**PUBG Placement Prediction**

Author: Group 11

001491111 Ke Yuan

001422663 Yifu Xu

[Introduction 1](#_Toc47671373)

[Description of Dataset 2](#_Toc47671374)

[Data fields 3](#_Toc47671375)

[Methodology 5](#_Toc47671376)

[Results and Analysis 6](#_Toc47671377)

[EDA 6](#_Toc47671378)

[Feature engineering 8](#_Toc47671379)

[Parameters tuning 10](#_Toc47671380)

[Train and Predict 11](#_Toc47671381)

[Parallel 11](#_Toc47671382)

[Results and Analysis 13](#_Toc47671383)

[improvements 14](#_Toc47671384)

[Conclusion 15](#_Toc47671385)

[Reference 16](#_Toc47671386)

## Introduction



Battle Royale games have taken this world into a new gaming era. 100 players are dropped onto an island with empty-handed. They must equip themselves with the items on the ground, survive to the end. The safe zone will shrink as the game is going on. only one group could enjoy the victory.

With the arise of PLAYERUNKNOWN’S BATTLEGROUNDS in the 2017, the fashion reached the summit. It became the top3 most-selling game in Steam history. For every single player, there are questions like how to get a better placement. What will influence our placement most? And is there a strategy that always works? Data can answer these questions. Here is a competition in Kaggle for predicting placement according players’ performance in over 65000 games. As big fans of Battle Royale, it will be a pleasure to analysis these data and find a way guiding us for a better placement.

## Description of Dataset

In a PUBG game, there are 100 players, but this is no guarantee. In game, players can pick up weapons, shoot enemies, walk or drive to avoid the gas, also can swim sometime, revive your teammates if they are down but not eliminated. All these data are formatted by the “The Blue Circle” official API. Every row presents a player. We are supposed to predict the final “winPlacePerc”. This attribute is distributed from 0(last place) to 1(first place). It is scaled according to your placement and the number of groups in total.

### Data fields

* **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-based external ranking of player. (Think of this as an Elo ranking where only kills matter.) If there is a value other than -1 in rankPoints, then any 0 in killPoints should be treated as a “None”.
* **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-fpp”, “duo-fpp”, and “squad-fpp”; other modes are from events or custom matches.
* **rankPoints** - Elo-like ranking of player. This ranking is inconsistent and is being deprecated in the API’s next version, so use with caution. Value of -1 takes place of “None”.
* **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-based external ranking of player. (Think of this as an Elo ranking where only winning matters.) If there is a value other than -1 in rankPoints, then any 0 in winPoints should be treated as a “None”.
* **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.

## Methodology

We used two different algorithms to solve this problem. One is LightGBM. As the most used algorithm in the Kaggle competition, it sure holds its ground once Microsoft team open-sourced it when compared to other Gradient Boosting algorithms such as XGBoost. It has many advantages:

* + Faster training and higher efficiency
  + Lower memory usage
  + Better accuracy
  + Support of parallel and GPU learning
  + Capable of handling large-scale data.

More information about LightGBM, please refer to <https://lightgbm.readthedocs.io/en/latest/>.

Another is CatBoost. It is also use gradient boosting on decision tree and is available as an open source library. Here is its documentation <https://catboost.ai/docs/concepts/about.html>

We can use multi-core CPU or GPU to accelerate the training. In another hand, we can use distributed machine to run the model for higher speed.

## Results and Analysis

### EDA

The first part when we encounter a machine learning problem is always EDA (Exploratory Data Analysis). So let’s start from here.

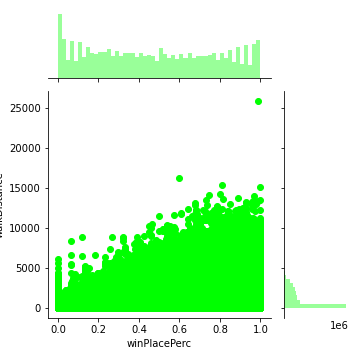
We are going to present some important results here. The notebook is run on the local machine.

We found kills, walk distance and other kinds of distance, heals and boosts have high correlation with our target “winPlacePerc”.

These are some jointplot of several attributes might have correlations in our understanding.

图片包含 游戏机

描述已自动生成

图片包含 游戏机, 截图

描述已自动生成

图片包含 游戏机

描述已自动生成

图片包含 游戏机

描述已自动生成图片包含 游戏机

描述已自动生成

Other variables also have their own attributes. So we plot the heatmap of all the variable.

图片包含 游戏机, 建筑, 笼子

描述已自动生成

These results will guide us in the next stage, feature engineering.

### Feature engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process. Feature Engineering is an art.

Feature engineering is the most important art in machine learning which creates the huge difference between a good model and a bad model.

There are 4 match types. Some attributes like “KillPlace” and “Kills” can not present players’ performance uniformly. So we transform them into the percentage ranking grouping by match.

Some attributes like “walkDistance, “rideDistance” and “swimDistance” reveal the same aspect of the players. They have same relationship with our target. We can simply plus them together as a new feature. Another example is “heals” and “boosts”, as we analysis before.

Players stay longer in the game will have higher values on some attributes like “walkDistance”. So we add new feature like “kills” over “walkDistance” and “headshotKillRate”, which can tell us who has better skills more straightforwardly.

Some features have weak correlation according to the heat map and our game experience. So we drop some features like “roadKills” and “vehicleDestroys”.

Other attributes we used to created new features should also be removed for simplicity.

We already know that there are hardly any missing values in our dataset. So we need only replace infinity values and None values in our computed attributes.

Then we add some aggregation features grouping by match and team. The level of your team and the whole match decided your placement in a large extent. “mean”, “max” and “std” are in our consideration. They also have a ranking in percentage for each team.

That is our most job about feature engineering. Finally, we extend from 29 columns in original dataset to 277 columns. It is a large collection, so we do not plan to show them all here. We believe they will lead us to a better predication.

### Parameters tuning

There are also parameters tuning guide for lightGBM in the documentation, including scenarios listed below:

* Tune Parameters for the Leaf-wise (Best-first) Tree
* For Faster Speed
* For Better Accuracy
* Deal with Over-fitting

Tuning different parameters can help us in different aspects.

We use RandomSearchCV to find the optimal parameters. This is most time-consuming part when executing the codes. Our codes were running on the Google collab. The reason we choose Google collab is the memory of local and Discovery can not meet our needs. And it is interactive. We can debug our codes conveniently.

The hyperparameters we concern about most and the parameters interval are listed here:

1. adj\_params = {'n\_estimators': range(100, 400, 10),
2. 'max\_depth': range(5, 15, 2),
3. 'num\_leaves': range(10,100,10),
4. 'min\_child\_weight': range(3, 20, 2),
5. 'min\_child\_samples': range(10, 30),
6. 'reg\_lambda': [0,0.001,0.01,0.05,0.08,0.1,0.3,0.5],
7. 'reg\_alpha': [0,0.001,0.01,0.05,0.08,0.1,0.3,0.5],
8. 'feature\_fraction':[0.5,0.6,0.7,0.8,0.9],
9. 'bagging\_fraction':[0.6,0.7,0.8,0.9,1.0]
10. }

### Train and Predict

In the last stage, we set the learning rate to 0.1 for faster convergence. Now we get the best parameters and save them in a file for further usage. But when it comes to predicting, we need a new model which learning rate set to 0.01 for better accuracy. This will be our final model. It will be saved as a file, too.

After this training, we can predict our target “winPlacePerc” on the test data set now. At the first stage, we split the 10% of the data as the test data. Only 90% of the data participate the parameters tuning and training. But the whole dataset need be preprocessed in the same way. It is essential to ensure the data consistency. The metric used to evaluate our model is MSE (Mean Squared Error).

This is also the metric of this competition to evaluate all the teams.

### Parallel

This is a lightgbm algorithm time cost when using different number of CPUs when searching parameters and training. We found that 4 CPU is the best. Too many CPU will cost more on scheduling resources.

This is the result when we used catboost algorithm. The best GPU number is 8.

When we use CatBoost algorithm accelerated with multi GPUs, time saving most at around 4 to 8 GPU. The time cost less than lightGBM because we did not run algorithm on the whole dataset.

### Results and Analysis

We use MSE (mean squared error) to evaluate our model. To our surprise, the MSE has a downward trend from 2 cores to 8 cores.

This result is extremely better than we expected. Because the best score, No.1 in this competition is 0.1385. It is so abnormal that we think there must be some problems. We might make some mistakes during the whole process. We are of course not the “genius” who created a model which can predict the placement so precisely. When we summary our project, we sum up two reasons for this “accuracy”. First reason is in the formal competition, competitors uses the test dataset provided by Kaggle, not splits data from the train data. Test data may have slight differences with train data in model which makes prediction more difficult. Another reason is there may have overfitting problem in our model.

### improvements

We had a hard work on this project. But we know clearly it can be improved in many aspects. Here are some ideas

Parallel. We use the multi core CPU and multi GPU to accelerate our learning process. But we can move one step further, use multi machine. LightGBM also supports distributed parallel learning. The parameter tree\_learner can decide the parallel type. This form is cited from the LightGBM documentation:

|  |  |
| --- | --- |
| Parallel Algorithm | How to Use |
| Data parallel | tree\_learner=data |
| Feature parallel | tree\_learner=feature |
| Voting parallel | tree\_learner=voting |

In our experiment, we used a quarter of the dataset downloaded. All the platform we are familiar with cannot meet our memory needs. The first rule of machine learning, the more data, the more accuracy. So using the whole dataset can definitely improve the result.

How we conduct feature engineering is dependent on our experience in great extent. But it is the main factor of the learning effect. So modifying the features may help a little.

As we know, machine learning is an iteration process. We can repeat the whole process while make a little improvement each time. In the end, we will get a much better model, no matter for features or parameters.

When we make predictions, we should take the edge case into consideration. In our topic, If there is only one group in the match, this situation can be exist due to cheaters, the “winPlacePerc” can be assigned as 1 directly.

## Conclusion

图片包含 游戏机, 截图

描述已自动生成

These are top 30 important features. As we can see, the most big factor is you have a strong teammate! Secondly what’s matters is your kills. These concludes that killing more will help a lot than just hide yourself.

## Reference

We reference a lot of resources during the project. Here we list them all:

Kaggle notebooks:

<https://www.kaggle.com/deffro/eda-is-fun>

<https://www.kaggle.com/rejasupotaro/effective-feature-engineering>

<https://www.kaggle.com/plasticgrammer/pubg-finish-placement-prediction-playground>

<https://www.kaggle.com/anycode/simple-nn-baseline-3>

<https://www.kaggle.com/kamalchhirang/5th-place-solution-0-0184-score>

documentation

<https://lightgbm.readthedocs.io/en/latest/>

<https://catboost.ai/docs/concepts/about.html>

forum:

<https://zhuanlan.zhihu.com/p/139453884>

<https://datascience.stackexchange.com/questions/63129/gridsearchcv-vs-randomsearchcv-and-how-it-works>

<https://www.analyticsvidhya.com/blog/2019/12/6-powerful-feature-engineering-techniques-time-series/#:~:text=Overview%201%20Feature%20engineering%20is%20a%20skill%20every,Each%20feature%20engineering%20technique%20is%20detailed%20using%20Python>

<https://zhuanlan.zhihu.com/p/76206257>

<https://zhuanlan.zhihu.com/p/65472471>