Pierre Tobgui

Final Report

**Project Information (Description, Details, Goals)** :

PUBG is a multiplayer battle royal game where about 100 players drop out of a plane into an online game which is a battleground. Ultimately, there is one winner or one team that wins depending on the game mode.

The goal for this project is to determine whether a user will win or lose in the game of PUBG. We will determine this by using past data in the game and that players current data. We want to not only determine whether the user is likely to win the game or not, but also find the most accurate method in determining this. To achieve our goal we’ll be comparing the accuracies between different machine learning algorithms. We will plot ROC curves and determine the AUC of those curves to represent the overall performance of a classifier.

The data given is over 65,000 games worth of player data. The prediction is the final placement based on in-game stats and player ratings. The data is split in test data and training data initially. The label we are trying to predict is if the user will win or not.

**Details about the data**

As stated, before we have over 65,000 games worth of data which means we have the stats of over 4 million players which are then divided into 29 data fields. Some of the data fields are self-explanatory like kills, assists, and revives while others need some explaining, here is a list of the data fields.

* DBNOs - Number of enemy players knocked down.
* 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.
* 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.
* 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.
* 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.

**Methods/Results:**

**Logistic Regression**

Since Logistic Regression is used to predict binary outcomes (1 or 0, True or False), this classifier is ill-suited to predict placements, which are continuous between 1 and 0. To obtain useful predictions out of this dataset, I aimed to instead model only the “win” state. The “win” state is a final placement (maxPlace) of 1 with the “win place percentage” (winPlacePerc) of 1.0. In other words, any placement other than 1st place is considered a loss.

The win and loss state can be defined by replacing the continuous target “winPlacePerc” with a discrete 1 or 0. Any float below <1.0 is replaced with a zero, indicating a loss state.

Number of read\_rows was capped at 100,000 rows. The training DataFrame contains 4,446,966 rows which caused system freezes when attempting to train on such a large dataset.

Features used to predict the target value were all integer and float columns excluding string columns. String columns such as “Id”, “groupId”, “matchId”, and “matchType” were identifiers that were not useful features in determining outcomes.

The training dataset was split into 75-25 training and testing cases. The Logistic Regression model was then trained and tested on only the “train\_df” dataset as well.

Prediction score was determined by the “accuracy\_score” module from the scikit.metrics library, and the result was 0.97188, obtaining 97% prediction accuracy.

**Random Forest Classifier**

The feature columns I used in testing and training the data include the features: boosts, damageDealt, headshotKills, killPlace, rankPoints, matchDuration, revives, numGroups, winPoints. I tried using different combinations of features and the ones I felt were more related to a player winning. After trying different combinations these yielded the best results without using up too much time since the data set we used was very large.

For the test and train split I split the data into 40% testing and 60% for training. I did so because this yielded the more accurate results without using too much time to run. For instance, if I made the training data even larger, it would take a very long time to run and yield accuracy results that were virtually the same.

Random Forest Classifier fits a number of decision tree classifiers and uses averaging to improve the predictive accuracy. Our results for decision tree was approximately 95% for prediction accuracy. After doing Random Forest with 19 estimators and random state of 3. The prediction accuracy was increased to 0.9705 which is approximately 97%.

After computing the accuracy for the predictions, I computed the ROC curve for the Random Forest Classifier. The area under the curve given was 0.87. The ROC curve is given by the true positive rate over the false positive rate.

**Decision Tree Classifier**

I ended up using three combinations of the features given, based on aggressive, non aggressive, and overall stats. The aggressive stats were the features 'assists', 'damageDealt', 'DBNOs', 'headshotKills', 'killPlace', 'kills', 'killStreaks', and 'roadKills'. The non aggressive stats were 'boosts', 'heals', 'matchDuration', 'revives', 'rideDistance', 'swimDistance', 'walkDistance'. While the overall stats were a combination of both feature sets.

The data of over 4,446,966 rows was split into 25% percent for the test size while the training size was 75%. The prediction accuracy was 95.8% for aggressive stats, 95% for non aggressive stats, and 95.6% for the stats overall. The small difference in percent accuracy would lead us to believe that neither stat grouping has an advantage over the other. After calculating the roc curve for each feature combination the average area under the curve was 0.60 with the true positive rate over the false positive rate.

**Linear Regression**

For this method we ignore the columns "Id", "groupId", "matchId", and "matchType" as they are used for identifying each specific matching in the dataset and will not be useful for determining the most and least important features or the performance of the method. After generating the feature matrix and normalizing the features, I split the dataset using the train\_test\_split with parameters of test\_size = 0.25 and random\_state = 6. After training the dataset, I found the the least important feature was “swimDistance” and the most important feature was “rankPoints” with the coefficients being -0.00030911 and 0.07529613 respectively. Then I predicted the “winPlacePerc” for the data and compared it with the actual “winPlacePerc”, this yielded a RMSE value of 0.15265547213740976 (~15.26%) meaning that it was a pretty accurate prediction. Finally I used 10-fold Cross Validation to verify the performance of the Linear Regression method and it yielded a RMSE value of 0.152954463992816 (~15.29%) which was almost identical to the RMSE previously obtained meaning that the average error margin is roughly 15%.