A Struggling Problem For Players:

How To Win a Rank Game In League Of Legends

Yanjie Qi Wuji Shan Zhaochi Ye

June 05, 2020

Abstract

Legal of Legends is highly competitive, fast paced action-strategy game designed for those who crave a hard fought victory. This dataset is from the Riot Gaming in the Diamond Rank Game of League of Legend. We are trying to design a model to predict how various factors affect the consequence of a game in the Diamond Rank Game of NA server. After viewing the dataset, we think methods K nearest neighbor, logistic regression, bagging, randomForest and regulation are potential methods for discovering the relationship.

Data Overview

Research Question:

- How does various factors in first ten minutes affect the consequence of a game in the Diamond Rank Game of NA server in League of Legends?

Data Exploration:

- Response variable:
  - 'blueWins': the result whether blue side wins ("1" stands for blue wins and "0" stands for red wins)
- Total predictor variables: 38
  - Numeric predictor variables: 30
  - Binary predictor variables: 8
- Predictor varibles:
  - blueWardsPlaced: Number of warding totems placed by the blue team on the map
  - blueWardsDestroyed: Number of enemy warding totems the blue team has destroyed
  - blueFirstBlood: First kill of the game. 1 if the blue team did the first kill, 0 otherwise
  - blueKills: Number of enemies killed by the blue team
  - blueDeaths: Number of deaths (blue team)
  - blueAssists: Number of kill assists (blue team)
  - blueEliteMonster: Number of elite monsters killed by the blue team (Dragons and Heralds)
  - blueDragons: Number of dragons killed by the blue team
  - blueHeralds: Number of heralds killed by the blue team
  - blueTowersDestroyed: Number of structures destroyed by the blue team (towers...)
  - blueTotalGold: Blue team total gold
  - blueAvgLevel: blue team average champion level
  - blueTotalExperience: Blue team total experience
  - blueTotalMinionsKilled: blue team total minions killed
  - blueTotalJungleMinionsKilled: Blue team total jungle monsters killed
  - blueGoldDiff: blue team gold difference compared to the enemy team
  - blueExperienceDiff: Blue team experience difference compared to the enemy team
  - blueCSPerMin: blue team CS (minions) per minute
  - blueGoldPerMin: blue team gold per minute
  - redWardsPlaced: Number of warding totems placed by the red team on the map
  - redWardsDestroyed: Number of enemy warding totems the red team has destroyed

  - redFirstBlood: First kill of the game. 1 if the red team did the first kill, 0 otherwise
  - redKills: Number of enemies killed by the red team
  - redDeaths: Number of deaths (red team)
  - redAssists: Number of kill assists (red team)
  - redEliteMonster: Number of elite monsters killed by the red team (Dragons and Heralds)
  - redDragons: Number of dragons killed by the red team
  - redHeralds: Number of heralds killed by the red team
  - redTowersDestroyed: Number of structures destroyed by the red team (towers...)
  - redTotalGold: red team total gold
  - redAvgLevel: red team average champion level
  - redTotalExperience: red team total experience
  - redTotalMinionsKilled: red team total minions killed
  - redTotalJungleMinionsKilled: red team total jungle monsters killed
  - redGoldDiff: red team gold difference compared to the enemy team
  - redExperienceDiff: red team experience difference compared to the enemy team
  - redCSPerMin: red team CS (minions) per minute
  - redGoldPerMin: red team gold per minute

Analysis plan:

- Descriptive statistics:
  - We plan to report mean and median on predictor variables, and report frequencies on binary response variables.
  - We plan to report boxplots of a potential predictor variable in the dataset.
  - We hope to develop/use supervised machine learning.
- Model building:
  - We plan to train our classifier via cross-validation.
- Model testing:
  - 2964 hold-out observations will be planned.

Reference:

- Michel's fanboi. (2020, April). League of Legends Diamond Ranked Games (10 min), Version 1. Retrieved May 27, 2020 from https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min

Data Pre-processing

- After loading the target dataset, we have the dataset with 40 columns and 9879 observations; among these 40 columns, there are 1 response variable, 38 potential predictor variables, and one 'gameID' variable that is absolutely unrelated to our research. Thus, we eliminated the 'gameID' column and summarised the missness in the data; it turns out that there is no missing value in this dataset.

Methods

- To split into training and test dataset, we took 70% of random sample in the dataset as our training set and the other 30% as test set. In the traing set, there are 6915 observations; in the test set, there are 2964 observations.
- To generally understand the response variable, we calculated the frequency of “blueWin”(blueWins == 1) and “blueLose” (blueWins == 0). Among the 6915 training observations, in 3447 games blue side won the game and in 3468 games red side won the game.
- To visualize the dataset, we summarized every column to show the overall data distribution as following:

```
##     blueWins     blueWardsPlaced blueWardsDestroyed blueFirstBlood
##  Min.   :0.000   Min.   :  5.0   Min.   : 0.00      Min.   :0.000
##  1st Qu.:0.000   1st Qu.: 14.0   1st Qu.: 1.00      1st Qu.:0.000
##  Median :0.000   Median : 16.0   Median : 3.00      Median :1.000
##  Mean   :0.499   Mean   : 22.3   Mean   : 2.83      Mean   :0.505
##  3rd Qu.:1.000   3rd Qu.: 20.0   3rd Qu.: 4.00      3rd Qu.:1.000
##  Max.   :1.000   Max.   :250.0   Max.   :27.00      Max.   :1.000
##    blueKills       blueDeaths     blueAssists    blueEliteMonsters
##  Min.   : 0.00   Min.   : 0.00   Min.   : 0.00   Min.   :0.00
##  1st Qu.: 4.00   1st Qu.: 4.00   1st Qu.: 4.00   1st Qu.:0.00
##  Median : 6.00   Median : 6.00   Median : 6.00   Median :0.00
##  Mean   : 6.18   Mean   : 6.14   Mean   : 6.64   Mean   :0.55
##  3rd Qu.: 8.00   3rd Qu.: 8.00   3rd Qu.: 9.00   3rd Qu.:1.00
##  Max.   :22.00   Max.   :22.00   Max.   :29.00   Max.   :2.00
##   blueDragons     blueHeralds    blueTowersDestroyed blueTotalGold
##  Min.   :0.000   Min.   :0.000   Min.   :0.000       Min.   :10730
##  1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000       1st Qu.:15416
##  Median :0.000   Median :0.000   Median :0.000       Median :16398
##  Mean   :0.362   Mean   :0.188   Mean   :0.051       Mean   :16503
##  3rd Qu.:1.000   3rd Qu.:0.000   3rd Qu.:0.000       3rd Qu.:17459
##  Max.   :1.000   Max.   :1.000   Max.   :4.000       Max.   :23701
##   blueAvgLevel  blueTotalExperience blueTotalMinionsKilled
##  Min.   :4.60   Min.   :10098       Min.   : 90
##  1st Qu.:6.80   1st Qu.:17168       1st Qu.:202
##  Median :7.00   Median :17951       Median :218
##  Mean   :6.92   Mean   :17928       Mean   :217
##  3rd Qu.:7.20   3rd Qu.:18724       3rd Qu.:232
##  Max.   :8.00   Max.   :22224       Max.   :283
##  blueTotalJungleMinionsKilled  blueGoldDiff    blueExperienceDiff
##  Min.   : 0.0                 Min.   :-10830   Min.   :-9333
##  1st Qu.:44.0                 1st Qu.: -1586   1st Qu.:-1290
##  Median :50.0                 Median :    14   Median :  -28
##  Mean   :50.5                 Mean   :    14   Mean   :  -34
##  3rd Qu.:56.0                 3rd Qu.:  1596   3rd Qu.: 1212
##  Max.   :92.0                 Max.   : 11467   Max.   : 8348
##   blueCSPerMin  blueGoldPerMin redWardsPlaced  redWardsDestroyed
##  Min.   : 9.0   Min.   :1073   Min.   :  6.0   Min.   : 0.00    
##  1st Qu.:20.2   1st Qu.:1542   1st Qu.: 14.0   1st Qu.: 1.00    
##  Median :21.8   Median :1640   Median : 16.0   Median : 2.00    
##  Mean   :21.7   Mean   :1650   Mean   : 22.4   Mean   : 2.72    
##  3rd Qu.:23.2   3rd Qu.:1746   3rd Qu.: 20.0   3rd Qu.: 4.00    
##  Max.   :28.3   Max.   :2370   Max.   :276.0   Max.   :24.00    
##  redFirstBlood     redKills       redDeaths       redAssists   
##  Min.   :0.000   Min.   : 0.00   Min.   : 0.00   Min.   : 0.00  
##  1st Qu.:0.000   1st Qu.: 4.00   1st Qu.: 4.00   1st Qu.: 4.00  
##  Median :0.000   Median : 6.00   Median : 6.00   Median : 6.00  
##  Mean   :0.495   Mean   : 6.14   Mean   : 6.18   Mean   : 6.66  
##  3rd Qu.:1.000   3rd Qu.: 8.00   3rd Qu.: 8.00   3rd Qu.: 9.00  
##  Max.   :1.000   Max.   :22.00   Max.   :22.00   Max.   :28.00  
##  redEliteMonsters   redDragons      redHeralds   redTowersDestroyed
##  Min.   :0.000    Min.   :0.000   Min.   :0.00   Min.   :0.000     
##  1st Qu.:0.000    1st Qu.:0.000   1st Qu.:0.00   1st Qu.:0.000     
##  Median :0.000    Median :0.000   Median :0.00   Median :0.000     
##  Mean   :0.573    Mean   :0.413   Mean   :0.16   Mean   :0.043     
##  3rd Qu.:1.000    3rd Qu.:1.000   3rd Qu.:0.00   3rd Qu.:0.000     
##  Max.   :2.000    Max.   :1.000   Max.   :1.00   Max.   :2.000     
##   redTotalGold    redAvgLevel   redTotalExperience redTotalMinionsKilled
##  Min.   :11212   Min.   :4.80   Min.   :10465      Min.   :107          
##  1st Qu.:15428   1st Qu.:6.80   1st Qu.:17210      1st Qu.:203          
##  Median :16378   Median :7.00   Median :17974      Median :218          
##  Mean   :16489   Mean   :6.92   Mean   :17962      Mean   :217          
##  3rd Qu.:17418   3rd Qu.:7.20   3rd Qu.:18764      3rd Qu.:233          
##  Max.   :22732   Max.   :8.20   Max.   :22269      Max.   :289          
##  redTotalJungleMinionsKilled  redGoldDiff     redExperienceDiff  redCSPerMin  
##  Min.   : 4.0                Min.   :-11467   Min.   :-8348     Min.   :10.7  
##  1st Qu.:44.0                1st Qu.: -1596   1st Qu.:-1212     1st Qu.:20.3  
##  Median :51.0                Median :   -14   Median :   28     Median :21.8  
##  Mean   :51.3                Mean   :   -14   Mean   :   34     Mean   :21.7  
##  3rd Qu.:57.0                3rd Qu.:  1586   3rd Qu.: 1290     3rd Qu.:23.3  
##  Max.   :92.0                Max.   : 10830   Max.   : 9333     Max.   :28.9  
##  redGoldPerMin 
##  Min.   :1121  
##  1st Qu.:1543  
##  Median :1638  
##  Mean   :1649  
##  3rd Qu.:1742  
##  Max.   :2273  
```

From the table above, we could have some general sense of how these different factors is normally presented in the first 10 minutes of the game. For example, the mean of blueWins is equal to 0.499, which verified the result that in our training set, the blue side has nearly the same win rate with the red side. Other one-sided data could show normally how players one side does various kinds of things to win the game and the range of these data distribute. For instance, blue side places ward with the range from 5 to 250 and the average 22.3.

- To better understand the dataset, we employed the boxplot to show the data distribution of total gold of blue under two situations: "blueWins" and "blueLose", since we reckoned the totalGold of blue might be one of the most related paramater.

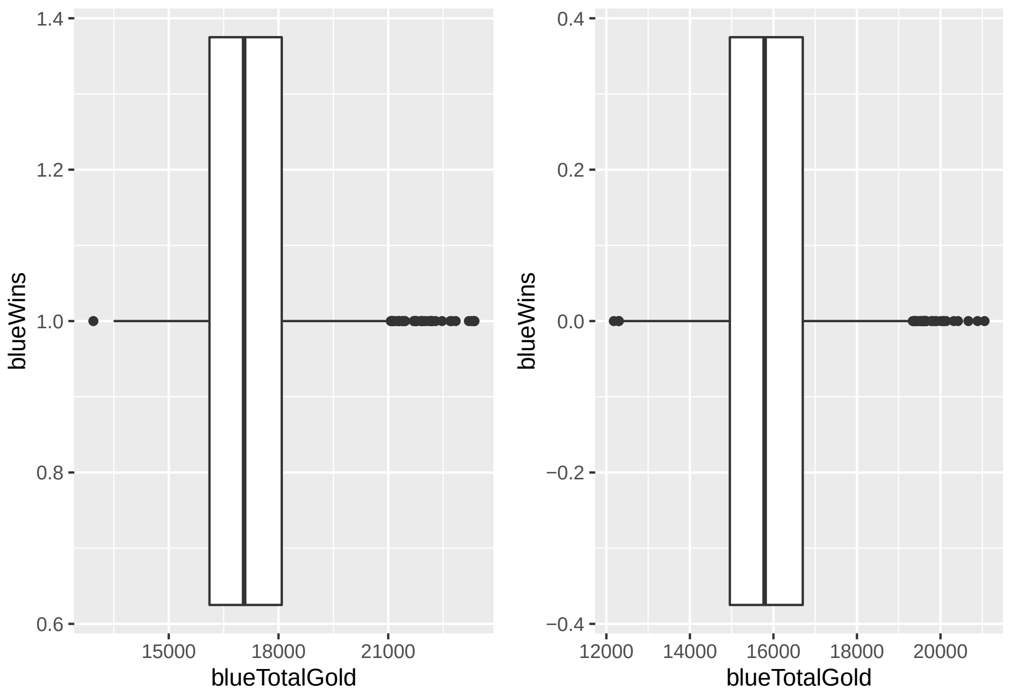

Figure 1: The Distribution of BlueTotalGold Under two situation: blueWins and redWins (blueWins: 1 = Wins, 0 = Lost)

1. Logistic Regression Method

Build and summarise a logistic regression model

Firstly, we attempt fit the model with logistic regression and explore the relationship between blueWins and other predictor variables; thus, we fit the response variable to the other variables and get the summary of model as below:

```
##
## Call:
## glm(formula = blueWins ~ ., family = binomial, data = Rank.train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.747  -0.870  -0.154   0.865   2.765
##
## Coefficients: (13 not defined because of singularities)
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   2.21e-01   1.48e+00    0.15   0.8814
## blueWardsPlaced              -1.82e-03   1.53e-03   -1.19   0.2334
## blueWardsDestroyed          -8.54e-03   1.40e-02   -0.61   0.5416
## blueFirstBlood               8.99e-02   6.29e-02    1.43   0.1528
## blueKills                   -2.23e-02   3.67e-02   -0.61   0.5432
## blueDeaths                   5.15e-02   3.67e-02    1.40   0.1605
## blueAssists                 -1.37e-02   1.38e-02   -0.99   0.3229
## blueEliteMonsters            1.49e-02   7.78e-02    0.19   0.8484
## blueDragons                  3.59e-01   1.12e-01    3.21   0.0013 **
## blueHeralds                        NA         NA      NA       NA
## blueTowersDestroyed         -2.16e-01   1.73e-01   -1.25   0.2130
## blueTotalGold               -1.13e-05   1.19e-04   -0.09   0.9244
## blueAvgLevel                -1.69e-02   2.18e-01   -0.08   0.9382
## blueTotalExperience         -8.51e-05   9.76e-05   -0.87   0.3833
## blueTotalMinionsKilled      -2.91e-03   2.32e-03   -1.26   0.2093
## blueTotalJungleMinionsKilled  2.62e-03   3.99e-03    0.66   0.5117
## blueGoldDiff                 5.33e-04   8.34e-05    6.39  1.7e-10 ***
## blueExperienceDiff           2.95e-04   7.24e-05    4.08  4.6e-05 ***
## blueCSPerMin                       NA         NA      NA       NA
## blueGoldPerMin                     NA         NA      NA       NA
## redWardsPlaced              -2.80e-04   1.60e-03   -0.17   0.8614
## redWardsDestroyed            1.31e-06   1.34e-02    0.00   0.9999
## redFirstBlood                      NA         NA      NA       NA
## redKills                           NA         NA      NA       NA
## redDeaths                          NA         NA      NA       NA
## redAssists                   1.18e-02   1.37e-02    0.86   0.3915
## redEliteMonsters            -8.18e-02   8.38e-02   -0.98   0.3289
## redDragons                  -1.50e-01   1.16e-01   -1.29   0.1971
## redHeralds                         NA         NA      NA       NA
## redTowersDestroyed           3.73e-01   1.87e-01    2.00   0.0460 *
## redTotalGold                       NA         NA      NA       NA
## redAvgLevel                 -4.58e-03   2.18e-01   -0.02   0.9832
## redTotalExperience                 NA         NA      NA       NA
## redTotalMinionsKilled        6.77e-03   2.32e-03    2.92   0.0035 **
## redTotalJungleMinionsKilled  9.48e-03   3.95e-03    2.40   0.0165 *
## redGoldDiff                        NA         NA      NA       NA
## redExperienceDiff                  NA         NA      NA       NA
## redCSPerMin                        NA         NA      NA       NA
## redGoldPerMin                      NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9586.2  on 6914  degrees of freedom
## Residual deviance: 7260.0  on 6889  degrees of freedom
## AIC: 7312
##
## Number of Fisher Scoring iterations: 4
```

Interpret coefficient

- In above result, blueDragons, blueGoldDiff, blueExperienceDiff, redTowersDestroyed, redTotalMinionsKilled, and redTotalJungleMinionsKilled are statistically significant at level 0.05.
- The coefficient for blueDragon is 0.359; it manifests that the dragon blue side takes positively affect their win rate.
- The coefficient for blueGoldDiff is 0.000533; it manifests that, similar with blueDragon, the more Gold blue side wins, the more likely they win the game; however, it does influence as much as blueDragon does.
- The coefficient for blueExperienceDiff is 0.000295; it is likely has the same result as the two variables above and it has less effect.
- The coefficient for redTowersDestroyed is 0.373; it means that the number of towers red side destroyed, in fact, helps to elevate the win rate of blue side.
- The coefficient for redTotalMinionsKilled is 0.00677; it has the similar effect with redTowersDestroyed.
- The coefficient for redTotalJungleMinionsKilled is 0.00948, which is similar with the previous two variables.

Construct confusion matrix for the training data

- Building up the confusion matrix for the training set, we attempt to see how the model functions generally:

```
##       true
## pred    0    1
##    0 2560  912
##    1  908 2535
```

- From the confusion matrix, we calculate the Accurate Rate for the model is 0.7368; the correct classified rate, out of all lost game, is 0.7382; the correct classified rate, out of all winned game, is 0.7354.

Estimate win rate for the test data

- Then, we employed the model above to fit the test set to see how the model fucntions and got the confusion matrix below for the test set:

```
##       true
## pred    0    1
##    0 1087  425
##    1  394 1058
```

- Throught calculation, we have the accurate rate 0.7237 and the error rate 0.2763.
- The accurate rate for test set is close to the accurate rate for training set. The model looks reasonable and could be the potential final model.

2. K-Nearest Neighbor Method

- We use LOOCV to find validation error for 1-NN, 2-NN, …, 50-NN.

- Set validation.error (a vector) to save validation errors in future and give possible number of nearest neighbours to be considered.
- The validation errors for k = 1 to k = 50 is as following:

```
##  [1] 0.3657 0.3640 0.3303 0.3304 0.3119 0.3151 0.3021 0.3018 0.3007 0.2973
## [11] 0.2969 0.2966 0.2915 0.2913 0.2875 0.2894 0.2871 0.2892 0.2882 0.2868
## [21] 0.2839 0.2847 0.2840 0.2847 0.2853 0.2852 0.2821 0.2830 0.2820 0.2801
## [31] 0.2797 0.2805 0.2811 0.2801 0.2790 0.2787 0.2791 0.2777 0.2758 0.2765
## [41] 0.2759 0.2758 0.2764 0.2778 0.2765 0.2787 0.2766 0.2766 0.2764 0.2765
```

- Find the best number of neighbors. If there is a tie, pick larger number of neighbors for simpler model.

- Therefore, we determine the best number for kNN is k = 42.

- Use the best k = 42 to get the confusion matrix:

```
##          true
## predicted    0    1
##         0 1084  469
##         1  397 1014
```

- From the confusion matrix, we know that 1084+1014=2098 games are predicted correctly (accurately), and 397+469=866 games are predicted incorrectly (with error).

- Then, though calculation, the accuracy rate is 0.7078 and the test error rate is 0.2922.

3. Random Forest & bagging

Random Forest

```
## 0.008599 0.05
## -0.04654 0.05
```

- In this section, OBB error is considered as the criteria for accuracy on test set and use varing *mtry* parameter to find the best *mtry*, which is used to control the complexity for single tree by control the number of variables used in single tree. According to the result, best *mtry* parameter are selected as 3 while the OBB error is least.

- According to the result error decreasing and keep stable as ntree increasing, thus ntree parameter is suitable in this model.

- Importance of these variable is showing above. It indicates that gold difference and expirence difference are the most important factors to affect the result of a game.

- It is indicated that error rate is 27.67% in 10-flod cross validation, which is near to the OBB error rate.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1088  427
```

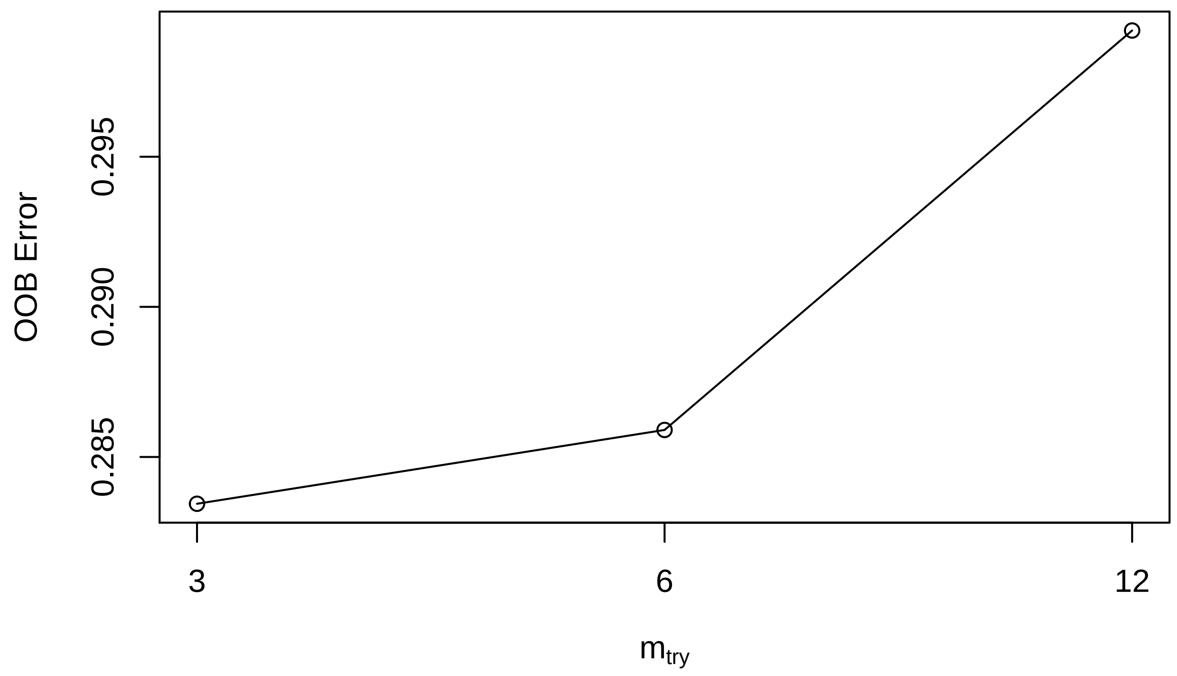

Figure 2: Tune for mtry parameter

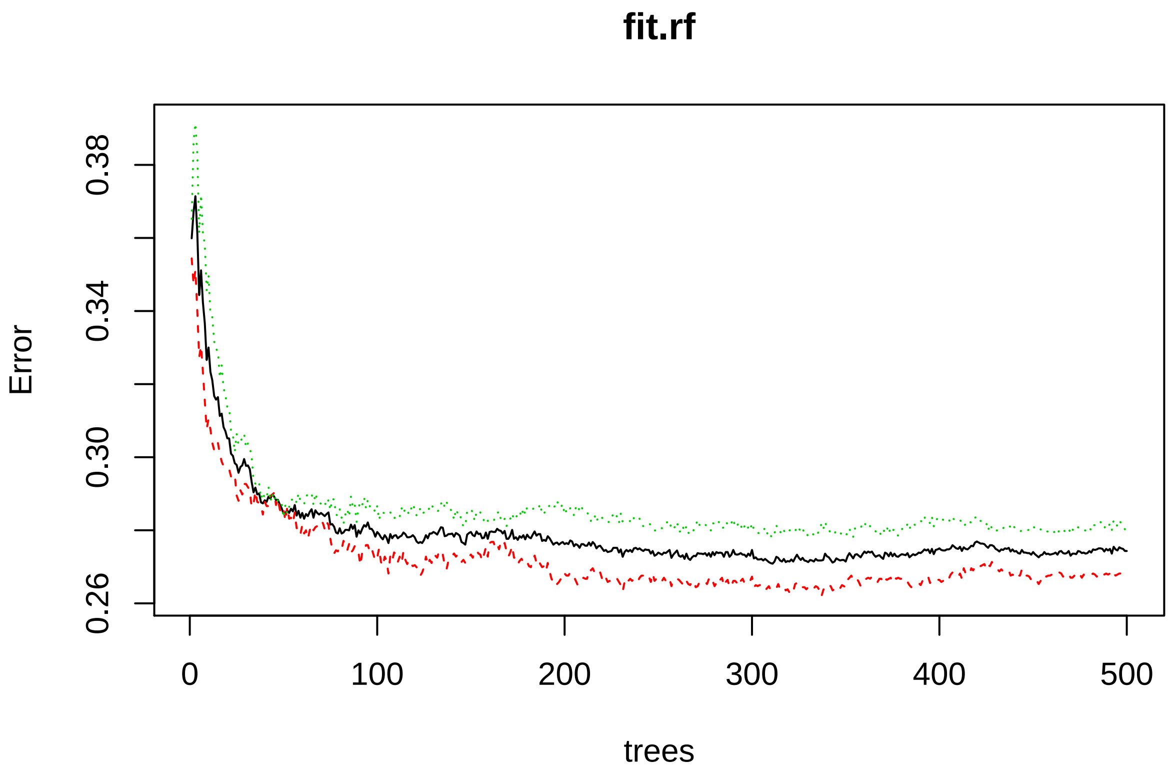

Figure 3: Number of tree and corresponding error

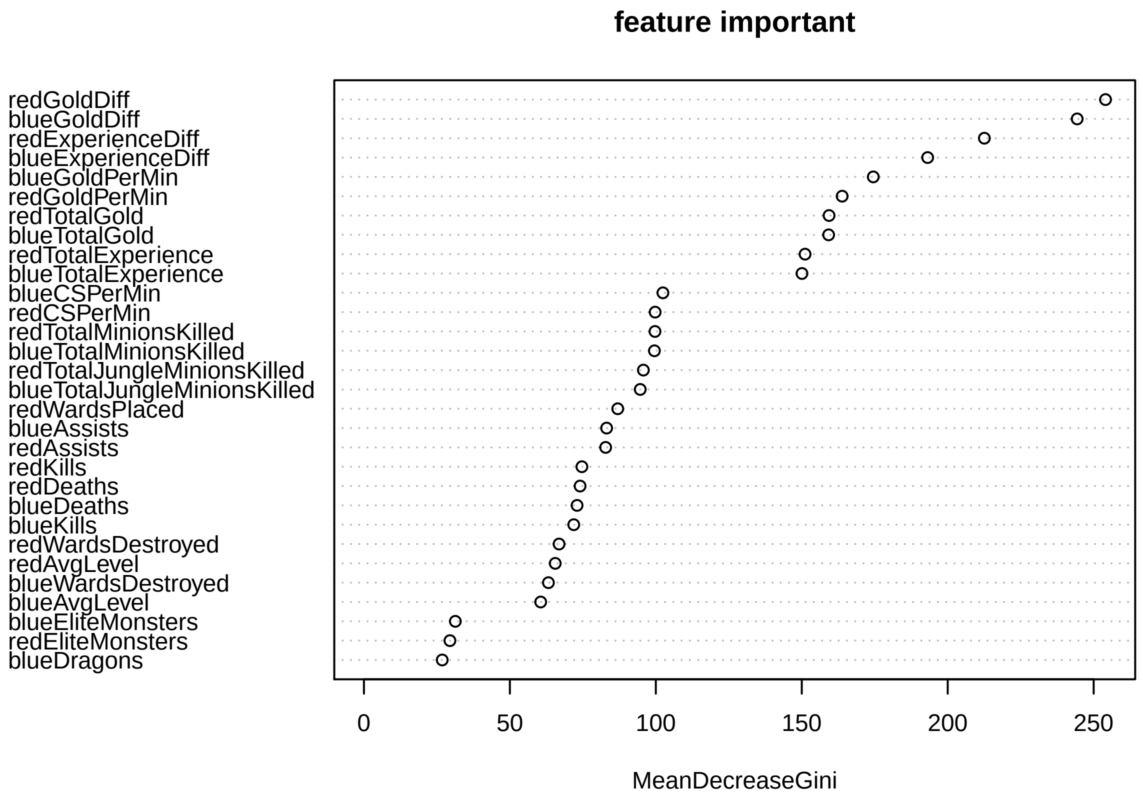

Figure 4: Importance of variable

```
##          1  393 1056
##
##                Accuracy : 0.723
##                  95% CI : (0.707, 0.739)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.447
##
##  Mcnemar's Test P-Value : 0.249
##
##             Sensitivity : 0.735
##             Specificity : 0.712
##          Pos Pred Value : 0.718
##          Neg Pred Value : 0.729
##              Prevalence : 0.500
##          Detection Rate : 0.367
##    Detection Prevalence : 0.511
##       Balanced Accuracy : 0.723
##
##        'Positive' Class : 0
##
```

- From the table above, though the calculation, we have the accuracy that is 0.723; the error rate is 0.277.

Bagging

Fit the training set through bagging using randomForest function. Set the mtry = p = 38, indicating that 38 predictors should be considered for each split of the tree. Besides, the independent variable importance in bagged tree is assessed. Thus have the following table and plot:

```
##
## Call:
##  randomForest(formula = blueWins ~ ., data = Rank.train, mtry = 38,      importance = T
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 38
##
##         OOB estimate of  error rate: 27.62%
## Confusion matrix:
##      0    1 class.error
## 0 2539  929      0.2679
## 1  981 2466      0.2846
```

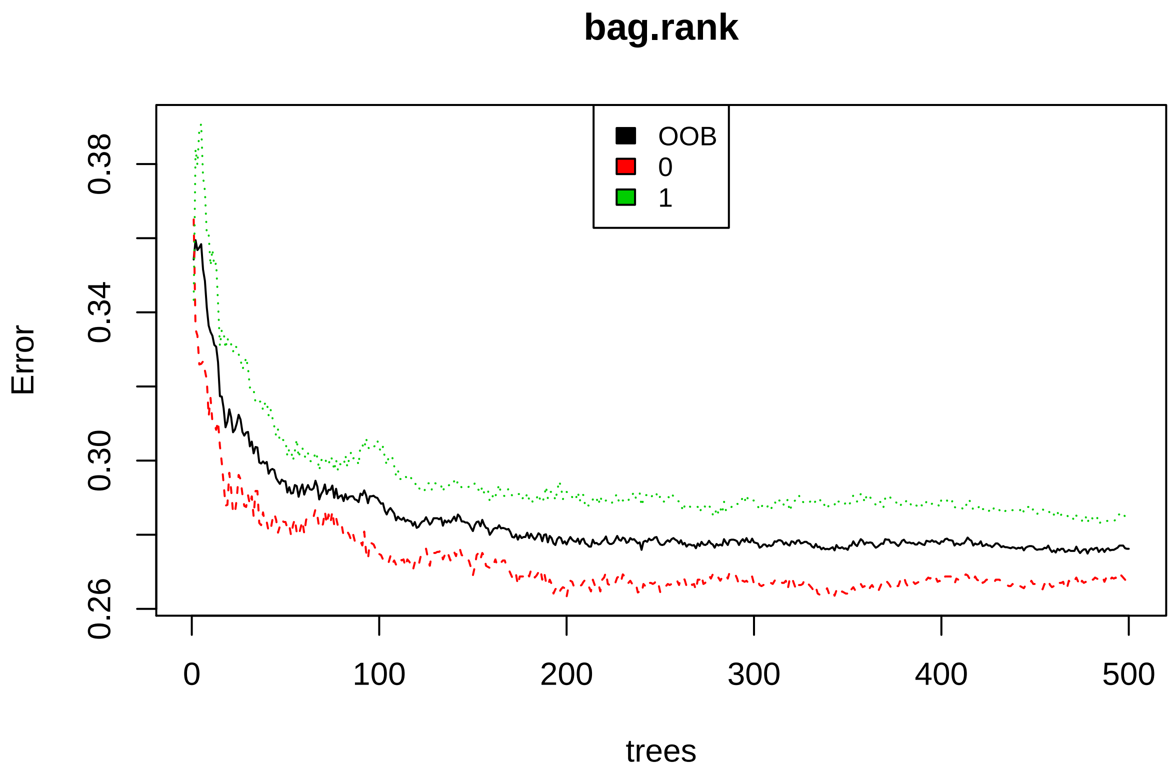

Figure 5: Error Rate for Different Trees

- The plot shows the error rate under different number of trees. The table gives us that the OOB error rate = 27.62%.

- Through Calculation, we could have the lowest OOB error rate = 0.2749 under the condition that trees number equals to 478.

Then, we fit the model to the test set to get the confusion matrix and calculate the error rate:

```
##     truth
## pred    0    1
##    0 1068  437
##    1  413 1046
```

```
## [1] 0.2868
```

- From the confusion matrix, through calculation, we could see that the error rate is 0.2814.

4. Regulation

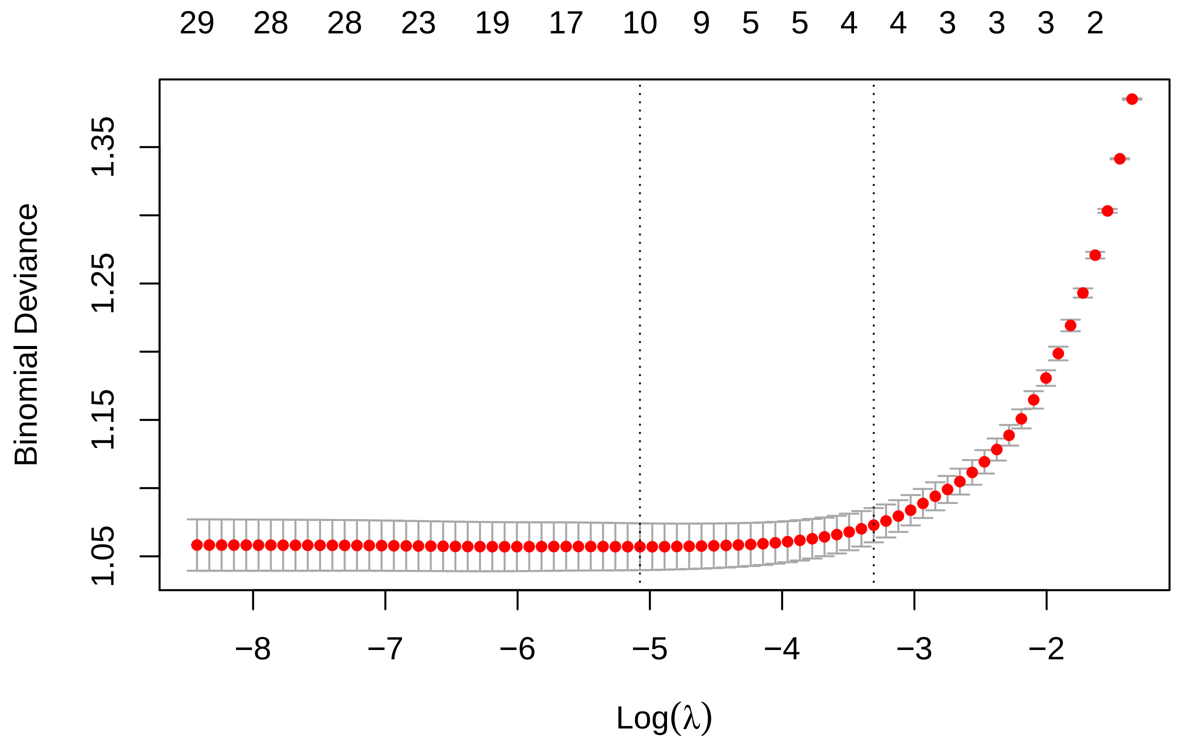

Figure 6: lamdba Value and correspondinh Binomial Deviance

- According to the result of 10-fold cross validation, lambda parameters are selected while Binomial Deviance in test set is least. Thus only 10 variable are retained in the model.
- According to coefficients of lasso logistic regression model, the most important factors affecting result of game are Dragons, Elite Monster and First Blood.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1080  423
##          1  401 1060
```

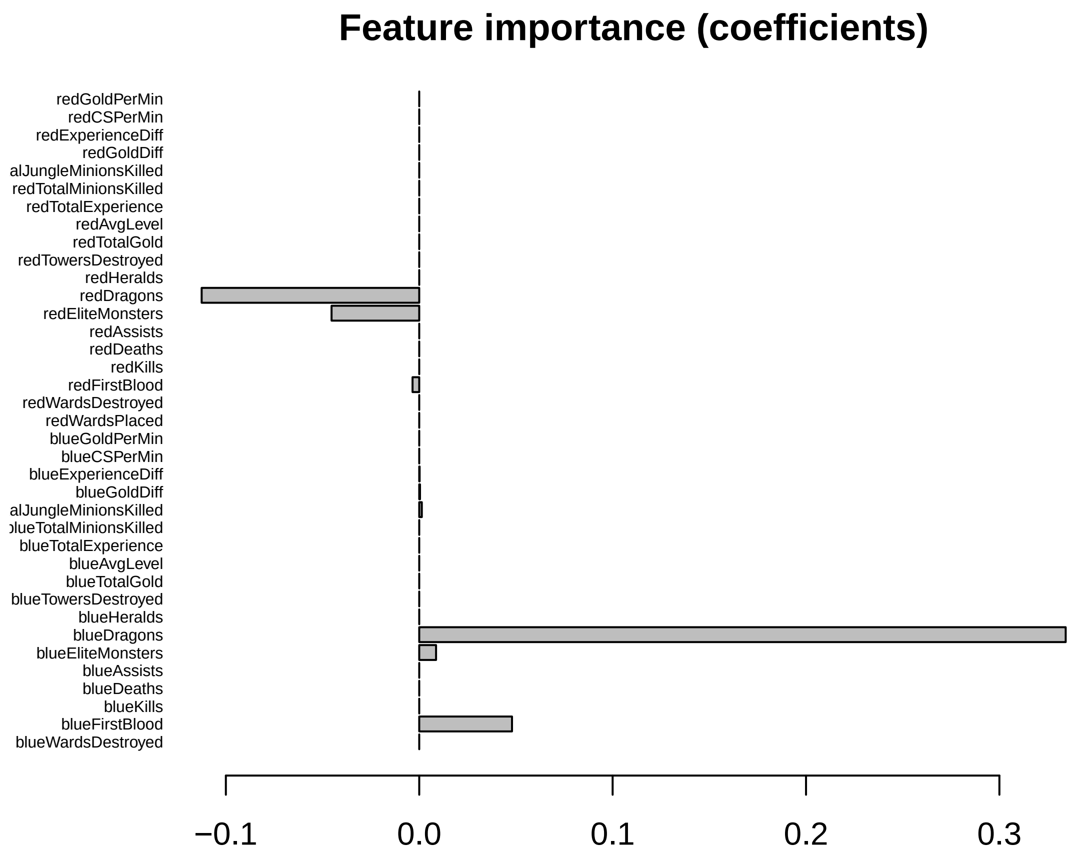

Figure 7: Feature Importance of each variable based on the lasso model

```
##
##                Accuracy : 0.722
##                  95% CI : (0.705, 0.738)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.444
##
##  Mcnemar's Test P-Value : 0.464
##
##             Sensitivity : 0.729
##             Specificity : 0.715
##          Pos Pred Value : 0.719
##          Neg Pred Value : 0.726
##              Prevalence : 0.500
##          Detection Rate : 0.364
##    Detection Prevalence : 0.507
##       Balanced Accuracy : 0.722
##
##        'Positive' Class : 0
##
```

- From the table, for this model, the accuracy rate is 0.722, the error rate is 0.278.

Data Analysis and Conclusion

- From the models above, the error rate of logistic regression model is 0.2763, error rate of k-NN model is 0.2922, error rate of randomForest is 0.277, error rate of bagging is 0.2814, error rate of regulation is 0.278.
- 0.2763 is the smallest error rate, so we can conclude that for this dataset, logistic regression model is the best model, which is our final model.
- In logistic regression model summary results, blueDragons, blueGoldDiff, blueExperienceDiff, redTowersDestroyed, redTotalMinionsKilled, and redTotalJungleMinionsKilled are statistically significant at level 0.05.
- As we analysed in the logistic regression part, we have the conclusion about the coefficient of significant variables:
  - The underline reason could be that in the game currently, if red side kept farming and not getting advanatages on Dragons, Gold in the first ten minutes, the blue side is more likely to win the game. As for the experiences, it might be the reason that Kills, in the game, now win more experience than the lane experience like miniors and jungle minors.

Appendix

```r
library(knitr)
options(digits = 4)  #limit the number of
```

```r
library(tidyverse)
library(ForImp)
library(dplyr)
library(randomForest)
library(gbm)
library(ISLR)
library(tree)
library(class)

# Load the data
RankDt <- read.csv("~/Box/Final Project/Dataset/high_diamond_ranked_10min.csv")

# Dimension of the data
dim(RankDt)

# Primary Data cleaning
# Exclude GameId column since it is obviously not related to the model
RankD <- RankDt %>%
  dplyr::select(-gameId)

# Summarize the missingness in the data
for (i in names(RankD)){
  print(which(is.na(RankD$i)))
}
## There is no missing value in the data set

# set the seed
set.seed(2020)
# training and test dataset
train <- sample(1:nrow(RankD), 0.7*nrow(RankD))
Rank.train <- RankD[train,]
Rank.test <- RankD[-train,]
# dimensions of training set
dim(Rank.train)
# dimensions of test set
dim(Rank.test)

# since our response varibale is a binary variable, we intend to
# have a table to show the frequency of wins and loses
# frequency of blue wins
table(Rank.train$blueWins )

# Summary of Dataset
summary(RankD)

blueTotalGold_Wins <- Rank.train %>%
  select(blueTotalGold, blueWins) %>%
  filter(blueWins == 1)
blueTotalGold_lose <- Rank.train %>%
  select(blueTotalGold, blueWins) %>%
  filter(blueWins == 0)
require(gridExtra)

plot1 <- qplot(x = blueTotalGold, y = blueWins,
               data = blueTotalGold_Wins, geom = "boxplot")
plot2 <- qplot(x = blueTotalGold, y = blueWins,
               data = blueTotalGold_lose, geom = "boxplot")
grid.arrange(plot1, plot2, ncol=2)

glm.fit <- glm(blueWins ~ .,
               data = Rank.train, family = binomial)

summary(glm.fit)

# Specify type="response" to get the estimated probabilities
prob.training <- predict(glm.fit, type = "response")
# Round the result to 2 decimal places
round(prob.training, digits = 2)

# Save the predicted labels using 0.5 as a threshold
Rank.train <- Rank.train %>%
  mutate(preBlueWins=as.factor(ifelse(prob.training<=0.5, 0, 1)))
# Confusion matrix (training error/accuracy)
table(pred=Rank.train$preBlueWins, true=Rank.train$blueWins)

# Accurate rate
Lg_AR <- (2560+2535) / (2560+912+908+2535)
Lg_AR # 0.7368
# Out of All lost game, correct classified rate
LCCR <- 2560 / (2560 + 908)
LCCR # 0.7382
# Out of All winned game, correct classified rate
WCCR <- 2535 / (2535 + 912)
WCCR # 0.7354

# Predict the win rate and round the predict results to 5 decimals
prob.test <- round(predict(glm.fit, Rank.test, type = "response"), digits = 5)
Rank.test <- Rank.test %>%
  mutate(winRate=prob.test)
# Predict bluw win or not column
Rank.test <- Rank.test %>%
  mutate(preBlueWins=as.factor(ifelse(prob.test<=0.5, 0, 1)))
```

```r
# confusion matrix (test error)
table(pred=Rank.test$preBlueWins, true=Rank.test$blueWins)

# test accurate rate
Lg_testMR <- (1087+1058) / (1087+425+394+1058)
Lg_testMR

# Reload training and test dataset in case there was change of specific variables
# set the seed
set.seed(2020)
# training and test dataset
train <- sample(1:nrow(RankD), 0.7*nrow(RankD))
Rank.train <- RankD[train,]
Rank.test <- RankD[-train,]

# YTrain is the observed labels for bluwWins on the training set,
# XTrain is the design matrix
YTrain = Rank.train$blueWins
XTrain = Rank.train %>% select(-blueWins)
XTrain <- scale(XTrain,center = TRUE, scale = TRUE)
str(XTrain)

meanvec <- attr(XTrain,'scaled:center')
sdvec <- attr(XTrain,'scaled:scale')
# YTest is the observed labels for blueWins on the test set,
# Xtest is the design matrix
YTest = Rank.test$blueWins
XTest = Rank.test %>% select(-blueWins) %>% scale(center = meanvec,
                                                  scale = sdvec)

validation.error = NULL
allK = 1:50
set.seed(66)
for (i in allK){
# Predict on the left-out validation set
pred.Yval = knn.cv(train=XTrain, cl=YTrain, k=i)
validation.error = c(validation.error, mean(pred.Yval!=YTrain))
}
validation.error

numneighbor = max(allK[validation.error == min(validation.error)])
numneighbor

set.seed(67)
pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=numneighbor)
conf.matrix = table(predicted=pred.YTest, true=YTest)
conf.matrix

sum(diag(conf.matrix)/sum(conf.matrix))
1 - sum(diag(conf.matrix)/sum(conf.matrix))

library(randomForest)
library(caret)

# parameters tune
set.seed(2020)
fit.rf<-tuneRF(x=Rank.train[,-1:-2],y=as.factor(Rank.train$blueWins),
               trace=F,doBest=T,plot=T)

plot(fit.rf)
varImpPlot(fit.rf,main='feature important',cex=0.75)

# Cross validation
CVgroup <- function(k,datasize,seed){
  cvlist <- list()
  set.seed(seed)
  n <- rep(1:k,ceiling(datasize/k))[1:datasize]
  temp <- sample(n,datasize)
  x <- 1:k
  dataseq <- 1:datasize
  cvlist <- lapply(x,function(x) dataseq[temp==x])
  return(cvlist)
}
cvtest <- function(i){
  train <- Rank.train[i,]
  test <- Rank.train[-i,]
  fit0 <- randomForest(x=train[,-1:-2],y=as.factor(train$blueWins),
                       mtry=fit.rf$mtry)
  pred<-predict(fit0,test)
  return(1-sum(diag(table(pred,test$blueWins)))/nrow(test))
}
cvlist<-CVgroup(k=10,data=nrow(Rank.train),seed=2020)
cv.res<-lapply(cvlist,cvtest)
average.error<-mean(unlist(cv.res))
average.error

#predict
pred<-predict(fit.rf,Rank.test)
confusionMatrix(pred,as.factor(Rank.test$blueWins))

# Reload training and test dataset in case there was change of specific variables
# set the seed
```

```r
set.seed(2020)
# training and test dataset
train <- sample(1:nrow(RankD), 0.7*nrow(RankD))
Rank.train <- RankD[train,]
Rank.test <- RankD[-train,]

glimpse(Rank.train)

Rank.train$blueWins <- as.character(Rank.train$blueWins)
Rank.train$blueWins <- as.factor(Rank.train$blueWins)
bag.rank <- randomForest(blueWins~., data = Rank.train, mtry=38,
                         importance = TRUE)
bag.rank

plot(bag.rank)
legend("top", colnames(bag.rank$err.rate), col = 1:4, cex = 0.8, fill = 1:4)

which.min(bag.rank$err.rate[,1])
bag.rank$err.rate[478,1]

# show how the model is well performed in test set
yhat.bag <- predict(bag.rank, newdata = Rank.test)


# Confusion matrix
bag.err <- table(pred=yhat.bag, truth=Rank.test$blueWins)
bag.err
test.bag.err <- 1- sum(diag(bag.err))/sum(bag.err)
test.bag.err

library(glmnet)

set.seed(2020)
cv.lasso<-cv.glmnet(x=as.matrix(Rank.train[,-1:-2]),
                    y=as.factor(Rank.train$blueWins),family='binomial')
plot(cv.lasso)
fit.lasso<-glmnet(x=as.matrix(Rank.train[,-1:-2]),
                  y=as.factor(Rank.train$blueWins),family='binomial',
                  lambda=cv.lasso$lambda.min)

# feature importance
barplot(as.numeric(fit.lasso$beta),horiz = T,
        names.arg = rownames(fit.lasso$beta),las=1,cex.names=0.5,
        main='Feature importance (coefficients)')

#predict
```

```
pred<-predict(fit.lasso,as.matrix(Rank.test[,-1:-2]))
pred<-as.factor(ifelse(pred>0,1,0))
confusionMatrix(pred,as.factor(Rank.test$blueWins))
```