ANALYSIS ON PUBG WIN PLACE PREDICTION

Group member: Xianghui Hou, Siyi He, Yifei Xu, Yi Wang

1. INTRODUCTION

PUBG is a Battle Royale-style video games which is prevalent over the world. In this game, there will be 100 players. Two standard modes are available to be chosen: first person perspective and third person perspective. Each mode has three team types which are solo, duo and squad. One of each player can carry two different guns, several flares and different first aid items. Through shooting, exploring and protecting team member, there will be corresponding ranking for each team.

In this research, we tend to focus on what feature can be a reason to cause the ranking of each group, and fit a relatively reliable model with small error rate to predict rankings for future PUBG games. Tested models are: linear regression, gradient boost machine, random forest, xgboost and stacking. It is easy to select the best model by comparing RMSE (root-mean-square error). Smaller RMSE the model has, more accurate the model is.

2. EMPIRICAL DATA ANALYSIS

2.1 BASIC INFORMATION ABOUT DATA

We pick the competition named *PUBG Finish Placement Prediction* as our project. This competition has two data sets, test and train datasets.
The train data set has 4446966 observations and 29 variables. The test dataset has 1934174 observations and 28 variables.

2.2 VARIABLE INTRODUCTION

These 29 variables in the original train data are some typical factors in the game PUBG, the introduction of the variables are as follows:

1) DBNOs - Number of enemy players knocked.
2) assists - Number of enemy players this player damaged that were killed by teammates.
3) boosts - Number of boost items used.
4) damageDealt - Total damage dealt. Note: Self inflicted damage is subtracted.
5) headshotKills - Number of enemy players killed with headshots.
6) heals - Number of healing items used.

7) killPlace - Ranking in match of number of enemy players killed.
8) killPoints - Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.)
9) killStreaks - Max number of enemy players killed in a short amount of time.
10) kills - Number of enemy players killed.
11) 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.
12) matchId - Integer ID to identify match. There are no matches that are in both the training and testing set.
13) revives - Number of times this player revived teammates.
14) rideDistance - Total distance traveled in vehicles measured in meters.
15) roadKills - Number of kills while in a vehicle.
16) swimDistance - Total distance traveled by swimming measured in meters.
17) teamKills - Number of times this player killed a teammate.
18) vehicleDestroys - Number of vehicles destroyed.
19) walkDistance - Total distance traveled on foot measured in meters.
20) weaponsAcquired - Number of weapons picked up.
21) winPoints - Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.)
22) groupId - Integer 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.
23) numGroups - Number of groups we have data for in the match.
24) maxPlace - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
25) 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.
26) matchDuration - Duration of match in seconds.
27) 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.

Also, we find that variable winPlacePerc is the variable that test dataset does not contain, so this one is our response to predict in this project.

2.3 CORRELATION ANALYSIS

In this part, our goal is to analyze the variables that have strong correlation with the response “winPlacePerc”. First, we load package highcharter to plot a correlation chart of all our variables.

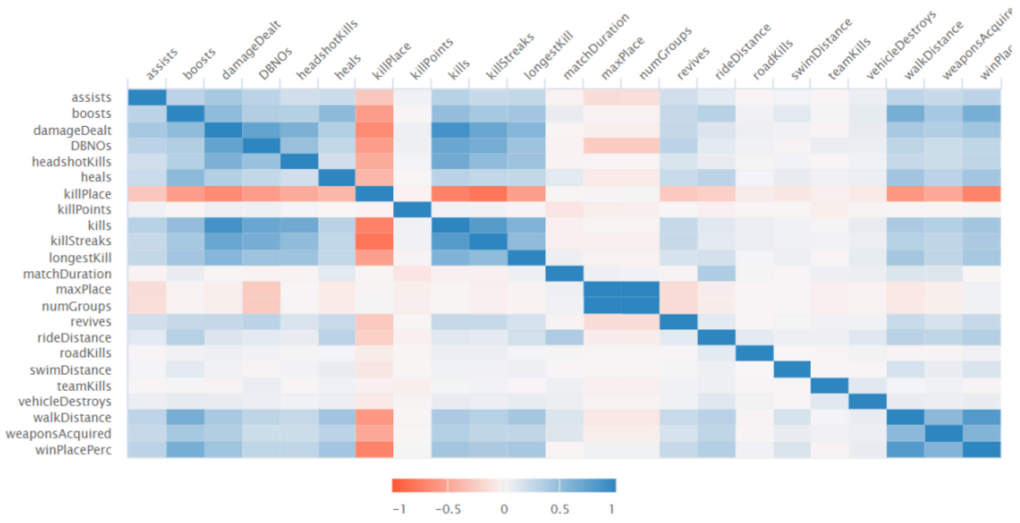


Figure 1: Correlation Plot

From the plot, walkDistance (0.81), killPlace (-0.72) and boosts (0.63) have a strong relationship with winPlacePerc. Next step, we do analysis that focus on these factors.

2.3.1 WALKDISTANCE FACTOR

2.3.1.1 POINT FOR WALKDISTACE

In theory the players who survive to the end should walk longer distances, so we plot walkdistance and winPlacePerc with the point counting to see whether they have a relationship.

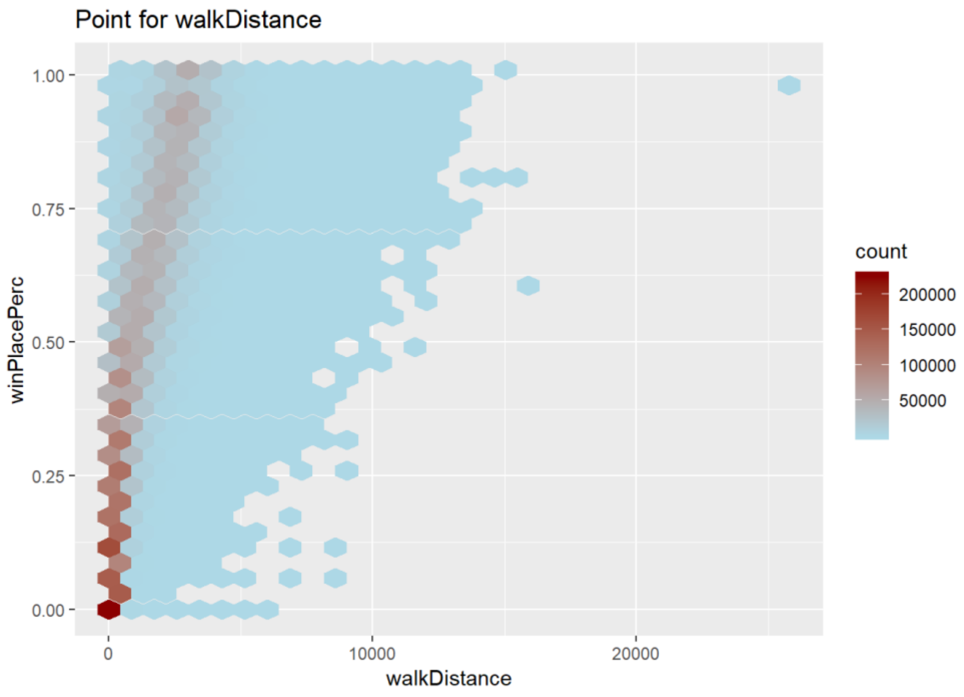


Figure 2: walkDistance and winPlacePerc

We see that the players with higher scores on the chart will walk longer distances,but more players will walk less than 3000,even for the winning players.This shows that blindly walking is not the right choice,and more often the winner will choose a reliable stronghold to ambush.

2.3.1.2 DISTRIBUTION OF METERS PER HOUR

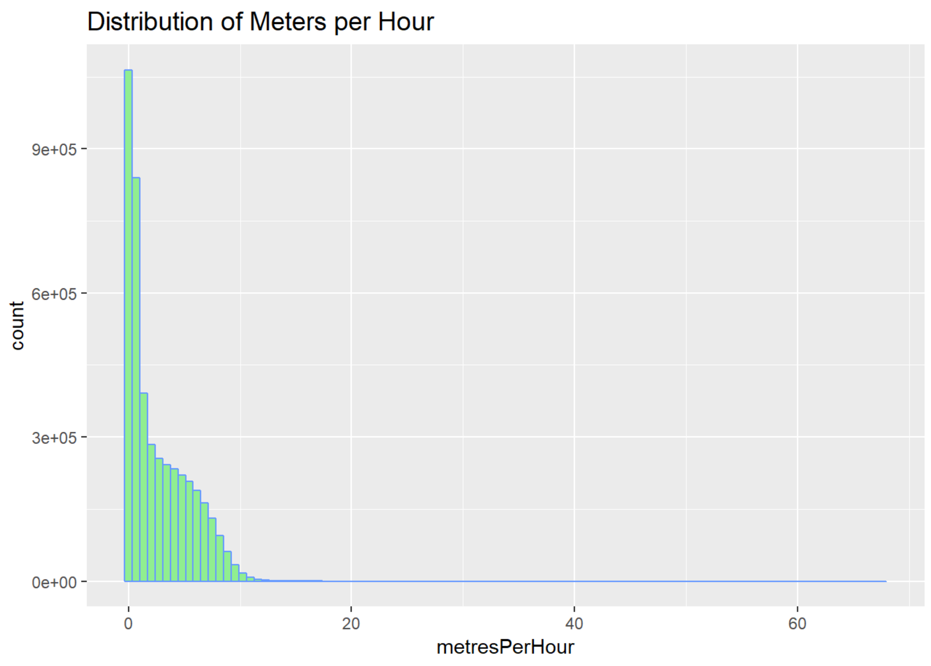


Figure 3 : Distribution of Meters Per Hour

From this plot, we find that most players travel no more than 22km/h. However, there are some people travel more than 22km/h, which is very impractical. We assume that these players are cheaters (there are quite a lot of cheaters in PUBG!)

2.3.2 KILLPLACE FACTOR

The variable killPlace means ranking in match of number of enemy players killed.

In the game,the more killPlace there are,the more advantages there are,and the easier it is to win the game.The red area in the chart is divided into three parts,which can be summarized as:

1) Your killPlace is low,and your score is likely to be below 0.5.
2) Players with killPlace between 20 and 40 score between 0.5 and 0.8.
3)Players with killPlace top 25 score above 0.8.

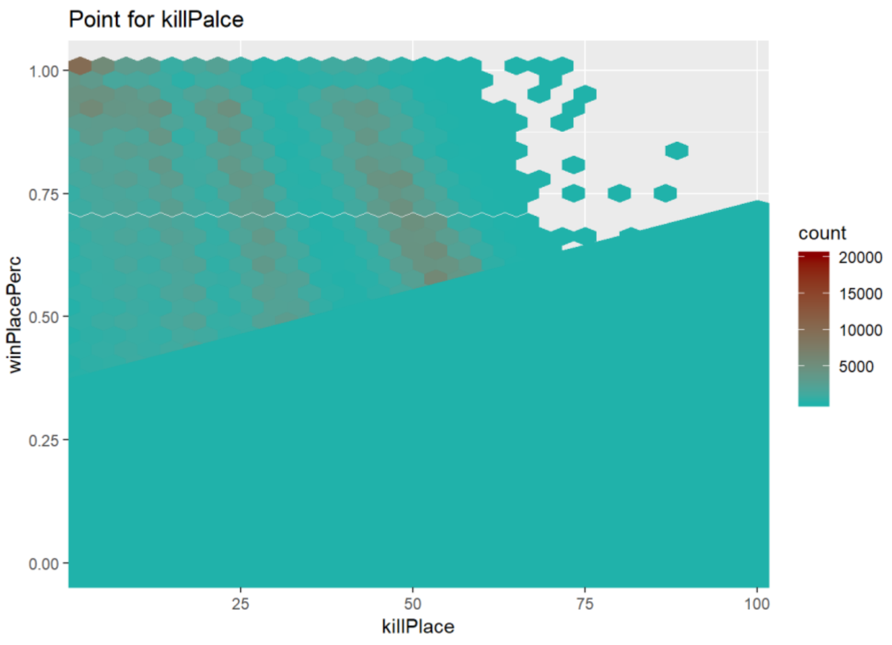


Figure 4 : Point for killPlace

2.3.3 BOOSTS FACTOR

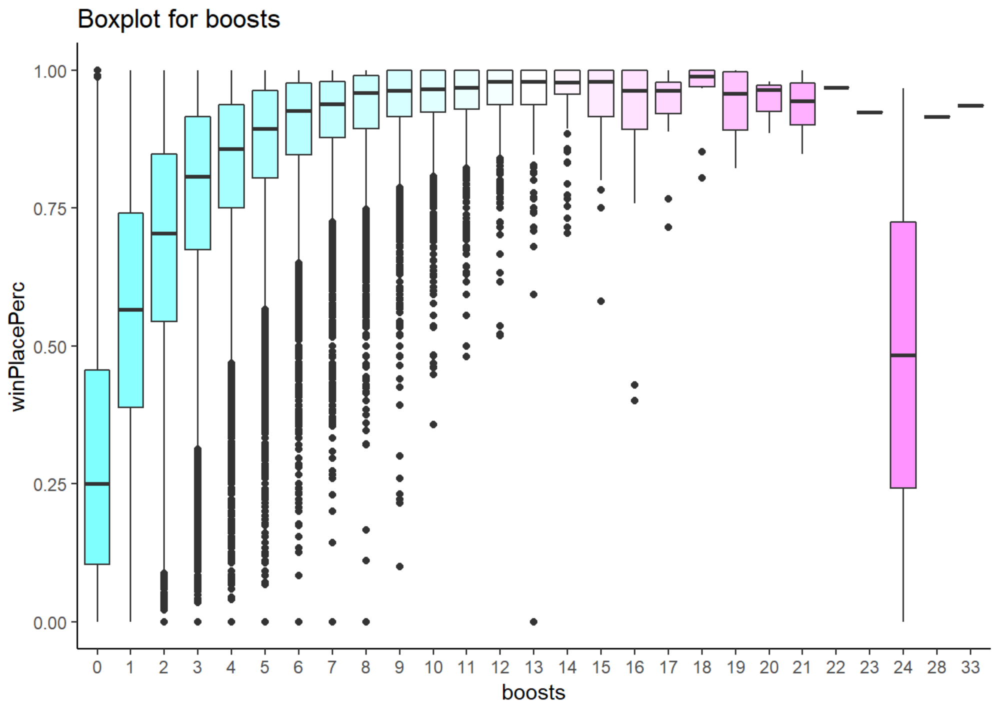


Figure 5 : Boxplot for Boosts

The variable boosts means number of boost items used.

To survive as a winner must actively leverage other items to increase their power, which is why the early stages of the game collect more resources.

From the plot, we see that more boosts a player used, the higher win place a player gained. There is an outlier that have 24 item boosts, but did not have a good win place.

2.4 PRIMARY PCA ANALYSIS

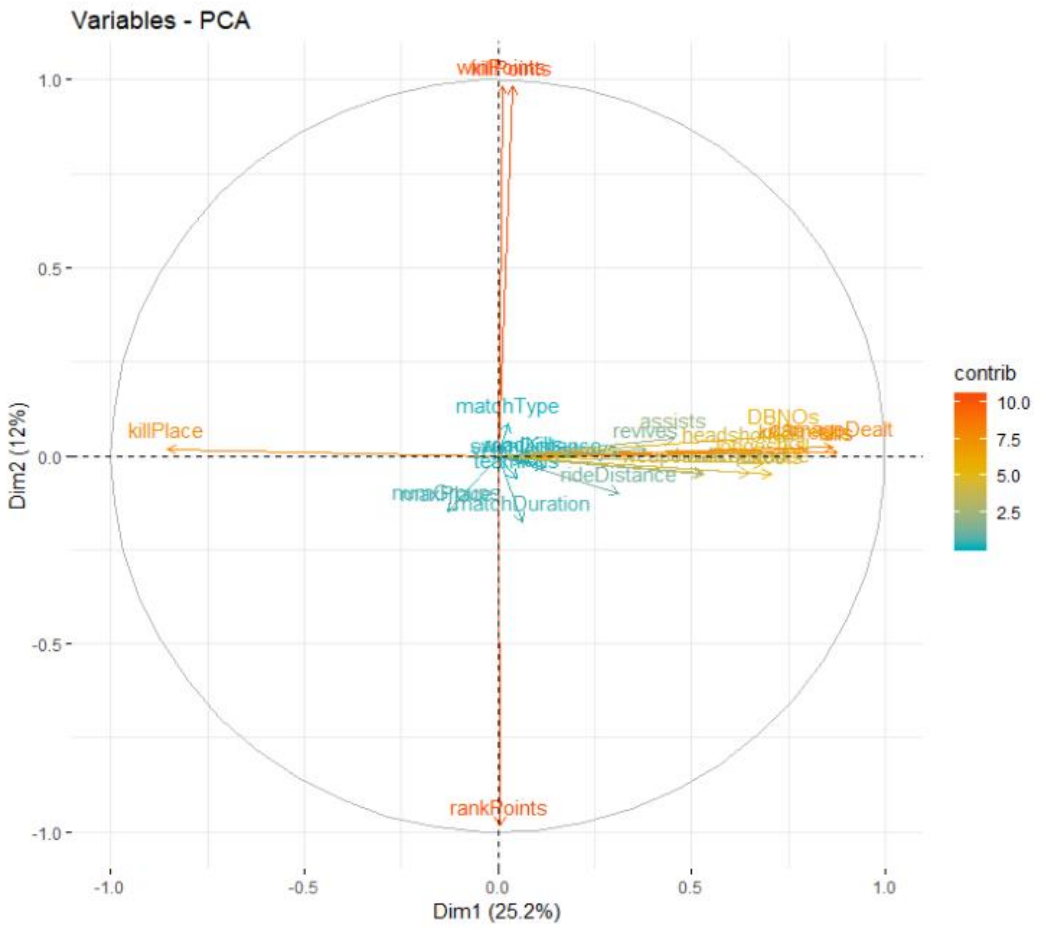


Figure 6 : Variables for PCA

As we can see, dimension 1 and dimension 2 are the most important dimensions. For these 2 dimensions, we want to explore which variables make the most contributions to the dimension. From the plot, the longest arrows must be observed to determine which variables are most important.

For dimension 1, damageDealt, killPlace, kills and killStreaks are the most contributed factors. As they are all related to the kill ability, we can rename dimension 1 as “kill ability”.

For dimension 2, killPoints, rankPoints and winPoints are the most contributed factors. Moreover, their points are around 32 while the other variables are only around 0.2. From the plot we can see that they have outstandingly long arrows.

2.5 WEIRD SITUATIONS ANALYSIS

In PUBG, there are two "special" groups of players, that is, zombies and cheaters. Zombies are players disconnected from start to end. Cheaters are players using "super powers" to gain more kills and win a good place in game. As these two groups are very different from normal players, they need to be extracted from the data set, and to be observed and analyzed alone.

2.5.1 ZOMBIES

We define a player that has 0 walk distance and 0 weapon acquired as zombies.

First, we want to explore how often zombies appear in a game.

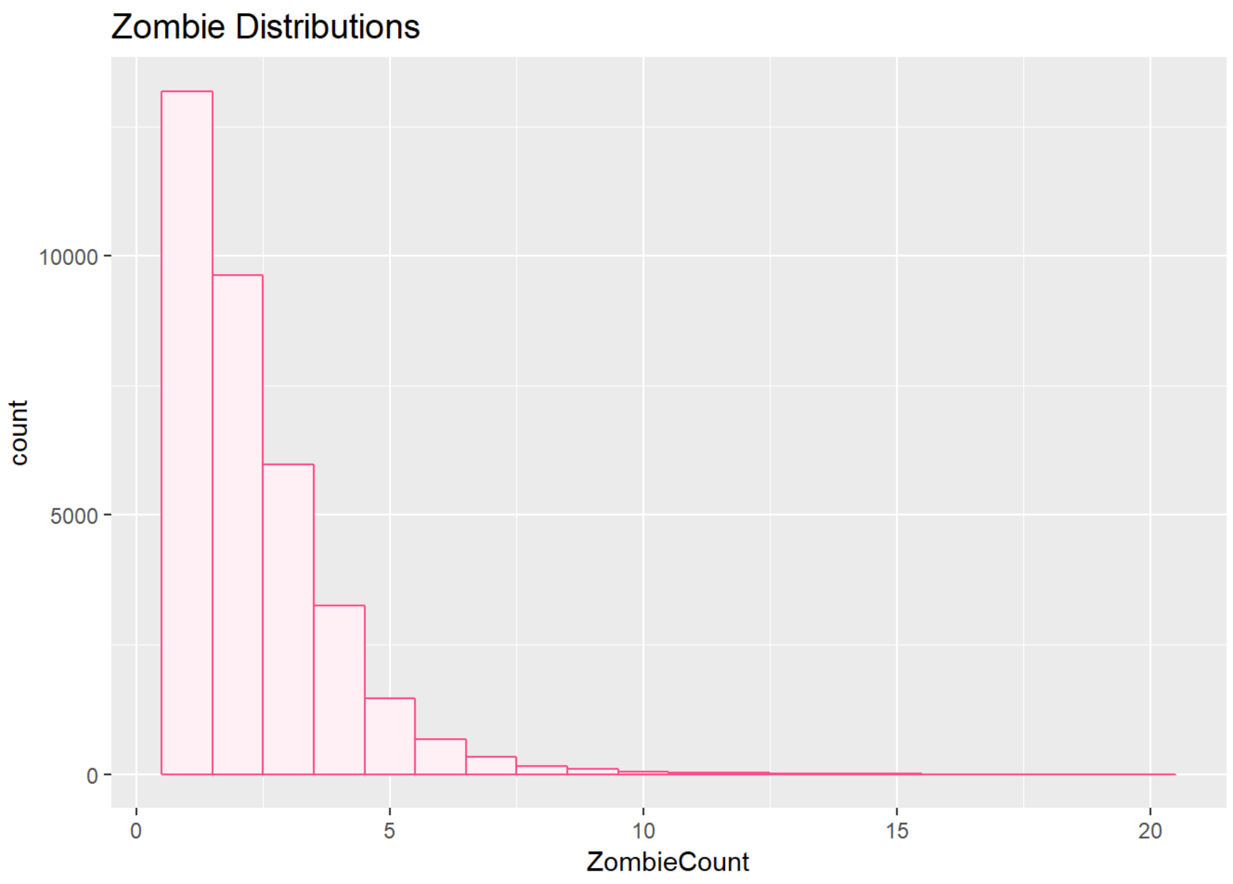


Figure 7 : Zombies Count

By using R, we find that 72.73% of matches in train data have zombie players. Also, from the plot, we observe that several zombies can appear in a same game.

Then, we want to explore the win place for the zombies. From the plot, we observe that the zombies are often first to die (most of them gained 0.00 win place percentile). However, we see that some teams with zombie(s) still won the game.

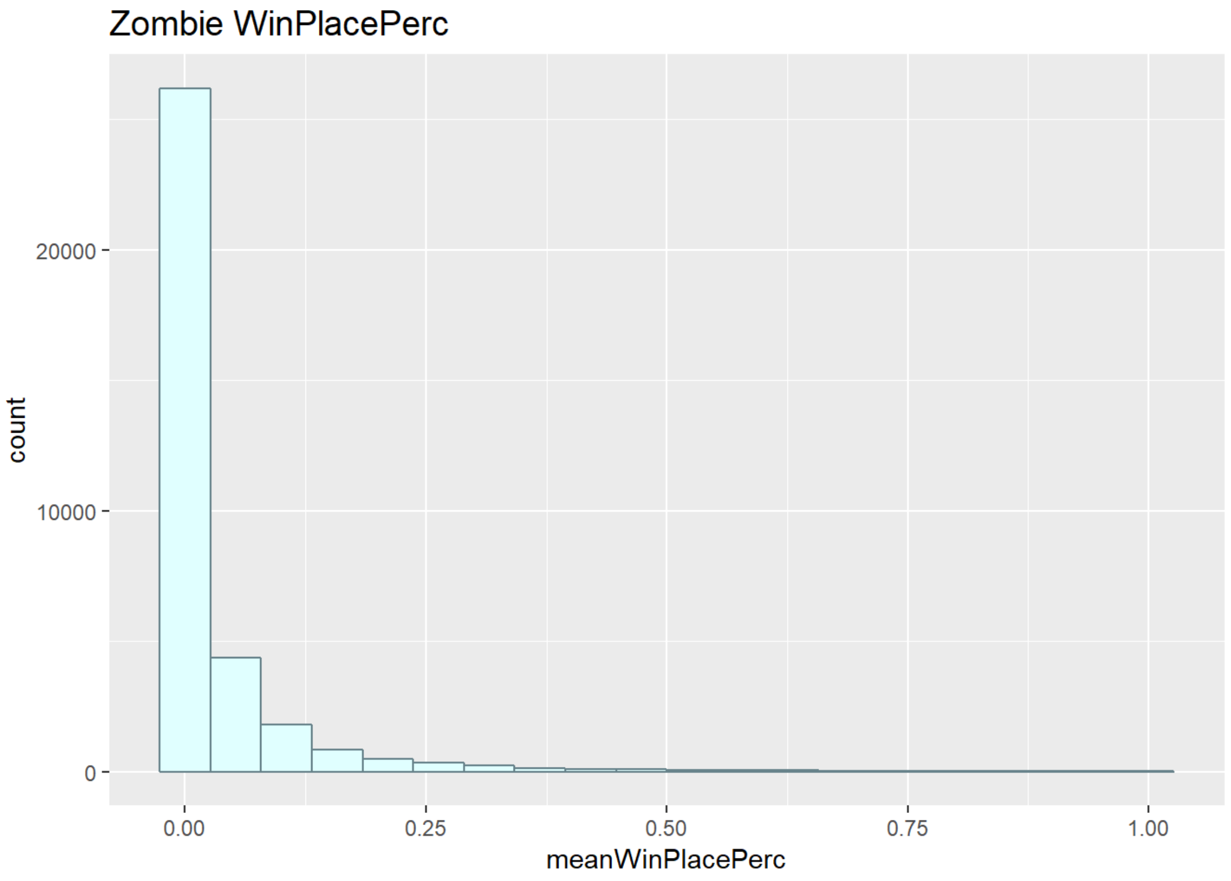


Figure 8 : Zombies Win Place

2.5.2 CHEATERS

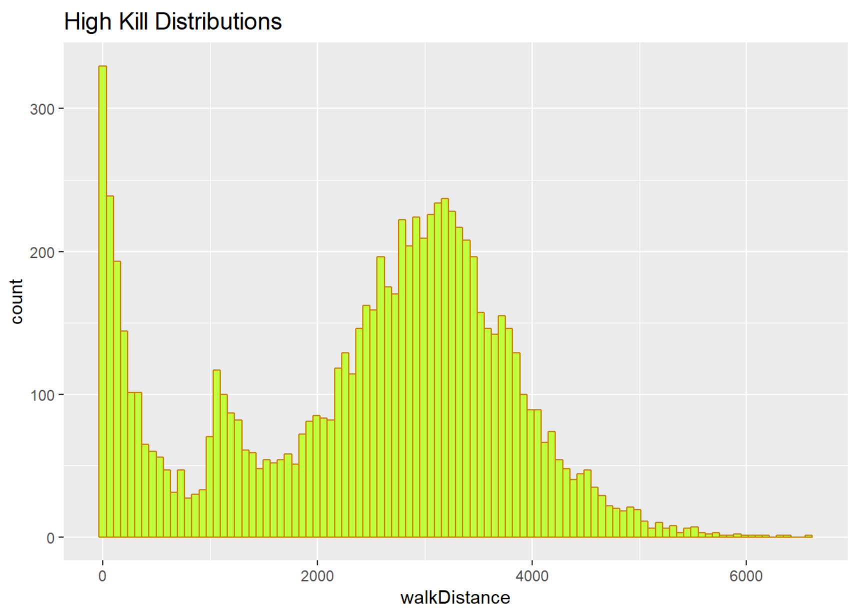


Figure 9 : Cheaters Walk Distance

I would like to mark those who kill 10 or more people within a game as a possible cheater (for I never kill such many people in one single game.)

First, we want to take a look at the walk distance of those players who have high kills. From figure 9, we observe that most cheaters have a walk distance around 3000m which is higher than the average of normal players.

Then, we want to explore how many headshots the cheaters could do. From figure 10, we observe that most of the cheaters have around 6 headshots in a single game. That is too high to be practical in a FPS game, because the top of professional players only have a headshots rate at 20%.

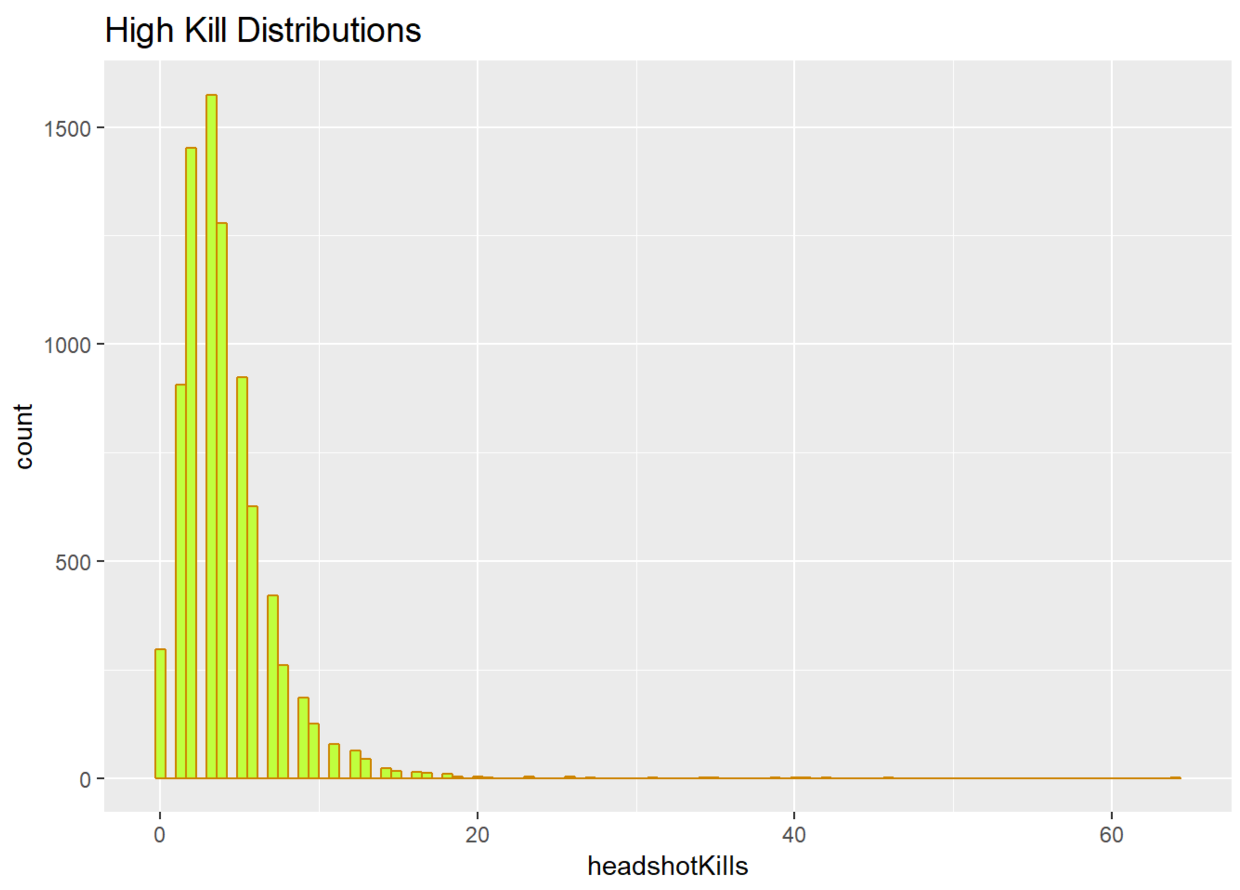


Figure 10 : Headshots Kills for Cheaters

3. PRIMARY FEATURE SELECTION AND MODEL

3.1 FEATURE SELECTION

3.1.1 PRIMARY FEATURE SELECTION RESULT

Besides some variables in the original dataset, we created some new features for further exploring. We redefine “walkdistance” that player with walkdistance per hour over 22km/hour is regarded as cheater, and set it to be 22000; others keep the original walkdistance. Furthermore, we define variables “combatScore” and “teamWorkScore” which give an overall score for each player regarding team cooperation. Also, we calculate average distance per minute, the amount of weapon that each player has per minute and the amount of collected item for each player. We define each of these three variables as: “averageDistancePerMinute”, “weaponPerMinute”, “itemsCollected”.

We find mean, minimum and maximum of the ranking for each of the existing variables within each group. For instance, if the number of vehicles destroyed for each member in a certain squad team is 1, 0, 0, 1. The ranking within this group should be 1, 3, 3, 4. Then, minimum of the ranking should be 1. The corresponding new variable “vehicleDestroysminRank is equal to 1.

The following features are the results of primary feature selection.

From 1) to 19), these are the mean of the ranking for each of the existing 19 variables within each certain group.

1) assistsMeanRank
2) boostsMeanRank
3) damageDealtMeanRank
4) DBNOsMeanRank
5) headshotKillsMeanRank
6) healsMeanRank
7) killPlaceMeanRank
8) killsMeanRank
9) longestKillMeanRank
10) revivesMeanRank
11) rideDistanceMeanRank
12) roadKillsMeanRank
13) swimDistanceMeanRank
14) teamKillsMeanRank
15) vehicleDestroysMeanRank
16) walkDistanceMeanRank
17) weaponsAcquiredMeanRank
18) rankPointsMeanRank
19) combatScoreMeanRank

From 20) to 38), these are the minimum of the ranking for each of the existing 19 original variables within each certain group.

20) assistsminRank
21) boostsminRank
22) damageDealtminRank
23) DBNOsminRank
24) headshotKillsminRank
25) healsminRank
26) killPlaceminRank
27) killsminRank
28) longestKillminRank
29) revivesminRank
30) rideDistanceminRank
31) roadKillsminRank

32) swimDistanceminRank
33) teamKillsminRank
34) vehicleDestroysminRank
35) walkDistanceminRank
36) weaponsAcquiredminRank
37) rankPointsminRank
38) combatScoreminRank

From 39) to 57), these are the maximum of the ranking for each of the existing 19 original variables within each certain group.

39) assistsmaxRank
40) boostsmaxRank
41) damageDealtmaxRank
42) DBNOsmaxRank
43) headshotKillsmaxRank
44) healsmaxRank
45) killPlacemaxRank
46) killsmaxRank
47) longestKillmaxRank
48) revivesmaxRank
49) rideDistancemaxRank
50) roadKillsmaxRank
51) swimDistancemaxRank
52) teamKillsmaxRank
53) vehicleDestroysmaxRank
54) walkDistancemaxRank
55) weaponsAcquiredmaxRank
56) rankPointsmaxRank
57) combatScoremaxRank

From 58) to 66), these variables are created regarding different team groups.

58) partySize: group size
59) maxPlace: maximum of worst placement we have data for in a certain match.
60) averageDistancePerMinute: mean calculated by average distance per minute of each member in a certain group.
61) weaponPerMinute: mean calculated by average weapon per minute of each member in a certain group.
62) ItemsCollected: mean calculated by the number of weapon, heals and boosts that each member has in a certain group.
63) killStreaks: mean calculated by percentage of max number of enemy players killed in a short amount of time of each member in a certain group.

64) isCheater: If the total walk distance of a group is greater than 2600 (average walk distance from sample dataset), we define variable “isCheater” as 1; elsewise as 0.
65) zombies: if the total walk distance and weapon number of a group are bothe equal to 0, we define “zombies” as 1; elsewise as 0.
66) numGroups: maximum of number of groups we have data for in a certain match.

From 67) to 71), the variables are dummy which only has 0 or 1. 1 represents yes; 0 represents no. For instance, no matter a player joins “solo-fpp” or “solo-fpp”, he does joins a solo game, so “isSolo” is chosen to be 1.

67) isSolo
68) isDuo
69) isSquad
70) isCrash
71) isFlare

3.1.2 FEATURE EXTRACTION

3.1.2.1 LASSO

Now, we have 71 features which is a lot. To make the model easier to interpret, work faster, and reduce overfitting, we did feature selection by using LASSO method. If there are grouped variables (highly correlated between each other) LASSO tends to select one variable from each group ignoring the others. During features selection process the variables that still have a non-zero coefficient after the shrinking process are selected to be part of the model. From Figure 11, we can clearly see the values of the best model of 27 variables can be extracted. After select the most informative features, we use linear cross validation to check the error decreases.

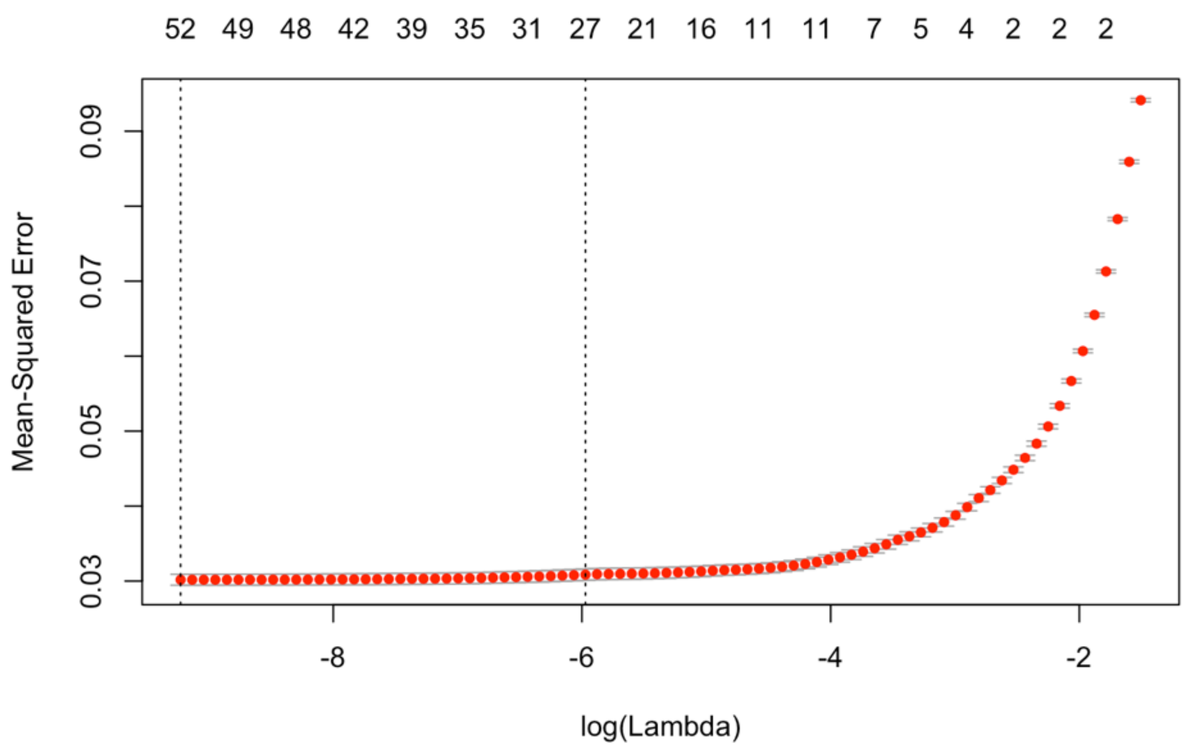


Figure 11 : LASSO

3.1.2.2 PCA

Since we have a high-dimensional dataset (with 27 variables), applying principle component analysis (PCA) is a good way to extract important variables (in form of components) from a large set of variables.

From Figure 12, as we can see, the dimension 1(PC1) is the most important . The long arrows ("vehicleDestroysMeanRank", "killPlaceMeanRank"," teamKillsmaxRank" and etc.) are determined to be important variables.

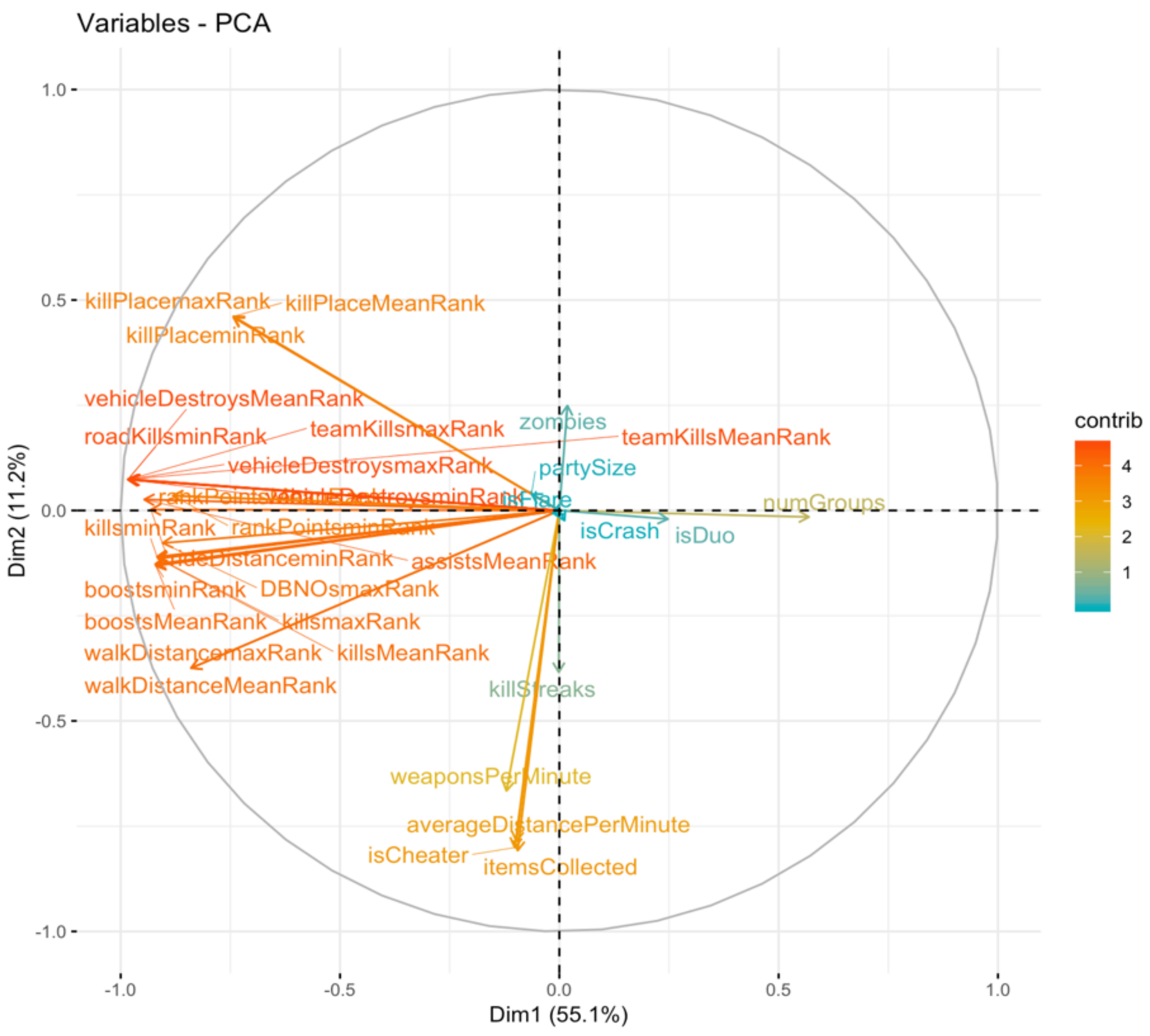


Figure 12 : PCA

Specifically, the contributions of PCA shows for dimension 1, the most explanatory variables are “teamKillsMeanRank”,”vehicleDestoryMeanRank”,”teamKillsmaxRank”, ”vehicleDestoryminRank ”,”vehicleDestorymaxRank” and “assistsMeanRank”.

3.2 MODEL ANALYSIS

3.2.1 Linear Regression

Linear regression is a linear approach that allows use to understand the relationship between dependent variable and one or more explanatory variables. Table 1 represents the 27 variables as the features.

DBNOsMeanRank	killPlaceMeanRank	killsMeanRank	isCrash

teamKillsMeanRank	vehicleDestroysMeanRank	walkDistanceMeanRank	isDuo
rankPointsMeanRank	killPlaceminRank	killsminRank	numGroups
vehicleDestroysminRank	walkDistanceminRank	rankPointsminRan	zombies
combatScoreminRank	killPlacemaxRank	teamKillsmaxRank	isCheater
vehicleDestroysmaxRank	walkDistancemaxRank	combatScoremaxRank	weaponsPerMinute
averageDistancePerMinute	itemsCollected	killStreaks	

Table 1: 27 Features

However, the RMSE of linear regression was 0.171 which was higher than other models, indicating that the response variable may not follow a linear relationship with the predictors and more complex models may be needed to mining such relationship in the data.

3.2.1.1 Linear Regression with interaction terms

In order to make use of the information contained in the interaction terms, we constructed a linear regression model including both the original features and interactions. So, the number of predictors increases from 27 to 180.

We then conducted linear regression to select important features by removing the predictors with p value higher than 0.05. These filtration step causes the decrease of the number of predictors from 180 to 63. Although this filtration step selects the important variables and reduce the predict error in training set , the increasing predictor error in validation indicates the model is overfitting. Therefore, we decided to remove all interaction terms to avoid the overfitting problem. The number of features we used in the following models is 27.

3.2.2 Gradient Boost Machine

Gradient Boost Machine produces a prediction model in the form of ensemble of weak prediction models (typically decision trees) It builds the model by allowing optimization of loss function. We used 27 features in Gradient Boost Machine model. We built 5 Gradient Boost Machine models with different parameters and 5-folds cross validation. These 5 different GBM models are one of the best models with lower predict error. So they were chosen in the stacking model.

ntrees	**Training RMSE**	**Valid RMSE**	**Cross Validation RMSE**
50	0.136	0.137	0.138
100	0.134	0.136	0.137
200	0.132	0.136	0.136
300	0.130	0.135	0.136
400	0.130	0.135	0.136

Table 2 : RMSE of Gradient Boosting Machine (27 variables)

3.2.3 Random Forest

Random forest can be used for regression which starts with multiple decision trees and ends with mean prediction for each single tree. It has two parameters which are mtry and ntree. Mtry is the number of variables randomly sampled as candidates at each split. The default value for regression is p/3 where p is number of variables in x. Ntree is the number of trees to grow. This parameter should not be too small, to ensure that every input row gets predicted at least a few times.

In our research, we build 5 random forest models. Based on 27 variables, we choose the default value of mtry as 7 which means we choose 7 random variables for each tree. The number of tree for each model are: 400, 600, 800, 1000, 1200.

The table listed as below is showing the comparison of training RMSE, validation RMSE regarding 5 models. As can be seen from the table, the RMSE are all around 0.14 which is small, but higher than the RMSE from gradient boost machine and xgboost. Therefore, these 5 models will not be considered in the stacking model.

mtry	**ntree**	**Training RMSE**	**Valid RMSE**
7	**400**	0.1400	0.1414
7	**600**	0.1400	0.1414

7	**800**	0.1400	0.1414
7	**1000**	0.1396	0.1412
7	**1200**	0.1396	0.1412

Table 3 : RMSE of Random Forest (27 variables)

3.2.4 XGBOOST

(1) XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.
In this part, we use 5 different Xgboost models with different shrinkage (eta). We take eta=0.01, 0.05, 0.11, 0.2. From Table 4, we can see when eta= 0.05, the model has the lowest Valid RMSE and cross validation test-RMSE. Moreover, according to RMSE, xgboost performs better than random forest and gradient boost machine models. They are chosen in the following stacking model.

ETA	**Training RMSE**	**Valid RMSE**	**Cross Validation test- RMSE**
0.01	0.134843	0.136958	0.139704+0.001107
0.05	0.127481	0.135200	0.136251+0.000701
0.11	0.114560	0.137080	0.136375+0.001262
0.2	0.111497	0.137080	0.136229+0.001053
0.25	0.111399	0.138065	0.136320+0.000995

Table 4 : RMSE of Xgboost (27 variables)

(2) In Figure 13, we found the top 5 important variables selected by Xgboost model is “averageDistancePerMinute”,”itemsCollecred”,”weaponsPerMinute” ,”killPlaceMeanRank” and “killPlacemaxRank”.

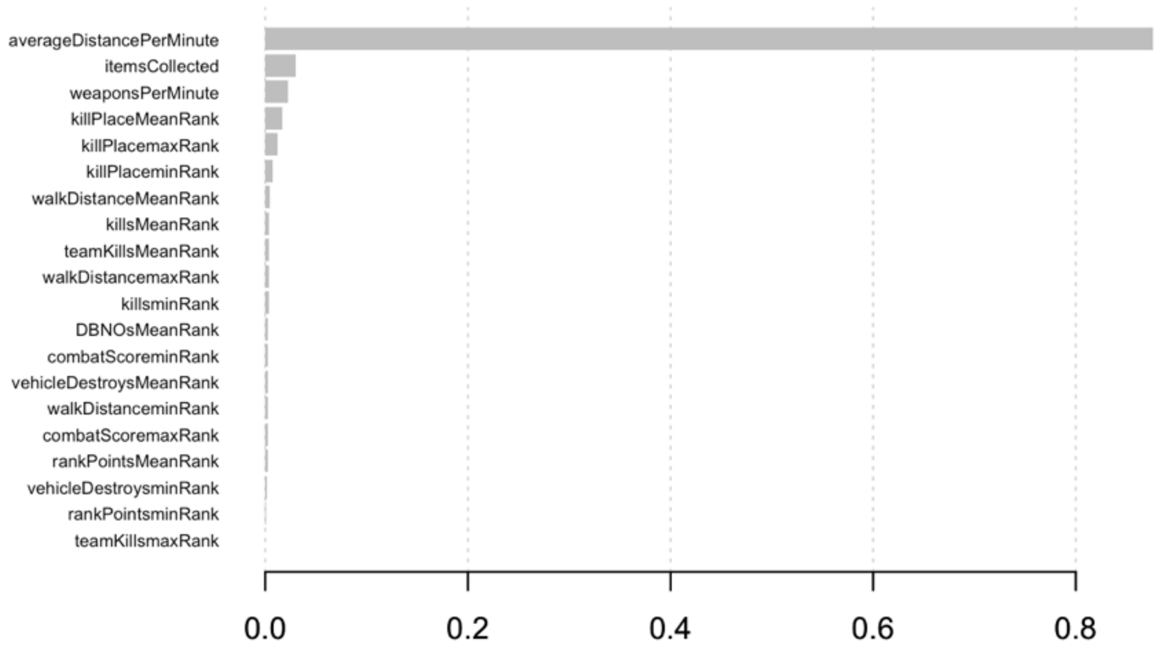
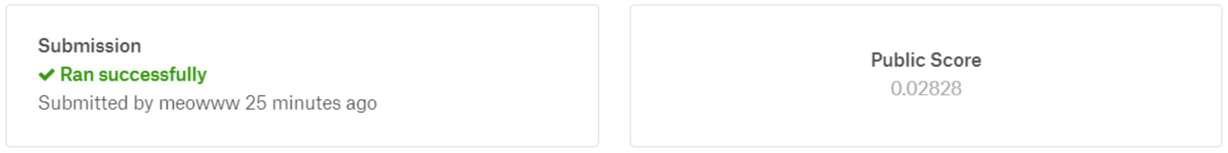


Figure 13 : Important variables selected in Xgboost

(3) We submitted a single Xgboost model to Kaggle, and gained a score of 0.02828.



Submission
✓ Ran successfully
Submitted by meowww 25 minutes ago

Public Score
0.02828

Figure 14 :Xgboost Score on Kaggle

3.2.5 MODEL ENSEMBLE

Based on the performance of the 15 models, we chose 5 xgboost models and 5 gradient boosted machine models to build a stacking model, endeavoring to further improve the estimation accuracy. Although the training RMSE decreased, the valid RMSE increased because of overfitting problem caused by the collinearity. From Table 5, we can see RMSE in validation set is 43.8% higher than that in training set.

Collinearity should be a common problem in stacking methods which is difficult to be avoided in this case, since all of the 10 prediction results in the first layer appear close to the true response variables with low RMSE, which indicates a collinearity in the 10 set of predictors.

Training RMSE	Valid RMSE	Cross Validation test- RMSE	Cross Validation train- RMSE
0.100936	0.145161	0.130456+0.000314	0.132875+0.001267

Table 5 : RMSE of Model Stacking (27 variables)

4. DISCUSSION and CONCLUSION

In this study, we found that the single Xgboost model perform best. Number of iterations were carefully chosen in order to make an accurate estimation of the predictors-response relationship while avoiding overfit. Trying other parameters randomly also helped with finding the most suitable models.

After using LASSO to do feature selection, using PCA to do dimension reduction, we decreased the number of variables from 71 to 27. The performance showed improvement, suggesting that unrelated variables may deteriorate the prediction accuracy.

Using interaction terms did reduce RMSE in train dataset but it also increased RMSE in validation set. Although we selected significant variables, the validation RMSE is still high. It is likely that we've badly over fit the data. Therefore, we finally gave up all interaction terms to avoid overfitting. Since the interaction terms did contain somehow useful information, we'd like to use them in the future to make model improvement.

How to choose important interaction terms? Features that are highly correlated or colinearity can cause overfitting. When a pair of variables are highly correlated (correlation > 0.7), we can remove one in the pair to reduce dimensionality without much loss information. Which one should we keep? The one with higher correlation to the target. The remaining interaction terms not only avoid multicollinearity but fit the model better.

After deciding the final set of features, we ran the test set provided online to test the score of our stacking model. The score of the final model is around 0.02828 which is much better than the one we got in our first meeting in which we only considered adding new features, but neglected to consider collinearity and useless features. After the process of feature selection, the xgboost model is relatively a successful model to fit the PUBG dataset to predict the ranking of each player.

In terms of the hurdles and problems, we found that the dataset takes large storage space and long time to load, which made it impossible to run the full training data on our computers. In addition, some R packages such as h2o are not convenient to install in Mac computer, frequently showing errors. It may be more helpful to improve our models if we can learn about other groups' methods

APPENDIX

- **R CODE FOR EMPIRICAL DATA ANALYSIS**

~~~
```{r,message=FALSE,warning=FALSE}
library(tidyverse)
library(readr)
library(dplyr)
library(gridExtra)
library(ggplot2)
library(data.table)
library(caret)
library(MASS)
library(h2o)
library(pdp)
library(ranger)
library(corrplot)
library(RColorBrewer)
library(factoextra)
library(randomForest)
```
```{r}
train <- read_csv(“/train_V2.csv")
test <- read_csv("test_V2.csv")
```
After preparing the train and test data set, we find that "winPlacePerc" is the response we will
predict.
calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match.
```{r,results='hide'}
str(train)
str(test)
```
```{r}
train$winPlacePerc[is.na(train$winPlacePerc)] <- 0
hchart(cor(train[,-c(1,2,3,16,19,28)]))
```

 walkDistance - Total distance traveled on foot measured in meters.
```{r}
ggplot(train,aes(walkDistance,winPlacePerc)) +
  stat_binhex() +
  scale_fill_gradient(low = "lightblue",high = "darkred") +
~~~

~~~~
  labs(title = "Point for walkDistance")
```
```{r}
train.n <- train %>% mutate(metresPerHour=(walkDistance/1000)/(matchDuration/60/60))
mph <-ggplot(data = train.n, aes(metresPerHour)) + geom_histogram(bins =
100,color="#6699ff", fill="lightgreen") + theme(plot.title = element_text(size=14)) + labs(title =
'Distribution of Meters per Hour', xlab = 'Meters per Hour', ylab = 'Count')
print(mph)
```
```{r}
kp <- ggplot(train,aes(as.factor(killPlace),winPlacePerc)) +
  stat_binhex() +
  scale_fill_gradient(low = "lightseagreen",high = "darkred") +
  labs(title = "Point for killPalce",x = "killPlace") +
  scale_x_discrete(breaks = c(0,25,50,75,100))
print(kp)
```
```{r}
col = cm.colors(27)
boost <- ggplot(train,aes(as.factor(boosts),winPlacePerc)) +
  geom_boxplot(fill = col) +
  theme_classic() +
  labs(title = "Boxplot for boosts",x = "boosts")
print(boost)
```
```{r}
zombies<-train%>%filter(walkDistance==0,weaponsAcquired==0)
totalMatches<-length(unique(train$matchId))

zombies<-
summarise(group_by(zombies,matchId),ZombieCount=n(),meanWinPlacePerc=mean(winPlace
Perc,na.rm=T))%>%mutate(row=row_number())
head(zombies)
print(paste0(round((nrow(zombies)/totalMatches)*100,2),"% of matches in train data have
zombie players."))
zomc<-ggplot(data = zombies, aes(ZombieCount)) + geom_histogram(bins =
20,color="#ff4d88", fill="lavenderblush1") + labs(title = 'Zombie Distributions') + theme(plot.title =
element_text(size=14))
zomw<-ggplot(data = zombies, aes(meanWinPlacePerc)) + geom_histogram(bins =
20,color="lightblue4", fill="lightcyan") + labs(title = 'Zombie WinPlacePerc') + theme(plot.title =
element_text(size=14))
print(zomc)
print(zomw)
```
~~~~

~~~
```{r}
killCheaters <- train %>% filter(kills>10)
distance <- ggplot(data = killCheaters, aes(walkDistance)) + geom_histogram(bins =
100,color="orange3", fill="olivedrab1") + labs(title = 'High Kill Distributions') + theme(plot.title =
element_text(size=14))
headshot <-ggplot(data = killCheaters, aes(headshotKills)) + geom_histogram(bins =
100,color="orange3", fill="olivedrab1") + labs(title = 'High Kill Distributions') + theme(plot.title =
element_text(size=14))
DBNO <- ggplot(data = killCheaters, aes(DBNOs)) + geom_histogram(bins =
100,color="orange3", fill="olivedrab1") + labs(title = 'High Kill Distributions') + theme(plot.title =
element_text(size=14))
print(distance)
print(headshot)
print(DBNO)
```
~~~

~~~
```{r}
train_pca <- subset(train, select = -c(Id, matchId, groupId, winPlacePerc))
train_pca$matchType <- as.numeric(as.factor(train_pca$matchType))
res.pca <- prcomp(train_pca, scale = T)
fviz_pca_var(res.pca,
        col.var = "contrib", # Color by contributions to the PC
        gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
)
```
```{r}
res.var <- get_pca_var(res.pca)
```
~~~

- **R CODE FOR MODEL ANALYSIS**

~~~
```{r}
#Load the libraries we will need.
library(tidyverse)
library(MASS)
library(data.table)
library(xgboost)
library(Metrics)
library(corrplot)
library(grid)
library(fmsb)
library(h2o)
~~~

~~~~
library(glmnet)
library(Matrix)
h2o.init(nthreads=-1)
```

```{r}
set.seed(123)
train <- sample_frac(train,0.02)
#remove missing values
train<-train%>%filter(!is.na(winPlacePerc))
train<-train%>%filter(numGroups>1)
```
```{r}
combat.scores<-function(x)
{
  x<-x%>%mutate(combatScore = ((damageDealt/100)*10)+(kills*10)+(-
teamKills*20)+(DBNOs*10),
                teamWorkScore = (assists*20)+(revives*50),
                walkDistance =
ifelse((walkDistance/1000)/(matchDuration/60/60)>2.66,2660,walkDistance),   ### based on
subsample

                averageDistancePerMinute =
(walkDistance+rideDistance+swimDistance)/(matchDuration/60),
                weaponsPerMinute = weaponsAcquired/(matchDuration/60),
                itemsCollected = (weaponsAcquired+heals+boosts)
  )
  return(x)
}
```

```{r}
match.types<-function(x)
{
  x<-x%>%mutate(isSolo = ifelse(x$matchType=='solo-fpp' | x$matchType=='solo' |

                                  x$matchType=='normal-solo-fpp' | x$matchType=='solo' |
                                  x$matchType=='normal-solo',1,0),

                isDuo = ifelse(x$matchType=='duo' | x$matchType=='duo-fpp' |
                                 x$matchType=='normal-duo-fpp' | x$matchType=='normal-duo' ,1,0),

                isSquad = ifelse(x$matchType=='squad-fpp' | x$matchType=='squad' |
                                   x$matchType=='normal-squad-fpp' | x$matchType=='normal-squad' ,1,0),
~~~~

```
              isCrash = ifelse(x$matchType=='crashfpp' | x$matchType=='crashtpp',1,0),
              isFlare = ifelse(x$matchType=='flaretpp' | x$matchType=='flarefpp',1,0))

  x<-unique(x%>%select(matchId,isSolo,isDuo,isSquad,isCrash,isFlare))
  return (x)
}
```

```{r}
matchType<-match.types(train%>% select(matchId,matchType))
train<-combat.scores(train)

lables<-summarise(group_by(train,matchId,groupId),winPlacePerc=max(winPlacePerc)) # y by
matchId, groupId


train<-train%>%select(-"Id",-"winPlacePerc") #remove y
#create new features for each group
groups<-summarise(group_by(train,matchId,groupId),partySize=n(),
                maxPlace=max(maxPlace),
                averageDistancePerMinute=mean(averageDistancePerMinute),
                weaponsPerMinute=mean(weaponsPerMinute),
                itemsCollected=mean(itemsCollected),
                killStreaks = mean(killStreaks/kills),
                isCheater=sum(ifelse(walkDistance>=2660,1,0)),  ### origin 22000
                zombies=sum(ifelse(walkDistance==0 & weaponsAcquired==0,1,0)),
                numGroups=max(numGroups))

####################
agg<-train%>%group_by(matchId,groupId)%>%summarise_if(is.numeric, funs(mean, min,
max)) # find mean min max of each group

agg.r<-agg%>%group_by(matchId)%>%mutate(assistsMeanRank=rank(assists_mean,
ties.method="max")/numGroups_mean*100,
                                  boostsMeanRank=rank(boosts_mean,
ties.method="max")/numGroups_mean*100,
                                  damageDealtMeanRank=rank(damageDealt_mean,
ties.method="max")/numGroups_mean*100,
                                  DBNOsMeanRank=rank(DBNOs_mean,
ties.method="max")/numGroups_mean*100,
                                  headshotKillsMeanRank=rank(headshotKills_mean,
ties.method="max")/numGroups_mean*100,
                                  healsMeanRank=rank(heals_mean,
ties.method="max")/numGroups_mean*100,
                                  killPlaceMeanRank=rank(killPlace_mean,
ties.method="max")/numGroups_mean*100,
                                  killsMeanRank=rank(kills_mean,
ties.method="max")/numGroups_mean*100,
                                  longestKillMeanRank=rank(longestKill_mean,
ties.method="max")/numGroups_mean*100,
                                  revivesMeanRank=rank(revives_mean,
ties.method="max")/numGroups_mean*100,
                                  rideDistanceMeanRank=rank(rideDistance_mean,
ties.method="max")/numGroups_mean*100,
                                  roadKillsMeanRank=rank(roadKills_mean,
ties.method="max")/numGroups_mean*100,
                                  swimDistanceMeanRank=rank(swimDistance_mean,
ties.method="max")/numGroups_mean*100,
                                  teamKillsMeanRank=rank(teamKills_mean,
ties.method="max")/numGroups_mean*100,
                                  vehicleDestroysMeanRank=rank(vehicleDestroys_mean,
ties.method="max")/numGroups_mean*100,
                                  walkDistanceMeanRank=rank(walkDistance_mean,
ties.method="max")/numGroups_mean*100,
                                  weaponsAcquiredMeanRank=rank(weaponsAcquired_mean,
ties.method="max")/numGroups_mean*100,
                                  rankPointsMeanRank=rank(rankPoints_mean,
ties.method="max")/numGroups_mean*100,
                                  combatScoreMeanRank=rank(combatScore_mean,
ties.method="max")/numGroups_mean*100,

                                  assistsminRank=rank(assists_min,
ties.method="max")/numGroups_mean*100,
                                  boostsminRank=rank(boosts_min,
ties.method="max")/numGroups_mean*100,
                                  damageDealtminRank=rank(damageDealt_min,
ties.method="max")/numGroups_mean*100,
                                  DBNOsminRank=rank(DBNOs_min,
ties.method="max")/numGroups_mean*100,
                                  headshotKillsminRank=rank(headshotKills_min,
ties.method="max")/numGroups_mean*100,
                                  healsminRank=rank(heals_min,
ties.method="max")/numGroups_mean*100,
                                  killPlaceminRank=rank(killPlace_min,
ties.method="max")/numGroups_mean*100,
                                        killsminRank=rank(kills_min,
ties.method="max")/numGroups_mean*100,
                                        longestKillminRank=rank(longestKill_min,
ties.method="max")/numGroups_mean*100,
                                        revivesminRank=rank(revives_min,
ties.method="max")/numGroups_mean*100,
                                        rideDistanceminRank=rank(rideDistance_min,
ties.method="max")/numGroups_mean*100,
                                        roadKillsminRank=rank(roadKills_min,
ties.method="max")/numGroups_mean*100,
                                        swimDistanceminRank=rank(swimDistance_min,
ties.method="max")/numGroups_mean*100,
                                        teamKillsminRank=rank(teamKills_min,
ties.method="max")/numGroups_mean*100,
                                        vehicleDestroysminRank=rank(vehicleDestroys_min,
ties.method="max")/numGroups_mean*100,
                                        walkDistanceminRank=rank(walkDistance_min,
ties.method="max")/numGroups_mean*100,
                                        weaponsAcquiredminRank=rank(weaponsAcquired_min,
ties.method="max")/numGroups_mean*100,
                                        rankPointsminRank=rank(rankPoints_min,
ties.method="max")/numGroups_mean*100,combatScoreminRank=rank(combatScore_min,
ties.method="max")/numGroups_mean*100,
                                    assistsmaxRank=rank(assists_max,
ties.method="max")/numGroups_mean*100,
                                        boostsmaxRank=rank(boosts_max,
ties.method="max")/numGroups_mean*100,
                                        damageDealtmaxRank=rank(damageDealt_max,
ties.method="max")/numGroups_mean*100,
                                        DBNOsmaxRank=rank(DBNOs_max,
ties.method="max")/numGroups_mean*100,
                                        headshotKillsmaxRank=rank(headshotKills_max,
ties.method="max")/numGroups_mean*100,
                                        healsmaxRank=rank(heals_max,
ties.method="max")/numGroups_mean*100,
                                        killPlacemaxRank=rank(killPlace_max,
ties.method="max")/numGroups_mean*100,
                                        killsmaxRank=rank(kills_max,
ties.method="max")/numGroups_mean*100,
                                        longestKillmaxRank=rank(longestKill_max,
ties.method="max")/numGroups_mean*100,revivesmaxRank=rank(revives_max,
ties.method="max")/numGroups_mean*100,
                                        rideDistancemaxRank=rank(rideDistance_max,
ties.method="max")/numGroups_mean*100,
```

~~~~
                                    roadKillsmaxRank=rank(roadKills_max,
ties.method="max")/numGroups_mean*100,
                                    swimDistancemaxRank=rank(swimDistance_max,
ties.method="max")/numGroups_mean*100,
                                    teamKillsmaxRank=rank(teamKills_max,
ties.method="max")/numGroups_mean*100,
                                    vehicleDestroysmaxRank=rank(vehicleDestroys_max,
ties.method="max")/numGroups_mean*100,
                                    walkDistancemaxRank=rank(walkDistance_max,
ties.method="max")/numGroups_mean*100,
                                    weaponsAcquiredmaxRank=rank(weaponsAcquired_max,
ties.method="max")/numGroups_mean*100,
                                    rankPointsmaxRank=rank(rankPoints_max,
ties.method="max")/numGroups_mean*100,
                                    combatScoremaxRank=rank(combatScore_max,
ties.method="max")/numGroups_mean*100
)
agg.r<-agg.r%>%inner_join(groups)
agg.r<-agg.r%>%inner_join(matchType)
rm(agg,train)
train<-agg.r
rm(agg.r)
gc()
```



```{r}
train <- train %>% select(-contains("_"))
train <- train %>% mutate_if(is.integer,funs(as.numeric(.)))

train<-train%>%inner_join(lables)
rm(lables)
```


```{r}
train$killStreaks[is.na( train$killStreaks)] <- 0
train <- train[,-c(1,2)]
```


```{r}
features <- train[,-c(72)]
response <- train$winPlacePerc
```
~~~~

~~~
```{r}
set.seed(123)

new.x<-as.matrix(subset(train,select = -c(winPlacePerc)))
lasso.mod <-cv.glmnet( new.x, response,alpha=1)
plot(lasso.mod) #

lasso.mod$lambda.min
myCoefs<-coef(lasso.mod)

myCoefs[which(myCoefs != 0 ) ]

myResults <- data.frame(
  features = myCoefs@Dimnames[[1]][ which(myCoefs != 0 ) ], #intercept included
  coefs    = myCoefs              [ which(myCoefs != 0 ) ]  #intercept included
)

features2 = myCoefs@Dimnames[[1]][ which(myCoefs != 0 ) ]
lasso_feature<-features2[-1]
```

```{r}
lasso_feature<-features2[-1]   # important selected features for lasso
train<-train[,c(lasso_feature)]
train$winPlacePerc<-response
```
```{r}

set.seed(123)
#(a)split to train_linear, valid, test for linear regression
spec<- c(sample_train = 0.6, sample_test  = 0.2, sample_valid = 0.2)
split <- sample(cut(
  seq(nrow(train)),
  nrow(train)*cumsum(c(0,spec)),
  labels = names(spec)
))

res <- split(train , split)
sample_train   <- res$sample_train
sample_test  <-  res$sample_test
sample_valid  <- res$sample_valid
train_reponse<-sample_train$winPlacePerc
test_reponse  <-  sample_test$winPlacePerc
valid_reponse<-sample_valid $winPlacePerc
~~~

~~~
```

##PCA for dataset with selected features
```{r}
library(factoextra)
train_pca <- subset(sample_train, select = -c( winPlacePerc))
res.pca <- prcomp(train_pca, scale = TRUE)

fviz_pca_var(res.pca,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
)
print("As we can see, the dimension 1 is the most important")

res.var <- get_pca_var(res.pca)
res.var$contrib
print("For the dimension 1, the most explanatory variables are :")
print("- teamKillsMeanRank")
print("- vehicleDestroysMeanRan")
print("- roadKillsminRank")
print("- vehicleDestroysminRank")
print("- vehicleDestroysmaxRank")
print("- assistsMeanRank")
```

# linear regression to reduce variables (Use its variables (var_select2) in final model)
```{r}
library(DAAG)
#####linear cross validation lasso
sample_train<-as.data.frame(sample_train)
train.lm <- lm(winPlacePerc ~., data=sample_train)
cross_v<-CVlm( sample_train,train.lm,m=3)
RMSE <-sqrt(mean((cross_v$cvpred-cross_v$winPlacePerc)^2))    #root-mean-square
deviation (RMSE)
RMSE

train.lm_sum=summary(train.lm)
train.lm_c=train.lm_sum$coefficients
train.lm_p=train.lm_c[,4]
train.lm_select=which(train.lm_p<0.05)
length(train.lm_select)
var_select2=names(train.lm_select[-1])
~~~

~~~~
sample_train<-sample_train[,c(which(colnames(sample_train) %in% var_select2))]
sample_valid<-sample_valid[,c(which(colnames(sample_valid) %in% var_select2))]
sample_test<-sample_test[,c(which(colnames(sample_test) %in% var_select2))]

sample_train$winPlacePerc<-train_reponse
sample_test$winPlacePerc<-test_reponse
sample_valid$winPlacePerc<-valid_reponse
```
## Gradient boosted Machine
```{r}
library(MASS)
library(tidyverse)
library(caret)
library(mgcv)

trainHex<-as.h2o(sample_train)
validHex<-as.h2o(sample_valid)
testHex<-as.h2o(sample_test)

#(1)base
gbm1 <- h2o.gbm(x = var_select2, y = "winPlacePerc", training_frame =
trainHex,validation_frame=validHex,nfolds = 5, seed = 1234)
gbm1

#(2) tuning 2nd
gbm2 <- h2o.gbm(
  x = var_select2,
  y = "winPlacePerc",
  training_frame = trainHex,
  validation_frame = validHex,
  ntrees = 100,
  learn_rate=0.1,
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric = "RMSE",
  sample_rate = 0.8,
  col_sample_rate = 0.8,
  seed = 1234,
  nfolds = 5,
  score_tree_interval = 10
)
gbm2

#(3) tuning 3rd

gbm3 <- h2o.gbm(
~~~~

```
  x = var_select2,
  y = "winPlacePerc",
  training_frame = trainHex,
  validation_frame = validHex,
  ntrees = 200,
  learn_rate=0.1,
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric = "RMSE",
  sample_rate = 0.8,
  col_sample_rate = 0.8,
  seed = 1234,
  nfolds = 5,
  score_tree_interval = 10
)
gbm3

#(4) tuning 4th

gbm4 <- h2o.gbm(
  x = var_select2,
  y = "winPlacePerc",
  training_frame = trainHex,
  validation_frame = validHex,
  ntrees = 300,
  learn_rate=0.1,
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric = "MAE",
  sample_rate = 0.8,
  col_sample_rate = 0.8,
  seed = 1234,
 nfolds=5,
  score_tree_interval = 10
)
gbm4

#(5) tuning 5th

gbm5 <- h2o.gbm(
  x = var_select2,
  y = "winPlacePerc",
  training_frame = trainHex,
  validation_frame = validHex,
  ntrees = 400,
  learn_rate=0.1,
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric = "RMSE",
  sample_rate = 0.8,
```

~~~~
  col_sample_rate = 0.8,
  seed = 1234,
  nfolds=5,
  score_tree_interval = 10
)
gbm5


h2o.rmse(h2o.performance(gbm1, newdata = validHex))
h2o.rmse(h2o.performance(gbm2, newdata = validHex))
h2o.rmse(h2o.performance(gbm3, newdata = validHex))
h2o.rmse(h2o.performance(gbm4, newdata = validHex))
h2o.rmse(h2o.performance(gbm5, newdata = validHex))
```

## RANDOM FOREST
```{r}
#model1
model_rf1.train <- randomForest(winPlacePerc~.,data=sample_train,mtry=7,ntree=400)
model_rf1.train

model_rf1.valid <- randomForest(winPlacePerc~.,data=sample_valid,mtry=7,ntree=400)
model_rf1.valid

#model2
model_rf2.train <- randomForest(winPlacePerc~.,data=sample_train,mtry=7,ntree=600)
model_rf2.train

model_rf2.valid <- randomForest(winPlacePerc~.,data=sample_valid,mtry=7,ntree=600)
Model_rf2.valid

#model3
model_rf3.train <- randomForest(winPlacePerc~.,data=sample_train,mtry=7,ntree=800)
model_rf3.train
model_rf3.valid <- randomForest(winPlacePerc~.,data=sample_valid,mtry=7,ntree=800)
model_rf3.valid

#model4
model_rf4.train <- randomForest(winPlacePerc~.,data=sample_train,mtry=7,ntree=1000)
Model_rf4.train

model_rf4.valid <- randomForest(winPlacePerc~.,data=sample_valid,mtry=7,ntree=1000)
model_rf4.valid
~~~~

~~~
#model5
model_rf5.train <- randomForest(winPlacePerc~.,data=sample_train,mtry=7,ntree=1200)
model_rf5.train

model_rf5.valid <- randomForest(winPlacePerc~.,data=sample_valid,mtry=7,ntree=1200)
model_rf5.valid
```

## XGBOOST
```{r}
d_train <- sample_train %>%
  dplyr::select(-winPlacePerc) %>%
  as.matrix %>%
  xgb.DMatrix(label=as.numeric(sample_train$winPlacePerc)-1)

sample_valid=data.frame(sample_valid)
d_val <- sample_valid %>%
  dplyr::select(-winPlacePerc) %>%
  as.matrix %>%
  xgb.DMatrix(label=as.numeric(sample_valid$winPlacePerc)-1)

#Model1:eta=0.01
set.seed(123)
xgb1_freq1 <- xgb.train(
  data = d_train,
  watchlist = list( train = d_train, valid = d_val), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.01,
  nround = 600,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1,
  eval_metric = "rmse",
  objective = "reg:linear",
  nthread = 3,
  max_depth = 5
)

cv_boost1 <- xgb.cv(data = d_train, nrounds = 600, nthread = 6, nfold = 5, metrics = list("rmse"),
                  max_depth = 3, eta = 0.01,predictions=TRUE )

#Model2:eta=0.05
set.seed(123)
xgb1_freq2 <- xgb.train(
~~~

```
  data = d_train,
  watchlist = list( train = d_train, valid = d_val), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.05,
  nround = 600,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1,
  eval_metric = "rmse",
  nthread = 3,
  max_depth = 5
)

cv_boost2 <- xgb.cv(data = d_train, nrounds = 600, nthread = 6, nfold = 5, metrics = list("rmse"),
                 max_depth = 3, eta = 0.2,predictions=TRUE )

#Model3:eta=0.11
set.seed(123)
xgb1_freq3<- xgb.train(
  data = d_train,
  watchlist = list( train = d_train, valid = d_val), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.11,
  nround = 600,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1,
  eval_metric = "rmse",
  nthread = 3,
  max_depth = 5
)
cv_boost3 <- xgb.cv(data = d_train, nrounds = 600, nthread = 6, nfold = 5, metrics = list("rmse"),
                 max_depth = 3, eta = 0.11,predictions=TRUE )

#Model4:eta=0.2
set.seed(123)
xgb1_freq4 <- xgb.train(
  data = d_train,
  watchlist = list( train = d_train, valid = d_val), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.2,
  nround = 600,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1,
  eval_metric = "rmse",

  nthread = 3,
  max_depth = 5
)

cv_boost4 <- xgb.cv(data = d_train, nrounds = 600, nthread = 6, nfold = 5, metrics = list("rmse"),
                  max_depth = 3, eta = 0.2,predictions=TRUE )

#Model5:eta=0.25
set.seed(123)
xgb1_freq5 <- xgb.train(
  data = d_train,
  watchlist = list( train = d_train, valid = d_val), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.25,
  nround = 600,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1,
  eval_metric = "rmse",
  nthread = 3,
  max_depth = 5
)cv_boost5 <- xgb.cv(data = d_train, nrounds = 600, nthread = 6, nfold = 5, metrics =
list("rmse"),
                  max_depth = 3, eta = 0.25,predictions=TRUE )
```

##STACKING

layer1

```{r}
##Stacking - combine the best 10 models - xgboost and gbm
#1st layer generation-for training

gbm_pred_test1<-predict(gbm1,testHex, probability=TRUE)
gbm_pred_test2<-predict(gbm2,testHex, probability=TRUE)
gbm_pred_test3<-predict(gbm3,testHex, probability=TRUE)
gbm_pred_test4<-predict(gbm4,testHex, probability=TRUE)
gbm_pred_test5<-predict(gbm5,testHex, probability=TRUE)

testHex_mat=(data.matrix(testHex))
testHex_mat2=testHex_mat[,-21]#remove y
```

```
xgb_pred_test1<-predict(xgb1_freq1,testHex_mat2, probability=TRUE)
xgb_pred_test2<-predict(xgb1_freq2,testHex_mat2, probability=TRUE)
xgb_pred_test3<-predict(xgb1_freq3,testHex_mat2, probability=TRUE)
xgb_pred_test4<-predict(xgb1_freq4,testHex_mat2, probability=TRUE)
xgb_pred_test5<-predict(xgb1_freq5,testHex_mat2, probability=TRUE)

df_pred1=data.frame(as.vector(gbm_pred_test1),as.vector(gbm_pred_test2),as.vector(gbm_pre
d_test3),as.vector(gbm_pred_test4),as.vector(gbm_pred_test5),
                    xgb_pred_test1,xgb_pred_test2,xgb_pred_test3,xgb_pred_test4,xgb_pred_test5)

names(df_pred1)=c(paste0("gbm_",c(1:5)),paste0("xgb_",c(1:5)))


#1st layer generation-for testing the stacking prediction

gbm_pred_valid1<-predict(gbm1,validHex, probability=TRUE)
gbm_pred_valid2<-predict(gbm2,validHex, probability=TRUE)
gbm_pred_valid3<-predict(gbm3,validHex, probability=TRUE)
gbm_pred_valid4<-predict(gbm4,validHex, probability=TRUE)
gbm_pred_valid5<-predict(gbm5,validHex, probability=TRUE)


tvalidHex_mat=(data.matrix(validHex))
tvalidHex_mat2=tvalidHex_mat[,-65]#remove y

xgb_pred_valid1<-predict(xgb1_freq1,tvalidHex_mat2, probability=TRUE)
xgb_pred_valid2<-predict(xgb1_freq2,tvalidHex_mat2, probability=TRUE)
xgb_pred_valid3<-predict(xgb1_freq3,tvalidHex_mat2, probability=TRUE)
xgb_pred_valid4<-predict(xgb1_freq4,tvalidHex_mat2, probability=TRUE)
xgb_pred_valid5<-predict(xgb1_freq5,tvalidHex_mat2, probability=TRUE)

df_valid=data.frame(as.vector(gbm_pred_valid1),as.vector(gbm_pred_valid2),as.vector(gbm_pr
ed_valid3),as.vector(gbm_pred_valid4),as.vector(gbm_pred_valid5),

xgb_pred_valid1,xgb_pred_valid2,xgb_pred_valid3,xgb_pred_valid4,xgb_pred_valid5)

names(df_valid)=c(paste0("gbm_",c(1:5)),paste0("xgb_",c(1:5)))

#convert to matrix
response_train_stack <- sample_test$winPlacePerc

train_stack <- data.matrix(df_pred1)
d_train_stack <- xgb.DMatrix(
  data = train_stack,
```

~~~~
  label = response_train_stack,
  missing = "NAN")

response_test_stack <- sample_valid$winPlacePerc
test_stack <- data.matrix(df_valid)
d_test_stack <- xgb.DMatrix(
  data = test_stack,
  label = response_test_stack,
  missing = "NAN")
```
### 2nd layer
```{r}
#Model3:eta=0.03
set.seed(123)
stack8 <- xgb.train(
  data = d_train_stack,
  watchlist = list( train = d_train_stack, valid = d_test_stack), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.03,
  nround = 600,
  subsample = 0.8,
  colsample_bytree = 0.8,
  seed = 1,
  eval_metric = "rmse",
  nthread = 3,
  max_depth = 5

)

 #cross-validation
cv_stacking <- xgb.cv(data = d_train_stack, nrounds = 600, nthread = 6, nfold = 5, metrics =
list("rmse"),
                  max_depth = 3, eta = 0.25,predictions=TRUE )
```

##SUBBMISSION
#Load the libraries we will need.
library(tidyverse)
library(data.table)
library(xgboost)
library(Metrics)
library(corrplot)
library(grid)
library(fmsb)
~~~~

```
#Quick check of the input directory.

train <- fread("../input/train_V2.csv")

#Combat Scores attempts to group together similar actions
combat.scores<-function(x)
{
  x<-x%>%mutate(combatScore = ((damageDealt/100)*10)+(kills*10)+(-
teamKills*20)+(DBNOs*10),
                teamWorkScore = (assists*20)+(revives*50),
                walkDistance =
ifelse((walkDistance/1000)/(matchDuration/60/60)>22,22000,walkDistance),
                averageDistancePerMinute =
(walkDistance+rideDistance+swimDistance)/(matchDuration/60),
                weaponsPerMinute = weaponsAcquired/(matchDuration/60),
                itemsCollected = (weaponsAcquired+heals+boosts)
  )
  return(x)
}

match.types<-function(x)
{
  x<-x%>%mutate(isSolo = ifelse(x$matchType=='solo-fpp' | x$matchType=='solo' |
                                 x$matchType=='normal-solo-fpp' | x$matchType=='solo' |
                                 x$matchType=='normal-solo',1,0),

                isDuo = ifelse(x$matchType=='duo' | x$matchType=='duo-fpp' |
                                x$matchType=='normal-duo-fpp' | x$matchType=='normal-duo' ,1,0),

                isSquad = ifelse(x$matchType=='squad-fpp' | x$matchType=='squad' |
                                  x$matchType=='normal-squad-fpp' | x$matchType=='normal-squad' ,1,0),


                isCrash = ifelse(x$matchType=='crashfpp' | x$matchType=='crashtpp',1,0),
                isFlare = ifelse(x$matchType=='flaretpp' | x$matchType=='flarefpp',1,0))

  x<-unique(x%>%select(matchId,isSolo,isDuo,isSquad,isCrash,isFlare))
  return (x)
}

cat("Loading Data.....")
train <- fread("../input/train_V2.csv")
```

```
train<-train%>%filter(!is.na(winPlacePerc))
#For training we will remove any matches were numGroups ==1 as we know these will always
be 0
train<-train%>%filter(numGroups>1)

matchType<-match.types(train%>%select(matchId,matchType))
train<-combat.scores(train)

lables<-summarise(group_by(train,matchId,groupId),winPlacePerc=max(winPlacePerc))
train<-train%>%select(-Id,-winPlacePerc)

groups<-summarise(group_by(train,matchId,groupId),partySize=n(),
            maxPlace=max(maxPlace),
            averageDistancePerMinute=mean(averageDistancePerMinute),
            weaponsPerMinute=mean(weaponsPerMinute),
            itemsCollected=mean(itemsCollected),
            killStreaks = mean(killStreaks/kills),
            isCheater=sum(ifelse(walkDistance>=22000,1,0)),
            zombies=sum(ifelse(walkDistance==0 & weaponsAcquired==0,1,0)),
            numGroups=max(numGroups))

agg<-train%>%group_by(matchId,groupId)%>%summarise_if(is.numeric, funs(mean, min,
max))

agg.r<-agg%>%group_by(matchId)%>%mutate(assistsMeanRank=rank(assists_mean,
ties.method="max")/numGroups_mean*100,
                          boostsMeanRank=rank(boosts_mean,
ties.method="max")/numGroups_mean*100,
                          damageDealtMeanRank=rank(damageDealt_mean,
ties.method="max")/numGroups_mean*100,
                          DBNOsMeanRank=rank(DBNOs_mean,
ties.method="max")/numGroups_mean*100,
                          headshotKillsMeanRank=rank(headshotKills_mean,
ties.method="max")/numGroups_mean*100,
                          healsMeanRank=rank(heals_mean,
ties.method="max")/numGroups_mean*100,
                          killPlaceMeanRank=rank(killPlace_mean,
ties.method="max")/numGroups_mean*100,
                          killsMeanRank=rank(kills_mean,
ties.method="max")/numGroups_mean*100,
                          longestKillMeanRank=rank(longestKill_mean,
ties.method="max")/numGroups_mean*100,
                          revivesMeanRank=rank(revives_mean,
ties.method="max")/numGroups_mean*100,
                                        rideDistanceMeanRank=rank(rideDistance_mean,
ties.method="max")/numGroups_mean*100,
                                        roadKillsMeanRank=rank(roadKills_mean,
ties.method="max")/numGroups_mean*100,
                                        swimDistanceMeanRank=rank(swimDistance_mean,
ties.method="max")/numGroups_mean*100,
                                        teamKillsMeanRank=rank(teamKills_mean,
ties.method="max")/numGroups_mean*100,
                                        vehicleDestroysMeanRank=rank(vehicleDestroys_mean,
ties.method="max")/numGroups_mean*100,
                                        walkDistanceMeanRank=rank(walkDistance_mean,
ties.method="max")/numGroups_mean*100,
                                        weaponsAcquiredMeanRank=rank(weaponsAcquired_mean,
ties.method="max")/numGroups_mean*100,
                                        rankPointsMeanRank=rank(rankPoints_mean,
ties.method="max")/numGroups_mean*100,
                                        combatScoreMeanRank=rank(combatScore_mean,
ties.method="max")/numGroups_mean*100,

                                        assistsminRank=rank(assists_min,
ties.method="max")/numGroups_mean*100,
                                        boostsminRank=rank(boosts_min,
ties.method="max")/numGroups_mean*100,
                                        damageDealtminRank=rank(damageDealt_min,
ties.method="max")/numGroups_mean*100,
                                        DBNOsminRank=rank(DBNOs_min,
ties.method="max")/numGroups_mean*100,
                                        headshotKillsminRank=rank(headshotKills_min,
ties.method="max")/numGroups_mean*100,
                                        healsminRank=rank(heals_min,
ties.method="max")/numGroups_mean*100,
                                        killPlaceminRank=rank(killPlace_min,
ties.method="max")/numGroups_mean*100,
                                        killsminRank=rank(kills_min,
ties.method="max")/numGroups_mean*100,
                                        longestKillminRank=rank(longestKill_min,
ties.method="max")/numGroups_mean*100,
                                        revivesminRank=rank(revives_min,
ties.method="max")/numGroups_mean*100,
                                        rideDistanceminRank=rank(rideDistance_min,
ties.method="max")/numGroups_mean*100,
                                        roadKillsminRank=rank(roadKills_min,
ties.method="max")/numGroups_mean*100,
                                  swimDistanceminRank=rank(swimDistance_min,
ties.method="max")/numGroups_mean*100,
                                  teamKillsminRank=rank(teamKills_min,
ties.method="max")/numGroups_mean*100,
                                  vehicleDestroysminRank=rank(vehicleDestroys_min,
ties.method="max")/numGroups_mean*100,
                                  walkDistanceminRank=rank(walkDistance_min,
ties.method="max")/numGroups_mean*100,
                                  weaponsAcquiredminRank=rank(weaponsAcquired_min,
ties.method="max")/numGroups_mean*100,
                                  rankPointsminRank=rank(rankPoints_min,
ties.method="max")/numGroups_mean*100,
                                  combatScoreminRank=rank(combatScore_min,
ties.method="max")/numGroups_mean*100,


                                  assistsmaxRank=rank(assists_max,
ties.method="max")/numGroups_mean*100,
                                  boostsmaxRank=rank(boosts_max,
ties.method="max")/numGroups_mean*100,
                                  damageDealtmaxRank=rank(damageDealt_max,
ties.method="max")/numGroups_mean*100,
                                  DBNOsmaxRank=rank(DBNOs_max,
ties.method="max")/numGroups_mean*100,
                                  headshotKillsmaxRank=rank(headshotKills_max,
ties.method="max")/numGroups_mean*100,
                                  healsmaxRank=rank(heals_max,
ties.method="max")/numGroups_mean*100,
                                  killPlacemaxRank=rank(killPlace_max,
ties.method="max")/numGroups_mean*100,
                                  killsmaxRank=rank(kills_max,
ties.method="max")/numGroups_mean*100,
                                  longestKillmaxRank=rank(longestKill_max,
ties.method="max")/numGroups_mean*100,
                                  revivesmaxRank=rank(revives_max,
ties.method="max")/numGroups_mean*100,
                                  rideDistancemaxRank=rank(rideDistance_max,
ties.method="max")/numGroups_mean*100,
                                  roadKillsmaxRank=rank(roadKills_max,
ties.method="max")/numGroups_mean*100,
                                  swimDistancemaxRank=rank(swimDistance_max,
ties.method="max")/numGroups_mean*100,
                                  teamKillsmaxRank=rank(teamKills_max,
ties.method="max")/numGroups_mean*100,
                                  vehicleDestroysmaxRank=rank(vehicleDestroys_max,
ties.method="max")/numGroups_mean*100,
                                  walkDistancemaxRank=rank(walkDistance_max,
ties.method="max")/numGroups_mean*100,
                                  weaponsAcquiredmaxRank=rank(weaponsAcquired_max,
ties.method="max")/numGroups_mean*100,
                                  rankPointsmaxRank=rank(rankPoints_max,
ties.method="max")/numGroups_mean*100,
                                  combatScoremaxRank=rank(combatScore_max,
ties.method="max")/numGroups_mean*100
)

agg.r<-agg.r%>%inner_join(groups)
agg.r<-agg.r%>%inner_join(matchType)

rm(agg,train)
train<-agg.r
rm(agg.r)
gc()

train <- train %>% select(-contains("_"))
train <- train %>% mutate_if(is.integer,funs(as.numeric(.)))

train<-train%>%inner_join(lables)
rm(lables)

set.seed(878)
train.ids<-as.matrix(unique(train %>% select(matchId)))
train.ids<-as.data.frame(train.ids,stringsAsFactors=F)

train.ids<-sample_frac(train.ids,0.15)

holdout<-train%>%inner_join(train.ids)
train<-train%>%anti_join(train.ids)
rm(train.ids)

set.seed(10101)

train<-as.data.frame(train,stringsAsFactors=F)

train.ids<-as.matrix(unique(train %>% select(matchId)))
train.ids<-as.data.frame(train.ids,stringsAsFactors=F)

train.ids<-sample_frac(train.ids,0.1)
```

```
valid<-train%>%inner_join(train.ids)
train<-train%>%anti_join(train.ids)
rm(train.ids)

cat("Splitting Labels.....")
cols<-names(train)

train.labels <- train$winPlacePerc
valid.labels <- valid$winPlacePerc


### use lasso to choose features for linear regression model:
library(glmnet)
library(Matrix)
set.seed(123)

new.train <- train %>% select(-c(groupId,matchId,winPlacePerc))
new.train$killStreaks[is.na(new.train$killStreaks)] <- 0

new.x<-as.matrix(new.train)
lasso.mod <-cv.glmnet(new.x, train.labels,alpha=1)
plot(lasso.mod) #

lasso.mod$lambda.min  #[1]  0.000137
#Indeed the values of the coefficients for the best model of 33 variables can be extracted.
myCoefs<-coef(lasso.mod)

myCoefs[which(myCoefs != 0 ) ]

myResults <- data.frame(
  features = myCoefs@Dimnames[[1]][ which(myCoefs != 0 ) ], #intercept included
  coefs    = myCoefs              [ which(myCoefs != 0 ) ]  #intercept included
)

features2 = myCoefs@Dimnames[[1]][ which(myCoefs != 0 ) ]
### selected features by lasso (35 features)
# features
linear_feature<-features2[-1] #54 features

#save(newtrain,"/Users/xiaolinwu/Desktop/6240/hw7/train.Rdata")
### retrain 27 features, 1 reponse
train<-train[,c(linear_feature)] #no match type and group id
train$winPlacePerc<-train.labels
```

```
valid<-valid[,c(linear_feature)] #no match type and group id
valid$winPlacePerc<-valid.labels

train_mm_freq <- data.matrix(select(train,-winPlacePerc))
rm(train)
gc()
train_dm_freq <- xgb.DMatrix(
  data = train_mm_freq,
  label = train.labels,
  missing = "NAN")
cols<-colnames(train_mm_freq)
rm(train,train_mm_freq)
gc()
valid_mm_freq <- data.matrix(select(valid,-winPlacePerc))
valid_dm_freq <- xgb.DMatrix(
  data = valid_mm_freq,
  label = valid.labels,
  missing = "NAN")


# Train XGBoost model
set.seed(1234)
xgb1_freq <- xgb.train(
  data = train_dm_freq,
  watchlist = list( train = train_dm_freq, valid = valid_dm_freq), eval_metric = "rmse",
  objective = "reg:linear",
  eta = 0.11,
  max_depth = 5,
  subsample = 0.85,
  nthread=8,
  colsample_bytree = 0.65,
  nrounds=800,
  print_every_n = 200,
  early.stop.round = 200
)

imp <- xgb.importance(cols, model = xgb1_freq)
xgb.plot.importance(imp, top_n = 50)



test <- fread("../input/test_V2.csv")
matchType<-match.types(test%>%select(matchId,matchType))
```

```
test.predictions.master <- test%>%select(Id,matchId,groupId)

test<-combat.scores(test)
test<-test%>%select(-Id)
groups<-summarise(group_by(test,matchId,groupId),partySize=n(),
                  maxPlace=max(maxPlace),
                  averageDistancePerMinute=mean(averageDistancePerMinute),
                  weaponsPerMinute=mean(weaponsPerMinute),
                  itemsCollected=mean(itemsCollected),
                  killStreaks = mean(killStreaks/kills),
                  isCheater=sum(ifelse(walkDistance>=22000,1,0)),
                  zombies=sum(ifelse(walkDistance==0 & weaponsAcquired==0,1,0)),
                  numGroups=max(numGroups))


agg<-test%>%group_by(matchId,groupId)%>%summarise_if(is.numeric, funs(mean, min, max))

#agg <- fix.ratings(agg)

agg.r<-agg%>%group_by(matchId)%>%mutate(assistsMeanRank=rank(assists_mean,
ties.method="max")/numGroups_mean*100,
                                      boostsMeanRank=rank(boosts_mean,
ties.method="max")/numGroups_mean*100,
                                      damageDealtMeanRank=rank(damageDealt_mean,
ties.method="max")/numGroups_mean*100,
                                      DBNOsMeanRank=rank(DBNOs_mean,
ties.method="max")/numGroups_mean*100,
                                      headshotKillsMeanRank=rank(headshotKills_mean,
ties.method="max")/numGroups_mean*100,
                                      healsMeanRank=rank(heals_mean,
ties.method="max")/numGroups_mean*100,
                                      killPlaceMeanRank=rank(killPlace_mean,
ties.method="max")/numGroups_mean*100,
                                      killsMeanRank=rank(kills_mean,
ties.method="max")/numGroups_mean*100,
                                      longestKillMeanRank=rank(longestKill_mean,
ties.method="max")/numGroups_mean*100,
                                      revivesMeanRank=rank(revives_mean,
ties.method="max")/numGroups_mean*100,
                                      rideDistanceMeanRank=rank(rideDistance_mean,
ties.method="max")/numGroups_mean*100,
                                      roadKillsMeanRank=rank(roadKills_mean,
ties.method="max")/numGroups_mean*100,
                                        swimDistanceMeanRank=rank(swimDistance_mean,
ties.method="max")/numGroups_mean*100,
                                        teamKillsMeanRank=rank(teamKills_mean,
ties.method="max")/numGroups_mean*100,
                                        vehicleDestroysMeanRank=rank(vehicleDestroys_mean,
ties.method="max")/numGroups_mean*100,
                                        walkDistanceMeanRank=rank(walkDistance_mean,
ties.method="max")/numGroups_mean*100,
                                        weaponsAcquiredMeanRank=rank(weaponsAcquired_mean,
ties.method="max")/numGroups_mean*100,
                                        rankPointsMeanRank=rank(rankPoints_mean,
ties.method="max")/numGroups_mean*100,
                                        combatScoreMeanRank=rank(combatScore_mean,
ties.method="max")/numGroups_mean*100,

                                        assistsminRank=rank(assists_min,
ties.method="max")/numGroups_mean*100,
                                        boostsminRank=rank(boosts_min,
ties.method="max")/numGroups_mean*100,
                                        damageDealtminRank=rank(damageDealt_min,
ties.method="max")/numGroups_mean*100,
                                        DBNOsminRank=rank(DBNOs_min,
ties.method="max")/numGroups_mean*100,
                                        headshotKillsminRank=rank(headshotKills_min,
ties.method="max")/numGroups_mean*100,
                                        healsminRank=rank(heals_min,
ties.method="max")/numGroups_mean*100,
                                        killPlaceminRank=rank(killPlace_min,
ties.method="max")/numGroups_mean*100,
                                        killsminRank=rank(kills_min,
ties.method="max")/numGroups_mean*100,
                                        longestKillminRank=rank(longestKill_min,
ties.method="max")/numGroups_mean*100,
                                        revivesminRank=rank(revives_min,
ties.method="max")/numGroups_mean*100,
                                        rideDistanceminRank=rank(rideDistance_min,
ties.method="max")/numGroups_mean*100,
                                        roadKillsminRank=rank(roadKills_min,
ties.method="max")/numGroups_mean*100,
                                        swimDistanceminRank=rank(swimDistance_min,
ties.method="max")/numGroups_mean*100,
                                        teamKillsminRank=rank(teamKills_min,
ties.method="max")/numGroups_mean*100,
                                        vehicleDestroysminRank=rank(vehicleDestroys_min,
ties.method="max")/numGroups_mean*100,
                                        walkDistanceminRank=rank(walkDistance_min,
ties.method="max")/numGroups_mean*100,
                                        weaponsAcquiredminRank=rank(weaponsAcquired_min,
ties.method="max")/numGroups_mean*100,
                                        rankPointsminRank=rank(rankPoints_min,
ties.method="max")/numGroups_mean*100,
                                        combatScoreminRank=rank(combatScore_min,
ties.method="max")/numGroups_mean*100,


                                        assistsmaxRank=rank(assists_max,
ties.method="max")/numGroups_mean*100,
                                        boostsmaxRank=rank(boosts_max,
ties.method="max")/numGroups_mean*100,
                                        damageDealtmaxRank=rank(damageDealt_max,
ties.method="max")/numGroups_mean*100,
                                        DBNOsmaxRank=rank(DBNOs_max,
ties.method="max")/numGroups_mean*100,
                                        headshotKillsmaxRank=rank(headshotKills_max,
ties.method="max")/numGroups_mean*100,
                                        healsmaxRank=rank(heals_max,
ties.method="max")/numGroups_mean*100,
                                        killPlacemaxRank=rank(killPlace_max,
ties.method="max")/numGroups_mean*100,
                                        killsmaxRank=rank(kills_max,
ties.method="max")/numGroups_mean*100,
                                        longestKillmaxRank=rank(longestKill_max,
ties.method="max")/numGroups_mean*100,
                                        revivesmaxRank=rank(revives_max,
ties.method="max")/numGroups_mean*100,
                                        rideDistancemaxRank=rank(rideDistance_max,
ties.method="max")/numGroups_mean*100,
                                        roadKillsmaxRank=rank(roadKills_max,
ties.method="max")/numGroups_mean*100,
                                        swimDistancemaxRank=rank(swimDistance_max,
ties.method="max")/numGroups_mean*100,
                                        teamKillsmaxRank=rank(teamKills_max,
ties.method="max")/numGroups_mean*100,
                                        vehicleDestroysmaxRank=rank(vehicleDestroys_max,
ties.method="max")/numGroups_mean*100,
                                        walkDistancemaxRank=rank(walkDistance_max,
ties.method="max")/numGroups_mean*100,
                                        weaponsAcquiredmaxRank=rank(weaponsAcquired_max,
ties.method="max")/numGroups_mean*100,
                                        rankPointsmaxRank=rank(rankPoints_max,
ties.method="max")/numGroups_mean*100,
                                        combatScoremaxRank=rank(combatScore_max,
ties.method="max")/numGroups_mean*100

)

agg.r<-agg.r%>%inner_join(groups)
agg.r<-agg.r%>%inner_join(matchType)


rm(agg,test)
test<-agg.r
rm(agg.r)
gc()

test <- test %>% select(-contains("_"))
test.predictions <- test%>%select(matchId,groupId,numGroups,partySize)
test<-as.data.frame(test,stringsAsFactors=F)

test <- test %>% mutate_if(is.integer,funs(as.numeric(.)))
test<-test[,c(linear_feature)] #no match type and group id


test_mm_freq <- data.matrix(select(test))
test_dm_freq <- xgb.DMatrix(
  data = test_mm_freq,
  missing = "NAN")

rm(test,test_mm_freq)
gc()

preds<-predict(xgb1_freq,test_dm_freq)
preds<-as.data.frame(preds)
names(preds)<-c("y")
preds<-preds%>%mutate(y=ifelse(y<0,0,y),
                      y=ifelse(y>1,1,y))

test.predictions$winPlacePerc <- preds$y
test.predictions.master<-test.predictions.master%>%inner_join(test.predictions)
sub<-test.predictions.master%>%select(Id,winPlacePerc)
#head(sub,50)
```

```
write.csv(sub,"score1.csv",row.names = F)

test.predictions<-test.predictions%>%group_by(matchId)%>%mutate(rank=rank(winPlacePerc,
ties.method="first"))
test.predictions<-test.predictions%>%mutate(
  gap = 1.0 / (numGroups - 1),
  winPlacePerc = (rank)*gap,winPlacePerc,
  winPlacePerc = ifelse(numGroups==1,0,winPlacePerc))

#t<-test.predictions%>%filter(matchId=="0008c31a9be4a7")
test.predictions.master<-test.predictions.master%>%select(-
winPlacePerc)%>%inner_join(test.predictions)
sub<-test.predictions.master%>%select(Id,winPlacePerc)

write.csv(sub,"score2.csv",row.names = F)
#head(sub,50)
```