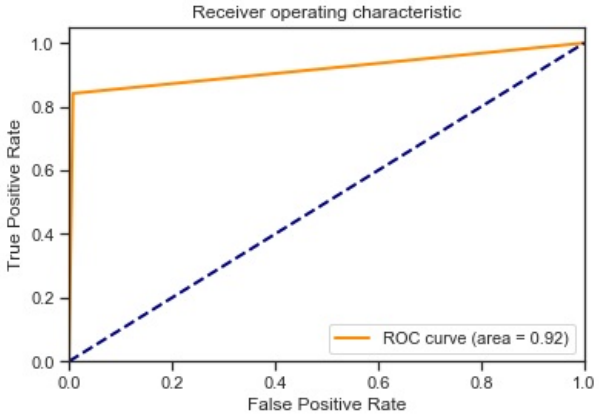


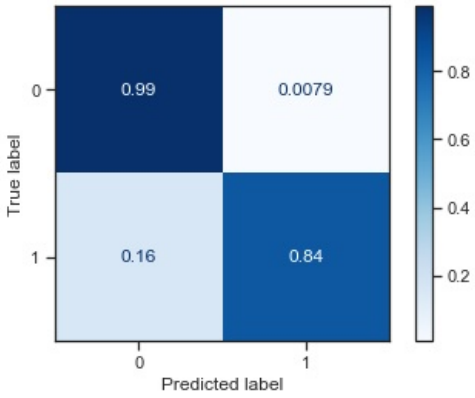

```
**************************************************
LogisticRegression(C=1000.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



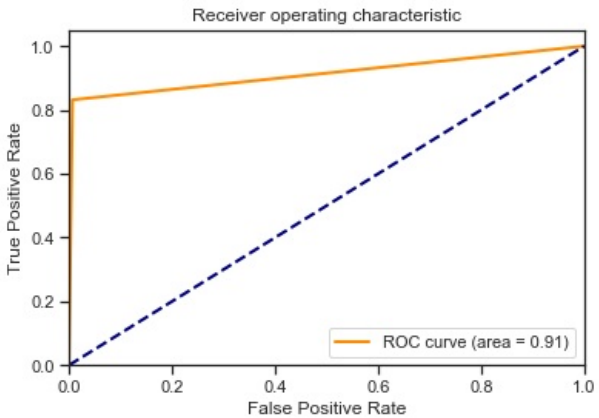


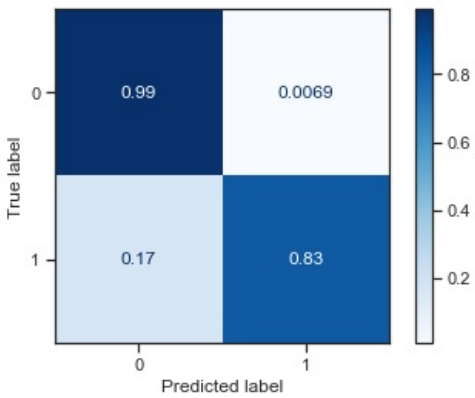
```
**************************************************
SVC(C=1000.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**************************************************
```



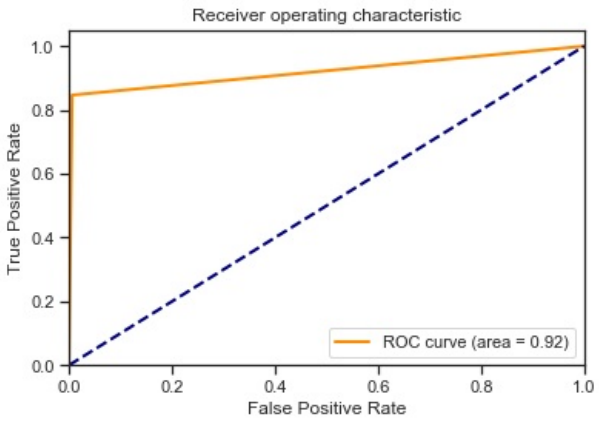


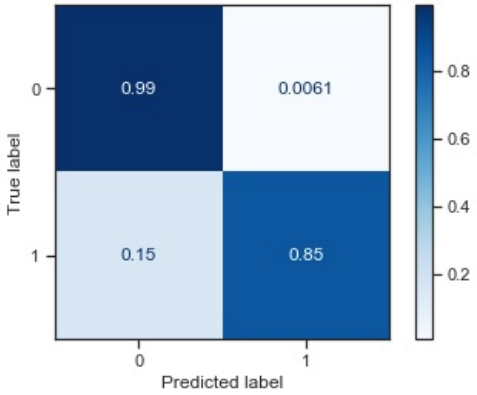

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=5, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```



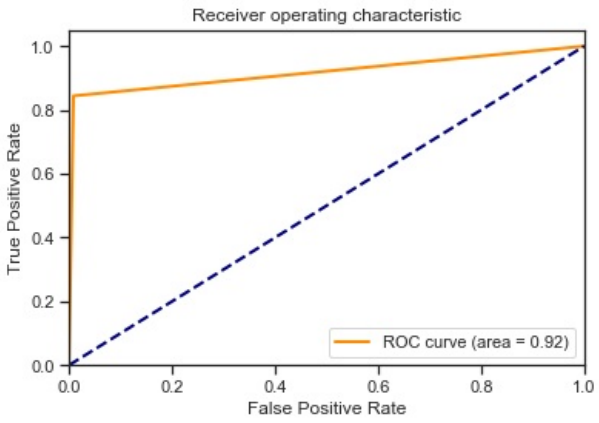


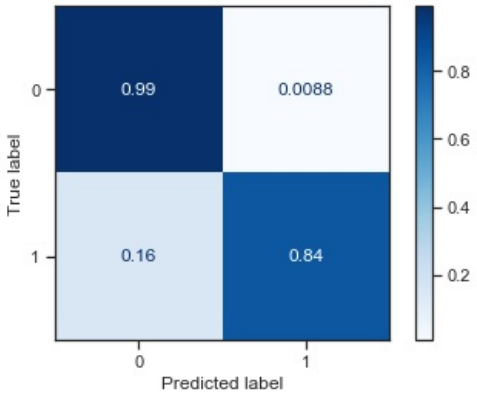
```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=318,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
**************************************************
```






```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=5, max_leaf_nodes=15,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```





11) Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

In [58]:

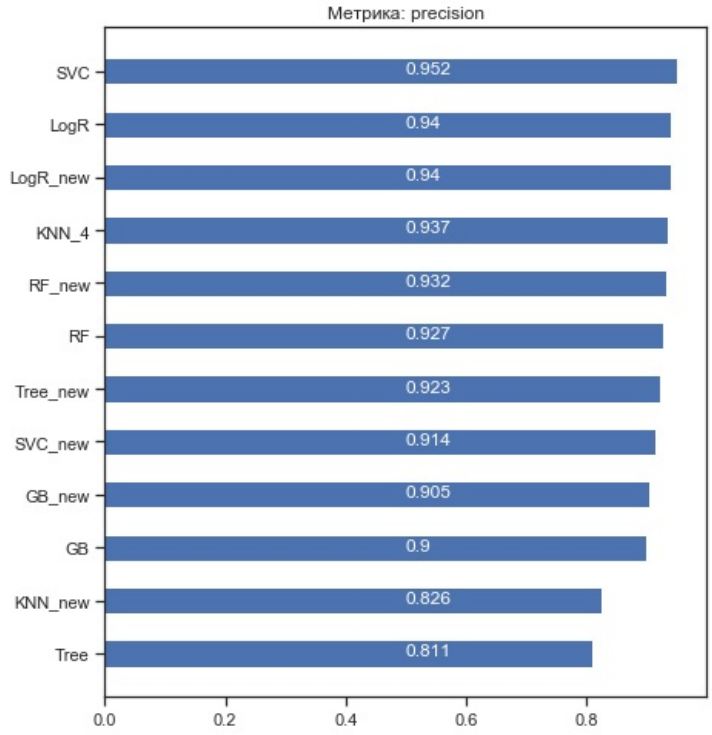
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

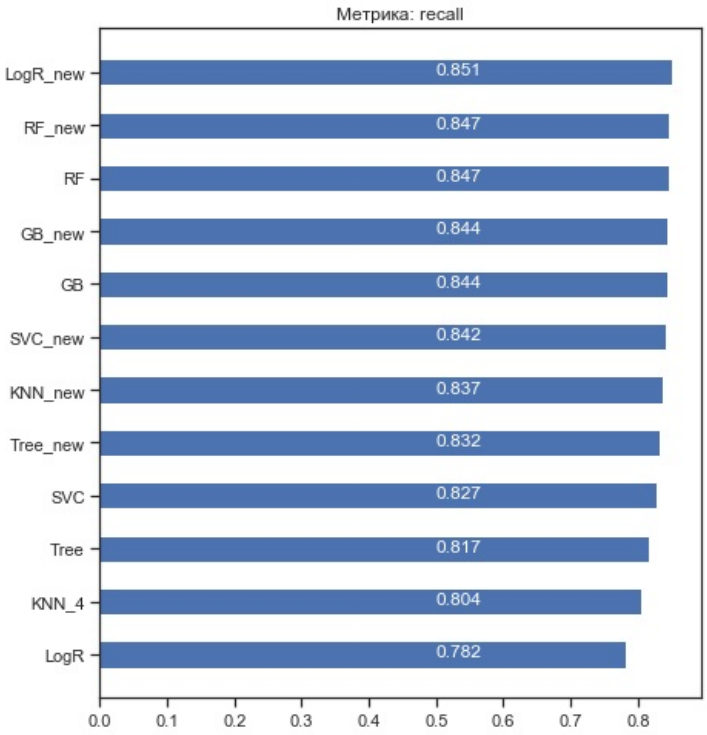
Out[58]:

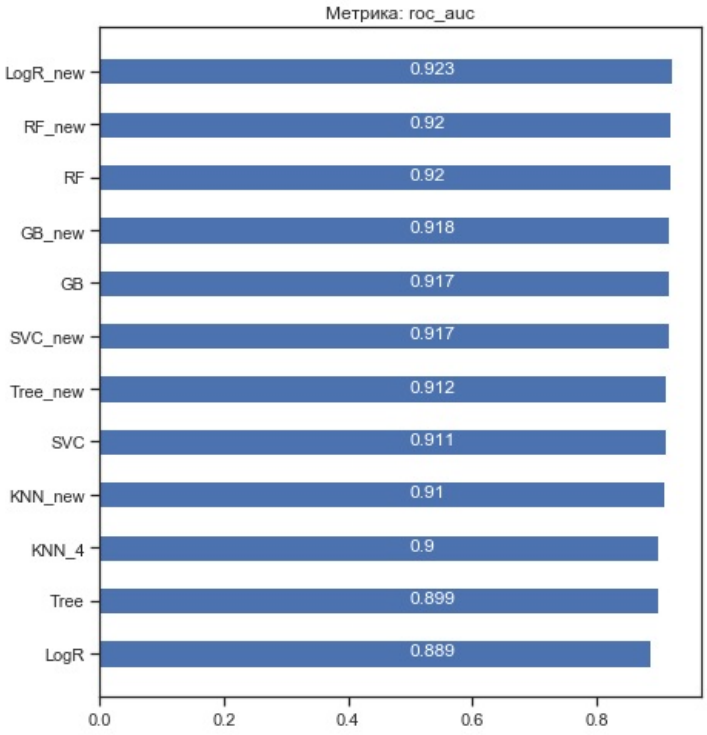
```
array(['precision', 'recall', 'roc_auc', 'f1', 'bal_accuracy'],
      dtype=object)
```

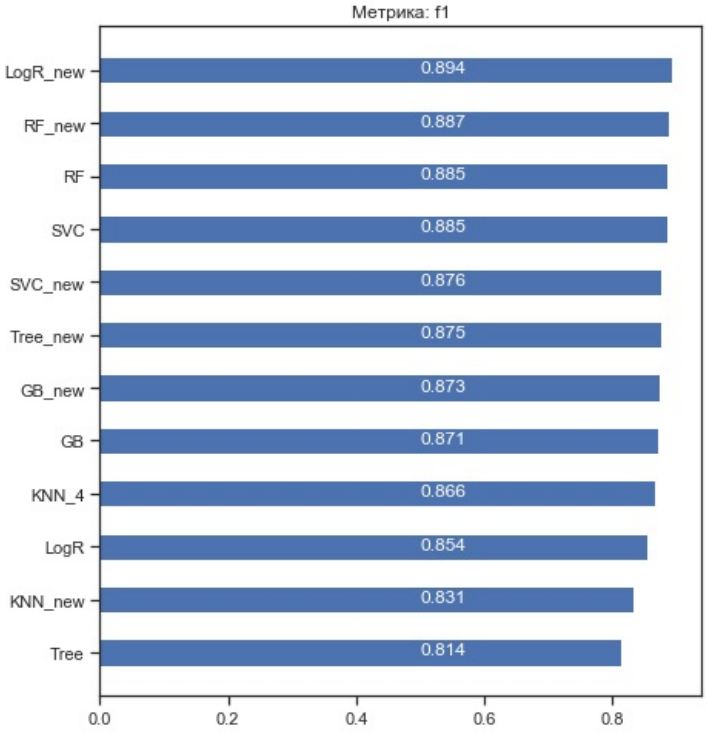
In [59]:

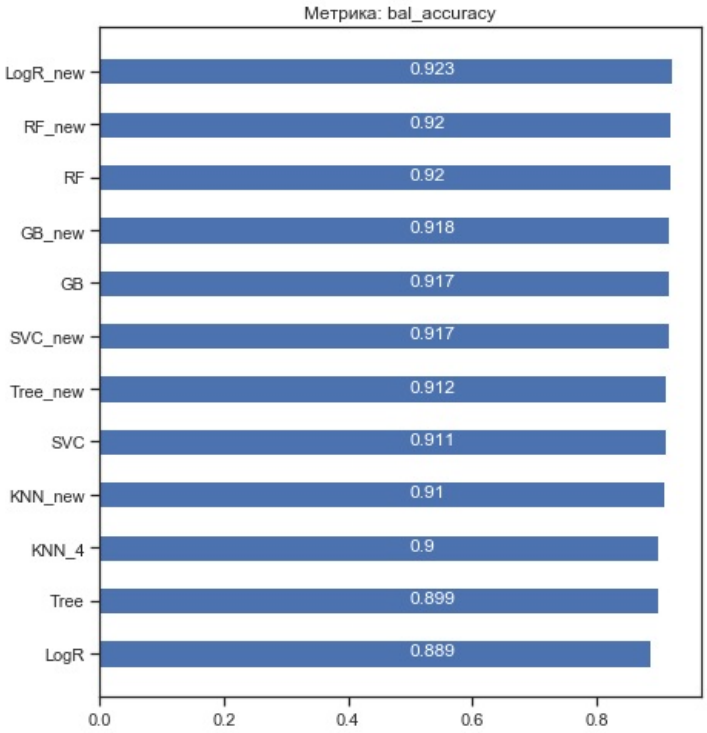
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 8))
```




Метрика: recall
LogR_new
RF_new
RF
GB_new
GB
SVC_new
KNN_new
Tree_new
SVC
Tree
KNN_4
LogR
0.851
0.847
0.847
0.844
0.844
0.842
0.837
0.832
0.827
0.817
0.804
0.782
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


Метрика: roc_auc
LogR_new
RF_new
RF
GB_new
GB
SVC_new
Tree_new
SVC
KNN_new
KNN_4
Tree
LogR
0.923
0.92
0.92
0.918
0.917
0.917
0.912
0.911
0.91
0.9
0.899
0.889
0.0
0.2
0.4
0.6
0.8





Таким образом, логистическая регрессия и случайный лес показывают наилучший результат.

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте мы выполнили типовую задачу машинного обучения. В первую очередь, выбрали набор данных для построения моделей машинного обучения, провели разведочный анализ данных и построили графики, необходимые для понимания структуры данных. Далее выбрали признаки, подходящие для построения моделей, масштабировали данные и провели корреляционный анализ данных. Это позволило нам сформировать промежуточные выводы о возможности построения моделей машинного обучения.

На следующем этапе мы выбрали метрики для последующей оценки качества моделей и наиболее подходящие модели для решения задачи классификации. Затем сформировали обучающую и тестовую выборки на основе исходного набора данных и построили базовое решение для выбранных моделей без подбора гиперпараметров.

Следующим шагом был подбор гиперпараметров для выбранных моделей, после чего мы смогли сравнить качество полученных моделей с качеством baseline-моделей. Большинство моделей, для которых были подобраны оптимальные значения гиперпараметров, показали лучший результат.

В заключение, мы сформировали выводы о качестве построенных моделей на основе выбранных метрик. Для наглядности результаты сравнения качества отобразили в виде графиков, а также сделали выводы в форме текстового описания. Четыре метрики из пяти показали, что для выбранного набора данных лучшими моделями оказались случайный лес и логистическая регрессия.

ЛИТЕРАТУРА

1. Ю.Е. Гапанюк, Лекции по курсу «Технологии машинного обучения», 2019 - 2020 учебный год
2. Электронный ресурс https://scikit-learn.org/stable/
3. Дж. Вандер Плас «Python для сложных задач. Наука о данных и машинное обучение»