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 группы ИУ5-61Б _____ Крюков Г. М.
(Группа) (Подпись, дата) (И.О.Фамилия)

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

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

Москва, 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 _____
(Индекс)
_____ _____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»

Студент группы ИУ5-61Б

_____Крюков Геннадий Максимович_____
(Фамилия, имя, отчество)

Тема курсового проекта: “Бинарная классификация”
_____
_____

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

Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на __42__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____

Дата выдачи задания «12» февраля 2020 г.

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

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

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

Содержание

Задание

В данной курсовой работе необходимо предпринять следующие шаги:

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 постройение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Введение

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

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

В рамках данной курсовой работы необходимо применить навыки, полученные в течение курса «Технологии машинного обучения», и обосновать полученные результаты.

Основная часть

Постановка задачи

В данной курсовой работе ставится задача определения принадлежности звезды к классу пульсаров по различным параметрам с помощью методов машинного обучения: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest". С помощью различных метрик выбор метода, который наиболее эффективно и качественно определяет значение целевого признака.

Решение поставленной задачи

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**

Описание выбранного датасета

Выбранный набор данных описывает статистику игр лучших игроков в дисциплине League of Legends.

League of Legends — ролевая видеоигра с элементами стратегии в реальном времени (MOBA), разработанная и выпущенная компанией Riot Games 27 октября 2009 года. Игра распространяется по модели free-to-play. Ежемесячная аудитория игры составляет 100 млн игроков по всему миру (сентябрь 2016).

Подобно игровым видам спорта, игра в League of Legends происходит в виде отдельных партий (сессий). Чтобы сыграть игру в определённом режиме, необходимо подать заявку и дождаться автоматического подбора игроков для следующей партии. Подбор осуществляется с учетом рейтинга игрока в соответствующем режиме. Поиск игры длится около минуты, сама же игровая сессия обычно длится 16-65 минут. Каждая игра начинается «с чистого листа», то есть все игроки имеют одинаковый наименьший начальный уровень, стартовое количество ресурсов.

Сейчас турниры League of Legends являются одними из самых массовых и зрелищных.
Трансляции финала мирового чемпионата Лиги Легенд третьего сезона (2013 год) посмотрело 32 миллиона зрителей, что сделало его самым популярным (к тому времени) киберспортивным событием в истории.

В 2013 году США признало профессиональных игроков League of Legends в качестве спортсменов, с возможностью выдачи им виз P-1.

Информация об атрибутах:

Каждая игра описывается 49 непрерывными переменными и одной переменной класса. Первые 2 являются «общими» параметрами. 24 – описания для одной из команд, другие 24 – описание для другой (причем один из них является целевым признаком).

Всего примеров 26 854.

- 13 454 – победа синей команды.
- 13 450 – победа красной команды.

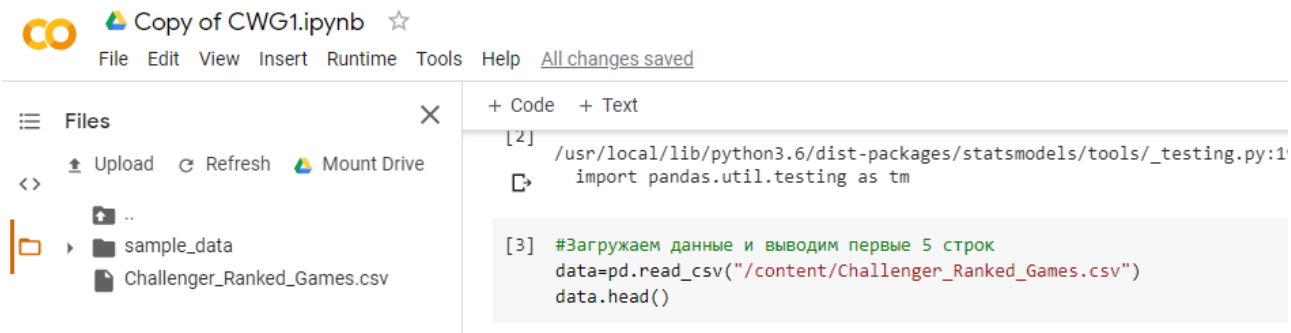
В рассматриваемом примере будем решать задачу классификации:

- Для решения **задачи классификации** в качестве целевого признака будем использовать "blueWins" (Класс). Поскольку признак содержит только значения 0 и 1, то это задача бинарной классификации.

Импорт библиотек

```
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 LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
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, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
%matplotlib inline
sns.set(style="ticks")
```

Загрузка данных.
Загрузим файл датасета с помощью библиотеки Pandas.



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

Основные характеристики датасета

```
#Загружаем данные и выводим первые 5 строк
data=pd.read_csv("/content/Challenger_Ranked_Games.csv")
data.head()
```

	gameId	gameDuraton	blueWins	blueFirstBlood	blueFirstTower	blueFirstBaron	blueFirstDragon	blueFirstInhibitor	blueDragonKills	blueBaronKills	blueTowerKills	blueInhibitorKills
0	4247263043	1323	0	1	0	0	0	0	0	0	0	0
1	4247155821	1317	1	0	0	0	1	0	2	0	4	0
2	4243963257	932	0	0	0	0	0	0	0	0	0	0
3	4241678498	2098	0	0	0	1	1	0	1	1	2	0
4	4241538868	2344	1	1	1	1	1	1	2	2	11	3

```
# Размер  датасета
data.shape
```

```
(26904, 50)
```

```
# Список колонок
data.columns
```

```
Index(['gameId', 'gameDuraton', 'blueWins', 'blueFirstBlood', 'blueFirstTower',
       'blueFirstBaron', 'blueFirstDragon', 'blueFirstInhibitor',
       'blueDragonKills', 'blueBaronKills', 'blueTowerKills',
       'blueInhibitorKills', 'blueWardPlaced', 'blueWardkills', 'blueKills',
       'blueDeath', 'blueAssist', 'blueChampionDamageDealt', 'blueTotalGold',
       'blueTotalMinionKills', 'blueTotalLevel', 'blueAvgLevel',
       'blueJungleMinionKills', 'blueKillingSpree', 'blueTotalHeal',
       'blueObjectDamageDealt', 'redWins', 'redFirstBlood', 'redFirstTower',
       'redFirstBaron', 'redFirstDragon', 'redFirstInhibitor',
       'redDragonKills', 'redBaronKills', 'redTowerKills', 'redInhibitorKills',
       'redWardPlaced', 'redWardkills', 'redKills', 'redDeath', 'redAssist',
       'redChampionDamageDealt', 'redTotalGold', 'redTotalMinionKills',
       'redTotalLevel', 'redAvgLevel', 'redJungleMinionKills',
       'redKillingSpree', 'redTotalHeal', 'redObjectDamageDealt'],
      dtype='object')
```

```
# Список колонок с типами данных
data.dtypes
```

```
gameId                        int64
gameDuraton                   int64
blueWins                      int64
blueFirstBlood                int64
blueFirstTower                int64
blueFirstBaron                int64
blueFirstDragon               int64
blueFirstInhibitor            int64
blueDragonKills               int64
blueBaronKills                int64
blueTowerKills                int64
blueInhibitorKills            int64
blueWardPlaced                int64
blueWardkills                 int64
blueKills                     int64
blueDeath                     int64
blueAssist                    int64
blueChampionDamageDealt       int64
blueTotalGold                 int64
blueTotalMinionKills          int64
blueTotalLevel                int64
blueAvgLevel                float64
blueJungleMinionKills         int64
blueKillingSpree              int64
blueTotalHeal                 int64
blueObjectDamageDealt         int64
redWins                       int64
redFirstBlood                 int64
redFirstTower                 int64
redFirstBaron                 int64
redFirstDragon                int64
redFirstInhibitor             int64
redDragonKills                int64
redBaronKills                 int64
redTowerKills                 int64
redInhibitorKills             int64
redWardPlaced                 int64
redWardkills                  int64
redKills                      int64
redDeath                      int64
redAssist                     int64
redChampionDamageDealt        int64
redTotalGold                  int64
redTotalMinionKills           int64
redTotalLevel                 int64
redAvgLevel                 float64
redJungleMinionKills          int64
redKillingSpree               int64
redTotalHeal                  int64
redObjectDamageDealt          int64
dtype: object
```

Посмотрим заполненность датасета. Возможно есть пропуски.

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

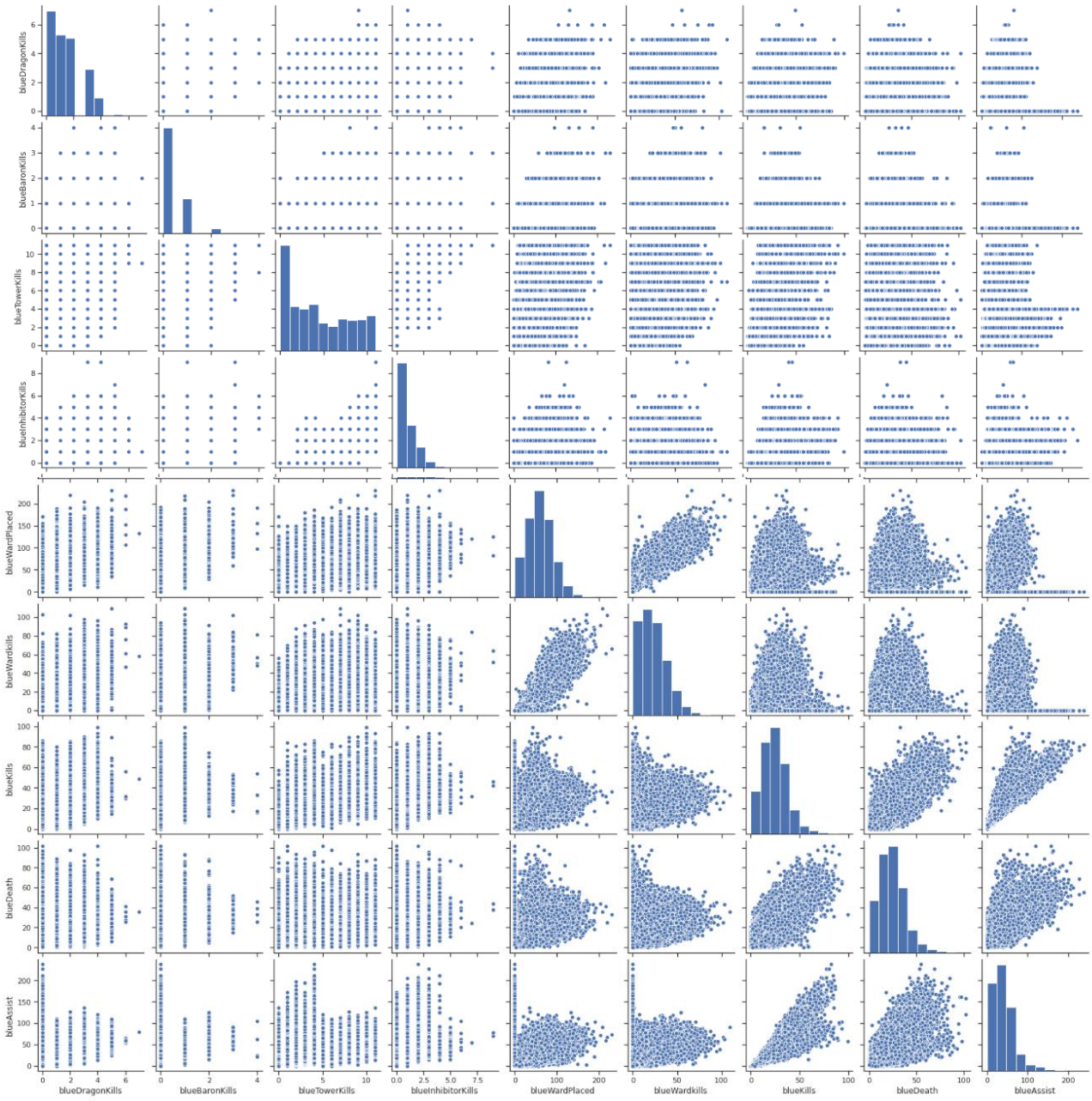
```
gameId                       0
gameDuraton                  0
blueWins                     0
blueFirstBlood               0
blueFirstTower               0
blueFirstBaron               0
blueFirstDragon              0
blueFirstInhibitor           0
blueDragonKills              0
blueBaronKills               0
blueTowerKills               0
blueInhibitorKills           0
blueWardPlaced               0
blueWardkills                0
blueKills                    0
blueDeath                    0
blueAssist                   0
blueChampionDamageDealt      0
blueTotalGold                0
blueTotalMinionKills         0
blueTotalLevel               0
blueAvgLevel                 0
blueJungleMinionKills        0
blueKillingSpree             0
blueTotalHeal                0
blueObjectDamageDealt        0
redWins                      0
redFirstBlood                0
redFirstTower                0
redFirstBaron                0
redFirstDragon               0
redFirstInhibitor            0
redDragonKills               0
redBaronKills                0
redTowerKills                0
redInhibitorKills            0
redWardPlaced                0
redWardkills                 0
redKills                     0
redDeath                     0
redAssist                    0
redChampionDamageDealt       0
redTotalGold                 0
redTotalMinionKills          0
redTotalLevel                0
redAvgLevel                  0
redJungleMinionKills         0
redKillingSpree              0
redTotalHeal                 0
redObjectDamageDealt         0
dtype: int64
```

Пропуски данных отсутствуют.

Построим некоторые графики для понимания структуры данных.

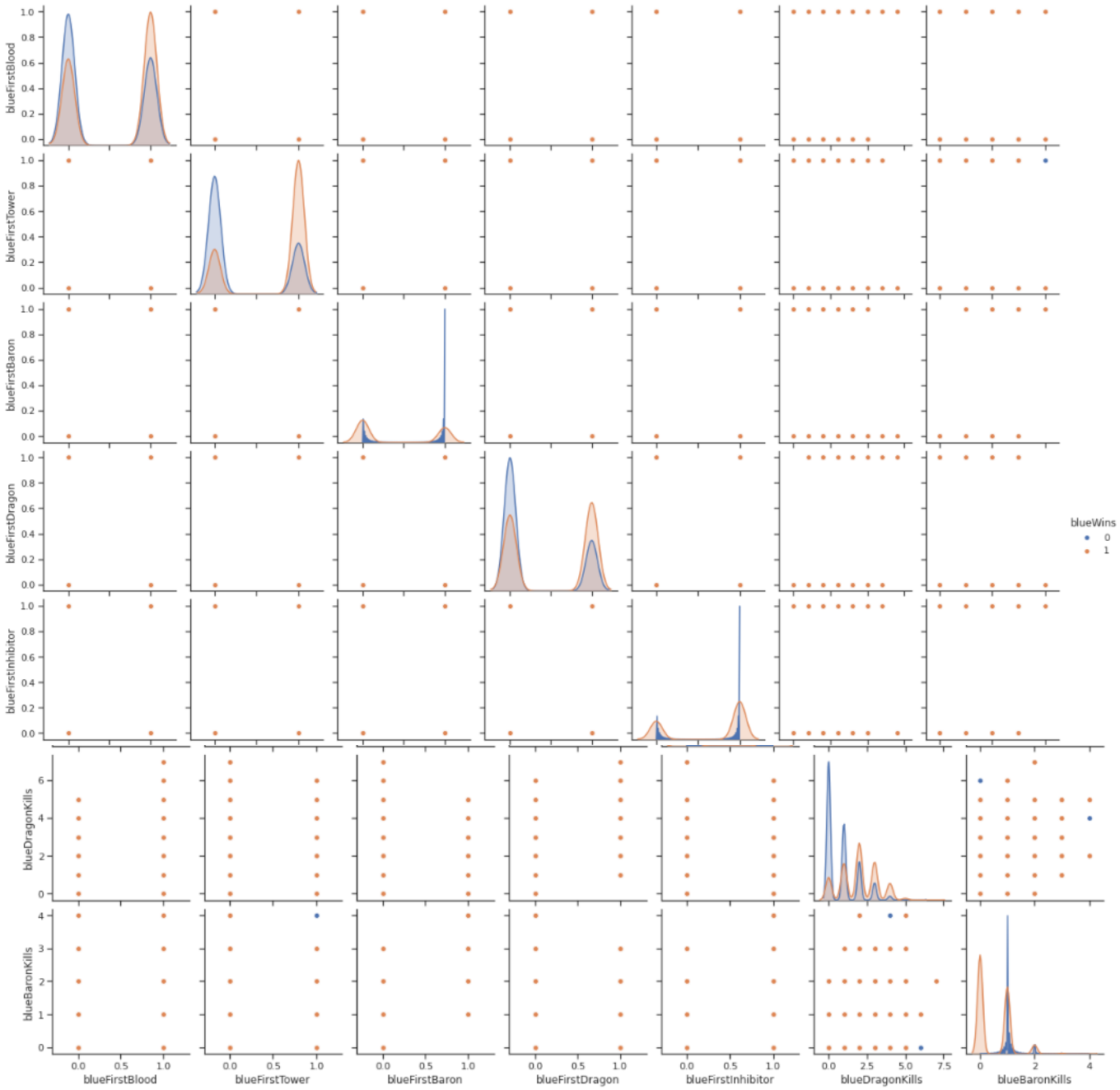
```
# Парные диаграммы
sns.pairplot(data1)
```

```
<seaborn.axisgrid.PairGrid at 0x7fba0c970c88>
```



```
#Комбинация гистограмм и диаграмм рассеивания для всего набора данных.
sns.pairplot(data2, hue="blueWins")
```

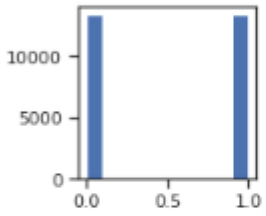
```
<seaborn.axisgrid.PairGrid at 0x7fba0c7ec748>
```



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

```
array([0, 1])
```

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




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

```
1    13454
0    13450
Name: blueWins, dtype: int64
```

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

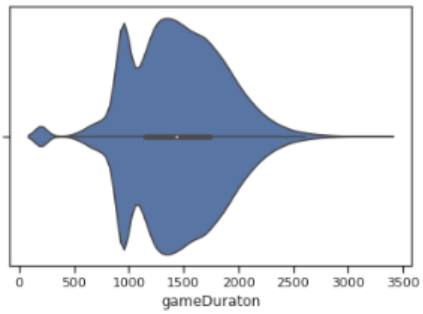
```
Класс 0 составляет 50.01%, а класс 1 составляет 49.99%.
```

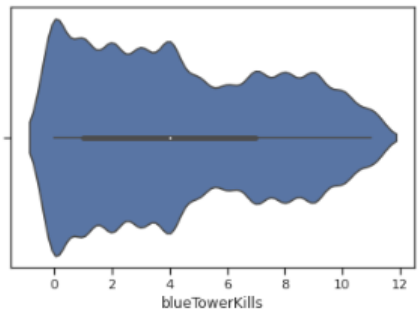
Дисбаланс классов практически отсутствует.

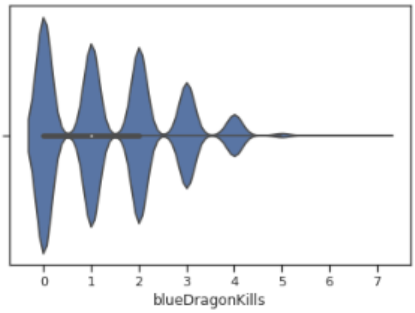
```
data.columns
```

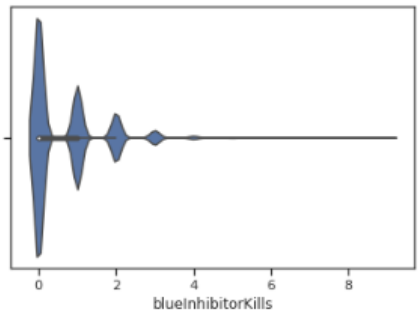
```
Index(['gameId', 'gameDuraton', 'blueWins', 'blueFirstBlood', 'blueFirstTower',
       'blueFirstBaron', 'blueFirstDragon', 'blueFirstInhibitor',
       'blueDragonKills', 'blueBaronKills', 'blueTowerKills',
       'blueInhibitorKills', 'blueWardPlaced', 'blueWardkills', 'blueKills',
       'blueDeath', 'blueAssist', 'blueChampionDamageDealt', 'blueTotalGold',
       'blueTotalMinionKills', 'blueTotalLevel', 'blueAvgLevel',
       'blueJungleMinionKills', 'blueKillingSpree', 'blueTotalHeal',
       'blueObjectDamageDealt', 'redWins', 'redFirstBlood', 'redFirstTower',
       'redFirstBaron', 'redFirstDragon', 'redFirstInhibitor',
       'redDragonKills', 'redBaronKills', 'redTowerKills', 'redInhibitorKills',
       'redWardPlaced', 'redWardkills', 'redKills', 'redDeath', 'redAssist',
       'redChampionDamageDealt', 'redTotalGold', 'redTotalMinionKills',
       'redTotalLevel', 'redAvgLevel', 'redJungleMinionKills',
       'redKillingSpree', 'redTotalHeal', 'redObjectDamageDealt'],
      dtype='object')
```

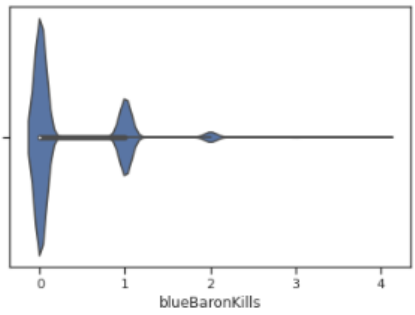
```
# Скрипичные диаграммы для числовых колонок
for col in [ 'gameDuraton', 'blueDragonKills', 'blueBaronKills', 'blueTowerKills',
       'blueInhibitorKills', 'blueWardPlaced', 'blueWardkills', 'blueKills',
       'blueDeath', 'blueAssist', 'blueChampionDamageDealt', 'blueTotalGold',
       'blueTotalMinionKills', 'blueTotalLevel', 'blueAvgLevel',
       'blueJungleMinionKills', 'blueKillingSpree', 'blueTotalHeal',
       'blueObjectDamageDealt', 'redWardPlaced', 'redWardkills', 'redKills', 'redDeath', 'redAssist',
       'redChampionDamageDealt', 'redTotalGold', 'redTotalMinionKills',
       'redTotalLevel', 'redAvgLevel', 'redJungleMinionKills',
       'redKillingSpree', 'redTotalHeal', 'redObjectDamageDealt']:
    sns.violinplot(x=data[col])
    plt.show()
```

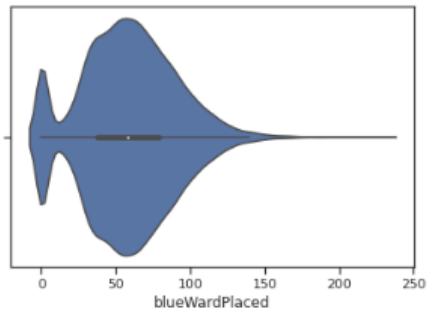

0 500 1000 1500 2000 2500 3000 3500
gameDuraton

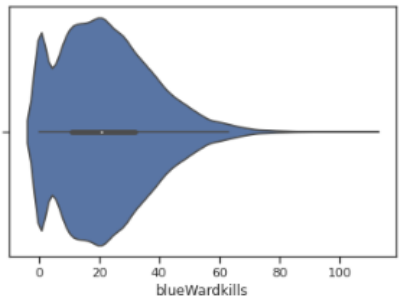

0 2 4 6 8 10 12
blueTowerKills

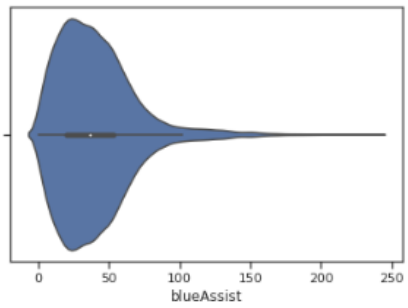

0 1 2 3 4 5 6 7
blueDragonKills

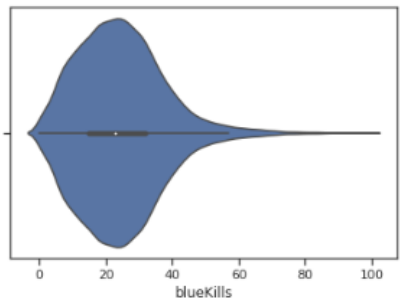

0 2 4 6 8
blueInhibitorKills

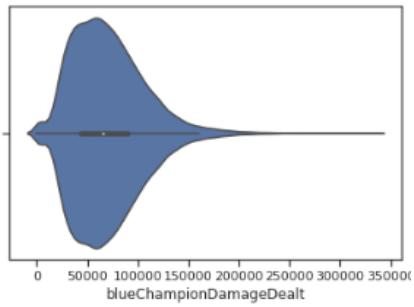

0 1 2 3 4
blueBaronKills

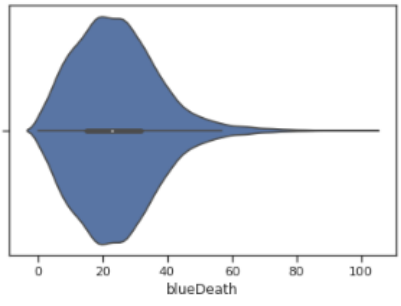

0 50 100 150 200 250
blueWardPlaced

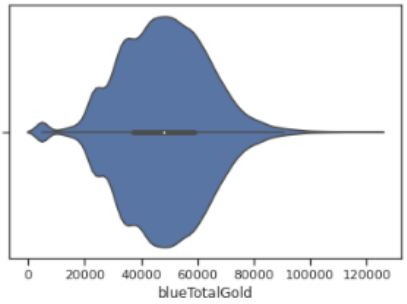

0 20 40 60 80 100
blueWardkills

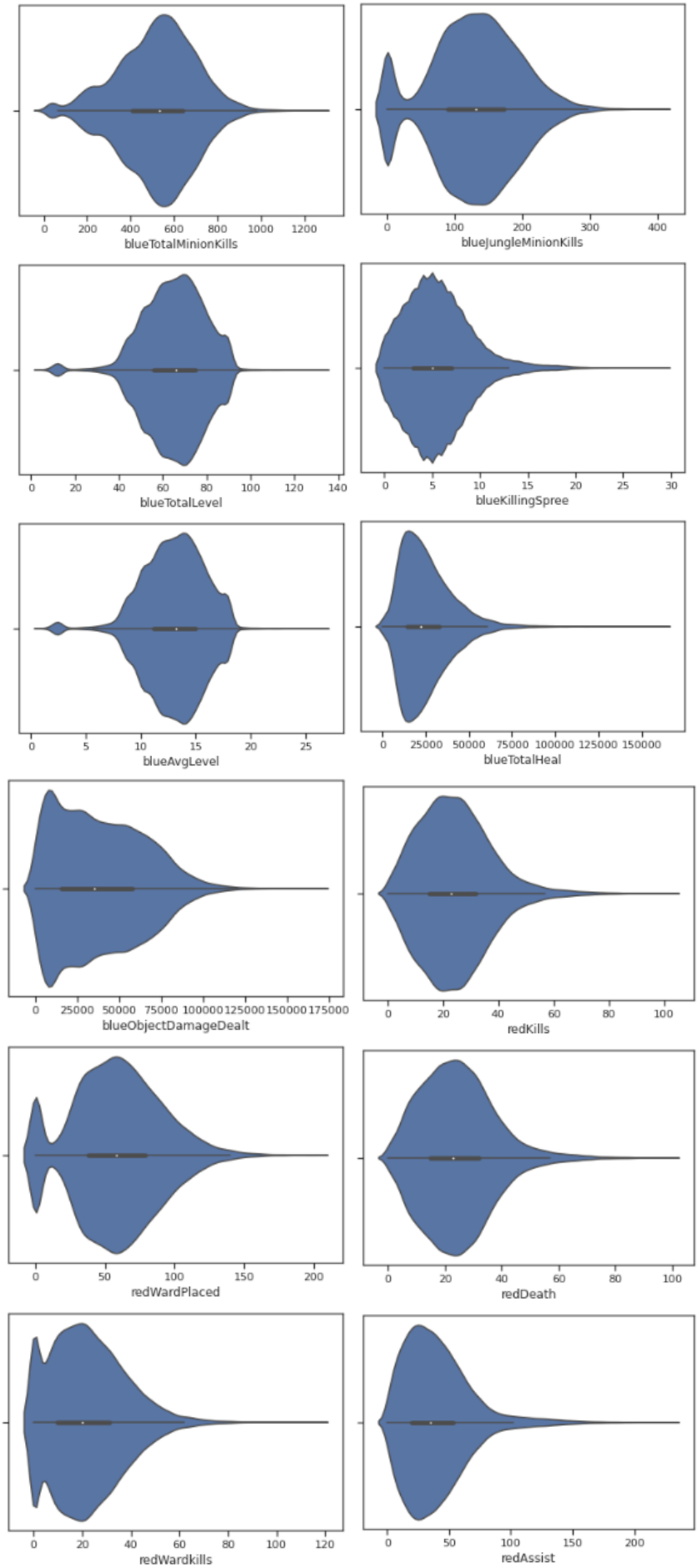

0 50 100 150 200 250
blueAssist

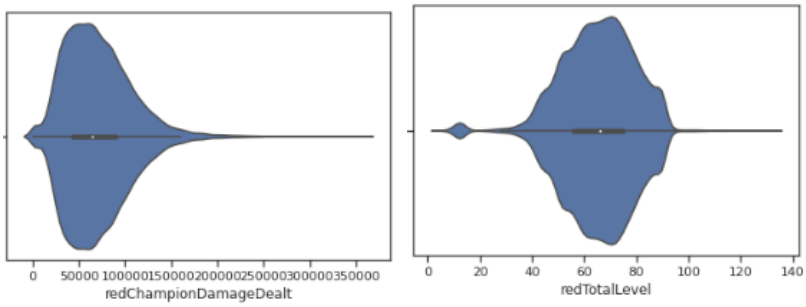

0 20 40 60 80 100
blueKills


0 50000 100000 150000 200000 250000 300000 350000
blueChampionDamageDealt

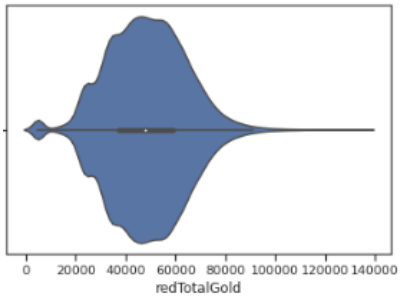

0 20 40 60 80 100
blueDeath

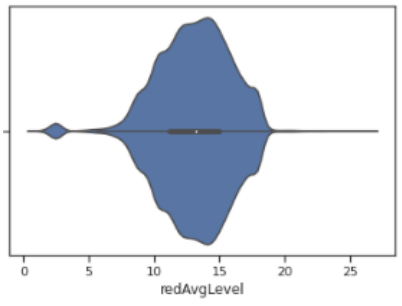

0 20000 40000 60000 80000 100000 120000
blueTotalGold

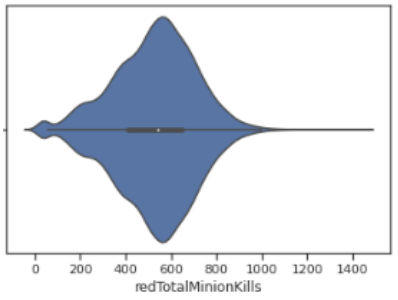

0 200 400 600 800 1000 1200
blueTotalMinionKills
0 100 200 300 400
blueJungleMinionKills
0 20 40 60 80 100 120 140
blueTotalLevel
0 5 10 15 20 25 30
blueKillingSpree
0 5 10 15 20 25
blueAvgLevel
0 25000 50000 75000 100000 125000 150000
blueTotalHeal
0 25000 50000 75000 100000 125000 150000 175000
blueObjectDamageDealt
0 20 40 60 80 100
redKills
0 50 100 150 200
redWardPlaced
0 20 40 60 80 100
redDeath
0 20 40 60 80 100 120
redWardkills
0 50 100 150 200
redAssist

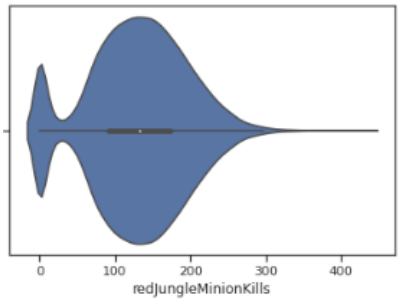

0 50000 100000 150000 200000 250000 300000 350000
redChampionDamageDealt

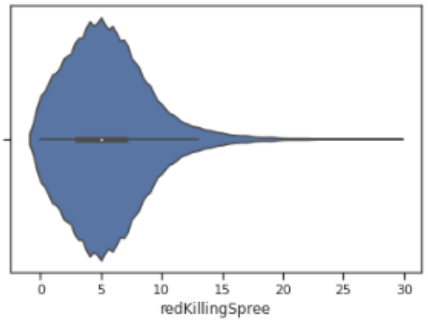
0 20 40 60 80 100 120 140
redTotalLevel

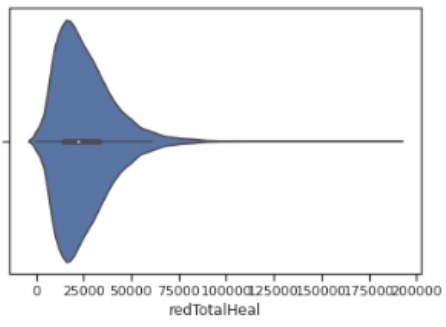

0 20000 40000 60000 80000 100000 120000 140000
redTotalGold

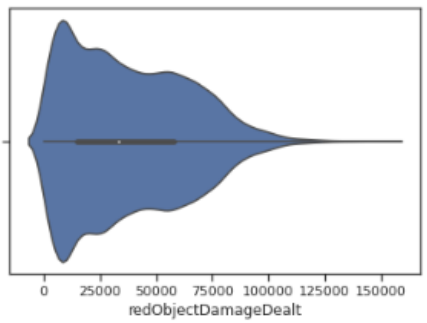

0 5 10 15 20 25
redAvgLevel


0 200 400 600 800 1000 1200 1400
redTotalMinionKills


0 100 200 300 400
redJungleMinionKills


0 5 10 15 20 25 30
redKillingSpree


0 25000 50000 75000 100000 125000 150000 175000 200000
redTotalHeal


0 25000 50000 75000 100000 125000 150000
redObjectDamageDealt

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 кодирования не требуется. Исключением является признак target_class, но в представленном датасете он уже закодирован на основе подхода LabelEncoding.

Вспомогательные признаки для улучшения качества моделей строить не будем.

Выполним масштабирование данных.

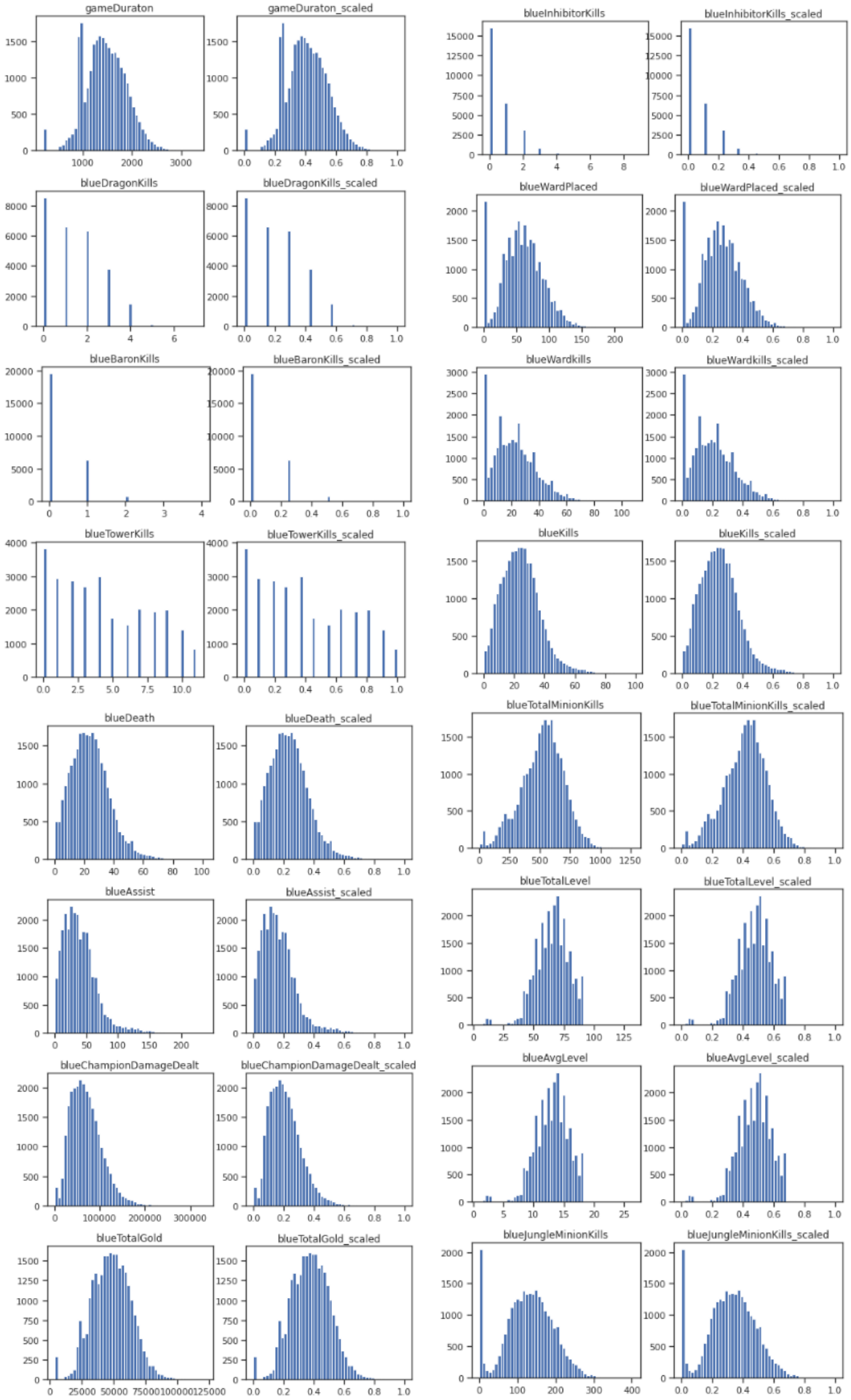
```
# Числовые колонки для масштабирования
scale_cols = ['gameDuraton', 'blueDragonKills', 'blueBaronKills', 'blueTowerKills',
       'blueInhibitorKills', 'blueWardPlaced', 'blueWardkills', 'blueKills',
       'blueDeath', 'blueAssist', 'blueChampionDamageDealt', 'blueTotalGold',
       'blueTotalMinionKills', 'blueTotalLevel', 'blueAvgLevel',
       'blueJungleMinionKills', 'blueKillingSpree', 'blueTotalHeal',
       'blueObjectDamageDealt', 'redWardPlaced', 'redWardkills', 'redKills', 'redDeath', 'redAssist',
       'redChampionDamageDealt', 'redTotalGold', 'redTotalMinionKills',
       'redTotalLevel', 'redAvgLevel', 'redJungleMinionKills',
       'redKillingSpree', 'redTotalHeal', 'redObjectDamageDealt']
bul_cols = ['blueWins', 'blueFirstBlood', 'blueFirstTower',
       'blueFirstBaron', 'blueFirstDragon', 'blueFirstInhibitor',
       'redWins', 'redFirstBlood', 'redFirstTower',
       'redFirstBaron', 'redFirstDragon', 'redFirstInhibitor']
```

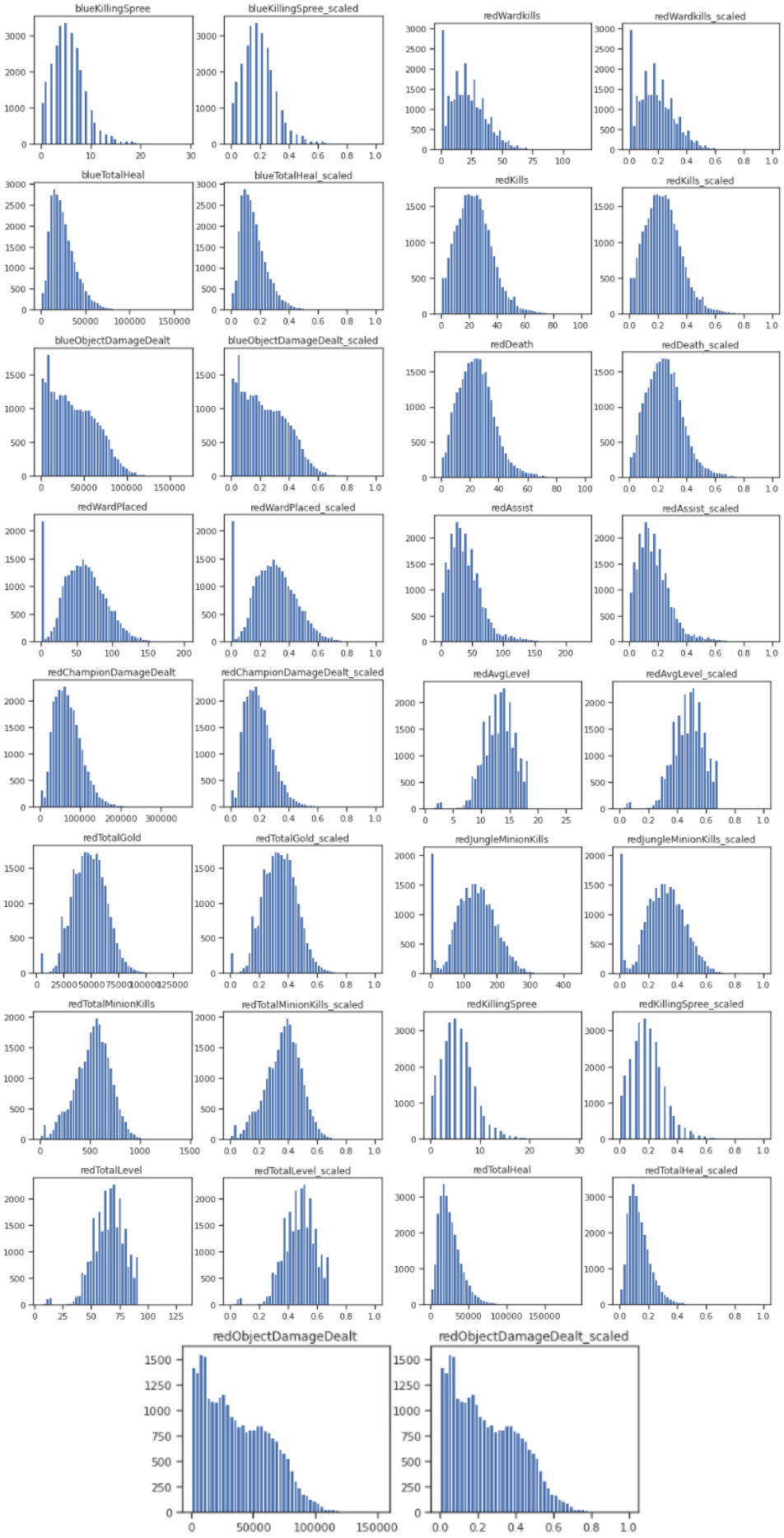
```
from sklearn.preprocessing import MinMaxScaler
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

```
# 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]
```

gameDuraton_scaled	blueDragonKills_scaled	blueBaronKills_scaled	blueTowerKills_scaled	blueInhibitorKills_scaled	blueWardPlaced_scaled	blueWardkills_scaled	blueKills_scaled
0.364192	0.000000	0.00	0.000000	0.000000	0.165217	0.119266	0.151515
0.362263	0.285714	0.00	0.363636	0.000000	0.247826	0.165138	0.191919
0.238509	0.000000	0.00	0.000000	0.000000	0.121739	0.064220	0.050505
0.613308	0.142857	0.25	0.181818	0.000000	0.560870	0.357798	0.262626
0.692382	0.285714	0.50	1.000000	0.333333	0.495652	0.321101	0.272727

```
# 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()
```


gameDuraton
gameDuraton_scaled
blueInhibitorKills
blueInhibitorKills_scaled
blueDragonKills
blueDragonKills_scaled
blueWardPlaced
blueWardPlaced_scaled
blueBaronKills
blueBaronKills_scaled
blueWardkills
blueWardkills_scaled
blueTowerKills
blueTowerKills_scaled
blueKills
blueKills_scaled
blueDeath
blueDeath_scaled
blueTotalMinionKills
blueTotalMinionKills_scaled
blueAssist
blueAssist_scaled
blueTotalLevel
blueTotalLevel_scaled
blueChampionDamageDealt
blueChampionDamageDealt_scaled
blueAvgLevel
blueAvgLevel_scaled
blueTotalGold
blueTotalGold_scaled
blueJungleMinionKills
blueJungleMinionKills_scaled


blueKillingSpree
blueKillingSpree_scaled
redWardkills
redWardkills_scaled
blueTotalHeal
blueTotalHeal_scaled
redKills
redKills_scaled
blueObjectDamageDealt
blueObjectDamageDealt_scaled
redDeath
redDeath_scaled
redWardPlaced
redWardPlaced_scaled
redAssist
redAssist_scaled
redChampionDamageDealt
redChampionDamageDealt_scaled
redAvgLevel
redAvgLevel_scaled
redTotalGold
redTotalGold_scaled
redJungleMinionKills
redJungleMinionKills_scaled
redTotalMinionKills
redTotalMinionKills_scaled
redKillingSpree
redKillingSpree_scaled
redTotalLevel
redTotalLevel_scaled
redTotalHeal
redTotalHeal_scaled
redObjectDamageDealt
redObjectDamageDealt_scaled

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

```
corr_cols_1 = scale_cols + bul_cols
corr_cols_1
```

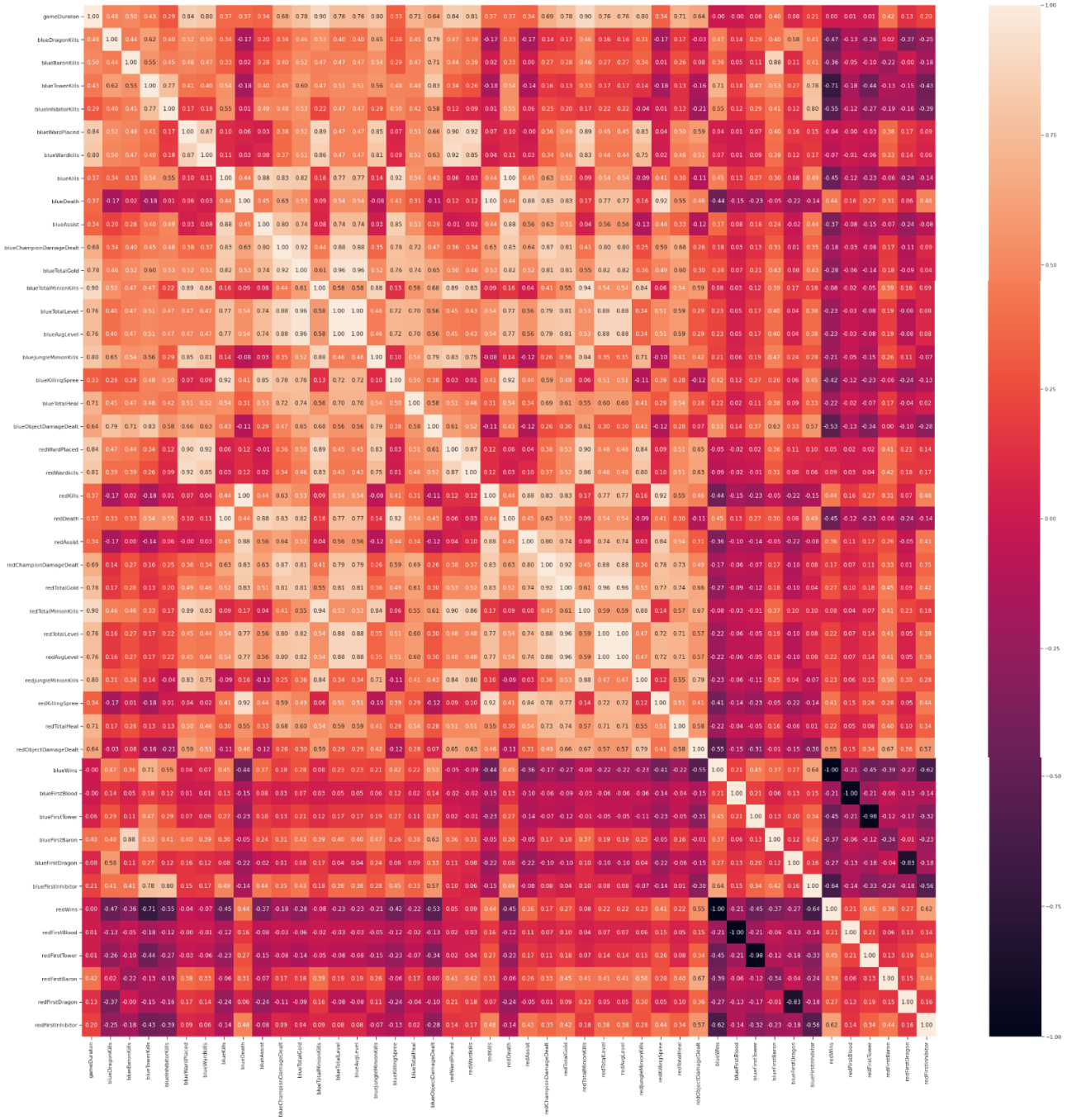
```
['gameDuraton',
 'blueDragonKills',
 'blueBaronKills',
 'blueTowerKills',
 'blueInhibitorKills',
 'blueWardPlaced',
 'blueWardkills',
 'blueKills',
 'blueDeath',
 'blueAssist',
 'blueChampionDamageDealt',
 'blueTotalGold',
 'blueTotalMinionKills',
 'blueTotalLevel',
 'blueAvgLevel',
 'blueJungleMinionKills',
 'blueKillingSpree',
 'blueTotalHeal',
 'blueObjectDamageDealt',
 'redWardPlaced',
 'redWardkills',
 'redKills',
 'redDeath',
 'redAssist',
 'redChampionDamageDealt',
 'redTotalGold',
 'redTotalMinionKills',
 'redTotalLevel',
 'redAvgLevel',
 'redJungleMinionKills',
 'redKillingSpree',
 'redTotalHeal',
 'redObjectDamageDealt',
 'blueWins',
 'blueFirstBlood',
 'blueFirstTower',
 'blueFirstBaron',
 'blueFirstDragon',
 'blueFirstInhibitor',
 'redWins',
 'redFirstBlood',
 'redFirstTower',
 'redFirstBaron',
 'redFirstDragon',
 'redFirstInhibitor']
```

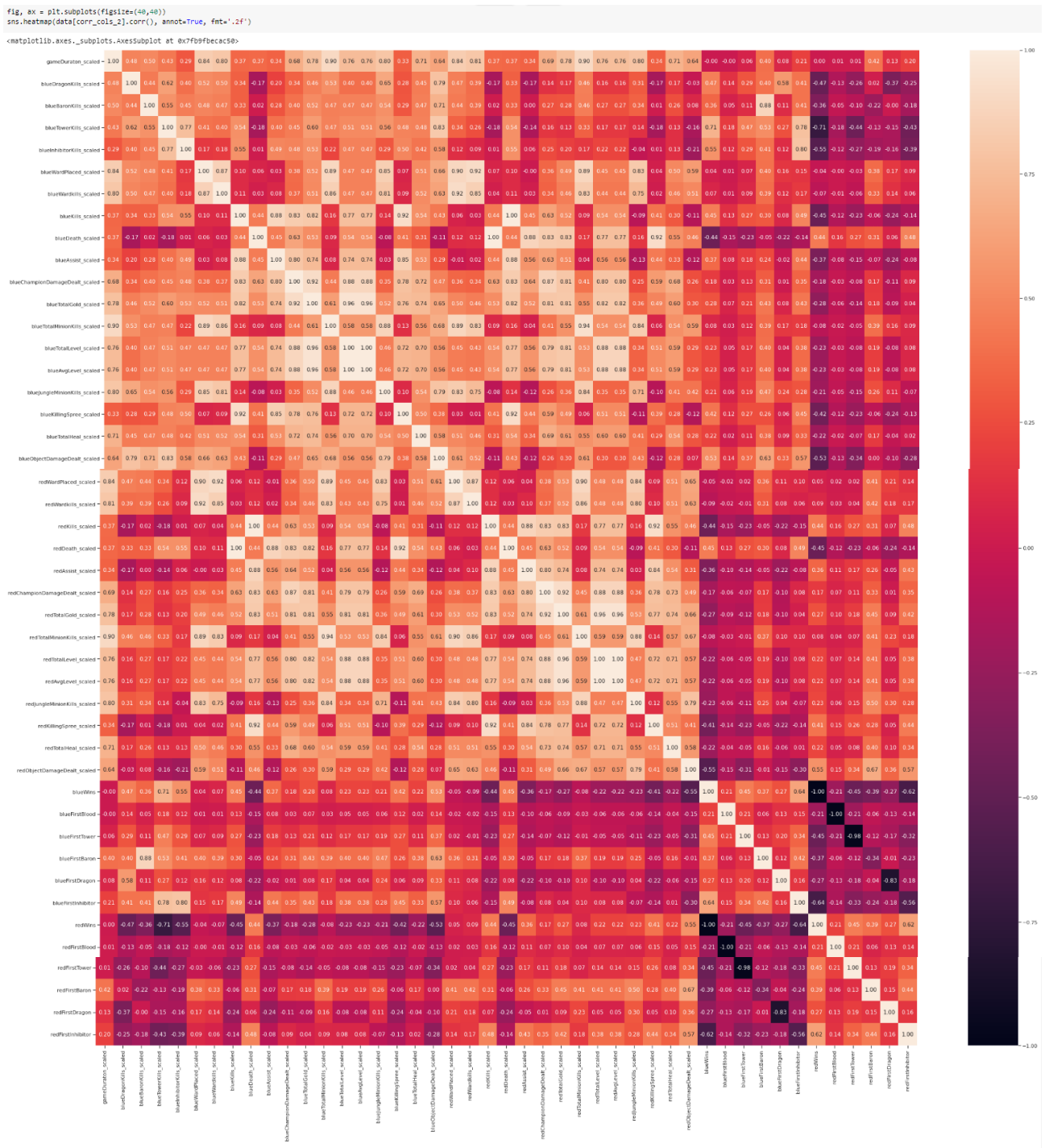
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + bul_cols
corr_cols_2
```

```
['gameDuraton_scaled',
 'blueDragonKills_scaled',
 'blueBaronKills_scaled',
 'blueTowerKills_scaled',
 'blueInhibitorKills_scaled',
 'blueWardPlaced_scaled',
 'blueWardkills_scaled',
 'blueKills_scaled',
 'blueDeath_scaled',
 'blueAssist_scaled',
 'blueChampionDamageDealt_scaled',
 'blueTotalGold_scaled',
 'blueTotalMinionKills_scaled',
 'blueTotalLevel_scaled',
 'blueAvgLevel_scaled',
 'blueJungleMinionKills_scaled',
 'blueKillingSpree_scaled',
 'blueTotalHeal_scaled',
 'blueObjectDamageDealt_scaled',
 'redWardPlaced_scaled',
 'redWardkills_scaled',
 'redKills_scaled',
 'redDeath_scaled',
 'redAssist_scaled',
 'redChampionDamageDealt_scaled',
 'redTotalGold_scaled',
 'redTotalMinionKills_scaled',
 'redTotalLevel_scaled',
 'redAvgLevel_scaled',
 'redJungleMinionKills_scaled',
 'redKillingSpree_scaled',
 'redTotalHeal_scaled',
 'redObjectDamageDealt_scaled',
 'blueWins',
 'blueFirstBlood',
 'blueFirstTower',
 'blueFirstBaron',
 'blueFirstDragon',
 'blueFirstInhibitor',
 'redWins',
 'redFirstBlood',
 'redFirstTower',
 'redFirstBaron',
 'redFirstDragon',
 'redFirstInhibitor']
```

```
fig, ax = plt.subplots(figsize=(40,40))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9fe0e27f0>
```





На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабированных данных совпадают.

2. Целевой признак классификации "blueWins" наиболее сильно коррелирует с признаками:
«blueTowerKills» (Синяя команда уничтожила вражеских строений) - 0.71
«blueInhibitorKills» (Синяя команда уничтожила кристаллов усиления) - 0.55
«blueKills» (Синяя команда уничтожила вражеских чемпионов) - 0.45
«blueAssist» (Игрок синей команды помог уничтожить вражеского чемпиона) - 0.37
«blueKillingSpree» (Игрок синей команды совершил массовое уничтожение) - 0.42

«blueObjectDamageDealt» (Синяя команда нанесла урона игровым объектам) - 0.53
«redDeath» (Игроки синей команды были уничтожены) - 0.45
«blueFirstTower» (Синяя команда первой сломала башню) - 0.45
«blueFirstBaron» (Синяя команда получила бонус Барона Нашора первой) - 0.37
«blueFirstInhibitor» (Синяя команда уничтожила кристалл усиления первой) - 0.64

Эти признаки обязательно следует оставить в модели классификации.

Достаточно 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:

1. Метрика precision:

$$precision = \frac{TP}{TP+FP}$$

Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Используется функция *precision_score*.

2. Метрика recall (полнота):

$$recall = \frac{TP}{TP+FN}$$

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция *recall_score*.

3. Метрика F1-мера

Для того, чтобы объединить precision и recall в единую метрику используется F_β-мера, которая вычисляется как среднее гармоническое от precision и recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{precision + recall}$$

где β определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при β=1:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Для вычисления используется функция *f1_score*.

4. Метрика ROC AUC

Используется для оценки качества бинарной классификации. Основана на вычислении следующих характеристик:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

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.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция *roc_auc_score*.

Сохранение и визуализация метрик

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

```
[40] 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, используемый для решения задачи бинарной классификации.
Метод выдает вероятность принадлежности объекта к нулевому/единичному классам.
Используется класс *LogisticRegression*.

- Машина опорных векторов

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

Для решения задачи классификации используем класс:
SVC - основной классификатор на основе SVM. Поддерживает различные ядра.

- Решающее дерево

Для текущего выбранного признака (колонки) из N признаков построить все варианты ветвления по значениям (для категориальных признаков) или по диапазонам значений (для числовых признаков).
Если подвыборке соответствует единственное значение целевого признака, то в дерево добавляется терминальный лист, который соответствует предсказанному значению.
Если в подвыборке больше одного значения целевого признака, то предыдущие пункты выполняются рекурсивно для подвыборки.

Для решения задачи классификации используется класс *DecisionTreeClassifier*.

- Случайный лес (ансамблевая)

Случайный лес можно рассматривать как алгоритмом бэггинга над решающими деревьями.

Но при этом каждое решающее дерево строится на случайно выбранном подмножестве признаков. Эта особенность называется "feature bagging" и основана на методе случайных подпространств.

Случайный лес для задачи классификации реализуется в scikit-learn с помощью класса *RandomForestClassifier*.

Задание параметра n_jobs=-1 распараллеливает алгоритм на максимально возможное количество процессоров.

- Градиентный бустинг (ансамблевая)

В отличие от методов бэггинга и случайного леса, которые ориентированы прежде всего на минимизизацию дисперсии (Variance), методы бустинга ориентированы прежде всего на минимизацию смещения (Bias) и, отчасти, на минимизизацию дисперсии.

Исторически первым полноценным алгоритмом бустинга считается алгоритм AdaBoost.

AdaBoost реализуется в scikit-learn с помощью класса *AdaBoostClassifier* для задач классификации.

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

Имеем масштабированные данные:

```
target = data['blueWins']
data = data.drop('blueWins', axis = 1)
```

```
data.columns
```

```
Index(['gameId', 'gameDuraton', 'blueFirstBlood', 'blueFirstTower',
       'blueFirstBaron', 'blueFirstDragon', 'blueFirstInhibitor',
       'blueDragonKills', 'blueBaronKills', 'blueTowerKills',
       'blueInhibitorKills', 'blueWardPlaced', 'blueWardkills', 'blueKills',
       'blueDeath', 'blueAssist', 'blueChampionDamageDealt', 'blueTotalGold',
       'blueTotalMinionKills', 'blueTotalLevel', 'blueAvgLevel',
       'blueJungleMinionKills', 'blueKillingSpree', 'blueTotalHeal',
       'blueObjectDamageDealt', 'redWins', 'redFirstBlood', 'redFirstTower',
       'redFirstBaron', 'redFirstDragon', 'redFirstInhibitor',
       'redDragonKills', 'redBaronKills', 'redTowerKills', 'redInhibitorKills',
       'redWardPlaced', 'redWardkills', 'redKills', 'redDeath', 'redAssist',
       'redChampionDamageDealt', 'redTotalGold', 'redTotalMinionKills',
       'redTotalLevel', 'redAvgLevel', 'redJungleMinionKills',
       'redKillingSpree', 'redTotalHeal', 'redObjectDamageDealt',
       'gameDuraton_scaled', 'blueDragonKills_scaled', 'blueBaronKills_scaled',
       'blueTowerKills_scaled', 'blueInhibitorKills_scaled',
       'blueWardPlaced_scaled', 'blueWardkills_scaled', 'blueKills_scaled',
       'blueDeath_scaled', 'blueAssist_scaled',
       'blueChampionDamageDealt_scaled', 'blueTotalGold_scaled',
       'blueTotalMinionKills_scaled', 'blueTotalLevel_scaled',
       'blueAvgLevel_scaled', 'blueJungleMinionKills_scaled',
       'blueKillingSpree_scaled', 'blueTotalHeal_scaled',
       'blueObjectDamageDealt_scaled', 'redWardPlaced_scaled',
       'redWardkills_scaled', 'redKills_scaled', 'redDeath_scaled',
       'redAssist_scaled', 'redChampionDamageDealt_scaled',
       'redTotalGold_scaled', 'redTotalMinionKills_scaled',
       'redTotalLevel_scaled', 'redAvgLevel_scaled',
       'redJungleMinionKills_scaled', 'redKillingSpree_scaled',
       'redTotalHeal_scaled', 'redObjectDamageDealt_scaled'],
      dtype='object')
```

На основе масштабированных данных выделим обучающую и тестовую выборки:

```
# Признаки для задачи классификации
task_clas_cols = ['blueTowerKills_scaled','blueInhibitorKills_scaled','blueKills_scaled', 'blueAssist_scaled',
                  'blueKillingSpree_scaled', 'blueObjectDamageDealt_scaled', 'redDeath_scaled',
                   'blueFirstTower', 'blueFirstBaron', 'blueFirstInhibitor']
```

```
# Выборки для задачи классификации
clas_data = data[task_clas_cols]
```

```
clas_data.head()
```

	blueTowerKills_scaled	blueInhibitorKills_scaled	blueKills_scaled	blueAssist_scaled	blueKillingSpree_scaled	b
0	0.000000	0.000000	0.151515	0.092437	0.137931	
1	0.363636	0.000000	0.191919	0.130252	0.103448	
2	0.000000	0.000000	0.050505	0.033613	0.000000	
3	0.181818	0.000000	0.262626	0.189076	0.103448	
4	1.000000	0.333333	0.272727	0.197479	0.172414	

```
#деление на тестовую и обучающую выборку
clas_X_train, clas_X_test, clas_Y_train, clas_Y_test = train_test_split(
    clas_data, target, test_size=0.2, random_state=1)
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

```
((21523, 10), (5381, 10), (21523,), (5381,))
```

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.**

```
from sklearn.linear_model import LogisticRegression
# Модели
clas_models = {'LogR': LogisticRegression(),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

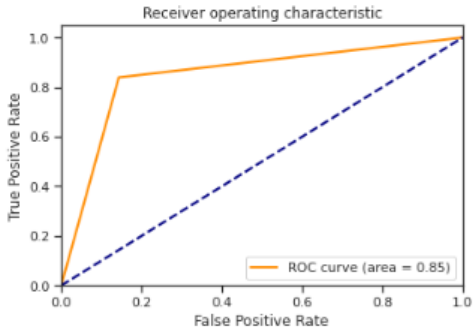
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

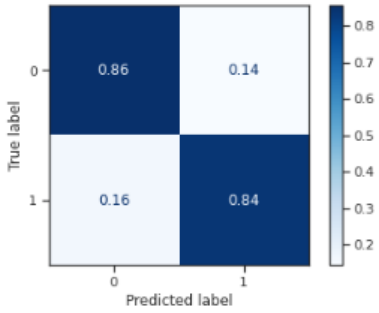
```
[73] def clas_train_model(model_name, model, clasMetricLogger):
         model.fit(clas_X_train, clas_Y_train)
         Y_pred = model.predict(clas_X_test)
         precision = precision_score(clas_Y_test.values, Y_pred)
         recall = recall_score(clas_Y_test.values, Y_pred)
         f1 = f1_score(clas_Y_test.values, Y_pred)
         roc_auc = roc_auc_score(clas_Y_test.values, Y_pred)

         clasMetricLogger.add('precision', model_name, precision)
         clasMetricLogger.add('recall', model_name, recall)
         clasMetricLogger.add('f1', model_name, f1)
         clasMetricLogger.add('roc_auc', model_name, roc_auc)

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

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

```
[75] # 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()
```

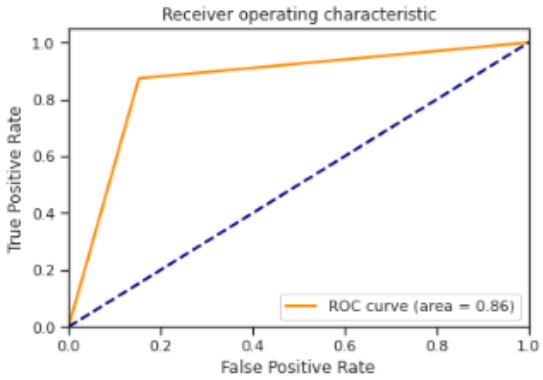
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

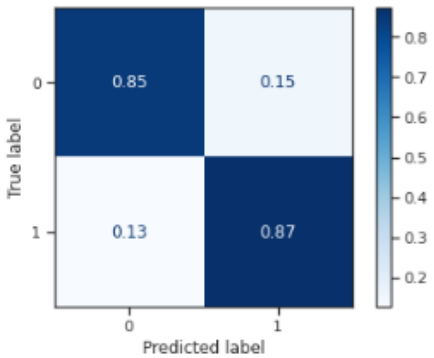
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None,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



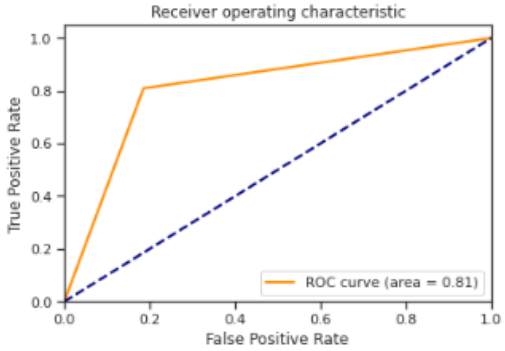


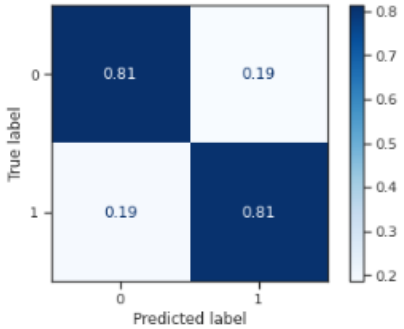

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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



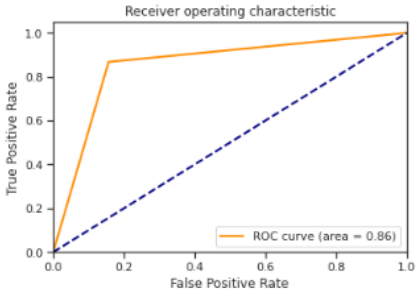


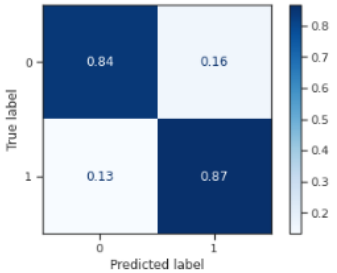
```
*****************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
             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



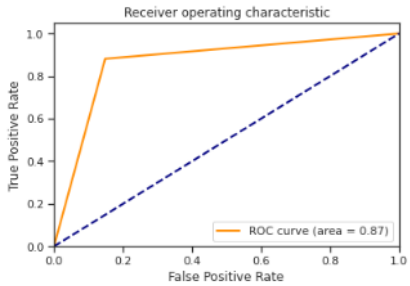


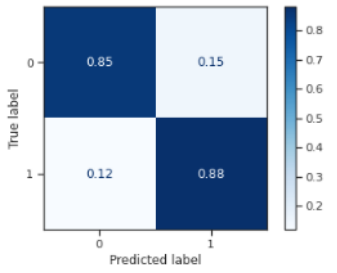

```
*****************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, 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.**

Кросс-валидация

```
#Кроссвалидация
scores_log = cross_val_score(LogisticRegression(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_log, np.mean(scores_log)
```

```
(array([0.85272254, 0.85057151]), 0.8516470252512649)
```

```
scores_svc = cross_val_score(SVC(gamma='auto'),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_svc, np.mean(scores_svc)
```

```
(array([0.84556774, 0.84499582]), 0.8452817782855525)
```

```
scores_tree = cross_val_score(DecisionTreeClassifier(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_tree, np.mean(scores_tree)
```

```
(array([0.81211671, 0.81377195]), 0.8129443306055695)
```

```
scores_rand_tree = cross_val_score(RandomForestClassifier(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_rand_tree, np.mean(scores_rand_tree)
```

```
(array([0.85922691, 0.85763405]), 0.8584304791882958)
```

```
scores_boost = cross_val_score(GradientBoostingClassifier(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_boost, np.mean(scores_boost)
```

```
(array([0.8653596 , 0.86125825]), 0.8633089229812012)
```

Подбор гиперпараметров:

```
parameters = {'gamma':[24, 23, 22]}
clf_gs_svc = GridSearchCV(SVC(), parameters, cv=3, scoring='accuracy')
clf_gs_svc.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=3, 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={'gamma': [24, 23, 22]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
# Лучшая модель
clf_gs_svc.best_estimator_
```

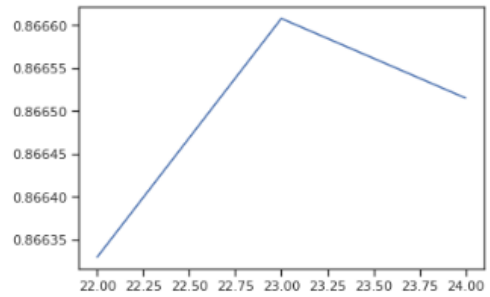
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=23, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
# Лучшее значение параметров
clf_gs_svc.best_params_
```

```
{'gamma': 23}
```

```
# Изменение качества на тестовой выборке в зависимости от параметра
n_range = np.array([24, 23, 22])
plt.plot(n_range, clf_gs_svc.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7fb9fabb0860>]
```




```
parameters = {'max_depth':[10,9,8,7,6], 'min_samples_split':[15,10,9,8,7,6,5,4,3]}
clf_gs_decision_tree = GridSearchCV(DecisionTreeClassifier(), parameters, cv=3, scoring='accuracy')
clf_gs_decision_tree.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=3, 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': [10, 9, 8, 7, 6],
                         'min_samples_split': [15, 10, 9, 8, 7, 6, 5, 4, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
# Лучшая модель
clf_gs_decision_tree.best_estimator_
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=7, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=9,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# Лучшее значение параметров
clf_gs_decision_tree.best_params_
```

```
{'max_depth': 7, 'min_samples_split': 9}
```

```
parameters_random_forest = {'n_estimators':[ 19, 20, 21],
                            'max_depth':[9, 10, 11],
                            'random_state':[8,  9, 10]}
best_random_forest = GridSearchCV(RandomForestClassifier(), parameters_random_forest, cv=3, scoring='accuracy')
best_random_forest.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=3, 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_jobs=None,
             param_grid={'max_depth': [9, 10, 11], 'n_estimators': [19, 20, 21],
                         'random_state': [8, 9, 10]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
# Лучшая модель
best_random_forest.best_estimator_
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, 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=20,
                       n_jobs=None, oob_score=False, random_state=9, verbose=0,
                       warm_start=False)
```

```
best_random_forest.best_params_
```

```
{'max_depth': 10, 'n_estimators': 20, 'random_state': 9}
```

```
parameters_gradient_boosting = {'n_estimators':[8, 10, 12],
                                'max_depth':[5, 7, 9]}
best_gradient_boosting = GridSearchCV(GradientBoostingClassifier(), parameters_gradient_boosting, cv=3, scoring='accuracy')
best_gradient_boosting.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=3, 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_iter_no_change=None,
                                                  presort='deprecated',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [5, 7, 9], 'n_estimators': [8, 10, 12]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
# Лучшая модель
best_gradient_boosting.best_estimator_
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=7,
                           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=10,
                           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)
```

```
best_gradient_boosting.best_params_
```

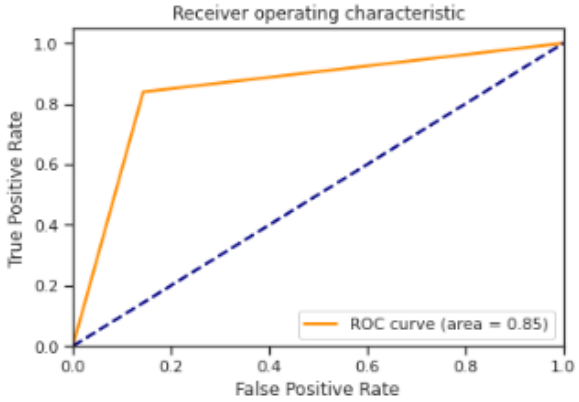
```
{'max_depth': 7, 'n_estimators': 10}
```

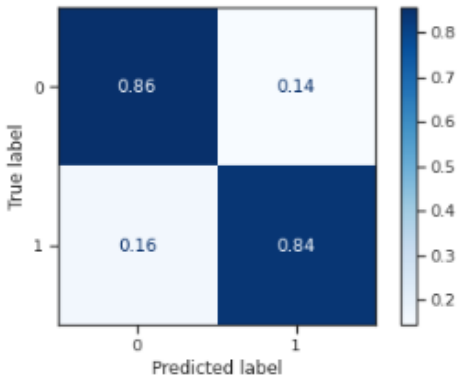
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

```
# Новые модели с подобранными гиперпараматерами
clas_models_grid = {'LogR':LogisticRegression(),
                    'SVC':clf_gs_svm_svc.best_estimator_,
                    'Tree':clf_gs_decision_tree.best_estimator_,
                    'RF':best_random_forest.best_estimator_,
                    'GB':best_gradient_boosting.best_estimator_}
```

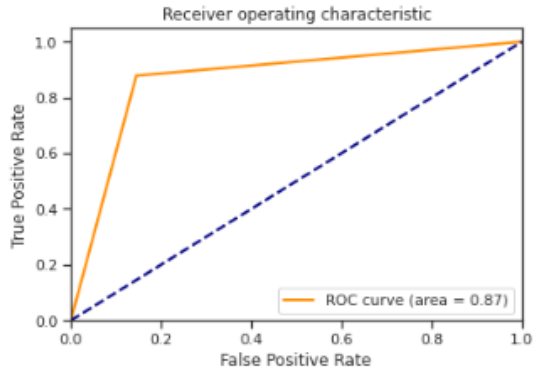
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

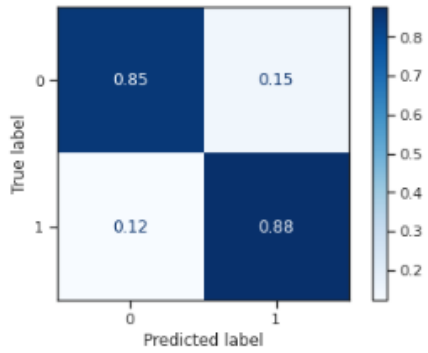
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None,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



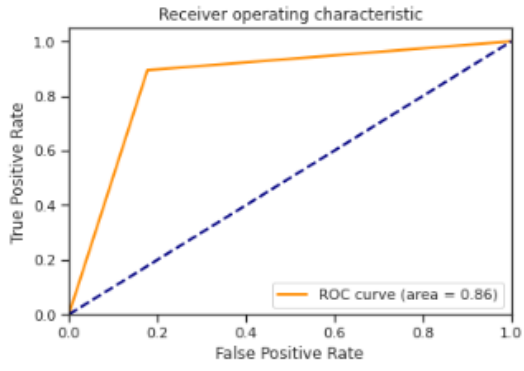


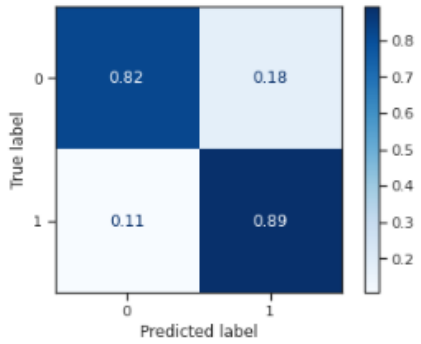
```
*****************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=10, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
*****************************************************
```



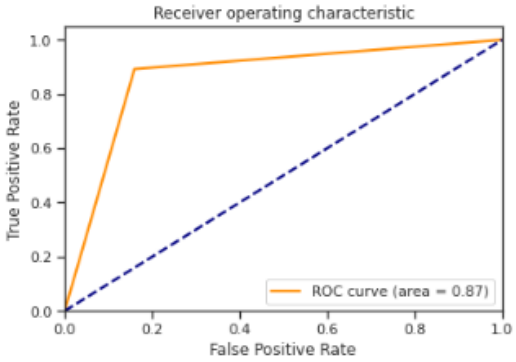


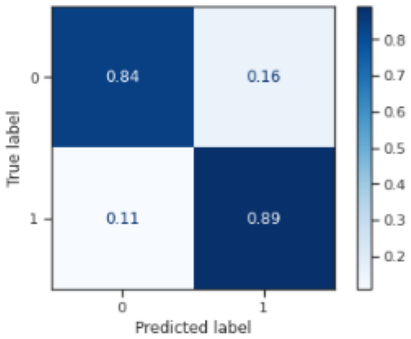

```
*****************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=7, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=9,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
*****************************************************
```



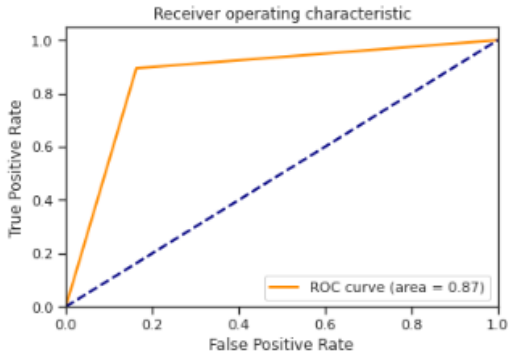


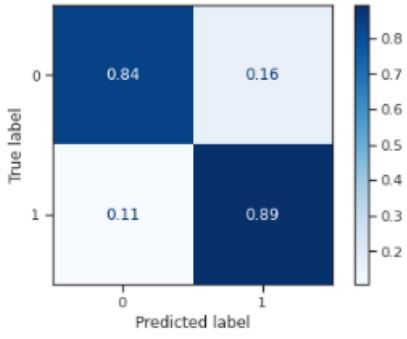
```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=20,
                       n_jobs=None, oob_score=False, random_state=9, verbose=0,
                       warm_start=False)
**************************************************
```






```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=7,
                           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=10,
                           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)
**************************************************
```

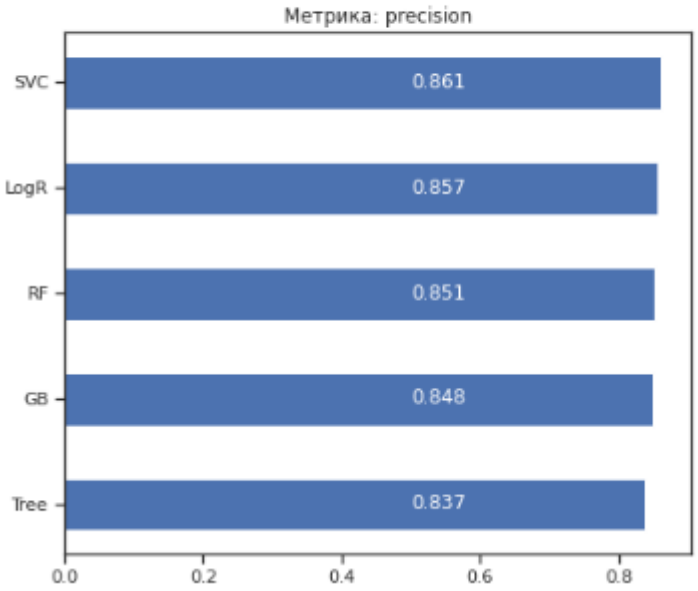




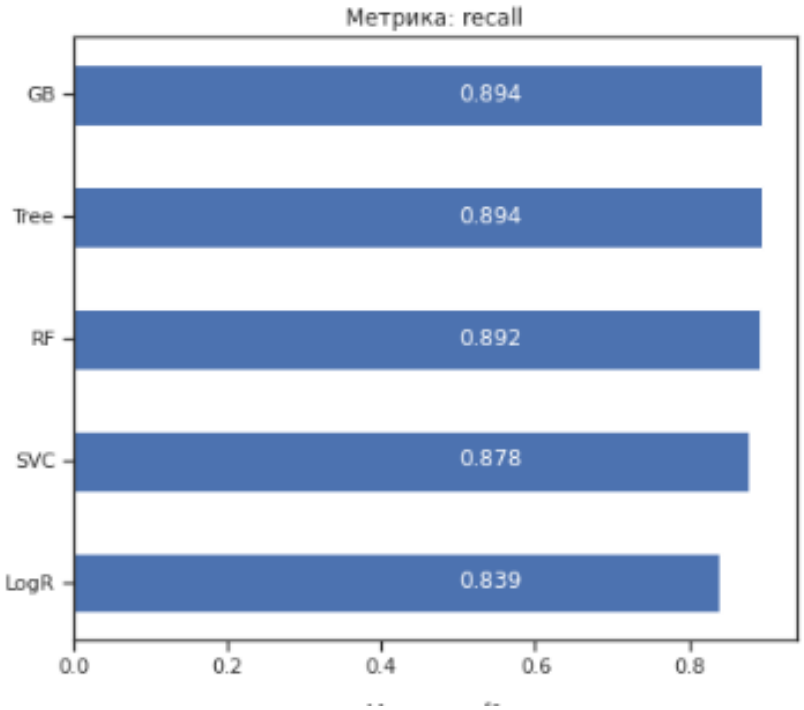
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания.

Лучшая модель по метрике precision: Логическая регрессия

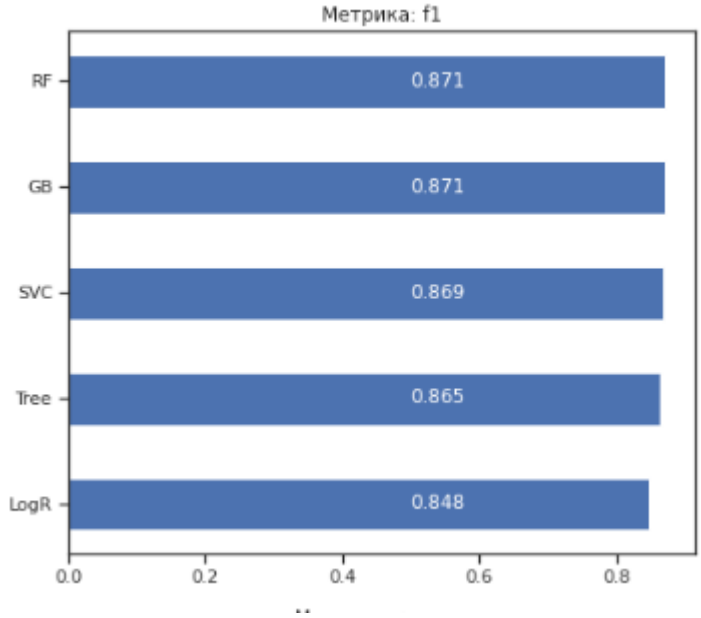
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



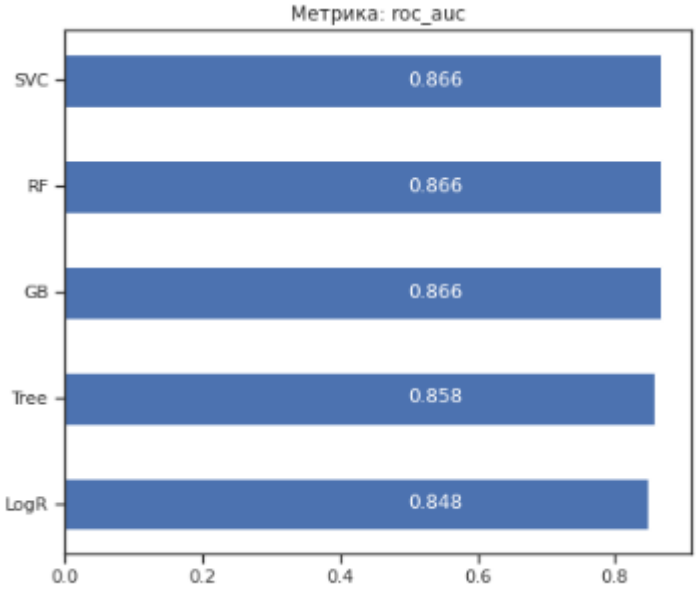
Лучшая модель по метрике recall: Случайный лес



Лучшая модель по метрике f1: Метод опорных векторов



Лучшая модель по метрике ROC AUC: Случайный лес



Вывод: на основании четырех метрик, лучшей оказались построенная на методе опорных векторов модель и модель случайного леса.

Заключение

Из всех рассмотренных алгоритмов: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest" для модели определения победителя матча лучших игроков по дисциплине League of Legends наиболее эффективными оказались алгоритм случайного леса и метод опорных векторов. Как известно Random forest борется с переобучением модели, следовательно можно сделать вывод о том, что датасет является довольно разрозненным, поэтому другие методы могли привести к возникновению проблемы переобучения, а "Random forest" успешно обошёл эту проблему.

Список литературы

1. Репозиторий курса "Технологии машинного обучения", бакалавриат, 6 семестр. Лекции по теории машинного обучения. Ю.Е. Гапанюк [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_TMO (дата обращения: 31.05.2020)
2. League Of Legends High elo Ranked Games (2020) [Электронный ресурс]. – Электрон. дан. - URL: https://www.kaggle.com/gyejr95/league-of-legends-challenger-ranked-games2020?select=Challenger_Ranked_Games.csv (дата обращения: 04.06.2020)
3. Машинное обучение (часть 1). А.М.Миронов [Электронный ресурс]. – Электрон. дан. - URL: http://www.intsys.msu.ru/staff/mironov/machine_learning_vol1.pdf (дата обращения: 31.05.2020)
4. Scikit learn[Электронный ресурс]. – Электрон. дан. - URL: https://scikit-learn.org/stable/index.html (дата обращения: 31.05.2020)