PUBG Finish Placement Prediction

EE 660 Project Type: Collaborative (Kaggle) – Done individually

Ishaan Vasant, ivasant@usc.edu

Dec 5 2018

1. Abstract

This project deals with the dataset of a game called PUBG where the stats of over 4 million players are listed as features. There are 28 features in total and the goal is to predict the finish placement of all the entries in the dataset. The preprocessing steps include data exploration, label encoding and filling of NaN values. Features relevant to the target are identified from the feature heatmap. Three models are trained on the given training set - Linear Regression model, Decision Tree Regressor and Random Forest Regressor. The Random Forest Regressor performs the best on the training set and is therefore used on the given test dataset.

2. Introduction

2.1. Problem Type, Statement and Goals

Given over 65,000 game sessions worth of anonymized player data, split into training and testing sets, the goal of this project is to predict the win placement percentage of each player. The best strategy here represents the pattern of performing certain actions or a certain number of actions across the various features(columns) of the dataset that could help the player secure the top spot. If employed by the player, this would help the player finish at the top spot and predict final placement from final in-game stats and initial player ratings.

The output of the model is represented on a scale from 1(top spot) to 0(last spot) for a given player in percentage.

The problem is interesting due to the large number of data points and that the data exhibits features that interact with one another.

2.2. Prior and Related Work - None

2.3. Overview of Approach

The entire given training data was further split into training and test sets the run the various models over. A label encoder was used to label string values. The final model used for this project was the Random Forest Regressor and the mean square error performance metric was used.

3. Implementation

3.1. Data Set

The dataset given was split into test and train sets. The training set had approximately 4.45 million data points, 28 features and 1 target. The test set had approximately 1.93 million data points and 28 features. Each of the variables are described below:-

- 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. 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.
- 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.
- 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.
- 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.
- 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.
- 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. [1]

3.2. Preprocessing, Feature Extraction, Dimensionality Adjustment

After the dataset was obtained, the first step taken was to explore the dataset and its features. The describe() function was used to analyze the mean, max, min, standard deviation and percentile(25%, 50%, 75%) values of each numeric feature. The second step was to use a label encoder to encode the different 'matchType' labels numerically. The sklearn.preprocessing package was used to import the label encoder and it was applied to the training and test datasets separately. Following this, a method was used to deal with the NaN values in the dataset. For each column, the NaN values were replaced by the mean value of the respective columns. This was done for all columns except for 'Id', 'matchId' and 'groupId' features. The next step was to plot a heatmap that depicts the correlation of each feature with one another.

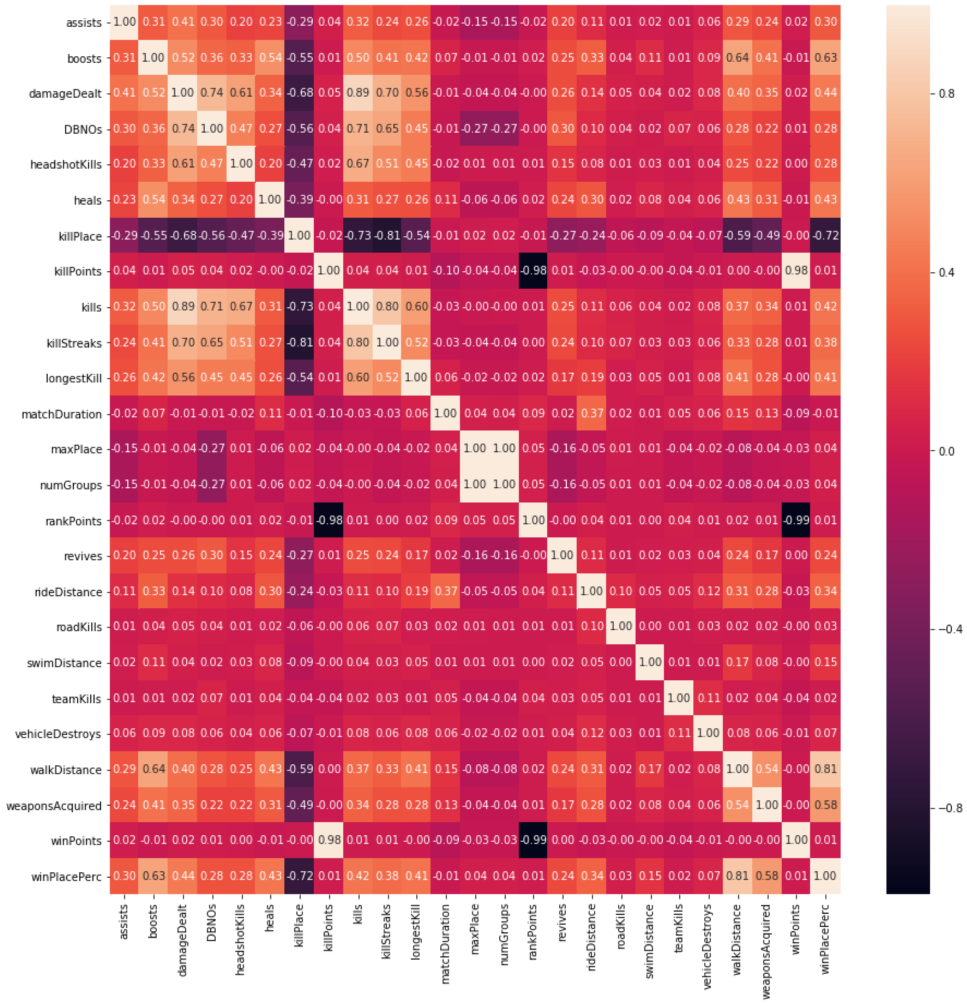

Fig: Feature Heatmap

‘winPlacePerc’ is the target to be predicted. The 5 features with the highest linear correlation with it are:-

1. walkDistance (0.81)
2. killPlace (-0.72)
3. boosts (0.63)
4. weaponsAcquired (0.58)
5. damageDealt (0.44)

3.3. Dataset Methodology

The procedure followed in the use of the dataset was a simple one. The given training dataset was split into 4 different data frames – Xtrain, Xtest, Ytrain and Ytest. This was done using the standard train_test_split() function imported from the sklearn.model_selection module. The size of the new training set was kept at 60 percent of the original training dataset and the size of the new test set was the remaining 40 percent of the original given training dataset. The model was fit and trained on the new training set and then predicted using the new test set.

3.4. Training Process

Three different models were tried and trained for this project – Linear Regression model, Decision Tree Regressor and Random Forest Regressor.

- Linear regression is the most widely used statistical technique; it is a way to model a relationship between two sets of variables. The result is a linear regression equation that can be used to make predictions about data.[2] The LinearRegression() function imported from sklearn.linear_model module was used for this method and the results are shown in the section below.
- Decision tree learning is the construction of a decision tree from class-labeled training tuples. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.[3] The DecisionTreeRegressor() function imported from sklearn.tree module was used for this method and the results are shown in the section below.
- Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual

trees. Random decision forests correct for decision trees' habit of overfitting to their training set.[4] The RandomForestRegressor() function imported from sklearn.ensemble module was used with 10 estimators and 4 tasks for this method and the results are shown in the section below.

3.5. Model Selection and Comparison of Results

As mentioned in section 3.4, three models were trained on the given training set - Linear Regression model, Decision Tree Regressor and Random Forest Regressor. Their performance is documented in the table below:-

Model	MSE Score on test set (from given training data)
Linear Regressor	0.016
Decision Tree Regressor	0.014
Random Forest Regressor	0.007

As expected, the Random Forest performed the best and since the Random Forest Regressor had the best Mean Square Error score on the training test set, the same was used on the original given test set.

4. Final Results and Interpretation

The Random Forest Regressor was used on the original given test set to predict the ‘winPlacePerc’ of all data points. When submitted to Kaggle, the performance score obtained by the model was 0.0606 (ranked 594 on the Kaggle Leaderboard) on their Ytest data frame. Minor changes in the number of estimators in the Random Forest Regressor made close to no difference on the overall performance of the model. On its own, this is a decent result for the Random Forest Regressor model. When compared to the best performing model on the Kaggle Competition Leaderboard it does fall short; best performing model had a score of 0.0167. The better performing models on the Leaderboard were neural networks, LightGBM and XGBoost.

5. Summary and Conclusions

Three models were trained on the dataset - Linear Regression model, Decision Tree Regressor and Random Forest Regressor. Each one outperformed its predecessor and finally the Random Forest Regressor was used to predict on the test set with a score of 0.0606. Using models such as LightGBM and XGBoost would definitely increase the performance and hence is an avenue to explore for the future.

6. References

[1] https://www.kaggle.com/c/pubg-finish-placement-prediction/data

[2] https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/regression-analysis/find-a-linear-regression-equation/

[3] https://en.wikipedia.org/wiki/Decision_tree_learning

[4] https://en.wikipedia.org/wiki/Random_forest