Final Project Proposal-Who wins the game? PUBG ranking prediction

Jingling Yang, Jingming Wei

1/24/2020

Section I. Introduction

PlayerUnknown's Battle grounds(PUBG) is a first/third person shooter battle royale style game where over 90 players in a single match try to survive to the end to win the game. Each player can play solo, duos, or squads. Players parachute from an airplane to land on distinct resource points to scavenge for equipment, weapons, first-aid, and vehicles. A circle, the boundary of the safe zone will appear a few minutes after the start of the game, and people outside the safe zone will suffer chronic damage. The safe zone will shrink as the game time elapses to enforce encounters of players. Players in the game can adopt different strategies of survival. They can choose to battle against any encounters, or evade hot battle grounds and take a zigzag path approaching the safe zone. This project aims to use machine learning methods to predict PlayerUnknown's Battlegrounds(PUBG) players' rankings in a game.

Typically, each player is ranked based on survival time in a match after the end of the game. And winning a position in top 10 is considered a good play. For this project, the task is divided into regression analysis and classification analysis. First, we define our dependent variables as whether a player in a match is ranked top 10. We aim to predict future players' probability of winning top 10. Secondly, we treat the ranking as numeric variables, typically ranged from 1 to 100(or less). The task is rather challenging since we here try to predict the exact ranking of a single player.

Section II. Data description and some preliminary analysis

The data titled *PUBG Finish Placement Prediction* is extracted from Kaggle. The data procider scraped the data from the PUBG API and well compiled them into two data sets, a train set with 1,934,174 observations and a test set with 1,048,575 observations. Each row of both data sets contains relevant statistics of each player in a single game. For our future analysis, we might scrape more features from the API to include more features for predictive purposes. But here, as an illustration of our research ideas, we just used the well compiled data for simplicity. The column names and meanings are summarized below.

```
#import the data both training and testing
train <- read.csv("/Users/jingmingw/Desktop/Project Proposal/pubg competition dataset/train_V2.csv")
test <- read.csv("/Users/jingmingw/Desktop/Project Proposal/pubg competition dataset/test_V2.csv")
```

Virable Name	Type	Meaning
Identificators		
ID	factor	Player's ID
group ID	factor	ID to identify a group within a match
match ID	factor	ID to identify a match
Responses		
winPlacePerc	numeric	Percentile winning placement, where 1 corresponds to 1st place, and 0 correspond to the last place in the match.
Preditors		
assists	interger	Number of enemy players this player damaged that were killed by teamates
boosts	integer	Number of boost items used
damageDealt	numeric	Total damage dealt. (Self inflicted damage is subtracted)
DBNOs	integer	Number of enemy players knoecked
headshotKills	integer	Number of enemy players killed with headshots

Virable Name	Type	Meaning
heals	integer	Number of healing items used
killPlace	integer	Ranking in match of number of enemy players killed
killPoints	integer	Kills-based external ranking of a player
killStreaks	integer	Max Number of enemy players killed in a short amount of time
kills	integer	Number of enemy players killed
longestKill	numeric	Longest distane between the killer and the victim at time of death
matchDuration	integer	Duration of match in seconds.
matchType	factor	String identifying the game mode that the data comes from
numGroups	integer	Number of groups in the match
rankPoints	integer	Elo-like ranking of a player
revives	integer	Number of times the player revived teammates
rideDistance	numeric	Total distance traveled in vehicles measured in meters
roadKills	integer	number of kills while in a vehicle
swimDistance	numeric	Total distance traveled by swimming measured in meters
teamKills	integer	Number of times the player killed by a teammate
vehicleDestroys	integer	Number of vehicle destroyed
walkDisrance	numeric	Total distance traveled on foot measured in meters
weaponsAquired	integer	Number of weapons picked up
winPoints	integer	Win-based external ranking of the player.

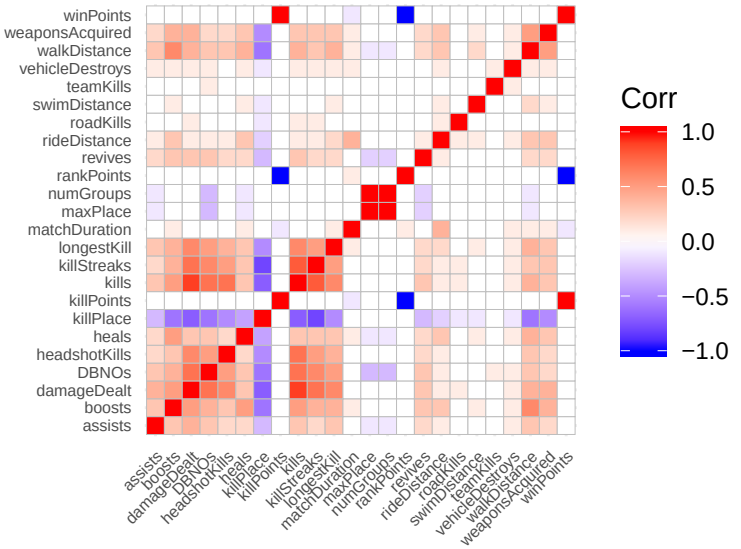
1. Check for collinearity

Then, we use pearson correlation to detect possiblly highly colinear features. We here only include a subset of numeric features and exclude our response variable **winPlacePerc**.

```
#summary statistics using pearson correlation
library(ggcorrplot)
```

```
## Loading required package: ggplot2
```

```
response <- names(train) %in% c("winPlacePerc")
features <- train[!response]
corr <- round(cor(features[unlist(lapply(features, is.numeric))]), 1)
ggcorrplot(corr,tl.cex=6)
```



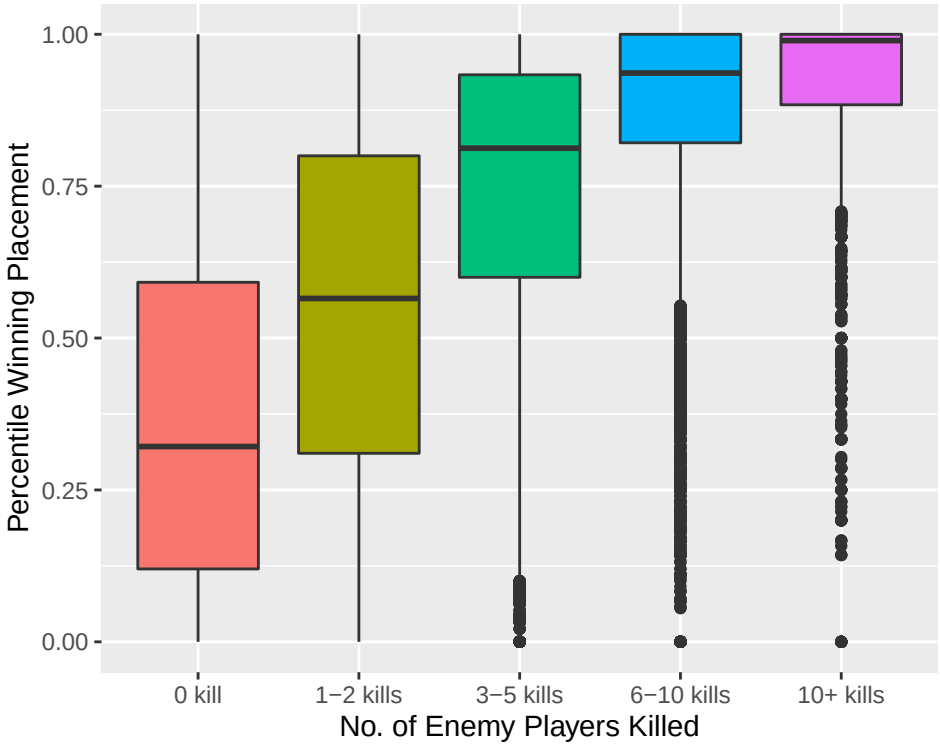
As we can see from the above graph, we need to exclude one of **numGroups** and **maxPlace**, and one of

killPoints and **winPoints** before proceeding to our future analysis.

2. Distribution of placement conditional on kills

In the train set, our response variable is percentile winning placement. It's highly likely that the number of kills is highly predictive of the final outcome. However, it's sometimes the case that the final winner survive without even killing many people by adopting a hiding strategy. So, it would be intersting to look at the distribution of ranking conditional on number of kills.

```
# plot
library(ggplot2)
train$group <- cut(train$kills,c(-1,0,2,5,10,max(train$kills)))
ggplot(train, aes(x=group, y=winPlacePerc, fill=group)) +
  geom_boxplot() +
  theme(legend.position = "none")+
  labs(x = "No. of Enemy Players Killed",y="Percentile Winning Placement")+
  scale_x_discrete(labels=c("0 kill","1-2 kills","3-5 kills","6-10 kills", "10+ kills"))
```



As we can see from the box plot above, although there is some overlap of rankings between different kill-number groups, it's generally the case that more kills are associated with a higher ranking. Though interestingly, there are many outliers in the 10 plus-kill group, implying that hiding might be sometimes a better strategy.

Section III. Working flow and methods

Admittedly, we have not explored PUBG API deeply. We might expand our data sets by scraping more predictors from the interface. But here, we used the data sets previously described to describe our working flows. (Please see attched for our flow chart)

Ideally, after we get our cleaned full data sets, we will randomly split the data sets into a train and a test set by a ratio of 7:3. We train our models in the train set and leave the test test for final scoring and comparison. Here, we just used the train and test data sets imported from Kaggle.

Before running regression and classification analysis, we wan to produce more features by unsupervised learning methods. For example, we can use clustering to classify players into different proficiency levels as a

new factor predictor. Based on that, we can create a new dummy indicating whether or not a master player is in the duo or squad, which might be crucial in predicting team wins. Hopefully, we can explore more underlying behaviors between features after we get the full collection of our data sets.

For our regression analysis, we first use OLS regression including all predictors. Though OLS is simple, it cannot help us differentiate the importance of the variables and thus risks the curse of high dimension of our features. As a potential improvement, we adopt forward and backward selections to identify the subset of features that are most statistically significant. Further, we use Ridge and Lasso regression for feature selections which penalize small coefficients. Tree based methods including regression tree, bagging, and random forests. In particular,

1. OLS: we do regression on all of the predictors to get the marginal effects of each predictor. We then apply those coefficients to the predictors in the test set to get predicted rankings, which are then compared with the true rankings for the calculation of prediction accuracy.
2. Forward and backward selection: for both cases, we set the stopping criterion to be $p = 0.01$.
3. Ridge and Lasso: we will use 10-fold cross validation to tune the parameter λ. After we get the optimal λ and feed it into our train set, we get the coefficient estimates. We can then apply those estimates into our test set for scoring.
4. Decision trees: we will use 10-fold cross validation to tune the parameter α, which dictates the penalty associated with the number of nodes included. After we get the best tree corresponding to a specific alpha, we apply the tree to our test set for scoring.
5. Bagging: We re-sample our train sets n times and each time, we grow decision trees without pruning and take the average of the prediction results. We iterate over different values of n to get the best n by some criteria. Then we apply the bagged trees to our test set for scoring.
6. random forests: The only difference between bagging and random forest is that when we are building our tree, at each node, we need to randomly choose a subset of predictors. Similarly, we tune the parameter n as described in Bagging above. Then we apply the best random forest (corresponding to a specific n) to the test test.

For our classification analysis, we will adopt KNN, Logit and LDA, and tree based methods stated in the previous paragraph. Tree based methods should follows the similar logic. So we only discuss the first three classifiers here.

1. KNN: we tune the parameter k, which determines the number of members included in a neighborhood. Then we appl
2. Logit and LDA: it would be interesting to compare these two methods since they basically follow the similar structures but differ in assumptions under which each is performed.

Reference

"PUBG Finish Placement Prediction (Kernels Only)." Kaggle. Accessed January 24, 2020. https://www.kaggle.com/c/pubg-finish-placement-prediction/data.

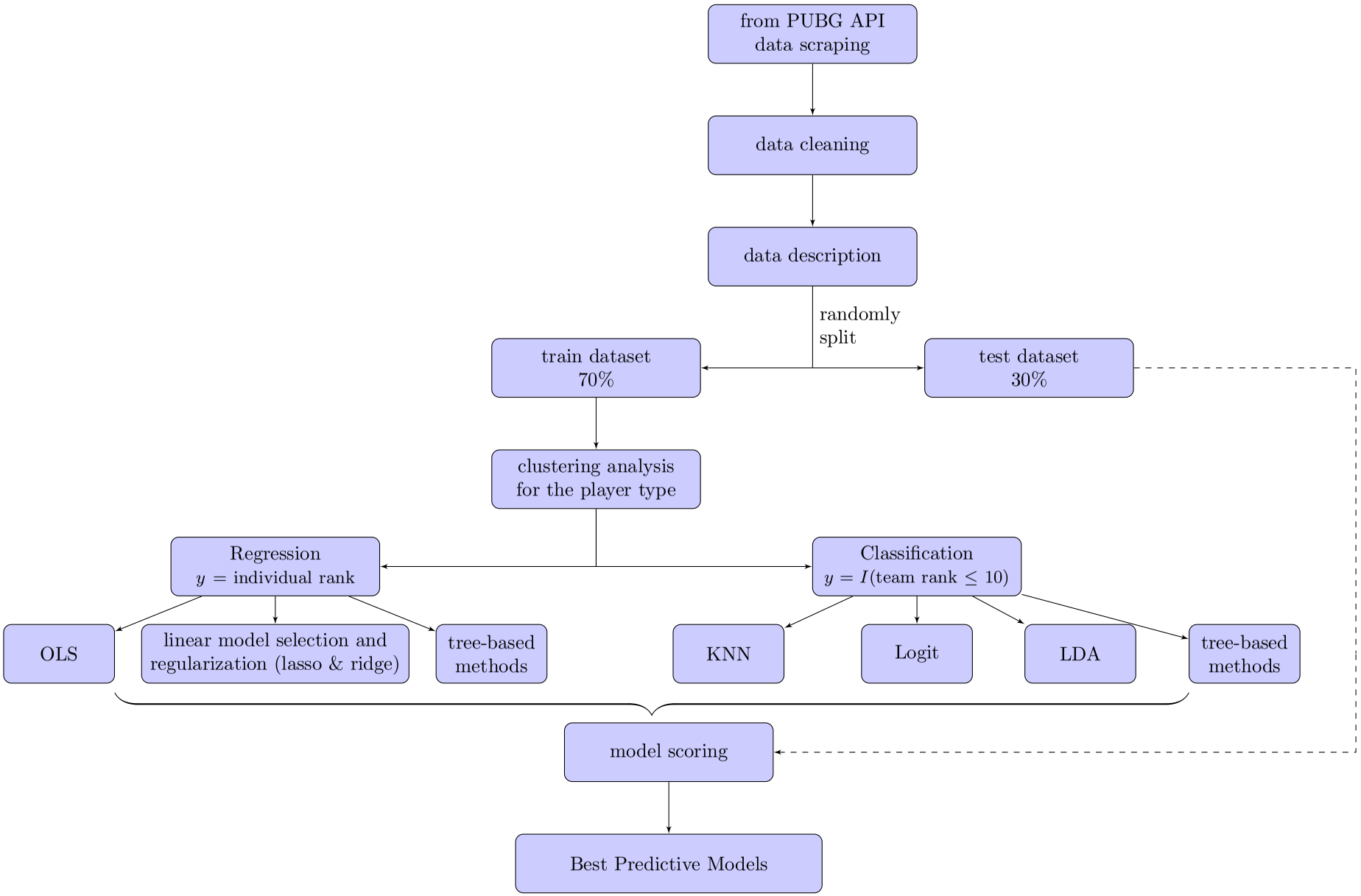


Figure 1: Work flow Chart