The University of Hong Kong

STAT 3612

Statistical Machine Learning

PUBG Finish Placement Prediction

Authors:
Chen Xiangyue 3035719052
Dong Xinhang 3035449217
Ye Mao 3035535066
Wang Mingxun 3035712872

December 7, 2021

Contents

1 Introduction

PlayerUnknown's Battlegrounds (PUBG) is one of the most popular multiplayer battle royale game in previous years. Analyzing such a familiar topic is quite an interesting task.

In order to better understand the details of the project, a brief summary of the gaming rules is needed: The match types consist of solo, duo, squad; first person or third person perspective playing. In the game, up to one hundred players parachute onto the battleground and scavenge for weapons and equipment to kill others. To discourage hiding throughout the game, the poisonous smoke will repress the available safety zone in size over time. In the enormous battleground, players can swim across rivers, drive vehicles to escape from the dangerous area. Notice that when you are playing in a squad, you can revive your teammates when they are knocked down by the enemies. However, solo players will be killed immediately. In the end, the only player or team remain wins the game. Our aim is to predict players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

2 Data Preparation

The dataset is retrieved from Kaggle which consists of 29 columns (Figure 1). Below will provide a brief description and summary of the important variables:

Identifiers:
MatchId is the id of each match with up to 100 players. GroupId identifies the team players in a match. MatchType indicates the game mode that the data comes from. Id is distributed randomly so it has no practical use here.

Attributes:
DBNOs represents the number of enemy players knocked. DamageDealt, headshotKills, killStreaks, kills, longestKill, vehicleDestroys can be interpreted in a literal sense. Boosts and heals mean the number of boosting and healing items used. RoadKills represent the number of kills whenever you are in a vehicle.

There are three distance measurements provided in the dataset. RideDistance measures the total distance traveled in vehicles, swimDistance measures the total distance traveled by swimming and walkDistance measures the walking distance. All three measurements use meter as the units.

Some team-play attributes are worth mentioning: assists means number of enemy this player has damaged who were killed by teammates. Revives means number of times this player has revived teammates.

WinPlacePerc is the target of prediction meaning the percentile winning placement, where 1 corresponds to the first place and 0 corresponds to last place in the match.

We plot a correlation heatmap to viualized the potential relationship bewteen the features. (Figure1)

2.1 Data Pre-Processing

Before we implement any statistical model to the data, we did some routine processing works. In the first step, we deleted the only missing value in the column "WinPlacePerc". Secondly, due to the player demanding nature of the game, it is not feasible to always gather 100 players in one game (Figure 2). Therefore, the game will automatically add robots to increase the number of "players". In order to avoid too many robots in one game, we dropped those matches with less than 75 players.

We created several new variables based on current features for latter analysis. 'num_player'

is the number of players in each match. 'group.player' is the number of players in every unique team. By subtracting 'num_player' from 'group.player', we get number of enemies faced by each group in a match and name the column as 'enemy.player'.(Figure 3) 'headshotRate' is headshotKills divided by kills, which is created to assess the skill of the players.(Figure 4) 'totalDistance' is an overall distance measurement, calculated by summing up walkDistance, rideDistance and swimDistance.(Figure 5) We will indeed use the two newly created variables for cheater detection. Another feature called 'health_items' is created by combining 'heals' and 'boosts' hence these two columns are dropped from the dataset.(Figure 6)

2.2 Cheater Detection

Hackers are rampant in PUBG, which is a bothering issue for the analysis. The hacks in the game create a bunch of outliers which will severely affect the stability of the model. For example, players with aim hacks can win nearly every fight they encountered in the game. Therefore, we developed a cheater detection scheme to identify the potential hackers for better model training.

The first one is anomalies in killing. The best strategy to spot the aim hack out is by looking at the head shot rate. Players with 100% head shot rate may be due to insufficient number of kills or using cheats. In order to avoid the "innocents", we consider a 100% headshot rate with more than 5 kills as cheating. From Figure 7, we can observe that more than 99% of the players with 100% headshot rate have lower than 5 kills. Killing without moving is another crucial criterion to identify cheaters, we detected 278 players having done so. Furthermore, we treat unrealistic number of kills or killing someone with an extraordinarily long distance as an evidence of cheating. According to Figure 8, the majority of players have no more than 20 kills. According to some gaming experiences, having 40 or more kills are normally using cheats, so we use 40 as our threshold. A large number of cheaters were removed by the above criteria.

The second one is anomalies in velocity. PUBG has a limitation of 6.3m/s for the walking speed, an upper bound of 2.9m/s for the swimming speed. Therefore, the players with average walking or swimming speed exceeding the limitation should be identified as cheaters. However, the total walking time and swimming time are not provided in the data set, so we use the average speed calculated by match duration as the upper bound. Given that $\frac{\text{Walking Distance}}{\text{Match Duration}} \leq \frac{\text{Walking Distance}}{\text{Walking Time}} = \text{Average Walking Speed}$, if $\frac{\text{Walking Distance}}{\text{Match Duration}}$ is larger than the limitation, the average walking speed must exceed the limitation.

3 Machine Learning

At first glance, we thought that the prediction task was relatively simple. However, there is one tiny detail which warrants our attention. For the same group of players, all of them will receive the same WinPlacePerc (our target variable) (Figure 9). Therefore, we have to analyse the group as a whole by developing a group-based dataset. In order to tackle the problem and obtain a better fit, we have to implement feature engineering.

3.1 Feature Engineering

The grouping issue can be handled by feature engineering. We first group the data set by matchtype, matchID, and groupID. In order to assign the same WinPlacePerc for each team member, we have to create new group-wise features to combine them as a whole.

We aim to establish a conversion scheme to represent the team-level characteristics for all relevant numerical features. It is quite intuitive that the strength of players within each group will vary. For example, a strong player may carry the whole team to the first place while one of his teammates might be killed at the beginning without contributing anything to the team. Therefore, we constructed weighted kills within each group. Selecting a suitable weight is a bit tricky, we used total distance of each team member divided by

the sum of total distances in the group as their own weight. The weighted kills will put heavy weight on those who survive longer, and we believe for those with a long total distance, they perform better than their teammates. Furthermore, we applied the max, min and mean functions to all variables in order to obtain a set of representative group-level features which increased our number of features significantly.(Figure 10) Adding too many features does not necessarily increase the prediction accuracy, but we can make adjustments afterwards.

After constructing the team level features, we are interested in comparing the performance of each group with other groups. We have constructed match-wise data for the comparison purpose. We selected several target features like: 'kills', 'killPlace', 'damageDealt', 'walkDistance' and 'healthitems' and obtained sum, maximum and mean of the variables within each match.(Figure 11) The comparisons are realized by dividing the group maximum value by the match sum value to produce the columns with prefix 'perc.gmax/msum'. Similarly, we divided the group max value by match max value to produce the columns with prefix 'perc.gmax/mmax'.(Figure 12) The purpose is to compare the strongest player in the group with the strongest player in other groups.

The raw killplace variable was presented in a very confused way. The data description indicates that killplace measures the relative ranking of kills in each match. However, we found out that even for those with zero kills, the killplaces are not equal. Therefore, we have created our own killplace to measure the relative ranks of number of kills. The scale used is from 0 to 1, a higher rank means a larger number of kills. Similar procedure has been done to kills, damageDealt and totalDistance also.(Figure 13) We come up with an idea that maybe stratifying killplace using the number of kills can be a good idea. Based on some preliminary data visualization, the majority of the players only kill a few number of enemies. Therefore, we reranked the killplace in each strata by creating the variables: 'kill_0_place', 'kill_1_place', 'kill_2_place'and 'kill_3_place'. For instance, for two players with zero kills who originally have killplace 80 and 81 will get a kill_0_place of 1 and 2 respectively (reranked in this sense) (Figure 14).

The last step is to encode the only categorical variable 'matchType', there are 14 unique matchtypes. One-hot encoding is applied to the dataset to create dummy variables. A '1' value is placed in the binary variable for the match mode and '0' values for the other modes.(Figure 15)

3.2 Model Fitting

Our finalized data set after feature engineering has 2,000,375 observations and 156 columns. Furthermore, after deleting some irrelevant columns, there are a total of 144 features used for model fitting and training. Because of the computationally expensiveness of adopting 10-fold cross validation or LOOCV, we decided to apply 2-fold cross validation in fine-tuning the data. The data set is first divided into two parts: the first part contains 700,000 observations and the following part contains 1,300,000 observations. We use the first 700,000 observations to fine-tune our model parameters, and the latter 1300,000 observations to examine the overall performance of the model. LGBM Regressor function from the lightgbm framework is used to carry out the following model fitting process.

3.2.1 Random Forest

The flexibility of the tree-based methods makes it quite suitable for the prediction task. Decision tree sometimes results in severe over fitting, so our first trial is to adopt the random forest which is a form of bagging method. Bagging is a technique which allows us to obtain a bootstrap aggregation of the decision trees.

$$\hat{f}(x) = \frac{1}{B}\sum_{i=1}^{n} \hat{f}^b(x)$$

Random forests further improve the bagging approach by decorrelating the decision trees. It selects n random subsets of predictors from the training set but when building the

decision trees, only a split in a tree is considered (James,2021).

3.2.2 Gradient Boosting Decision Tree

While random forest is a common-used bagging technique, gradient boosting is a boosting technique which can also be applied to solve classification and regression tasks. In this project, after careful comparison with other boosting types, gradient boosting is used for decision trees. Different from the random forest, the trees are grown using the information from the previous trees, so it forms a sequential training process.

3.3 Model Evaluation

The Kaggle competition used mean absolute error (MAE) as the evaluation criterion, so we adopt the same metric to allow us to compare our models with others.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Our initial trial with Random Forest brought us a MAE of 0.03502 in the validation set. The initial result with the gradient boosting decision tree is 0.034. Because of the training speed of the latter one is much faster than the random forest, we decided to adopt the gradient boosting decision tree.

For simplicity, we'll skip the parameter tuning in random forest evaluation and emphasize on our efforts made in improving the performance of our gradient boosting decision tree model.

3.3.1 Feature Selection

Since we have a total of 144 features, the interactions of some of the variables may not possess any intuitive meaning and may lead to deterioration of predictability and robustness. therefore we filter out variables with little importance in model predictions. Using the first 700,000 observations, we first randomly split them into test and training set with a ratio of 2 to 8. After fitting the model, we obtain the feature importance of the variables, and repeat the process for 10 time. We simply take the average of the feature importance of the 10 results and sort them in order. The initial parameters for model fitting are as follow:

boosting_type = "gdbt", objective = "regression", num_leaves = 150, verbose = -1, bagging_freq = 1, bagging_fraction = 0.7, feature_freq = 0.7, learning_rate = 0.05

We filter out the top 100 features with the highest importance scores. By repeating the model training procedure above, the obtained MAE is 0.03456. (Figure 16) In order to further simplify our model, the most important 50 features are selected and the model is refitted. The estimated MAE is 0.03453. (Figure 17) With further trials of smaller values, our judgement is that the ability of improving accuracy through tuning number of features included has reached its threshold, so we turned to parameter tuning using the 50 filtered features.

3.3.2 Parameter Tuning

There are several parameters that are of crucial importance to the performance of the gradient boosting decision tree. We believe that the following variables may impact the model performance heavily:

- num_leaves: maximum number of leaves allowed in the tree
  Too many leaves will lead to overfitting and insufficient number of leaves will lead

to underfitting.(Figure 18)

- max_depth: maximum depth of the decision tree
  By controlling the depth of the tree, overfitting can be possibly avoided. The default is -1, meaning that the depth of each tree will not be restricted. (Figure 19)

- learning_rate: boosting learning rate
  It measures how fast the model learns. The magnitude of modification is controlled by the learning parameter. If the value is too large, the model might converge too fast to reach a suboptimal model. (Figure 20)

- bagging_fraction:
  Bagging generally does a good job of improving the performance of unstable, overfitting classifiers. Changing bagging fraction may be able to avoid overfitting. (Figure21)

For the above parameters, we test the impact of each parameter by plotting the resulting MAE with different values of the parameter. The optimal parameters are:

num_leaves=500,max_depth=-1(indicating no limit),leaning_rate=0.125, bagging_fraction=0.6

For the above parameters, we re-examine the performance of model's relationship with the number of features included in the model,and obtain the Figure 22.

4 Results

In the end, we achieved the result of 0.03187 as the MAE of the validation set. The feature importance does not vary too much comparing to the previous feature selection part. From the feature importance plot, we found a very surprising result that almost all the important features are related to killPlace or variables generated from the killPlace,

especially kills_0_Place. Therefore, we go back to the original data and carefully investigate in what exactly killPlace is. We discover that actually it leaks some information of the WinPlacePerc (target variable). We can see that for the same number of kills, killPlace and winPlacePerc are perfectly correlated. Therefore, that's the reason why we have such a good performance for a reasonably difficult prediction task (Figure 23).

5 Conclusion

In conclusion, the prediction task teaches us analyzing the raw data set is very important in machine learning. If we fail to observe that the WinPlacePerc of every group is the same, we will obtain a worse result. The information leakage issue is something else which can be observed before feature engineering and model fitting. However, we have failed to do so, maybe the model performance can be further improved if we dig deeper into the variable killPlace. If we really submit our model to the Kaggle competition, we can achieve a rank of 588 (Figure 24).

6 Reference

James, G., Witten, D., Hastie, T., amp; Tibshirani, R. (2021). An introduction to statistical learning: With applications in R. Springer.