PUBG 玩家排名分析预测

刘满楸 2016201085

Contents

【摘要】PUBG（绝地求生）游戏风靡全球，如何通过海量玩家数据对游戏策略提供建议是目前研究的一大热点。本研究首先根据数据集中比赛类型对数据进行分类处理，对数据的特征进行探究并建立新的变量，接着使用线性回归、随机森林等算法对数据进行机器学习，并对玩家排名做出预测。

一、 背景简介及研究意义

本研究的 PUBG(绝地求生)游戏是指在 2017 年由韩国蓝洞公司发行的的一款电脑端多人在线射击竞技类游戏。需要区分的是，目前中国区该游戏由腾讯代理发行；国内的光子游戏开发工作室推出该游戏的手机客户端 PUBG MOBILE——中国区的 PUBG 及手机端 PUBG 不在本次讨论范围内。

PUBG 游戏模式如下：玩家在地图上收集武器、补给等，在不断缩小的安全区中与其他玩家对抗，最终存活者赢得比赛。注:

- 每一局游戏有 100 名系统匹配的玩家，若真人玩家人数不够 100，缺少的人数则由系统机器人补至 100，机器人对真人玩家只会造成较小的攻击伤害。
- 根据组队人数的不同，游戏有单人、双人、四人、自由组队型游戏。
- 根据画面显示的不同，游戏有 TPP（第三人称视角）和 FPP（第一人称视角）的模式可供玩家选择。
- 地图有多种类型可选，如海岛、沙漠等。
- 玩家在游戏开局的时候可自主选择在地图上的跳伞位置
- 武器包括不同杀伤力的枪、炮弹等，补给可用来恢复玩家体力及生命力，不同补给会有不同的恢复效果。
- 安全区是指在一局游戏中，系统每隔一定时间会在地图上缩小安全区，安全区外的区域将会通过毒气等方式杀死玩家；因此安全区起到聚拢一开始跳伞位置分散的玩家的作用。
- 玩家在与其他玩家之间对抗时，杀死对方的方式有通过枪击、车撞击等。
- 若玩家存活到最后，则在本场游戏中排名第一

目前该游戏在全球有固定的竞技赛事，吸引着全球各个国家优秀职业选手参加并获得奖金。对该游戏数据集进行分析预测，有助于用数据分析的方式探索在游戏中取胜的关键因素。

二、 数据集概览

该数据集来自于 Kaggle 比赛（链接：https://www.kaggle.com/c/pubg-finish-placement-prediction/data）。通过对 46 万余场比赛的数据进行机器学习，对测试集数据的玩家排名进行分析预测。根据官方数据描述，训练数据集变量及其解释如下：

变量名	变量描述
DBNOS	Number of enemy players knocked.
ASSISTS	Number of enemy players this player damaged that were killed by teammates.
BOOSTS	Number of boost items used.
DAMAGEDEALT	Total damage dealt. Note: Self inflicted damage is subtracted.
HEADSHOTKILLS	Number of enemy players killed with headshots.
HEALS	Number of healing items used.
ID	Player's Id
KILLPLACE	Ranking in match of number of enemy players killed.
KILLPOINTS	Kills ranked
KILLSTREAKS	Max number of enemy players killed in a short amount of time.
KILLS	Number of enemy players killed.
LONGESTKILL	Longest distance between player and player killed at time of death. This may be misleading, as downing a player and driving away may lead to a large longestKill stat.
MATCHDURATION	Duration of match in seconds.
MATCHID	ID to identify match. There are no matches that are in both the training and testing set.
MATCHTYPE	String identifying the game mode that the data comes from. The standard modes are “solo”, “duo”, “squad”, “solo
RANKPOINTS	Elo
REVIVES	Number of times this player revived teammates.
RIDEDISTANCE	Total distance traveled in vehicles measured in meters.
ROADKILLS	Number of kills while in a vehicle.
SWIMDISTANCE	Total distance traveled by swimming measured in meters.
TEAMKILLS	Number of times this player killed a teammate.
VEHICLEDESTROYS	Number of vehicles destroyed.
WALKDISTANCE	Total distance traveled on foot measured in meters.
WEAPONSACQUIRED	Number of weapons picked up.
WINPOINTS	Win like ranking of player.
GROUPID	ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time.
NUMGROUPS	Number of groups we have data for in the match.
MAXPLACE	Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
WINPLACEPERC	The target of prediction. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match. It is

	calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match.

根据游戏的进程以及变量统计方式，我们可以对变量进行定性分类。每一个玩家有独有 Id 和特征属性。一场游戏的进程包括玩家匹配、开始游戏、游戏结束及名次计算，根据统计的环节将数据在流程中分为以下几类：

ID	【'Id', 'groupId', 'matchId'】
个人表现（绝对型）	【'assists', 'boosts', 'damageDealt', 'DBNOs','headshotKills', 'heals', 'kills','killStreaks', 'longestKill','revives', 'rideDistance','roadKills','swimDistance','teamKills', 'vehicleDestroys', 'walkDistance','weaponsAcquired'】
个人表现（相对型）	【'killPlace'】
个人特征	【'killPoints', 'rankPoints','winPoints'】
游戏变量	【'matchDuration', 'matchType', 'maxPlace', 'numGroups'】
目标变量	【'winPlacePerc'】

其中，个人表现（绝对型）变量指的是该类变量只反映玩家在该局比赛中的实际数据，与该局其他玩家表现无关；例如 Kills 代表玩家在该局击杀的数量。个人表现（相对型）变量指的是该变量是根据本局排名确定----的，如 killPlace 是玩家击杀数量在本局所有玩家中的排名。个人特征变量指的是玩家在本局游戏以外的、基于该玩家历史表现的分数变量，如 Killpoints 变量是指以 kill 数为标准，该玩家在历史游戏中的得分情况。游戏变量指的是对该局游戏每一个玩家都相同的、反映该局游戏的特征的变量。

三、 数据可视化及数据处理

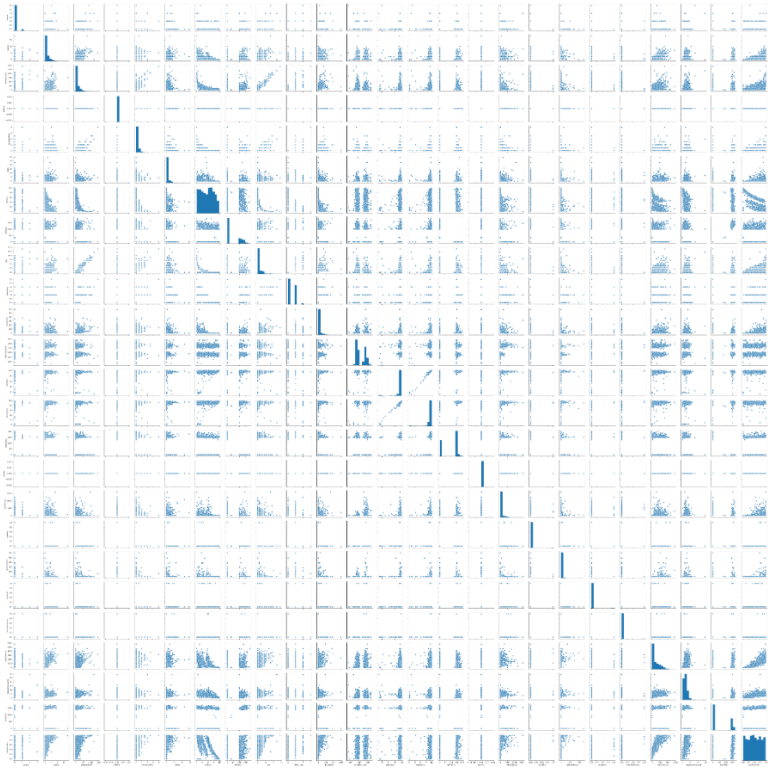
1) 数据概览

Id	groupId	matchId
4446966 unique values	2026745 unique values	47965 unique values

该数据集一共有四万余场游戏，目标变量：本次分析的目标变量是玩家在游戏结束后的排名，该排名在数据集中已经标准化，其中 1 代表第一名，0 代表最后一名。

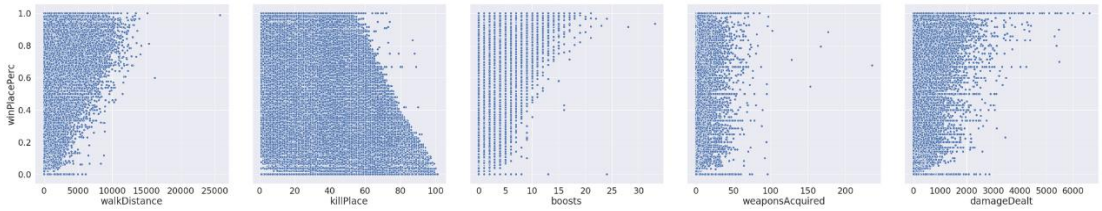
2) 变量探索

对所有变量两画散点分布图，并计算相关系数，得到结果如下：



从散点图及相关系数可以发现，大多数变量之间的相关系数较低（0.6 以下），大多数变量与目标变量之间相关性低。挑选部分变量与目标变量之间的图像进一步查看可知，变量值较小

的时候，其对应的目标变量的方差较大，例如在行走距离较小的时候，玩家的排名在 0 至 1 之间均有分布。这种情况可能是因为我们忽略了 matchid 这一变量直接对所有玩家的数据求相关系数。

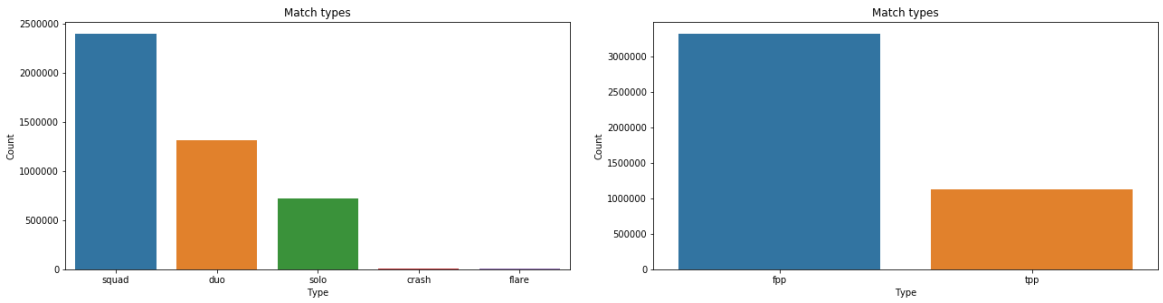


因此，在训练模型前，需要对变量进一步处理，使得每一局中不同玩家之间各个特征变量的差异变大，由此训练变量与目标变量之间的相关系数有一定增强。

3) 比赛类型

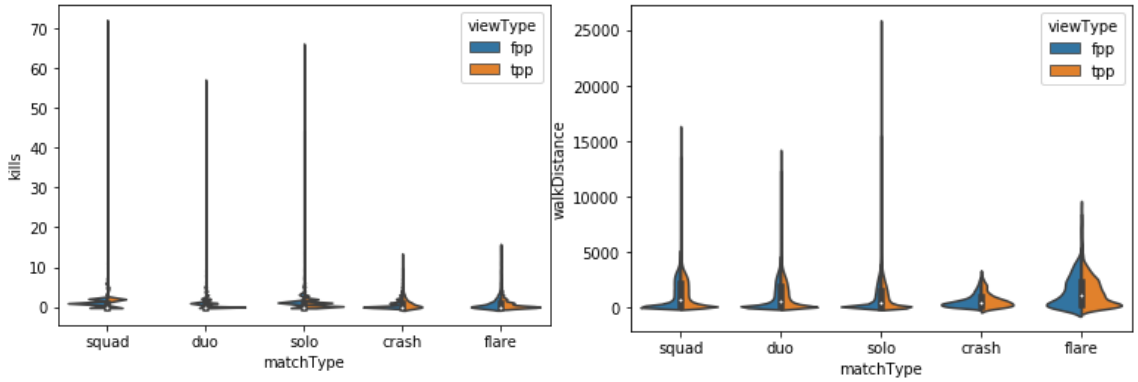
该数据集中有多种不同的游戏类别：根据组队模式可分为单人、双人、四人、任意组队（该类数据较少）；根据游戏视角可分为第一人称及第三人称。

数据集中各组队模式的数据量及各游戏视角的数据量为：



在组队模式下，同组每一个队员的排名都与存活时间最长的队员的排名相同，即在组队模式下，玩家可以靠队友的优异表现获得较好的排名。因此在考虑组队模式下玩家的排名的时候，需要考虑该组其他玩家的表现。本项目将会分别对不同组队模式的数据进行训练建模预测。

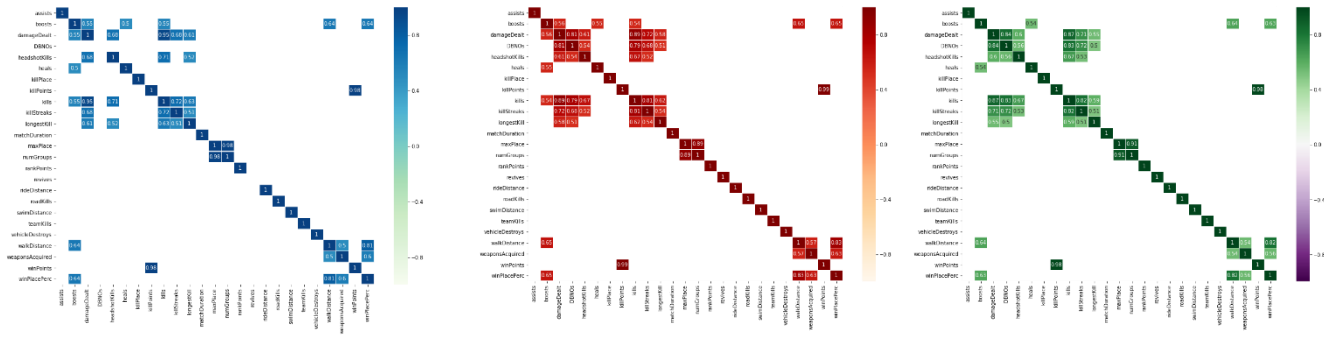
在各组队模式的数据被分开训练后，每一个模型的训练数据量下降。为了确认第一人称游戏视角与第三人称游戏视角的数据是否可以合并讨论从而增加每一组队模式下的训练数据量，将数据分为第一人称和第三人称，并对其中的两个反映玩家在游戏中的表现得变量作小提琴图。图中每一个小提琴图左边蓝色区域为第一人称游戏模式的数据分布，橘色区域为第三人称游戏模式的数据分布。可以看出，TPP 与 FPP 模式之间的数据在 kills 上的分布有较大差异，在 walk distance 差异较小，因此在后续分析仍然保留 tpp 与 fpp 的游戏分类变量。



4) 变量相关性

将数据分为三种游戏类型：SOLO\DUO\QUAD，分别探究变量相关性

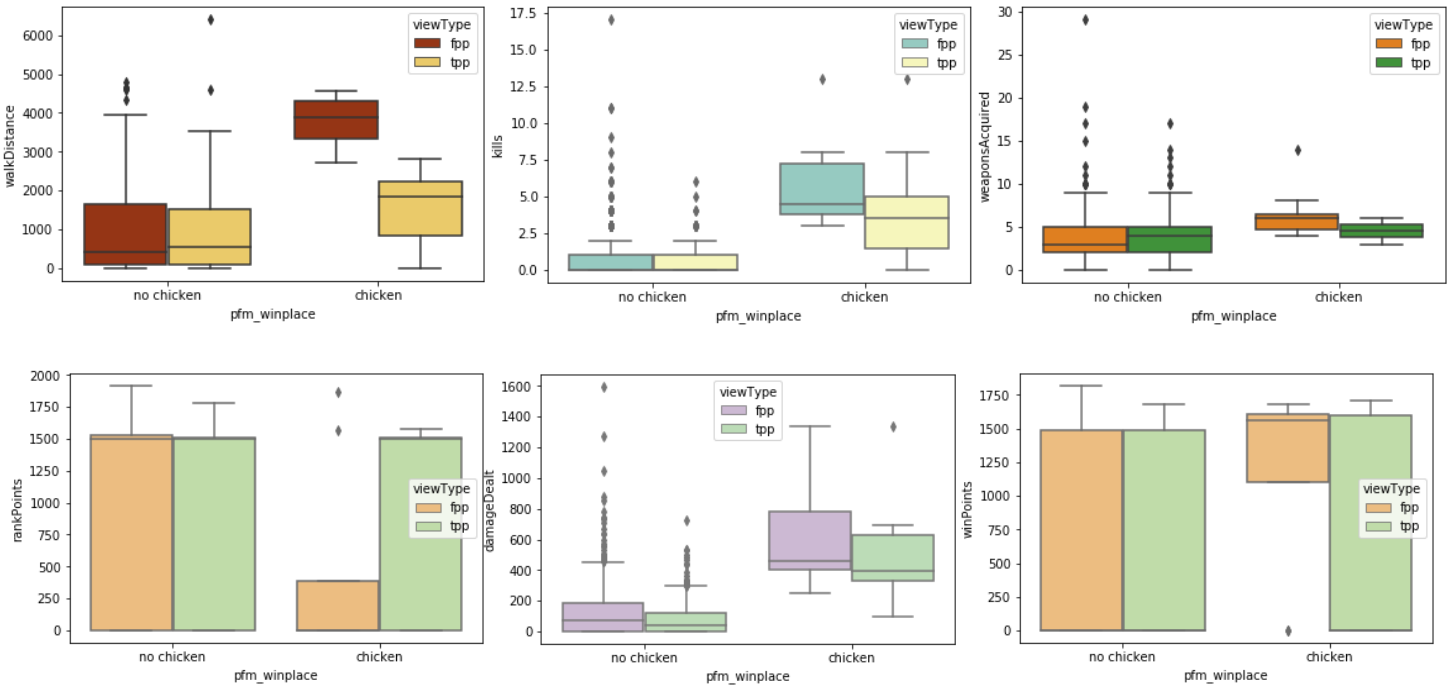
```
df_solo = df.loc[df['matchType']== 'solo']
df_duo = df.loc[df['matchType']== 'duo']
df_squad = df.loc[df['matchType']== 'squad']
df_other = df.loc[df['matchType']== 'other']
```



可以看到 walkDistance,killPlace,boosts,weaponsAcquired 和目标变量 winPlacePerc 有较强相关性。进一步对不同组队模式的玩家数据进行分析，探究玩家在获得较高名次时的变量特征

5) SOLO 模式变量特征探索

在 SOLO 模式的数据中，根据玩家排名，将玩家分为两类：chicken – 排名第一，no chicken – 排名不为第一。对两类玩家的各个变量画分布图，可以直观看到优秀玩家与一般玩家之间的变量差距。



由上面图片图可以看到，非优秀玩家在各个变量上都体现出明显的长尾分布，优秀玩家分布较为集中。优秀玩家在步行距离、击杀数量、武器获得数量、造成的破坏上相对于非优秀玩家。而在玩家的个人属性上，优秀玩家和非优秀玩家之间的差距并不明显，说明在一局游戏中，游戏的获胜有较大的随机性，经验较为丰富、历史游戏表现较好的玩家并不在某一局游戏中占据绝对获胜优势。

因此，对单人局变量进行处理：

标准化处理：标准化击杀人数 = 击杀人数/本场比赛人数，标准化头部击杀率 = 头部击杀/击杀人数

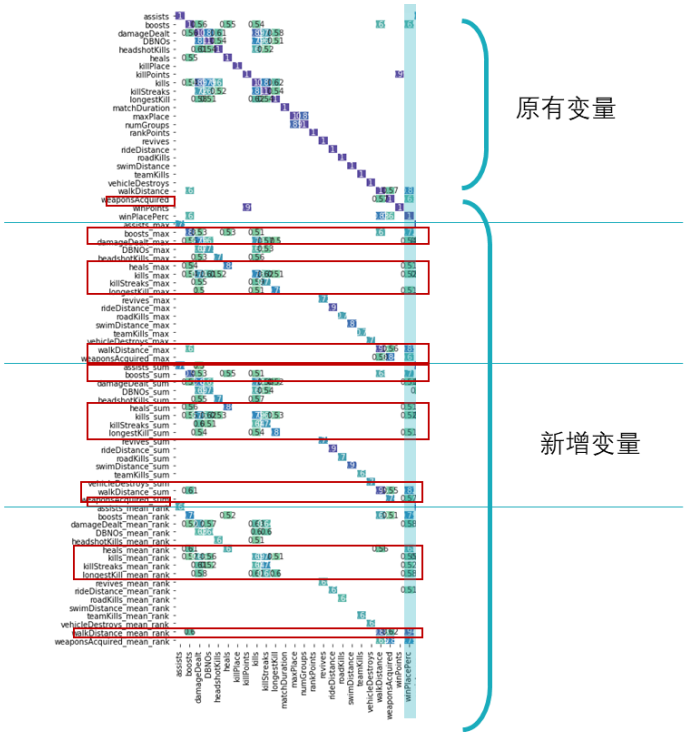
加总型处理：团队协作 = 复活数+助攻数，总移动距离 = 行走距离+驾驶距离+游泳距离

比值型处理：行走距离/体力恢复道具数，行走距离/击杀数

```
df['killPlace_norm'] = df['killPlace'] / df['maxPlace']
df['killPlace_norm'].fillna(0, inplace=True)
df['killPlace_norm'].replace(np.inf, 0, inplace=True)

df['headKills_norm'] = df['headshotKills'] / df['kills']
df['headKills_norm'].fillna(0, inplace=True)

df['teamwork'] = df['assists'] + df['revives']

df['totalDistance'] = df['walkDistance']+df['rideDistance']+df['swimDistance']
df['walk_heals'] = df['walkDistance'] / df['heals']
df['walk_heals'].fillna(0, inplace=True)
df['walk_heals'].replace(np.inf, 0, inplace=True)
df['walk_kills'] = df['walkDistance'] / df['kills']
df['walk_kills'].fillna(0, inplace=True)
df['walk_kills'].replace(np.inf, 0, inplace=True)
```

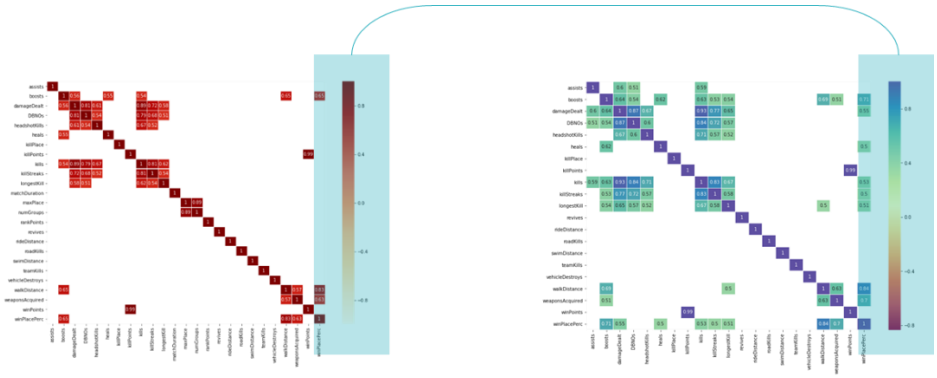
6) 多人组队模式变量特征探索

在探索多人组队模式的变量特征时，涉及到一个如何综合考虑本队实力的问题。可以采取的对各个变量处理方式有：组内取最大值，组内取最小值，组内取平均值，组内取极差。分别对各个变量构造了以上四种处理方式后，将新变量与目标变量求相关性，得到如下结果：

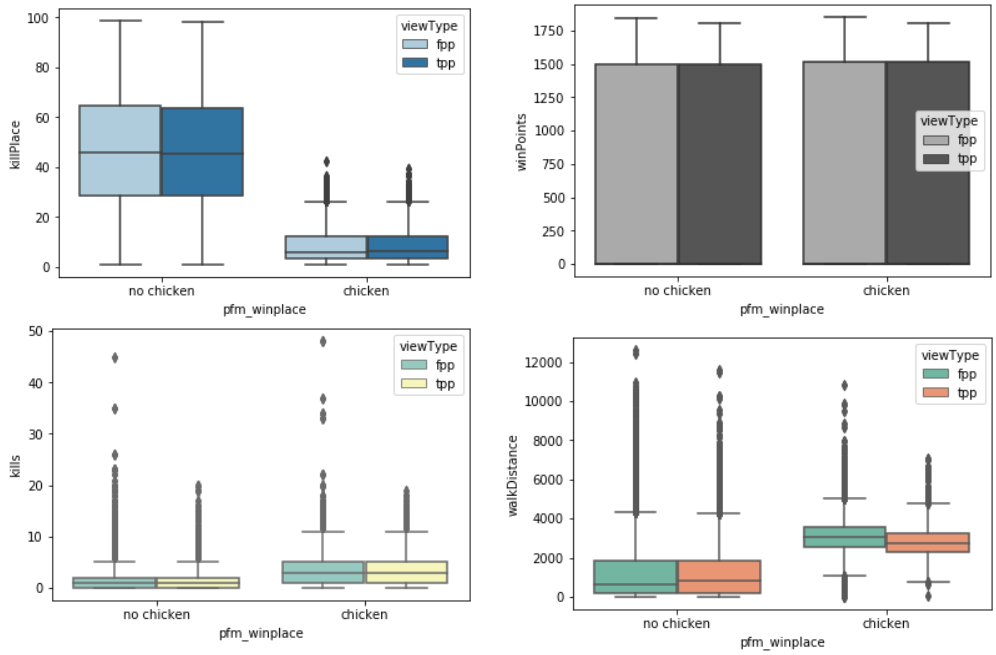


可以发现，四种处理方式中，相关性变高的变量基本吻合，相关性变化程度相近。因此四种处理方式在相关性提升上效果相近。本研究采取平均值的方式来代表整组表现。

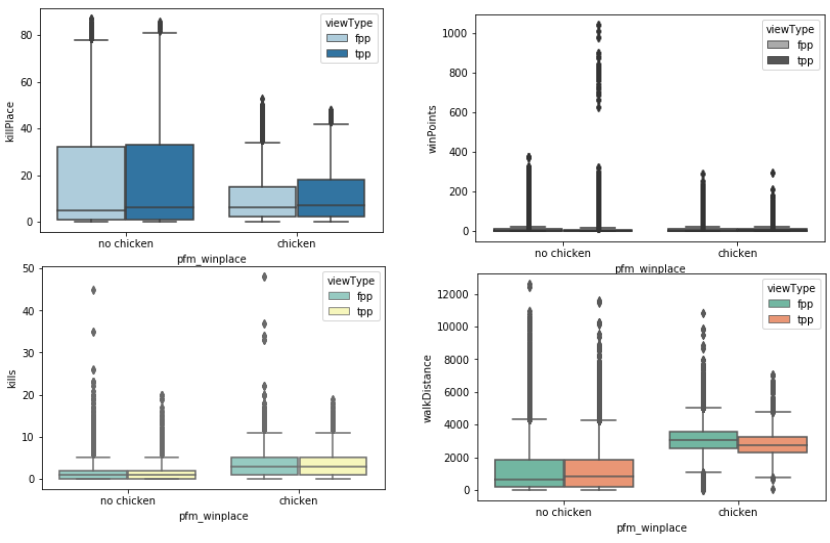
对多人游戏组内取平均值后，直接与目标变量有较强相关性的变量增多：



同对 SOLO 模式玩家的变量特征探索，将玩家分为两部分后，探索变量在优秀玩家和非优秀玩家之间的差异。变量处理方式分别为平均值和极差：



对于平均值的组内变量，我们发现优秀玩家在相对击杀率、行走距离上与非优秀玩家均有部分差距。



对于极差的组内变量，我们发现优秀玩家的方差变小，然而其他变量和非优秀玩家差距较小。

四、 训练集数据预处理

在分割训练集及测试集的时候，我们应注意同一游戏里的玩家不应该被分隔开，因此采用以下代码，使得我们按照游戏 ID 对测试集进行分割：

```
def split_df_val(data, fraction):
    matchIds = data['matchId'].unique().reshape([-1])
    df_size = int(len(matchIds)*fraction)

    random_idx = np.random.RandomState(seed=2).permutation(len(matchIds))
    df_matchIds = matchIds[random_idx[:df_size]]
    val_matchIds = matchIds[random_idx[df_size:]]

    data_df = data.loc[data['matchId'].isin(df_matchIds)]
    data_val = data.loc[data['matchId'].isin(val_matchIds)]
    return data_df, data_val
```

五、 基于机器学习算法的排名预测结果及分析

本研究使用到的机器学习模型有：多元回归、XGBOOST、LightGBM。首先对未经处理过的数据进行训练和预测，然后对经过特征工程后的数据集进行训练和预测，探究预测准确度提高的程度。

本研究采用平均绝对差来评价预测的准确性，公式为：

$$Error = Mean(|predict - true|)$$

1) 未经处理过的原数据

对于未经处理过的原数据，暂时不对参数进行调整，只用各模型的默认参数进行预测。

1. 多元回归

最基本的多元回归

$$h(\theta) = \theta 1x1 + \theta 2x2 + \theta 3x3 + \ldots + b = \theta Tx + b$$

通过最小二乘法，计算出最小的损失函数值，从而确定模型中各个参数的值。

其中计算系数方式为：

$$\theta = (X^T X)^{-1} X^T y$$

其中损失函数为：

$$\mathrm{J}(\theta) = \sum_{i=1}^{m} (\mathrm{h}_\theta(x_i) - \mathrm{y}_i)^2$$

使用 linear regression 对原数据建立模型（在 sklearn 中调用 sklearn.linear_model.LinearRegression），得到平均绝对值 error 为 0.092

```
estimator_1 = LinearRegression()
estimator_1.fit(X_train_reg,y_train_reg)
y_pred_reg = estimator_1.predict(X_test_reg)
#Error
Error_log = mse(y_pred_reg,y_test_reg)
Error_log
```

```
0.09211322699130407
```

基于梯度下降的多元回归

使用加入梯度下降法的 regression 对原数据建立模型，通过定义学习速率，随机初始化一个点，不断更新系数，最终找到成本最低的模型：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

但在使用该方法时，计算结果没有收敛，误差极大，调整参数后仍然无法计算出结果，因此在后续的建模中不予使用。

2. XGBOOST

XGBoost 全名叫极端梯度提升，通过建立 K 个回归树，使得数群的预测值接近真实。预测模型为

$$y_i = \sum_{k=1}^{K} f_k(x_i)$$

损失函数为：

$$Obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y_i}) + \sum_{k=1}^{K} \cap (f_k)$$

```
import xgboost
model = xgboost.XGBRegressor()
model.fit(X_train,y_train)
#shap_values = shap.TreeExplainer(model).shap_values(pd.DataFrame(X_
#shap.summary_plot(shap_values, X_train)
#xgb.plot_importance(model)
y_xgb_pred = model.predict(X_test)
Error_xgb = mse(y_xgb_pred,y_test)
```

进行预测后，误差结果为：0.068

3. LightGBM

该算法是 XGBOOST 的优化版

```
import lightgbm as lgb
model = lgb()
model.fit(X_train, y_train)
y_pred4 = mnb.predict(X_test)
Error_4 = mse(y_pred4,y_test)
Error_4
```

经过模型拟合，误差为

4. 随机森林算法

该算法原理是随机抽取 n 个样本建立 n 个决策树，在 m 和特征中，每次分裂时选择最好的特征分裂，接着让每颗决策树做到最大限度的生长，最终将多个分类树组成随机森林。

```
from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor(oob_score = True)
RF.fit(X_train,y_train)
y_pred4 = RF.predict(X_test)
#Error
Error_reg4 = mse(y_pred4,y_test)
Error_reg4
```

经过模型预测，得到结果为 0.0535

2) 已经进行特征工程的数据 – 单人游戏模式

1. 多元回归

首先使用多元回归训练模型，使用该模型对分离出的测试集进行预测，误差为 0.0784.

2. XGBoost

```
model = xgb.XGBRegressor(max_depth=17, gamma=0.3, learning_rate= 0.1)
model.fit(X_train,y_train)
#shap_values = shap.TreeExplainer(model).shap_values(pd.DataFrame(X_train, columns=X_train.columns))
#shap.summary_plot(shap_values, X_train)
#xgb.plot_importance(model)
#y_xgb_pred = model.predict(X_test)
#Error_xgb = mse(y_xgb_pred,y_test)
```

得到误差为 0.0538

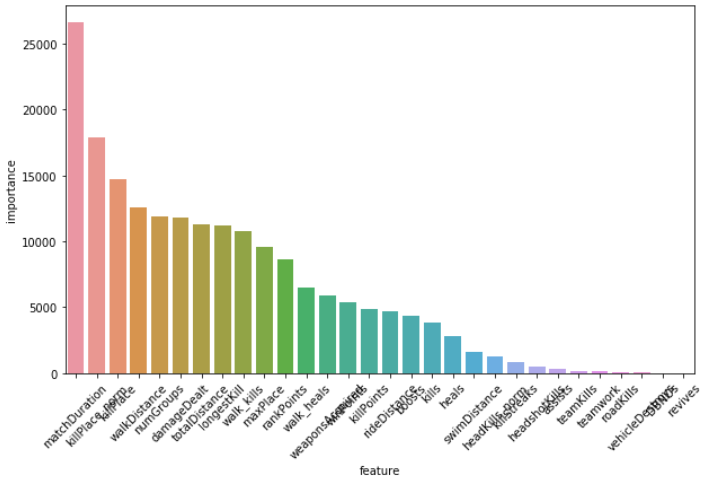
3. LightBGM

对 LightBGM 进行调参如下：

```
import lightgbm as lgb
train_set = lgb.Dataset(X_train, label=y_train)
#del X_train,y

valid_set = lgb.Dataset(X_test, label=y_test)
#del X_train_test,y_test


params = {
        "objective" : "regression",
        "metric" : "mae",
        "num_leaves" : 149,
        "learning_rate" : 0.03,
        "bagging_fraction" : 0.9,
        "bagging_seed" : 0,
        "num_threads" : 4,
        "colsample_bytree" : 0.5,
        'min_data_in_leaf':1900,
        'min_split_gain':0.00011,
        'lambda_l2':9
}

model = lgb.train(  params,
                    train_set = train_set,
                    num_boost_round=9400,
                    early_stopping_rounds=200,
                    verbose_eval=100,
                    valid_sets=[train_set,valid_set]
                  )
```

得到误差为 0.04，其中各个变量的重要性如下：

可以看到进行处理后的数据，模型预测正确率有一定提升，且进行特征工程的变量在重要性排名中较为靠前。



4. 随机森林

3) 已经进行特征工程的数据 – 多人模式

1. 线性回归：误差 0.108
2. XGBOOST：误差 0.079
3. LightGBM：误差 0.078
4. 随机森林：误差 0.084

总结以上误差结果表格如下:

方法	未经处理的总数据	特征工程后的单人游戏	特征工程后的多人游戏
线性回归	0.092	0.0784	0.108
XGBOOST	0.068	0.0538	0.079
LightGBM	0.0	0.0412	0.078
随机森林	0.0535	0.0462	0.084

根据结果，我们可以看到单人游戏的预测结果最好，其中 LightGBM 和随机森林的预测误差最小。说明在单人游戏的特征挖掘和数据处理上较为成功。多人游戏的预测结果较差，可能因为对多人游戏的数据是直接将所有队员的成绩取了平均而忽略了组内差异对结果的影响。进一步提升成绩应需要对组内差异建立变量。

附——源代码介绍:

源代码总体由两部分组成，一部分是数据可视化和特征工程，一部分是机器学习的算法。在源代码中通过横线标注出了每一部分的内容。

```
#------------------------------------------------------------调包------------------------
-------------------------
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from plotnine import *
import numpy as np

#-----------------------------------------------数据可视化及特征工程-----------------
-------------------------
#读入并备份
df = pd.read_csv('train_V2.csv')
df_origin = df

#查看比赛类别
```

```
#-------------------------------------------------------------------------
--
#画计数图
def count_show(x):#x 为列名
    m_types = df_origin.loc[:,x].value_counts().to_frame().reset_index()
    m_types.columns = ["Type","Count"]
    #m_types
    plt.figure(figsize=(10,5))
    ticks = m_types.Type.values
    ax = sns.barplot(x="Type", y="Count", data=m_types)
    ax.set_xticklabels(ticks,rotation = 60,fontsize=10)
    ax.set_title("Match types")
    plt.show()
#第一人称及第三人称转换
def type_transfer(x):
    if 'fpp' in x:
        return 'fpp'
    else:
        return 'tpp'
#新生成一列作为一三的标识
df['viewType'] = df['matchType'].apply(lambda x:type_transfer(x))###################
#把原有列改为游戏的标识
df['matchType'] = df['matchType'].str.replace(r'normal-|-fpp|fpp|tpp', '')
#画风琴图
def vio_plot(x_name,y_name,t_name,data_name):
    sns.violinplot(x = x_name,y = y_name, data = data_name, hue = t_name,split = True)
#看一下风琴图效果
vio_plot('matchType','kills','viewType',df_origin)
#vio_plot('matchType','kills','viewType',df_origin)

#-------------------------------------------------------------------------
-------------------------------
#画热力图
def hmap(x):
    corr = df.loc[df['matchType']== x].corr()
    f, ax= plt.subplots(figsize = (14, 10))
    sns.heatmap(corr, linewidths = 0.05, annot=True, ax = ax, mask=corr<0.5)
#分类看看效果
hmap('solo')
hmap('duo')
hmap('squad')

#-------------------------------------------------------------
#分组
df_solo = df.loc[df['matchType']== 'solo']
```

```
df_duo = df.loc[df['matchType']== 'duo']
df_squad = df.loc[df['matchType']== 'squad']
df_other = df.loc[df['matchType']== 'other']


#------------------------------------------------------------
#对组队创造变量

#一些 FE
#'assists', 'boosts', 'damageDealt', 'DBNOs','headshotKills', 'heals','kills','killStreaks',
#'longestKill','revives','rideDistance','roadKills',
#'swimDistance','teamKills', 'vehicleDestroys','walkDistance','weaponsAcquired'

#sum boosts kills walkDistance teamKills weaponsAcquired
#max walkDistance
#个人游戏特征工程
def eng_div():
    #一些个人表现型 FE#################################################################
    df_solo['killPlace_norm'] = df_solo['killPlace'] / df_solo['maxPlace']
    df_solo['killPlace_norm'].fillna(0, inplace=True)
    df_solo['killPlace_norm'].replace(np.inf, 0, inplace=True)

    df_solo['headKills_norm'] = df_solo['headshotKills'] / df_solo['kills']
    df_solo['headKills_norm'].fillna(0, inplace=True)

    df_solo['teamwork'] = df_solo['assists'] + df_solo['revives']

    df_solo['totalDistance'] = df_solo['walkDistance']+df_solo['rideDistance']+df_solo['swimDistance']

    df_solo['walk_heals'] = df_solo['walkDistance'] / df_solo['heals']
    df_solo['walk_heals'].fillna(0, inplace=True)
    df_solo['walk_heals'].replace(np.inf, 0, inplace=True)

    df_solo['walk_kills'] = df_solo['walkDistance'] / df_solo['kills']
    df_solo['walk_kills'].fillna(0, inplace=True)
    df_solo['walk_kills'].replace(np.inf, 0, inplace=True)

def min_by_team(df):
    cols_to_drop = ['Id', 'groupId', 'matchId','matchType', 'winPlacePerc']
    features = [col for col in df.columns if col not in cols_to_drop]
    agg = df.groupby(['matchId','groupId'])[features].min()
    return df.merge(agg, suffixes=['', '_min'], how='left', on=['matchId', 'groupId'])

def median_by_team(df):
    cols_to_drop = ['Id', 'groupId', 'matchId', 'winPlacePerc']
```

```
    features = [col for col in df.columns if col not in cols_to_drop]
    agg = df.groupby(['matchId', 'groupId'])[features].median()
    return df.merge(agg, suffixes=['', '_median'], how='left', on=['matchId', 'groupId'])

def mean_by_team(df):
    cols_to_drop = ['Id', 'groupId', 'matchId', 'winPlacePerc']
    features = [col for col in df.columns if col not in cols_to_drop]
    agg = df.groupby(['matchId', 'groupId'])[features].mean()
    return df.merge(agg, suffixes=['', '_mean'], how='left', on=['matchId', 'groupId'])


def sum_by_team(df):
    cols = ['assists', 'boosts', 'damageDealt', 'DBNOs','headshotKills', 'heals','kills','killStreaks', \
    'longestKill','revives','rideDistance','roadKills',\
    'swimDistance','teamKills', 'vehicleDestroys','walkDistance','weaponsAcquired' ]
    features = [col for col in df.columns if col in cols]
    agg = df.groupby(['matchId', 'groupId'])[features].sum()
    return df.merge(agg, suffixes=['', '_sum'], how='left', on=['matchId', 'groupId'])

def max_by_team(df):
    cols = ['assists', 'boosts', 'damageDealt', 'DBNOs','headshotKills', 'heals','kills','killStreaks', \
    'longestKill','revives','rideDistance','roadKills',\
    'swimDistance','teamKills', 'vehicleDestroys','walkDistance','weaponsAcquired' ]
    features = [col for col in df.columns if col in cols]
    agg = df.groupby(['matchId', 'groupId'])[features].max()
    return df.merge(agg, suffixes=['', '_max'], how='left', on=['matchId', 'groupId'])


def rank_by_match(df):
    cols = ['assists', 'boosts', 'damageDealt', 'DBNOs','headshotKills', 'heals','kills','killStreaks', \
    'longestKill','revives','rideDistance','roadKills',\
    'swimDistance','teamKills', 'vehicleDestroys','walkDistance','weaponsAcquired' ]
    features = [col for col in df.columns if col in cols]
    agg = df.groupby(['matchId','groupId'])[features].mean()
    agg = agg.groupby('matchId')[features].rank(pct=True)
    return df.merge(agg, suffixes=['', '_mean_rank'], how='left', on=['matchId', 'groupId'])

#----------------------------------------------------------------
#比较优秀玩家和非优秀玩家
def cpr(df):
    #df_solo_sample = df_solo.sample(1000)
    df['pfm_winplace'] = pd.cut(df['winPlacePerc'],[0,0.99,1],right=True,labels = ['no chicken','chicken'])
    plt.subplots(331)
    sns.boxplot(x = 'pfm_winplace', y= 'winPoints', data = df)
    plt.subplots(332)
```

```
    sns.boxplot(x = 'pfm_winplace', y= 'weaponsAcquired', data = df)
    plt.subplots(333)
    sns.boxplot(x = 'pfm_winplace', y= 'walkDistance', data = df)
    plt.subplots(334)
    sns.boxplot(x = 'pfm_winplace', y= 'damageDealt', data = df)
    plt.subplots(335)
    sns.boxplot(x = 'pfm_winplace', y= 'killPlace_norm', data = df)
    plt.subplots(336)
    sns.boxplot(x = 'pfm_winplace', y= 'teamwork', data = df)
    plt.subplots(337)
    sns.boxplot(x = 'pfm_winplace', y= 'walk_heals', data = df)
    plt.subplots(338)
    sns.boxplot(x = 'pfm_winplace', y= 'walk_kills', data = df)
    plt.subplots(339)
    sns.boxplot(x = 'pfm_winplace', y= 'teamwork', data = df)
#运行
df_solo_sample = df_solo.sample(1000)
cpr(df_solo_sample)
#-------------------------------------------------------------
#多人游戏数据处理
def sum_by_group(df):
    #a = df.groupby(['groupId'])[]
    cols = ['assists', 'boosts', 'damageDealt', 'DBNOs','headshotKills', 'heals','kills','killStreaks', \
    'longestKill','revives','rideDistance','roadKills',\
    'swimDistance','teamKills', 'vehicleDestroys','walkDistance','weaponsAcquired' ]
    features = [col for col in df.columns if col in cols]
    agg = df.groupby(['groupId'])[features].sum()
    return agg

df_duo1 = sum_by_group(df_duo)
df_duo_sample = df_duo1.sample(1000)
cpr(df_duo_sample)

df_squad1 = sum_by_group(df_squad)
df_squad_sample = df_squad1.sample(1000)
cpr(df_squad_sample)

#-------------------------------------------------机器学习部分-------------------------
-----------------------------------------
#机器学习模型
#分割 traindata 和测试 data
def split_df_val(data, fraction):
    matchIds = data['matchId'].unique().reshape([-1])
    df_size = int(len(matchIds)*fraction)
```

```
    random_idx = np.random.RandomState(seed=2).permutation(len(matchIds))
    df_matchIds = matchIds[random_idx[:df_size]]
    val_matchIds = matchIds[random_idx[df_size:]]

    data_df = data.loc[data['matchId'].isin(df_matchIds)]
    data_val = data.loc[data['matchId'].isin(val_matchIds)]
    return data_df, data_val

def train_test(df):
    X_train, X_train_test = split_train_val(df, 0.91)
    #train data
    y_train = X_train['winPlacePerc']
    X_train = X_train.drop(columns=['Id','groupId','matchType','viewType','matchId', 'winPlacePerc'])
    #test data
    X_test = X_test.drop(columns=['Id','groupId','matchType','viewType','matchId', 'winPlacePerc'])
    X_test = np.array(X_test)
    y_test = X_test['winPlacePerc']
    y_test = np.array(y_test)
    return X_train,X_test,y_train,y_test

#定义评估函数
def mse(predictions, targets):
    return abs(predictions - targets).mean()

#线性回归
from sklearn import linear_model
logistic = linear_model.LogisticRegression()

logistic = linear_model.LogisticRegression( penalty='l1', class_weight='balanced')
logistic.fit(X_train,y_train)
y_log_pred = logistic.predict(X_test)
#Error
Error_log = mse(y_log_pred,y_test)


#xgboost
model = xgboost.XGBRegressor(max_depth=17, gamma=0.3, learning_rate= 0.1)
model.fit(X,y)
xgboost.plot_importance(model)
y_xgb_pred = model.predict(X_test)
Error_xgb = mse(y_xgb_pred,y_test)

#随机森林
from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor(oob_score = True)
```

```
RF.fit(X_train,y_train)
y_pred4 = RF.predict(X_test)
#Error
Error_reg4 = mse(y_pred4,y_test)
Error_reg4


#LGB light
from sklearn.model_selection import train_test_split
X_train, X_train2, y, y2 = train_test_split(X_train, y, test_size=0.1, shuffle=False)
print("X_train np time")
X_train = np.array(X_train)
print("y np time")
y = np.array(y)

print("X_train2 np time")
X_train2 = np.array(X_train2)
print("y2 np time")
y2 = np.array(y2)

y = np.concatenate((y, y2), axis=0)
del y2
gc.collect()
X_train = np.concatenate((X_train, X_train2), axis=0)
del X_train2
gc.collect()


train_set = lgb.Dataset(X_train, label=y)
del X_train,y
gc.collect()
valid_set = lgb.Dataset(X_train_test, label=y_test)
del X_train_test,y_test
gc.collect()

params = {
        "objective" : "regression",
        "metric" : "mae",
        "num_leaves" : 149,
        "learning_rate" : 0.03,
        "bagging_fraction" : 0.9,
        "bagging_seed" : 0,
        "num_threads" : 4,
        "colsample_bytree" : 0.5,
        'min_data_in_leaf':1900,
```

```
        'min_split_gain':0.00011,
        'lambda_l2':9
}

model = lgb.train(  params,
                    train_set = train_set,
                    num_boost_round=9400,
                    early_stopping_rounds=200,
                    verbose_eval=100,
                    valid_sets=[train_set,valid_set]
                  )

del train_set,valid_set
gc.collect()

print("Calculating Feature Importance and save it in a file")
featureImp = list(model.feature_importance())
featureImp, features_label = zip(*sorted(zip(featureImp, features_label)))
with open("FeatureImportance.txt", "w") as text_file:
    for i in range(len(featureImp)):
        print(f"{features_label[i]} =  {featureImp[i]}", file=text_file)

print("Done calculating")
del featureImp,features_label
gc.collect()


X_test,features_label = featureModify(False)
X_test = X_test.drop(columns=['matchId'])
X_test = np.array(X_test)
y_pred=model.predict(X_test, num_iteration=model.best_iteration)
del X_test
gc.collect()
```