PUBG: A Guide to Free Chicken Dinner

Wenxin Wei, Xin Lu, Yang Li

December 13, 2018

Abstract

PUBG(Playerunknowns Battlegrounds) is a video game that has become popular in the past year. For this game, the final rank is the most important indicator of the player's ability. This project focuses on predicting the final rank and finding optimal strategies of the game. With data from PUBG, we uses several machine learning methods including ridge regression, lasso regression, lightGBM model, and random forest model to make relevant predictions.

1 Introduction

As the most popular multiplayer online battle arena video game, PUBG has attracted more and more players to join. The game features the last man standing deathmatch mode, in which one hundred players fight against each other in a battle royale. Players can also choose to match solo, or with small team of up to four people. In order to win the first place (a.k.a. eat Chicken Dinner), players must choose appropriate strategies during the game and make every effort to survive till the end. Various gaming strategies may include: collecting weapons and shields versus collecting sufficient medical packs, engaging in every possible fight versus avoiding battles, taking cars to relocate versus sneaking across bushes and forests, etc.

Given players statistics within a game, we plan to predict their final ranks. To be more specific, we take the data from Kaggle[1] competition, which contains anonymized PUBG game stats. The inputs to our algorithms are features that influence the players finishing placement. We then use both regression models and tree models to output a predicted final rank. After predicting final ranks, we perform an additional step to classify game strategies used by top players, which may give the players a choice to adopt optimal strategies.

2 Related work

A significant part of our work is feature engineering. The source data provided by Kaggle contains 4.45 million entries of training data, with 28 features in each entry. Some entries in the data are invalid, and a few entries are generated by cheaters. We filtered out problematic data entries first, then applied techniques to remove outliers, e.g., players who had no move in the whole game but got kills, who had unreasonable head-shot rate or long-distance kills. Next, normalization of features[2] is applied to numeric features, taking mean, variance and match size into consideration. Combined features are also added to help the algorithm consider team performance at the group level. Furthermore, we added cross validation to our train data[11].

To predict the rank of each player based on his statistic data, we firstly considered linear regression algorithms. Besides the normal linear regression, we tried Lasso regression[3], elastic net[4], Ridge regression[5], and SGD regression. Ridge regression got the best MAE because it handles regression data that suffer from multicollinearity better[6].

Tree algorithm is another series of widely-used algorithms that can be used as regression model. We explored the random forest algorithm[7] first, and used fast.ai utils[8] to extract out important features. Based on these features, we compared random forest model with lightGBM[9], and got our best result (MAE = 0.0204) with the lightGBM model.

3 Dataset and Features

The dataset comes from Kaggle competition, which contains anonymized PUBG game stats, formatted so that each row contains one players post-game stats. The columns are some features that have impact on the players finishing rank, such as number of enemy players killed, total distance traveled on foot/in vehicle/by swimming, number of weapons picked up, etc. Two example waveform plots showing distribution of 'damageDealt' and 'walkDistance' by players. (Figure 1)

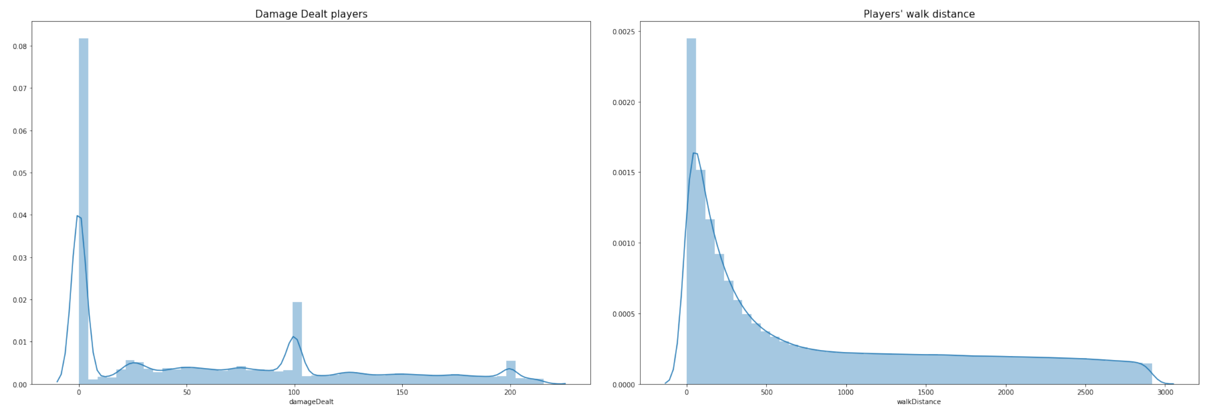

Figure 1: 'damageDealt' distribution (left) 'walkDistance' distribution (right)

We first perform fully-covered data cleaning. Some observations in our dataset have really weird characteristics. The players could be cheaters, maniacs or just anomalies. Removing these outliers will most likely improve results. We inspect the data and figure out the types of players to be removed: killing without moving, anomalies in aim(more than 10 roadKills, more than 45 kills and 100% headshot rate), anomalies in total travel distance, anomalies in weapons, etc.

We apply a few feature engineering methods to process the data:

1. Add group-statistic data: As the final score is the same for all members within a group, for each group in a match, we get the mean/sum/max/min value of each feature within the same group, then rank the values within the same match to get the mean/sum/max/min value ranking as a new feature for each group.

2. Since the number of players joined in each match is different, we first normalize feature values based on the amount of players joined in a match, and then normalize the features by mean and variance.

3. Convert categorical feature such as 'match type' into one hot vector.

4. To better capture information hidden behind group of features, we add combined features such as totalDistance = rideDistance + walkDistance + swimDistance, 'healsandboosts' = 'heals' + 'boosts'

5. Define new feature 'fights' to quantify amount of battles players involved

6. Define feature 'escape' to indicate how likely a player tend to run away from danger

4 Methods

4.1 Regression Models

We have tried five most commonly used regression models. Linear regression was selected as our baseline model with a baseline MAE score at around 0.09, and based on that, we continued to try ridge regression, lasso regression, elastic net regression, SGD regression. Ridge regression has the best out-of-box result with the help of aforementioned feature engineering, likely because it handles multicollinearity data better.

Ridge regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

4.2 Tree Models

Decision tree are a popular tool in machine learning, specifically in decision analysis, to help identify a strategy most likely to reach a goal. In this case, we have applied random forest algorithm and lightGBM (light gradient boosting machine) algorithms. LightGBM results in quite good MAE score, which shows significant improvement on linear regression models.

LightGBM is a new algorithm that combines GBDT algorithm with GOSS(Gradient-based One-Side Sampling) and EFB(Exclusive Feature Bundling).[9] Specifically, LightGBM uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage. Theoretically, in lightGBM algorithm with GOSS method, first, we rank the training instances according to their absolute values of their gradients in descending order; second, we keep the top-$a * 100\%$ instances with the larger gradients and get an instance subset A; then, for the remaining set A^c consisting $(1-a) * 100\%$ instances with smaller gradients, we further randomly sample a subset B with size $b * |A^c|$; finally, we split the instances according to the estimated variance gain over the subset $A \cup B$,

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n_r^j(d)}\right),$$

i.e.,
thus, the computation cost can be largely reduced. More importantly, GOSS will not lose much training accuracy and will outperform random sampling.

4.3 Semi-supervised Mixture Gaussian

A Mixture Gaussian model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians. Semi-supervised Mixture Gaussian model adds a portion of labeled data into the existing mixture model to help algorithm learn accurately and converge faster. Below is the update rule for GMM.[10]

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)} + \alpha \sum_{i=1}^{\tilde{m}} \mathbf{1}\{\tilde{z}^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)} + \alpha \sum_{i=1}^{\tilde{m}} \mathbf{1}\{\tilde{z}^{(i)} = j\}}$$

$$\phi_j := \frac{\sum_{i=1}^{m} w_j^{(i)} + \alpha \sum_{i=1}^{\tilde{m}} \mathbf{1}\{\tilde{z}^{(i)} = j\}}{m + \alpha \tilde{m}} \qquad \Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T + \alpha \sum_{i=1}^{\tilde{m}} \mathbf{1}\{\tilde{z}^{(i)} = j\} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)} + \alpha \sum_{i=1}^{\tilde{m}} \mathbf{1}\{\tilde{z}^{(i)} = j\}}$$

5 Results and Discussion

Our main motivation is to measure the accuracy of predicting players' winning placement and primary metric is MAE of predicted winning placement. The winning placement is a decimal value ranges from 0 to 1 while 0 corresponds to last place and 1 corresponds to the first place. Our best result comes from LightGBM model with a MAE of 0.0204; this result was ranked 57 in Kaggle leader board as of Dec 10.

In our baseline attempt, we feed only raw features into various models, including Linear Regression, Ridge Regression, Lasso Regression, Random Forest and Light Gradient Boosting Machine Model. Then we tune our models by adding engineered features step by step and observe that models gradually improved. The last step we perform is applying 5-fold cross validation to the data set. Please refer to the following table for detailed result we recorded from each step. For conciseness, we only selected best model from regression family and tree family, plus Linear Regression as reference.

MAE on 20% validation set	Linear Regression	Ridge Regression	Light GBM
Raw features	0.09000	0.08989	0.05654
Raw + mean	0.05736	0.05736	0.04158
Raw + mean + sum + max - min	0.04845	0.04845	0.02896
Everything above + match_mean + size	0.04825	0.04825	0.02755
Everything above + cross validation	0.04812	0.04810	0.0204

In the Light GBM model, we used learning rate 0.05. Although we understand model could be trained faster with slightly higher rate, we choose to use a conservative number just to make sure algorithm converges properly. In order to reduce bias, we introduced k-fold cross validation and found out 5-fold performed better than 10-fold.

Another part of our motivation is to categorize the different gaming strategies of top players. Some example strategies are camping (hide in one place and wait for others to come), actively fighting (engage in more fight to collect weapons and equipment) and escape (avoid fights in order to survive till the end). We expect players who use camping strategy have less total moved distance and high final rank; on the other hand, active fighter will have high total damage and best equipment.

We first trained two unsupervised learning models, K-means and Mixture of Gaussian, and tried to observe the relationship between winning placement and strategy-related features. However, we couldn't find obvious pattern in model predicted players' categories. In other words, when plotting the strategy related features of a predicted class, there is no clear insight in data distribution. After further investigation, we realized the problem is in our data. The game data we used is post-game statistics, when means it doesn't have the power to describe what happened during the game. Given players choose strategies flexibly, we cannot expect a player stick to one strategy throughout the game. Therefore, we spend extra step in categorize players' gaming strategies holistically. By using semi-supervised Mixture of Gaussian learned in class, we manually assign 4 classes according to a few important features, and observe if these classes can differentiate winning placement. And we found the following patterns:

1. Game prefer fighters. The cluster labeled with most aggressive players has a mean winning placement over 0.95, while the most conservative players cluster only has winning placement around 0.4.
2. Don't stay in one place. Although our model indicates keep avoiding confrontation does not increase final rank, staying in one place will bring players troubles. The cluster labeled no-escape only has mean winning placement around 0.26.

Please refer following Figure 2 to see the clusters plotted with winning placement as Y-axis, and 'fights'/ 'escape' as X-axis.

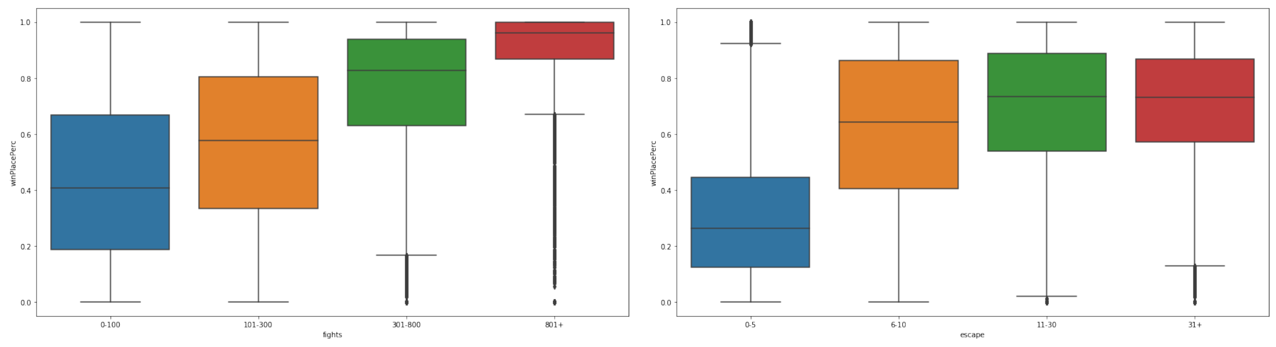

Figure 2: 'fights' and 'winPlacePerc' in different clusters (left) 'escape' and 'winPlacePerc' in different clusters(right)

6 Conclusion

With effective feature engineering and appropriate model selection, we achive decent results and observe that lightGBM performs best in predicting final ranks. Moreover, we explore some interesting gaming strategies which we will give out to PUBG players as Free Chicken Dinner! For future work, we will train the AI to play PUBG and win. With the results we have arrived at, we will train the bots to maximize the exponentially decayed sum of future rewards with data produced entirely from self-play, that is, the bots are playing against themselves to learn to play by the strategies that maximize their chances of winning. By building AIs that succeed in complex and dynamic games like PUBG, we believe it can bring humanity closer to the ultimate goal of making AGIs that function in the messiness of the real world.

7 Code

https://www.dropbox.com/sh/zuktbhf6h0fo0iz/AADU5PCw2lK5evL5RWaikBMYa?dl=0

8 Contribution

All members in our team performs extraordinary. Wenxin Wei worked on data cleaning and training with regression models. Xin Lu worked on feature engineering and training with light-GBM model. Yang Li worked on feature engineering, training with random forest, unsupervised learning algorithms and cross validation.

References

[1] "PUBG Finish Placement Prediction." https://www.kaggle.com/c/pubg-finish-placement-prediction.

[2] "Feature Scaling", https://en.wikipedia.org/wiki/Feature_scaling

[3] Chris Hans, "Bayesian lasso regression", Biometrika, Volume 96, Issue 4, 1 December 2009, Pages 835845.

[4] Hui Zou Trevor Hastie, "Regularization and variable selection via the elastic net", Journal of the Royal Statistical Society Volume67, Issue2, April 2005, Pages 301-320.

[5] AE Hoerl, RW Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems", Technometrics, Volume 12, Issue 1, 1970, Pages 55-67.

[6] "Ridge Regression", NCSS Statistical Software, Chapter 335.

[7] Andy Liaw and Matthew Wiener, "Classification and Regression by randomForest", R news, Page 1, Vol. 2/3, December 2002.

[8] fast.ai. https://www.fast.ai/.

[9] Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in Neural Information Processing Systems. 2017.

[10] Jui-Ting Huang and Mark Hasegawa-Johnson, On Semi-Supervised Learning of Gaussian Mixture Models for Phonetic Classification
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.9743rep=rep1type=pdf

[11] Feature importance of LightGBM
https://www.kaggle.com/ashishpatel26/feature-importance-of-lightgbm