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 ИУ_____

КАФЕДРА ИУ5_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Решение задачи машинного обучения

Студенты группы ИУ5-61Б _____ _____ Матиенко А.П.
(Группа) (Подпись, дата) (И.О.Фамилия)

_____ _____ Молева А.А.
(Подпись, дата) (И.О.Фамилия)

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

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

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»_____

Студенты группы ИУ5-61Б_____

___Матиенко Андрей Павлович, Молева Анастасия Алексеевна_____
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____
_____

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 __31___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « 12 » февраля 2020 г.

Руководитель курсового проекта _____ _____Матиенко А.П.
(Подпись, дата) (И.О.Фамилия)
_____ _____Молева А.А.
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Гапанюк Ю.Е.
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

Задание установленного образца

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

- 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.
- Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
- Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
- 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.
- Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
- Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
- Формирование обучающей и тестовой выборок на основе исходного набора данных.
- Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производятся

обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

- 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.

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

- 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.

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.

Основная часть. Описание постановки задачи и последовательности действий по решению поставленной задачи

Описание набора данных

В данной работе для исследований был выбран следующий датасет:

https://www.kaggle.com/c/mlcourse-dota2-win-prediction/data

Задача: предсказание победы по данным игры

По статистике в игре предсказать, какая из команд победит: Radiant или Dire?
Файл *train.csv* содержит `39675` строк и `245` столбцов. В него включены:

Входные переменные (на основе физико-химических тестов):

- `r1_hero`: герой игрока (расшифровка в dictionaries/heroes.csv)

- `r1_level`: максимальный достигнутый уровень героя (за первые 5 игровых минут)

- `r1_xp`: максимальный полученный опыт

- `r1_gold`: достигнутая ценность героя

- `r1_lh`: число убитых юнитов

- `r1_kills`: число убитых игроков

- `r1_deaths`: число смертей героя

- `r1_items`: число купленных предметов

- Признаки события "первая кровь" (first blood). Если событие "первая кровь" не успело произойти за первые 5 минут, то признаки принимают пропущенное значение

- `first_blood_time`: игровое время первой крови

- `first_blood_team`: команда, совершившая первую кровь (0 — Radiant, 1 — Dire)

- `first_blood_player1`: игрок, причастный к событию

- `first_blood_player2`: второй игрок, причастный к событию

- Признаки для каждой команды (префиксы `radiant_` и `dire_`)

- `radiant_bottle_time`: время первого приобретения командой предмета "bottle"

- `radiant_courier_time`: время приобретения предмета "courier"

- *`radiant_flying_courier_time`: время приобретения предмета "flying_courier"*
- *`radiant_tpscroll_count`: число предметов "tpscroll" за первые 5 минут*
- *`radiant_boots_count`: число предметов "boots"*
- *`radiant_ward_observer_count`: число предметов "ward_observer"*
- *`radiant_ward_sentry_count`: число предметов "ward_sentry"*
- *`radiant_first_ward_time`: время установки командой первого "наблюдателя", т.е. предмета, который позволяет видеть часть игрового поля*

- Итог матча (данные поля отсутствуют в тестовой выборке, поскольку содержат информацию, выходящую за пределы первых 5 минут матча)

- *`duration`: длительность*
- *`radiant_win`: 1, если победила команда Radiant, 0 — иначе*
- *Состояние башен и барраков к концу матча (см. описание полей набора данных)*
    - *`tower_status_radiant`*
    - *`tower_status_dire`*
    - *`barracks_status_radiant`*
    - *`barracks_status_dire`*

Выходная переменная (на основе отзывов потребителей):

Победа или проигрыш

Для данного набора данных мы будем решать задачу классификации – определение победы команды radiant.

Ход работы

Импортируем необходимые для работы библиотеки:

Библиотеки

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, roc_auc_score, precision_score, recall_score, f1_score, roc_curve
from sklearn.linear_model import Lasso, RidgeClassifier, SGDClassifier, LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold, StratifiedKFold, GridSearchCV, Random
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
from datetime import datetime
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
%matplotlib inline
```

Загрузка данных

In [2]:
```
train = pd.read_csv('datasets/train_features.csv', index_col='match_id_hash')
test = pd.read_csv('datasets/test_features.csv', index_col='match_id_hash')
target = pd.read_csv('datasets/train_targets.csv', index_col='match_id_hash')
```

In [3]:
```
print("Размер выборки Train:", train.shape)
print("Размер выборки Test:", test.shape)
print("Размер выборки target:", target.shape)
```

```
Размер выборки Train: (39675, 245)
Размер выборки Test: (10000, 245)
Размер выборки target: (39675, 5)
```

In [4]:
```
train.head()
```

Out[4]:

match_id_hash	game_time	game_mode	lobby_type	objectives_len	chat_len	r1_hero_id	r1_kills	r1_deaths	r1_assists	r1_denies	...	d
a400b8f29dece5f4d266f49f1ae2e98a	155	22	7	1	11	11	0	0	0	0	...	0
b9c57c450ce74a2af79c9ce96fac144d	658	4	0	3	10	15	7	2	0	7	...	0
6db558535151ea18ca70a6892197db41	21	23	0	0	0	101	0	0	0	0	...	0
46a0ddce8f7ed2a8d9bd5edcbb925682	576	22	7	1	4	14	1	0	3	1	...	8
b1b35ff97723d9b7ade1c9c3cf48f770	453	22	7	1	3	42	0	1	1	0	...	0

5 rows × 245 columns

Функция проверяет есть ли в выборке категориальные признаки

In [6]:
```
def cat_type_exist(data):
    counter = 0
    counter_int = 0
    counter_float = 0
    for col in train.columns:
        if train.dtypes[col] not in good_types:
            print(col, train.dtypes[col])
            counter += 1
        else:
            if train.dtypes[col]==np.dtype('int64'):
                counter_int += 1
            else:
                counter_float += 1
    if counter > 0:
        return "Dataset has {} no int types".format(counter)
    else:
        return "Dataset hasn't categorial types\n{}-{} признаков\n{}-{} признаков".format(np.dtype('int64'), counter_int,
                                                                                          np.dtype('float64'), counter_float)
```

In [7]:
```
good_types = [np.dtype('int64'), np.dtype('float64')]
print(cat_type_exist(train))
```

```
Dataset hasn't categorial types
int64-215 признаков
float64-30 признаков
```

В обучающей выборке нет нулевых элементов

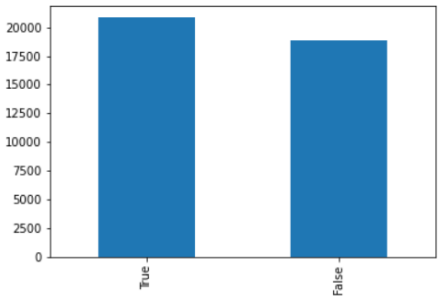
In [5]:
```
print(train.isnull().sum().any())
for col in train.columns:
    if train[col].isnull().any():
        print(col, train[col].isnull().sum())
```

```
False
```

Data explore

In [8]:
```
target['radiant_win'].value_counts().plot(kind='bar')
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x277d7ba9e80>
```



In [9]:
```
target['radiant_win'].value_counts()
```

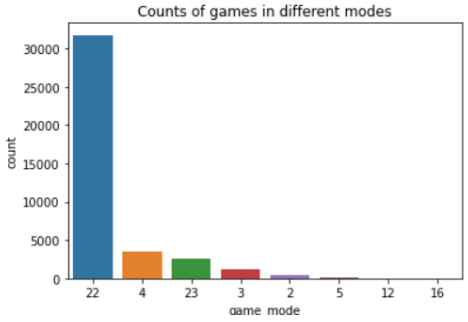
Out[9]:
```
True     20826
False    18849
```

Есть небольшой дисбаланс, но он невысокий, поэтому ничего не будем менять

Количество игр в разных режимах

In [10]:
```
sns.countplot(data=train, x='game_mode', order=train['game_mode'].value_counts().index)
plt.title('Counts of games in different modes')
```

Out[10]:
```
Text(0.5, 1.0, 'Counts of games in different modes')
```



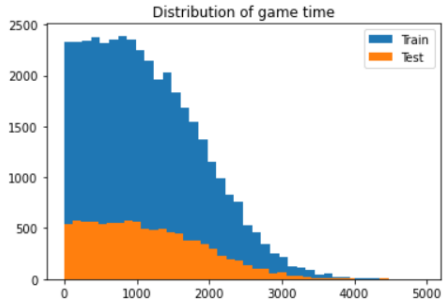
In [11]:
```
train_modes = train['game_mode'].value_counts().reset_index().rename(columns={'index': 'game_mode', 'game_mode': 'count'})
train_modes
```

Out[11]:

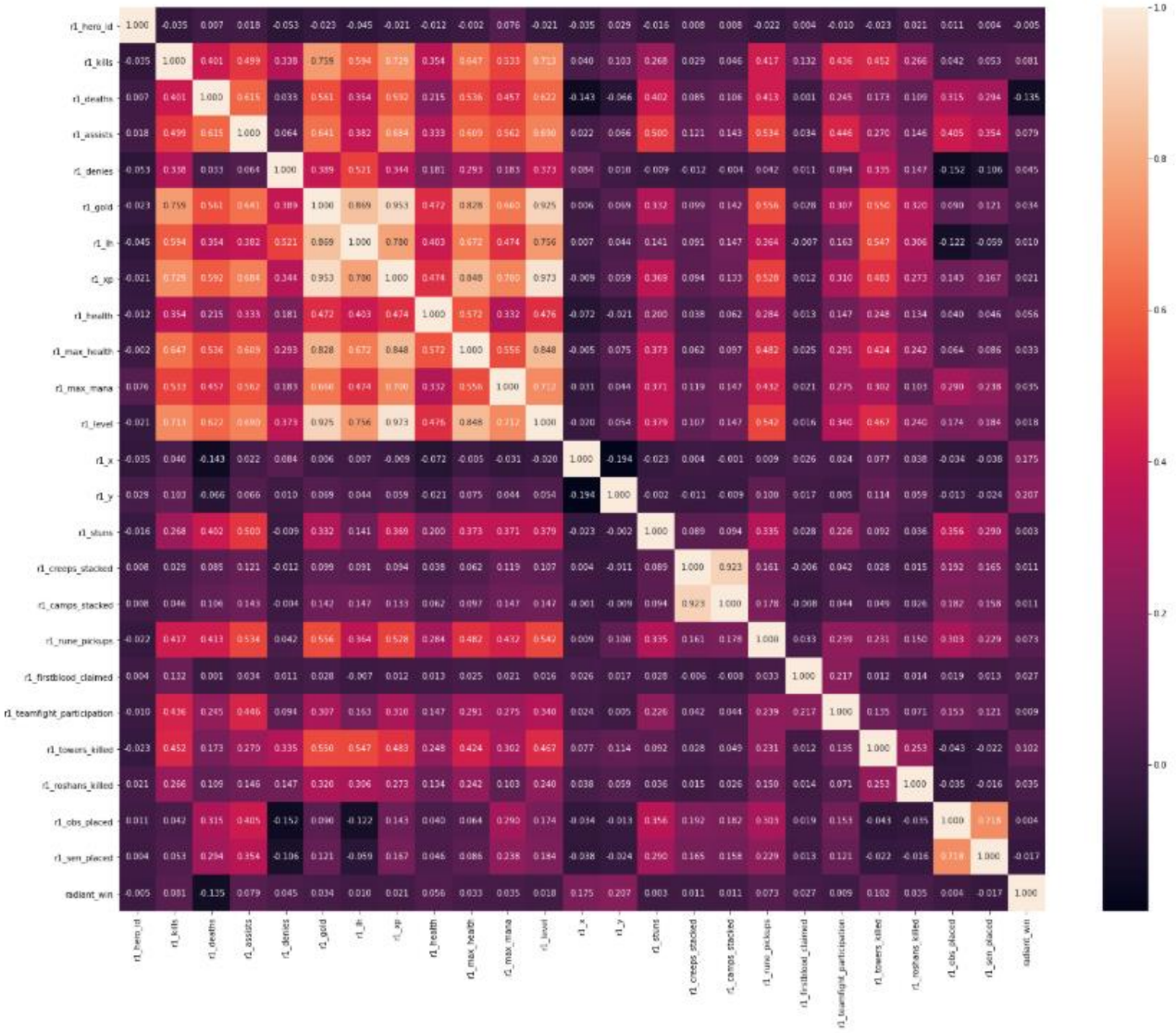
	game_mode	count
0	22	31762
1	4	3564
2	23	2546
3	3	1200
4	2	408
5	5	188
6	12	5
7	16	2

Распределение игрового времени в разных выборках

In [12]:
```
plt.hist(train['game_time'], bins=40, label='Train');
plt.hist(test['game_time'], bins=40, label='Test');
plt.title('Distribution of game time');
plt.legend();
```



Корреляционная матрица



Заменит True, False в целевой функции на 1, 0

In [14]:
```
target = target.radiant_win
target = target.map(lambda x: 1 if x==True else 0)
```

In [15]:
```
target
```

Out[15]:
```
match_id_hash
a400b8f29dece5f4d266f49f1ae2e98a    0
b9c57c450ce74a2af79c9ce96fac144d    1
6db558535151ea18ca70a6892197db41    1
46a0ddce8f7ed2a8d9bd5edcbb925682    1
b1b35ff97723d9b7ade1c9c3cf48f770    0
                                   ..
defd0caeed6ea83d7d5fbdec013fe7d1    0
bc7a87ed5f9c2bca55f9f7a93da0b0c5    1
e2ca68ac1a6847f4a37f6c9c8ee8695b    0
47ad6454ede66c1c78fdaa9391dfc556    1
9928dfde50efcbdb2055da23dcdbc101    1
Name: radiant_win, Length: 39675, dtype: int64
```

Измеряем accuracy, precision, recall, f1 и AUC-ROC

```
def ft_scores(clf, X_train, y_train, X_test, y_test, predict_proba=False, scaled=False):

    info = []
    if scaled == True:
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print('\tAccuracy:', accuracy)
    print('\tPrecision:', precision)
    print('\tRecall:', recall)
    print('\tF1:', f1)
    if predict_proba == False:
        auc_roc = roc_auc_score(y_test, y_pred)
        print('\tAUC-ROC:', auc_roc)

    if predict_proba == True:
        probability_predictions = clf.predict_proba(X_test)
```

```
        auc_roc = roc_auc_score(y_test, probability_predictions[:, 1])
        print('\tAUC-ROC:', auc_roc)

        fpr, tpr, _ = roc_curve(y_test, probability_predictions[:, 1])



        plt.plot(fpr, tpr, label=clf)
        plt.plot([0, 1], [0, 1], '--', color='grey')
        plt.xlim([-0.05, 1.05])
        plt.ylim([-0.05, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC curve')
        plt.legend(loc="lower right", bbox_to_anchor=(2, 0.95))
    info.append(accuracy)
    info.append(precision)
    info.append(recall)
    info.append(f1)
    info.append(auc_roc)
    return info
```

Алгоритмы

```
def print_models(X_train, X_test, y_train, y_test):

    sqrt_len_features = int(np.sqrt(len(X_train.columns)))

    knn = KNeighborsClassifier(n_jobs=-1)
    ridge = RidgeClassifier(random_state=42)
    sgd_clf = SGDClassifier(random_state=42)
    log_clf = LogisticRegression(max_iter=1e3,random_state=42)
    svc = LinearSVC(random_state=42)
    dec_tree = DecisionTreeClassifier(random_state=42)

    bag_clf = BaggingClassifier(DecisionTreeClassifier(max_features=sqrt_len_features), n_estimators=100,
                        n_jobs=-1,random_state=42)

    rnd_forest = RandomForestClassifier(n_estimators=100, max_features=sqrt_len_features,
random_state=42)
```

```
boost_clf = GradientBoostingClassifier(n_estimators=100, max_depth=3, random_state=42)

xgb_clf = xgb.XGBClassifier(random_state=42, n_estimators=100)


models = ['KNN', 'Ridge', 'SGDClassifier', 'LogisticRegression', 'LinearSVC',
          'DecisionTreeClassifier', 'BaggingClassifier',
         'RandomForestClassifier', 'GradientBoostingClassifier', 'XGBClassifier']
columns = ['Accuracy', 'Precision', 'Recall', 'F1', 'AUC-ROC']

info_big = []

list_clfs = [knn, ridge, sgd_clf, log_clf, svc, dec_tree, bag_clf, rnd_forest, boost_clf, xgb_clf]
for clf in list_clfs:
    print("============================================================")
    start_time = datetime.now()

    scaled = False

    if clf == svc:
        scaled=True

    predict_proba = False
    if clf in [knn, log_clf, dec_tree, bag_clf, rnd_forest, boost_clf]:
        predict_proba = True

    if clf != xgb_clf:
        print(clf, ":")
    else:
        print("XGBClassifier(n_estimators=100):")

    info = ft_scores(clf, X_train, X_test, y_train, y_test, predict_proba=predict_proba, scaled=scaled)
    info_big.append(info)
    print("Time", datetime.now() - start_time)
    print("============================================================\n")
```

return pd.DataFrame(info_big, columns=columns, index=models)

Отбор признаков

In [18]:
```
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(train, target, test_size=0.3, stratify=target)
```

In [19]:
```
lasso = Lasso(alpha=1)
lasso.fit(X_train_2, y_train_2)
```

Out[19]:
```
Lasso(alpha=1)
```

In [20]:
```
np.sum(np.abs(lasso.coef_) > 1e-6)
```

Out[20]:
```
45
```

In [21]:
```
print('Отбор признаков:')
for i, col in enumerate(X_train_2.columns):
    if np.abs(lasso.coef_[i]) > 1e-6:
        print(col)
```

```
Отбор признаков:
r1_gold
r1_xp
r1_health
r1_max_health
r1_max_mana
r2_gold
r2_xp
r2_health
r2_max_health
r3_gold
r3_xp
r3_health
r3_max_mana
r4_gold
r4_xp
r4_health
r4_max_mana
r5_gold
r5_xp
r5_health
r5_max_mana
d1_gold
d1_xp
d1_health
d1_max_mana
d2_gold
d2_xp
d2_health
d2_max_health
d2_max_mana
d3_gold
d3_xp
d3_health
d3_max_health
d3_max_mana
d4_gold
d4_xp
d4_health
d4_max_health
d4_max_mana
d5_gold
d5_xp
d5_health
d5_max_health
d5_max_mana
```

gold, xp, health, max_health, max_mana - важные признаки

Оставим только эти признаки для всей команды

In [22]:
```
train_copy = train.copy()
```

In [23]:
```
train_copy
```

Out[23]:

match_id_hash	game_time	game_mode	lobby_type	objectives_len	chat_len	r1_hero_id	r1_kills	r1_deaths	r1_assists	r1_denies	...
a400b8f29dece5f4d266f49f1ae2e98a	155	22	7	1	11	11	0	0	0	0	...
b9c57c450ce74a2af79c9ce96fac144d	658	4	0	3	10	15	7	2	0	7	...
6db558535151ea18ca70a6892197db41	21	23	0	0	0	101	0	0	0	0	...
46a0ddce8f7ed2a8d9bd5edcbb925682	576	22	7	1	4	14	1	0	3	1	...
b1b35ff97723d9b7ade1c9c3cf48f770	453	22	7	1	3	42	0	1	1	0	...
...	...	...	...	...	...	...	...	...	...	...	...
defd0caeed6ea83d7d5fbdec013fe7d1	1783	22	0	8	23	3	1	9	9	0	...
bc7a87ed5f9c2bca55f9f7a93da0b0c5	377	22	7	1	0	14	0	2	1	1	...
e2ca68ac1a6847f4a37f6c9c8ee8695b	643	22	7	1	23	63	1	4	0	2	...
47ad6454ede66c1c78fdaa9391dfc556	2405	22	7	12	4	22	3	8	14	7	...
9928dfde50efcbdb2055da23dcdbc101	1775	22	0	10	13	32	3	4	15	0	...

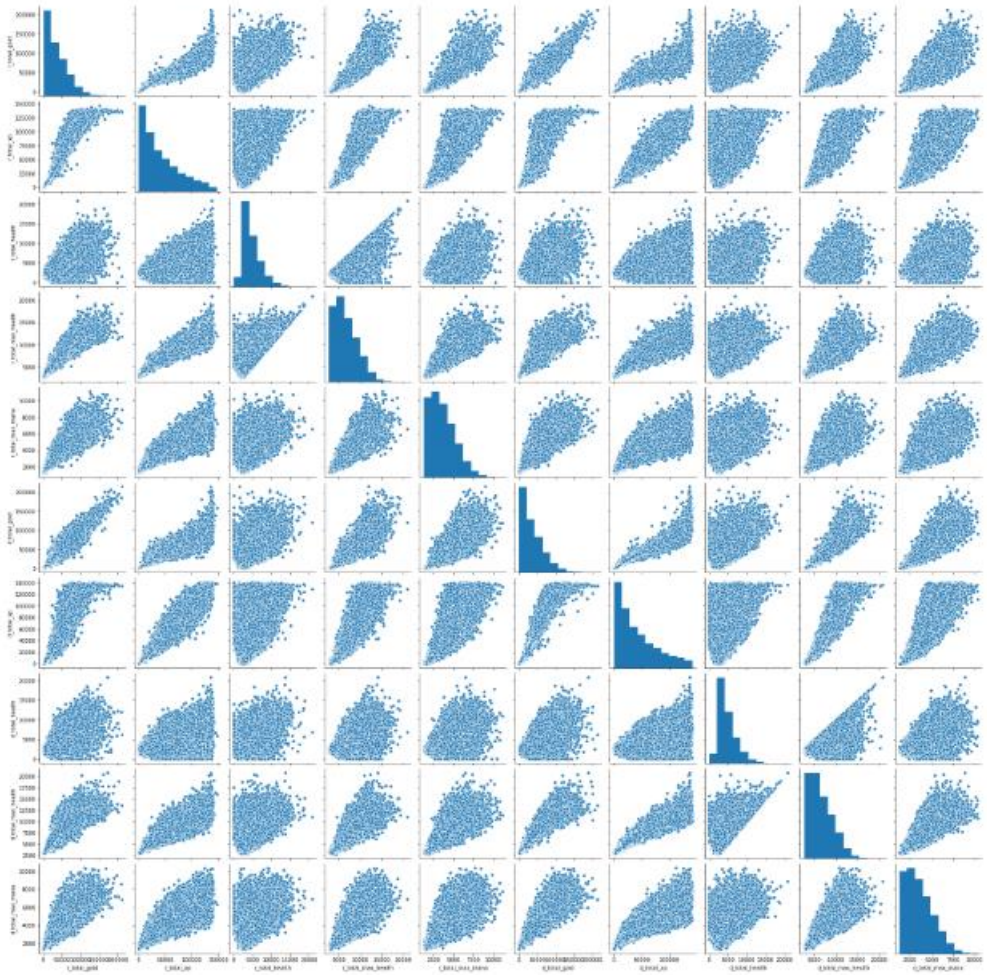
39675 rows × 245 columns

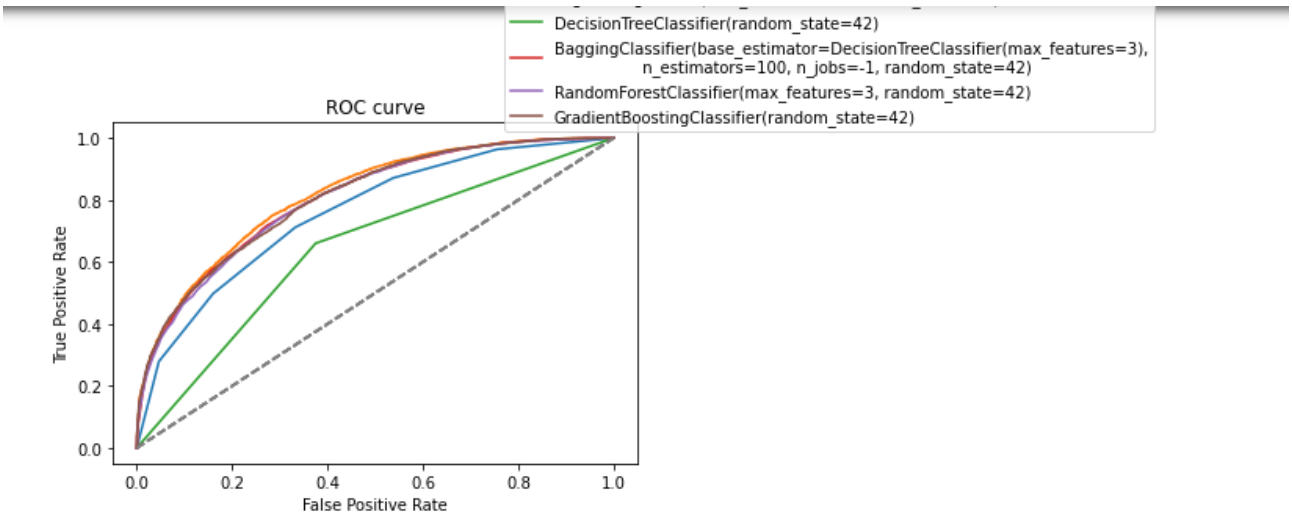
In [24]:
```
for c in ['gold', 'xp', 'health', 'max_health', 'max_mana']:
    r_columns = [f'r{i}_{c}' for i in range(1, 6)]
    d_columns = [f'd{i}_{c}' for i in range(1, 6)]

    train_copy['r_total_' + c] = train_copy[r_columns].sum(1)
    train_copy['d_total_' + c] = train_copy[d_columns].sum(1)
    train_copy['total_' + c + '_ratio'] = train_copy['r_total_' + c] / train_copy['d_total_' + c]
```

In [25]:
```
columns = ['r_total_gold', 'r_total_xp', 'r_total_health', 'r_total_max_health', 'r_total_max_mana',
           'd_total_gold', 'd_total_xp', 'd_total_health', 'd_total_max_health', 'd_total_max_mana']
train_copy[columns].head()
```

In [26]:
```
sns.pairplot(train_copy[columns])
```

Out[26]: <seaborn.axisgrid.PairGrid at 0x277d8e7fd68>

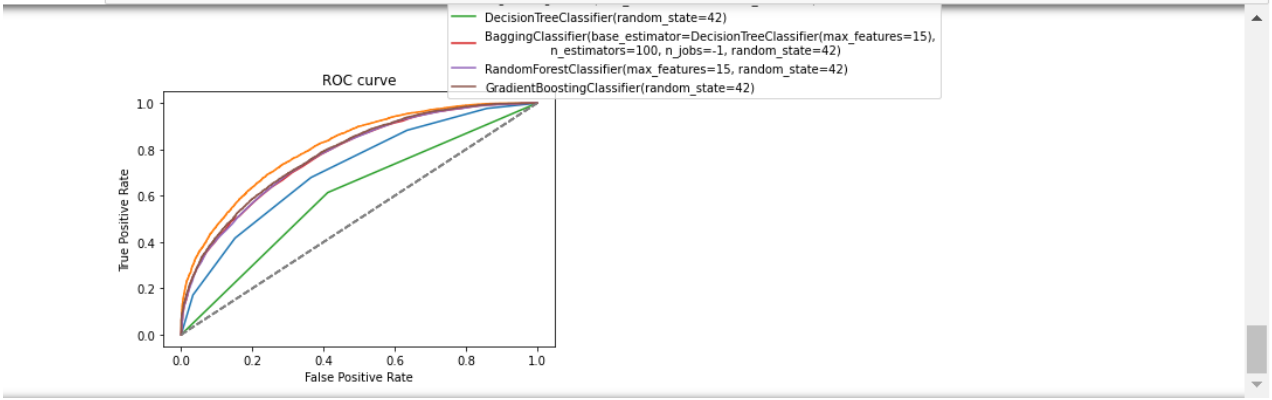




Посмотрим результат по первоначальной модели

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(train, target, test_size=0.3, stratify=target)
```

In [30]:
```
%%time
print_models(X_train, y_train, X_test, y_test)
```

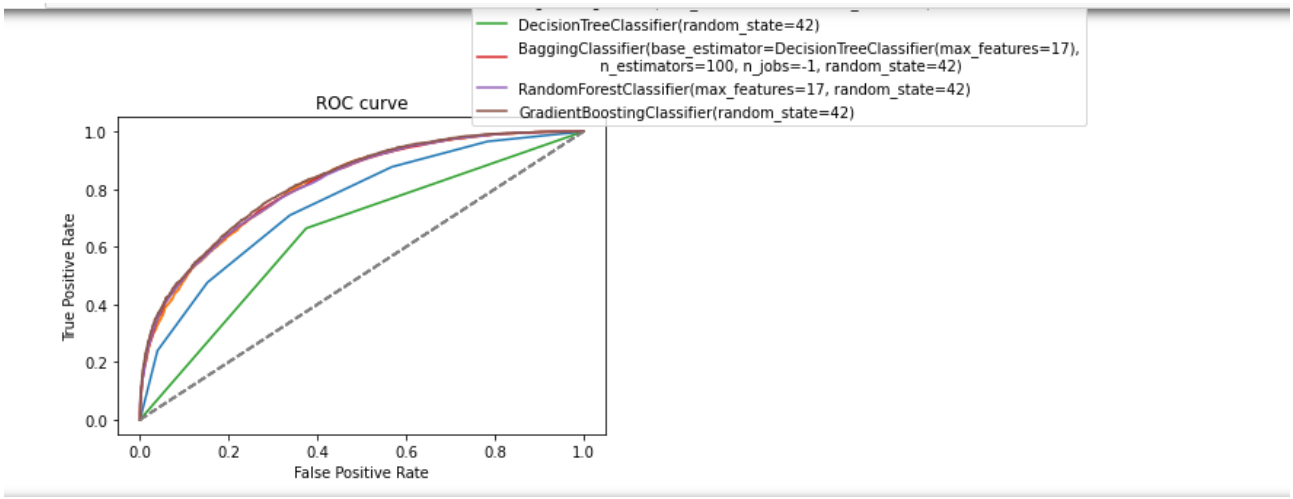


Добавим новых признаков: команды

In [31]:
```
for c in ['kills', 'deaths', 'assists', 'denies', 'gold', 'lh', 'xp',
          'health', 'max_health', 'max_mana', 'level', 'x', 'y', 'stuns',
          'creeps_stacked', 'camps_stacked', 'rune_pickups','firstblood_claimed',
          'teamfight_participation', 'towers_killed', 'roshans_killed', 'obs_placed', 'sen_placed']:
    r_columns = [f'r{i}_{c}' for i in range(1, 6)]
    d_columns = [f'd{i}_{c}' for i in range(1, 6)]

    train['r_total_' + c] = train[r_columns].sum(1)
    train['d_total_' + c] = train[d_columns].sum(1)
    train['total_' + c + '_ratio'] = train['r_total_' + c] / train['d_total_' + c]
```

In [32]:
```
train.shape
```

Out[32]:
```
(39675, 314)
```



Лучшая модель - GradientBoostingClassifier

Попробуем ее улучшить поиском по сетке

36]:
```
cv = KFold(n_splits=3, shuffle=True, random_state=42)
parameters = {
    "loss":["deviance"],
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "min_samples_split": np.linspace(0.1, 0.5, 12),
    "min_samples_leaf": np.linspace(0.1, 0.5, 12),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse",  "mae"],
    "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
    "n_estimators":[10]
}

grad_boost = GradientBoostingClassifier(random_state=42)
grid_search = RandomizedSearchCV(grad_boost, param_distributions=parameters, scoring='accuracy', cv=cv, n_jobs=-1)
```

37]:
```
%%time
grid_search.fit(train, target)
```

```
Wall time: 57 s
```

In [39]:
```
grid_search.best_score_
```

Out[39]:
```
0.6956521739130435
```

Попытаемся улучшить LogisticRegression

In [40]:
```
cv = KFold(n_splits=3, shuffle=True, random_state=42)
parameters=[{"C":np.logspace(-3,3,7)},
    {"penalty":["l1","l2", "elasticnet"]},
]

log_reg = LogisticRegression(random_state=42)
grid_search = GridSearchCV(log_reg, param_grid=parameters, scoring='accuracy', cv=cv, n_jobs=-1)
```

In [41]:
```
%%time
grid_search.fit(train, target)
```

```
Wall time: 27.3 s
```

Out[41]:
```
GridSearchCV(cv=KFold(n_splits=3, random_state=42, shuffle=True),
             estimator=LogisticRegression(random_state=42), n_jobs=-1,
             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])},
                         {'penalty': ['l1', 'l2', 'elasticnet']}],
             scoring='accuracy')
```

In [42]:
```
grid_search.best_estimator_
```

Out[42]:
```
LogisticRegression(C=0.1, random_state=42)
```

In [43]:
```
grid_search.best_score_
```

Out[43]:
```
0.7240075614366729
```

Выводы по проделанной работе

В ходе курсовой работы были закреплены полученные в течение курса знания и навыки. Для исследования использовались следующие модели: стохастический градиентный спуск, случайный лес, градиентный бустинг, метод ближайших соседей, метод опорных векторов (SVC). Для оценки качества использовались метрики: ROC-кривая и balanced_accuracy. Для наглядности были построены кривые обучения и валидации.
После подбора гиперпараметров наилучшую точность показал *градиентный бустинг*, при этом *случайный лес* без подбора гиперпараметров смог показать аналогичное качество (подбор параметров наоборот немного ухудшил качество модели).

Список использованных источников

1. Конспект лекций по дисциплине “Технологии машинного обучения”. 2020: https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_TMO
2. Документация scikit-learn: https://scikit-learn.org/stable/index.html
3. Метрики в задачах машинного обучения: https://habr.com/ru/company/ods/blog/328372/