University of Miami
Herbert Business School

Relationship of League of Legends Win Rate and Various in Game Characteristics

Tim Yu, Jenny Tu, Jiahao Ye, Jiahao Chang

MAS548-T

Professor Xu Ganggang

May 10, 2022

Contents

1 Initial Report **4**

1.1 Introduction **4**

1.2 Motivation and Objective **4**

1.3 Attributes and Predictors **5**

1.3.1 Attributes and Predictors Table **5**

1.3.2 Attributes and Predictors Table **5**

1.3.3 The Descriptions for the Attributes **6**

1.3.4 The Descriptions for the Predictor **9**

1.4 Boxplot (Win vs Other Attributes) **9**

1.5 Histogram **10**

1.6 Finding from QQ-Plot visualization **11**

1.7 Log transformation **12**

2 Full Logistic Regression Model **13**

2.1 Fix Singularities **17**

2.2 Check for Multicollinearity **18**

2.3 Identify Outliers **21**

2.4 Identify Non-linear Relationship **23**

2.5 New Model Performance **23**

3 Model Selection **25**

3.1 Perform Stepwise Selection Using AIC **25**

3.2 Perform Stepwise Selection Using BIC **26**

3.3 Training Misclassification Rates & Training AUC values **28**

3.4 Split Data into Training Data + Test Data **28**

3.5 Cross Validation **29**

4 Principal Component Analysis **29**

4.1 Standardization **29**

4.2 PCs Selection **30**

4.3 Test Error Comparison **30**

5 Tree Models **31**

5.1 Single Tree Model **31**

5.2 Pruned Tree Model **32**

5.3 Bagging tree **33**

5.4 Random Forest Tree Model **35**

5.5 Final Tree Model **37**

6 Summary **38**

References **40**

Contribution **40**

Appendix **41**

1 Initial Report

1.1 Introduction

League of Legends is one of the world's most popular video games, developed by Riot Games. It features a team-based competitive game mode based on strategy and outplaying opponents. Players work with their team to break the enemy Nexus before the enemy team breaks theirs. Our group has found a dataset that includes 18 attributes that correlated to the outcome of 9879 high elo ranked games, such as player's KDA, first blood, minions killed per minutes, wards placement as a team, wards destroyed as a team, first dragon, first rift-herald, and so on.

All the samples in the dataset is i.i.d which were collected from the high elo player in the United States, website NA.OP.GG. 9879 observed variables are enough for us to fit the logistic regression model. The reason why we choose the high elo player instead of the low elos is that a little advantage to a high elo player will snowball tremendously in a short period. The data from high elo players would be more reliable in counting those attributes because the player in the lower elo sometimes does not know how to end the game even they are at a huge leading. The average duration of a high elo ranked game is around 20 minutes; the average duration of a ranked game is around 35 minutes.

1.2 Motivation and Objective

The main purpose of this project is to identify how to increase the win rate and climb to a higher rank in League of Legends. A logistic regression model will be implemented in our research, to find the probability of winning a game with given characteristics.

1.3 Attributes and Predictors

1.3.1 Attributes and Predictors Table

In our data set, there are 18 attributes and 1 predictor. The simple attributes and predictor description table are as follows:

Attribute	Type of Attribute	Levels
Wards Placed	Numerical	NA
Wards Destroyed	Numerical	NA
First Blood	Categorical	2-Levels
Player Kills	Numerical	NA
Player Death	Numerical	NA
Player Assists	Numerical	NA
Killed Elite Monsters as a Team	Numerical	NA
First Dragon	Categorical	2-Levels
First Rift Heralds	Categorical	2-Levels
Player Towers Destroyed	Numerical	NA
Player Total Gold	Numerical	NA
Team Avg Level	Numerical	NA
Team Total Experience	Numerical	NA
Total Minions Killed	Numerical	NA
CS Per Min	Numerical	NA
Gold Per Min	Numerical	NA
Opponent Wards Placed	Numerical	NA
Opponent Wards Destroyed	Numerical	NA
Predictor	Type of Predictor	Levels
Win	Categorical	2-Levels

1.3.2 Attributes and Predictors Table

Each game is described by the following 18attributes:

Wards_Placed Wards_Destroyed First_Blood Player_Kills Player_Death Player_Assists Killed_Elite_Monsters_As_A_Team First_Dragon First_Rift_Heralds Player_Towers_Destroyed Player_Total_Gold Team_Avg_Level Team_Total_Experience

Total_Minions_Killed CS_Per_Min Gold_Per_Min Opponent_Wards_Placed Opponent_Wards_Destroyed First_First_Rift_Heralds.

The table below shows the first 6 rows of the datasetwhich gives us a good look at the types of values each attribute can take.

	Win <fctr>	Wards_Placed <int>	Wards_Destroyed <int>	First_Blood <fctr>	Player_Kills <int>	Player_Death <int>	Player_Assists <int>
1	0	28	2	1	9	6	11
2	0	12	1	0	5	5	5
3	0	15	0	0	7	11	4
4	0	43	1	0	4	5	5
5	0	75	4	0	6	6	6
6	1	18	0	0	5	3	6

Total_Minions_Killed <int>	CS_Per_Min <dbl>	Gold_Per_Min <dbl>	Opponent_Wards_Placed <int>	Opponent_Wards_Destroyed <int>
195	19.5	1721.0	15	6
174	17.4	1471.2	12	1
186	18.6	1611.3	15	3
201	20.1	1515.7	15	2
210	21.0	1640.0	17	2
225	22.5	1589.9	36	5

Player_Total_Gold <int>	Team_Avg_Level <dbl>	Team_Total_Experience <int>	Total_Minions_Killed <int>	CS_Per_Min <dbl>
17210	6.6	17039	195	19.5
14712	6.6	16265	174	17.4
16113	6.4	16221	186	18.6
15157	7.0	17954	201	20.1
16400	7.0	18543	210	21.0
15899	7.0	18161	225	22.5

1.3.3 The Descriptions for the Attributes

Wards Placed – A numerical attribute represents how many wards did the observation placed in the game. A wards cost 0 to 100 gold depending on a different kind of ward.

Wards Destroyed – A numerical attribute represents how many opponent's wards did the observation destroyed in the game. Every time a player destroys an opponent's ward, they will reward with a fixed amount of gold.

First Blood – A Categorical attribute represents whether the observation got the first blood in the game. "1" represents the observation made the kill, and "0" represents the observation did not make the kill.

Player Kills – A numerical attribute represents how many opponents did the observation killed in the game. Every kill will reward the observation some amount of gold depending on different in-game situations.

Player Death – A numerical attribute represents how many times the observation was killed by either opponent, minion, or monster.

Player Assists – A numerical attribute represents how many times the observation helped their teammate kill opponents. Player assist kills will reward the observation a small amount of gold depending on different in-game situations.

Killed Elite Monsters as a Team – A numerical attribute represents how many elite monsters the observation team killed. Although this attribute only contains three numbers (0, 1, 2) in the data set, the attribute is numerical since this number will increase as the playtime increases.

First Dragon - A Categorical attribute represents which team kills the first dragon. "1" represents the observation's team made the kill, and "0" represents the opponent's team made the kill. A dragon kill will reward the team with a fixed amount of gold and buff.

First Rift Heralds - A Categorical attribute represents which team kills the first rift heralds. "1" represents the observation's team, and "0" represents the opponent's team. A rift herald will reward the team with a fixed amount of gold and buff.

Player Towers Destroyed – A numerical attribute represents how many towers did the observation destroyed in the game. Every time a player destroys an opponent's tower will reward with a fixed amount of gold.

Player Total Gold – A numerical attribute represents how many golds the observation received in the game.

Team Avg Level – A numerical attribute represents the average level for the observation's team.

Team Total Experience – A numerical attribute represents how many experiences the observation's team received in the game.

Total Minions Killed – A numerical attribute represents how many minions the observation killed in the game. Every time a player kills a minion, reward in a fixed amount of gold depends on the type of minion.

CS Per Min – A numerical attribute represents how many minions and neutral monsters did the observation killed in the game per minute.

Gold Per Min – A numerical attribute represents how much gold the observation received in the game per minute.

Opponent Wards Placed – A numerical attribute represents how many wards the opponent placed in the game.

Opponent Wards Destroyed – A numerical attribute represents how many observation's team's wards did the opponent destroyed in the game.

1.3.4 The Descriptions for the Predictor

Win – A categorical predictor represents whether the observation won the game or not. "1" represents the observation won, and "0" represents the opponent won the game.

1.4 Boxplot (Win vs Other Attributes)

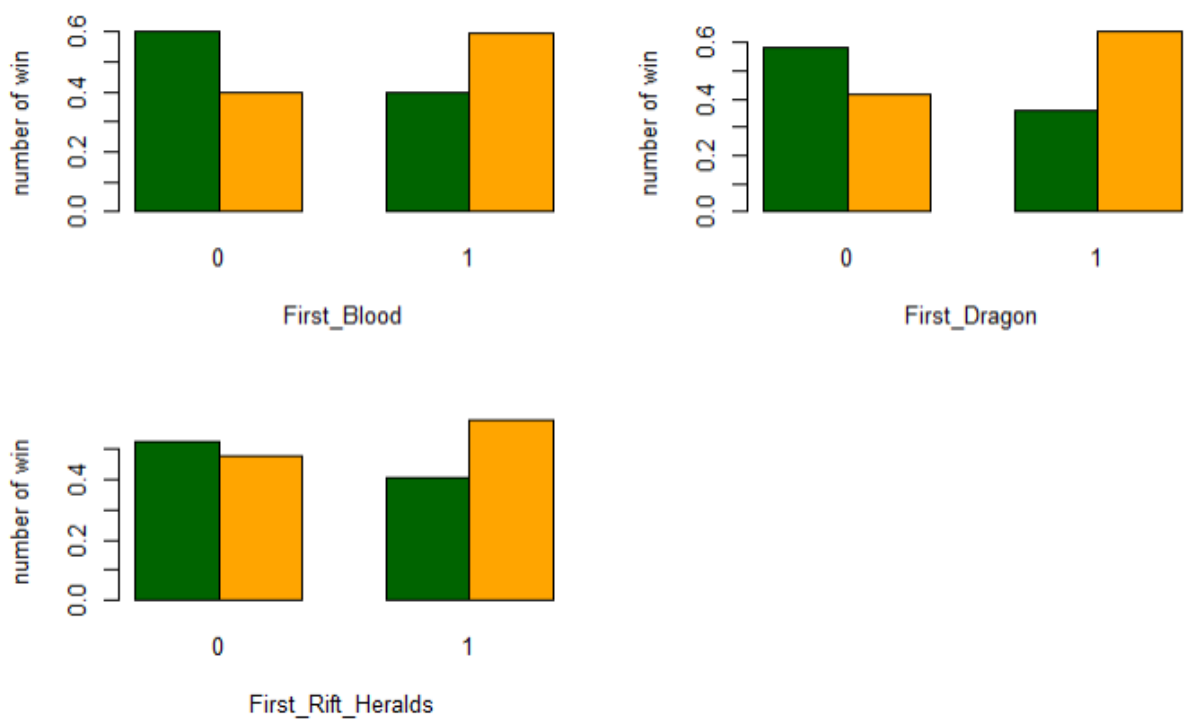


Figure 1

In Figure 1, we can see that there seems to be a somewhat positive association between the win and First_Blood, First_Dragon, and First_Rigt_Heralds. Intuitively, this makes sense when the team get the first dragon of this game. Which provide the advantage to gain the gold and buy powerful weapon, and then, they have greater chance to win the game.

1.5 Histogram

Figure	Numeric Variables	Mean	Median	Distribution	Skewness		
6.01	Wards_Placed	22.29	16.00	Bimodal			
6.02	Wards_Destroyed	2.82	3.00	Normal	2.85	Right Skewed	Highly Skewed
6.03	Player_Kills	6.18	6.00	Normal	0.54	Right Skewed	Moderately Skewed
6.04	Player_Death	6.14	6.00	Normal	0.51	Right Skewed	Moderately Skewed
6.05	Player_Assists	6.65	6.00	Normal	0.89	Right Skewed	Moderately Skewed
6.06	Killed_Elite_Monsters_As_A_Team	0.55	0.00	Exponential			
6.07	Player_Towers_Destroyed	0.05	0.00	Exponential			
6.08	Player_Total_Gold	16,503.46	16,398.00	Normal	0.47	Right Skewed	Approximately Symmetric
6.09	Team_Avg_Level	6.92	7.00	Normal	-0.34	Left Skewed	Approximately Symmetric
6.10	Team_Total_Experience	17,928.11	17,951.00	Normal	-0.25	Left Skewed	Approximately Symmetric
6.11	Total_Minions_Killed	216.70	218.00	Normal	-0.27	Left Skewed	Approximately Symmetric
6.12	CS_Per_Min	21.67	21.80	Normal	-0.27	Left Skewed	Approximately Symmetric
6.13	Gold_Per_Min	1,650.35	1,639.80	Normal	0.47	Right Skewed	Approximately Symmetric
6.14	Opponent_Wards_Placed	22.37	16.00	Bimodal			
6.15	Opponent_Wards_Destroyed	2.72	2.00	Normal	2.95	Right Skewed	Highly Skewed

In the 1.5 Histogram, it shows that there are six variables are almost symmetric, and three moderately skewed, and two highly skewed. When we fit those high skewed variable into our model, we may consider to use some transformation method to make the distribution more normal distributed. But the most attributes are pretty nice, which is applicable for the model.

The response variable is Win. Histograms for Player Kills Histogram and Killed Elite Monster as Team Histogram are shown in Figure 2.* From Figure 2, one can see that above ten player kill distribution. There is significant difference between the red(lose) and blue(win) zone. Which indicates that the more player you kill, the more likely you win. At the 15 player kill zone, we can barely see the red zone. At this zone, the teams with 15 or more play killed are going to win in most case.

That is, Player Kills has a right skeweddistribution. We can use the log-transformation to make it normal distribution. But after we fit the log(Player Kills) into our model. We find the Error rate increase. As a result, we keep the Player Kills.

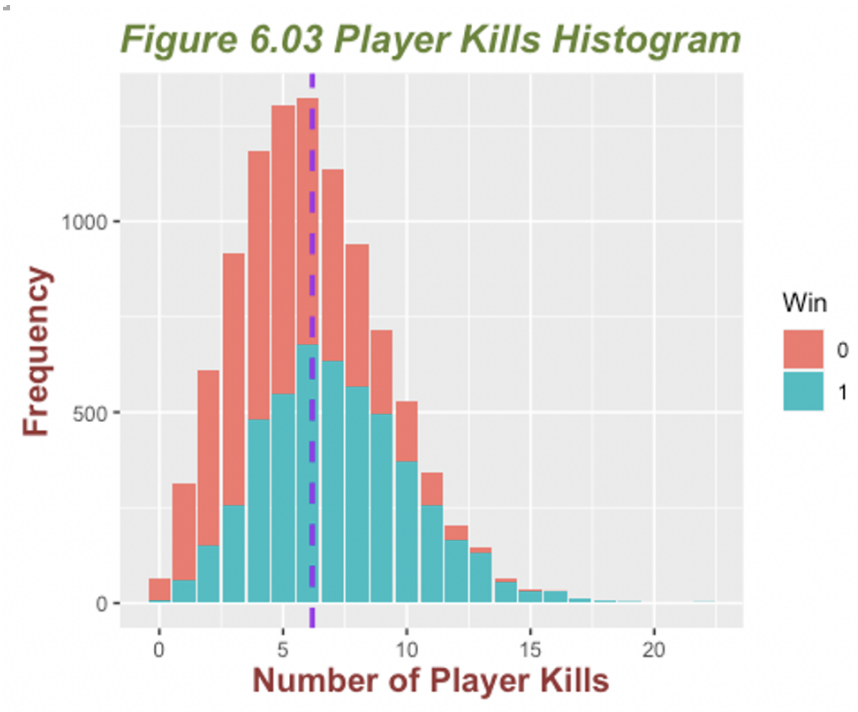
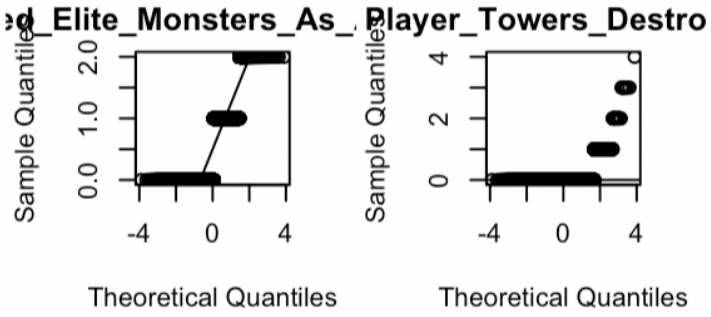


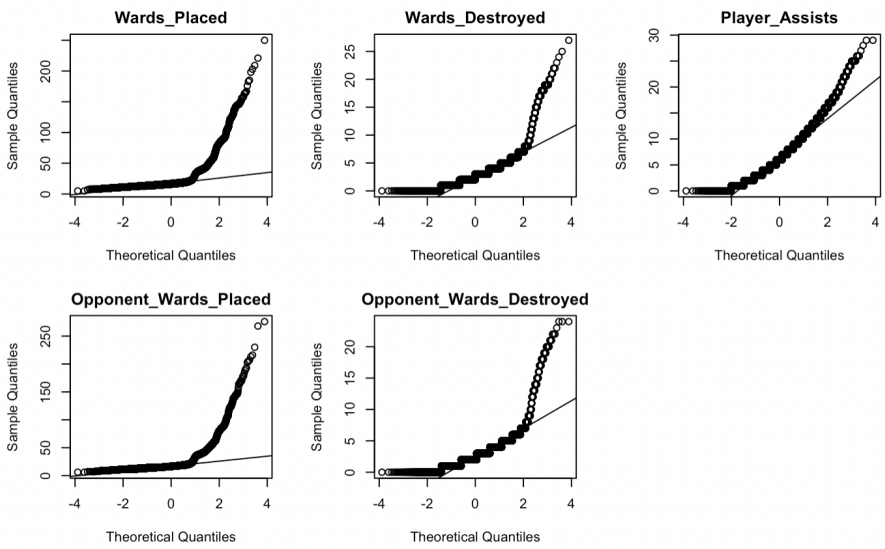
Figure 2

1.6 Finding from QQ-Plot visualization



By looking at all the QQ plots, we found two independent variables of strange - monsters killed by players as a team and towers destroyed by players. The discrete pattern shown on my QQ plot made us think that maybe we should convert these variables into categorical variables. In addition, each elite monster has its own buff, which would boost the entire team; if a player destroyed the tower at the beginning, this would give the team a compatible advantage.

1.7 Log transformation



There are five independent variables that we can log-transform based on a review of QQ plot. The term ward is a deployable unit that removes the fog of war from an area of the map. We decided to do a log transformation to following: wards destroyed, wards placed, Player assistance. We encountered some difficulties during the log transformation because of the large number of zeros in "destroyed" and "player assisted". Ward destroyed can be zero because the player must spend gold to buy the scan item, which they can choose not to. Ward Placement cannot be zero because each player receives a Ward Placement item at the beginning of the

match and wards are refreshed every 1.5 minutes. Player assistance can also be zero, if a top liner stays on top and no jungle comes to help. The top player will not receive any assistance if he is split pushing.

2 Full Logistic Regression Model

Since the response variable "Win" is categorical and nominal, having only two values of 0 and 1 to indicate lose or win the game, this is a classification problem, and we would like to find a good logistic regression model in order to know how to increase the win rate. The basic logistic regression equation is

$$logit(p(x)) = log\frac{p(x)}{1-p(x)} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

where $x = (x_1, x_2, \ldots, x_p)$ are p predictors.

The first thing we did is fitting a logistic regression model with "Win" as the response and all other variables as predictors. From the model summary shown below and by looking at the p-values, we can find that *Player_Death, Player_Assists, First_Dragon, Player_Towers_Destroyed, Player_Total_Gold,* and *Total_Minions_Killed* appear to be associated with the probability of winning the game. The model seems to be reasonable because if you ask LoL players what are the most important factors that affect the result of the game, these are the ones they usually answer. However, many predictors have large insignificant p-values, and they could be potentially removed from the model like *Killed_Elite_Monsters_As_A_Team* and *Team_Avg_Level*.

The estimated coefficients can be interpreted, for example, for the predictor variable "Player_Death", the estimated coefficient is -2.452e-01, assuming holding other factors constant,

for each unit increase of player's death, the odds of the player winning the game will increase by a factor of exp(-2.452e-01) = 0.7825 = 78.25% on average. For categorical explanatory variables like "First_Dragon", the interpretation of the estimated coefficient is as follows. The estimated coefficient for "First_Dragon" is 4.815e-01, assuming holding other factors constant, compared to the no first dragon group (base group), the odds of the player in the first dragon group winning the game will increase by a factor of exp(4.815e-01) = 1.6185 times on average.

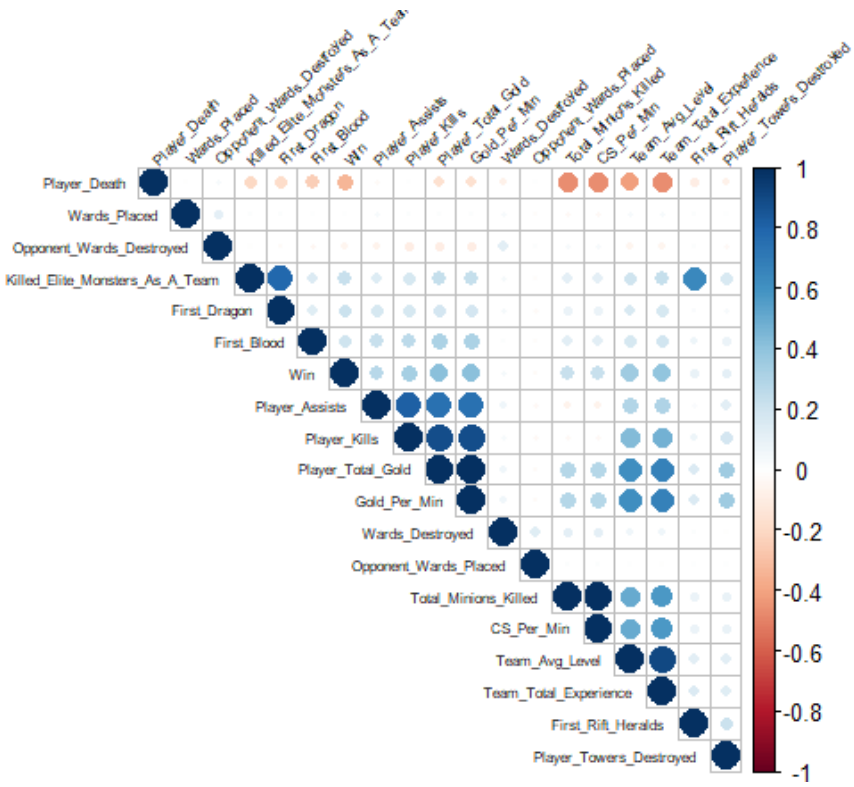
```
##
## Call:
## glm(formula = Win ~ ., family = "binomial", data = lol)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7014  -0.9005  -0.1403   0.8996   2.6753
##
## Coefficients: (3 not defined because of singularities)
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -1.030e+01  8.237e-01 -12.504  < 2e-16 ***
## Wards_Placed                     -1.513e-03  1.323e-03  -1.143 0.252919
## Wards_Destroyed                   6.485e-03  1.124e-02   0.577 0.563948
## First_Blood1                      4.651e-02  5.028e-02   0.925 0.354917
## Player_Kills                      5.557e-03  2.590e-02   0.215 0.830097
## Player_Death                     -2.452e-01  1.102e-02 -22.245  < 2e-16 ***
## Player_Assists                   -3.698e-02  1.119e-02  -3.304 0.000954 ***
## Killed_Elite_Monsters_As_A_Team1  4.474e-03  7.699e-02   0.058 0.953664
## Killed_Elite_Monsters_As_A_Team2  1.871e-01  1.317e-01   1.420 0.155501
## First_Dragon1                     4.815e-01  8.223e-02   5.856 4.73e-09 ***
## First_Rift_Heralds1                      NA         NA      NA       NA
## Player_Towers_Destroyed1         -4.605e-01  1.505e-01  -3.059 0.002217 **
## Player_Towers_Destroyed2          9.973e-01  1.061e+00   0.940 0.347226
## Player_Towers_Destroyed3          9.890e+00  1.800e+02   0.055 0.956181
## Player_Towers_Destroyed4          8.251e+00  5.354e+02   0.015 0.987704
## Player_Total_Gold                 6.992e-04  6.382e-05  10.955  < 2e-16 ***
## Team_Avg_Level                   -1.228e-02  1.795e-01  -0.068 0.945489
## Team_Total_Experience             7.962e-05  5.527e-05   1.441 0.149724
## Total_Minions_Killed             -4.487e-03  1.730e-03  -2.593 0.009516 **
## CS_Per_Min                               NA         NA      NA       NA
## Gold_Per_Min                             NA         NA      NA       NA
## Opponent_Wards_Placed            -1.983e-03  1.289e-03  -1.538 0.124007
## Opponent_Wards_Destroyed         -1.149e-02  1.130e-02  -1.017 0.309188
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13695  on 9878  degrees of freedom
## Residual deviance: 10668  on 9859  degrees of freedom
## AIC: 10708
##
## Number of Fisher Scoring iterations: 12
```

We then tried to use log-transformed variables instead of the original ones in the model to see if the model performance gets better, although the logistic regression does not require the normality assumption of the residuals. Based on the AIC values in the summary, as the AIC of the model using log-transformed variables is higher, we decided to continue our investigation on the original full model.

```
Call:
glm(formula = Win ~ . - Opponent_Wards_Placed - Opponent_Wards_Destroyed -
    Wards_Placed - Wards_Destroyed - Player_Assists, family = "binomial",
    data = lol)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7394  -0.8976  -0.1464   0.9027   2.6904

Coefficients: (3 not defined because of singularities)
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                     -9.855e+00  8.405e-01 -11.726  < 2e-16 ***
First_Blood1                     5.283e-02  5.021e-02   1.052  0.29268
Player_Kills                    -5.785e-04  2.582e-02  -0.022  0.98212
Player_Death                    -2.419e-01  1.093e-02 -22.125  < 2e-16 ***
Killed_Elite_Monsters_As_A_Team1  9.513e-03  7.694e-02   0.124  0.90159
Killed_Elite_Monsters_As_A_Team2  2.011e-01  1.317e-01   1.527  0.12682
First_Dragon1                    4.710e-01  8.219e-02   5.731 9.98e-09 ***
First_Rift_Heralds1                     NA         NA      NA       NA
Player_Towers_Destroyed1        -4.342e-01  1.504e-01  -2.887  0.00389 **
Player_Towers_Destroyed2         1.045e+00  1.061e+00   0.985  0.32481
Player_Towers_Destroyed3         9.977e+00  1.799e+02   0.055  0.95577
Player_Towers_Destroyed4         8.302e+00  5.354e+02   0.016  0.98763
Player_Total_Gold                6.741e-04  6.318e-05  10.670  < 2e-16 ***
Team_Avg_Level                  -1.641e-02  1.795e-01  -0.091  0.92715
Team_Total_Experience            9.473e-05  5.478e-05   1.729  0.08376 .
Total_Minions_Killed            -4.205e-03  1.733e-03  -2.426  0.01528 *
CS_Per_Min                              NA         NA      NA       NA
Gold_Per_Min                            NA         NA      NA       NA
logOpponent_Wards_Placed        -6.914e-02  4.979e-02  -1.389  0.16494
logOpponent_Wards_Destroyed     -6.928e-02  4.634e-02  -1.495  0.13490
logWards_Placed                 -3.671e-02  5.015e-02  -0.732  0.46416
logWards_Destroyed               4.870e-02  4.660e-02   1.045  0.29596
logPlayer_Assists               -1.693e-01  7.078e-02  -2.392  0.01675 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 13695  on 9878  degrees of freedom
Residual deviance: 10672  on 9859  degrees of freedom
AIC: 10712

Number of Fisher Scoring iterations: 12
```

2.1 Fix Singularities

One unusual thing that caught our attention is those NAs in the summary of the full model. They are due to singularities between our predictor variables, and singularity means perfect multicollinearity, so two or more predictors in this model have an exact linear relationship. In order to fix this problem, we first used the **cor()** function to see the correlation matrix of all predictors and identify which variables in the dataset have a perfect correlation with each other, finding ones or negative ones. For example, we found that *Gold per min* is perfectly correlated with *Player_Total_Gold*, which makes sense. Then, we just simply dropped one of those variables causing the problem from the regression model. So, we removed *CS per min, Gold per min*, and *First Rift Heralds*.



```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 13695  on 9878  degrees of freedom
Residual deviance: 10668  on 9859  degrees of freedom
AIC: 10708

Number of Fisher Scoring iterations: 12
```

2.2 Check for Multicollinearity

When going through the correlation matrix, we also noticed some high correlations among some of the predictors, which suggest that we have multicollinearity in our model. Since multicollinearity would make regression model results less reliable, we went ahead and checked

```
Call:
glm(formula = Win ~ . - CS_Per_Min - Gold_Per_Min - First_Rift_Heralds,
    family = "binomial", data = lol)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7014  -0.9005  -0.1403   0.8996   2.6753

Coefficients:
                                   Estimate Std. Error z value Pr(>|z|)
(Intercept)                      -1.030e+01  8.237e-01 -12.504  < 2e-16 ***
Wards_Placed                     -1.513e-03  1.323e-03  -1.143 0.252919
Wards_Destroyed                   6.485e-03  1.124e-02   0.577 0.563948
First_Blood1                      4.651e-02  5.028e-02   0.925 0.354917
Player_Kills                      5.557e-03  2.590e-02   0.215 0.830097
Player_Death                     -2.452e-01  1.102e-02 -22.245  < 2e-16 ***
Player_Assists                   -3.698e-02  1.119e-02  -3.304 0.000954 ***
Killed_Elite_Monsters_As_A_Team1  4.474e-03  7.699e-02   0.058 0.953664
Killed_Elite_Monsters_As_A_Team2  1.871e-01  1.317e-01   1.420 0.155501
First_Dragon1                     4.815e-01  8.223e-02   5.856 4.73e-09 ***
Player_Towers_Destroyed1         -4.605e-01  1.505e-01  -3.059 0.002217 **
Player_Towers_Destroyed2          9.973e-01  1.061e+00   0.940 0.347226
Player_Towers_Destroyed3          9.890e+00  1.800e+02   0.055 0.956181
Player_Towers_Destroyed4          8.251e+00  5.354e+02   0.015 0.987704
Player_Total_Gold                 6.992e-04  6.382e-05  10.955  < 2e-16 ***
Team_Avg_Level                   -1.228e-02  1.795e-01  -0.068 0.945489
Team_Total_Experience             7.962e-05  5.527e-05   1.441 0.149724
Total_Minions_Killed             -4.487e-03  1.730e-03  -2.593 0.009516 **
Opponent_Wards_Placed            -1.983e-03  1.289e-03  -1.538 0.124007
Opponent_Wards_Destroyed         -1.149e-02  1.130e-02  -1.017 0.309188
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

if there is severe multicollinearity using the **vif()** function in the R **car** package. Due to multi-level categorical variables in our dataset – *Killed_Elite_Monsters_As_A_Team* and *Player_Towers_Destroyed*, the function provides generalized variance inflation factor (GVIF) calculations automatically. GVIF^(1/(2*Df)) makes GVIFs comparable across dimensions, where Df (degrees of freedom) is the number of estimated coefficients of a variable in the model, and by squaring GVIF^(1/(2*Df)), we were able to apply the general VIF rule of thumb to determine the degree of multicollinearity (Buteikis). In the following, the (GVIF^(1/(2*Df)))^2 value is called VIF for short.

```
                                    GVIF Df GVIF^(1/(2*Df))
Wards_Placed                    1.024918  1        1.012382
Wards_Destroyed                 1.059674  1        1.029405
First_Blood                     1.135291  1        1.065500
Player_Kills                    8.760599  1        2.959831
Player_Death                    1.489792  1        1.220570
Player_Assists                  3.213921  1        1.792741
Killed_Elite_Monsters_As_A_Team 2.841964  2        1.298388
First_Dragon                    2.782860  1        1.668191
Player_Towers_Destroyed         1.416185  4        1.044456
Player_Total_Gold              11.725710  1        3.424282
Team_Avg_Level                  4.058370  1        2.014540
Team_Total_Experience           5.641276  1        2.375137
Total_Minions_Killed            2.329756  1        1.526354
Opponent_Wards_Placed           1.023436  1        1.011650
Opponent_Wards_Destroyed        1.043294  1        1.021418
```

From the VIF result shown below, four variables, which are *Player_Total_Gold, Player_Kills, Team_Total_Experience,* and *Team_Avg_Level*, have larger VIF values that are greater than 4. Among them, *Player_Total_Gold* has the highest VIF, which is more than 10, indicating there is a serious multicollinearity problem requiring correction.

Wards_Placed	1.024918
Player_Kills	8.760599
Killed_Elite_Monsters_As_A_Team	1.685813
Player_Total_Gold	11.725710
Total_Minions_Killed	2.329756
Wards_Destroyed	1.059674
Player_Death	1.489792
First_Dragon	2.782860
Team_Avg_Level	4.058370
Opponent_Wards_Placed	1.023436
First_Blood	1.135291
Player_Assists	3.213921
Player_Towers_Destroyed	1.090888
Team_Total_Experience	5.641276
Opponent_Wards_Destroyed	1.043294

Considering why the VIF of *Player_Total_Gold* is so large, it is understandable because there are many and various ways to get gold. As a result, this variable is related to many other predictors. For instance, if the player kills the opponent or assists his team members to kill the opponent, or the player kills minions, he will gain a certain amount of gold. Therefore, *Player_Total_Gold* does not provide unique or independent information in the regression model. The other three variables also need some investigation. To deal with multicollinearity, what we did is to remove the variable with the highest VIF and see whether the VIF values after that become better.

Wards_Placed	1.024261
Player_Kills	3.780267
Killed_Elite_Monsters_As_A_Team	1.683920
Team_Avg_Level	4.035971
Opponent_Wards_Placed	1.022878
Wards_Destroyed	1.059095
Player_Death	1.423882
First_Dragon	2.774598
Team_Total_Experience	5.088237
Opponent_Wards_Destroyed	1.042522
First_Blood	1.110787
Player_Assists	2.813523
Player_Towers_Destroyed	1.015308
Total_Minions_Killed	1.741093

Decreases a lot!

Still high!

After removing *Player_Total_Gold*, we could find that the VIF of *Player_Kills* decreases a lot since they are highly linearly related as previously discussed. The VIF of *Team_Total_Experience* is still high, so we tried to exclude it from the logistic regression model.

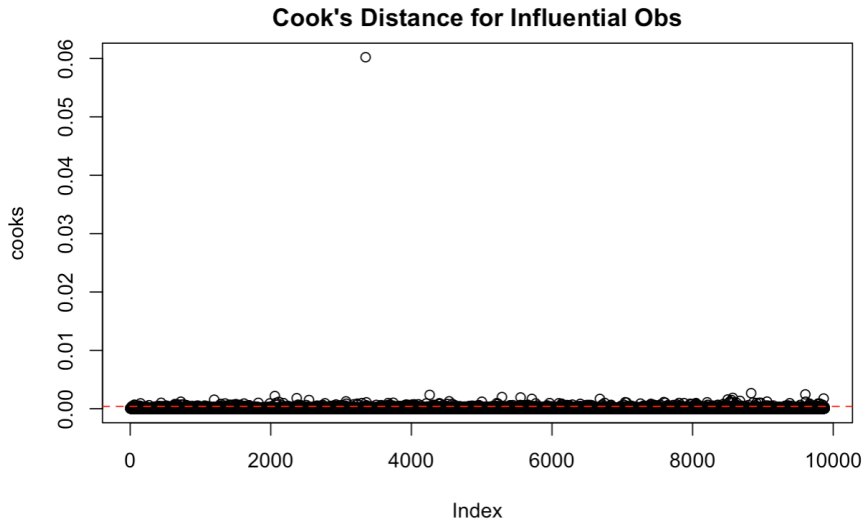
The reason why it has a high VIF value is similar to the reason discussed for *Player_Total_Gold* – experience is obtained in many different ways, so *Team_Total_Experience* is related to some other predictor variables.

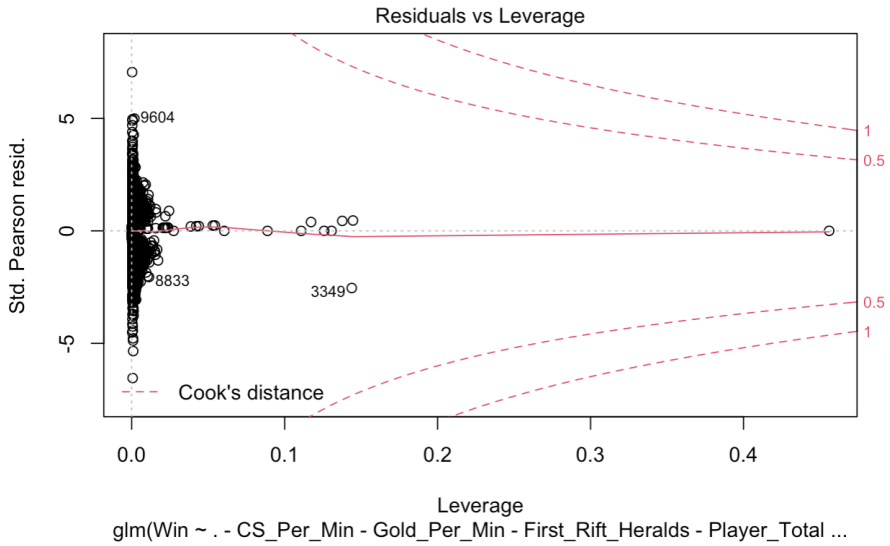
```
                    Wards_Placed                 Wards_Destroyed                     First_Blood
                        1.022868                        1.058942                        1.102906
                    Player_Kills                    Player_Death                  Player_Assists
                        3.380412                        1.378126                        2.743869
Killed_Elite_Monsters_As_A_Team                    First_Dragon         Player_Towers_Destroyed
                        1.680731                        2.770514                        1.015270
                  Team_Avg_Level            Total_Minions_Killed           Opponent_Wards_Placed
                        1.613700                        1.523453                        1.022857
       Opponent_Wards_Destroyed
                        1.042030
```

After removing *Player_Total_Gold* and *Team_Total_Experience*, the VIF values look great – all less than 4.

2.3 Identify Outliers

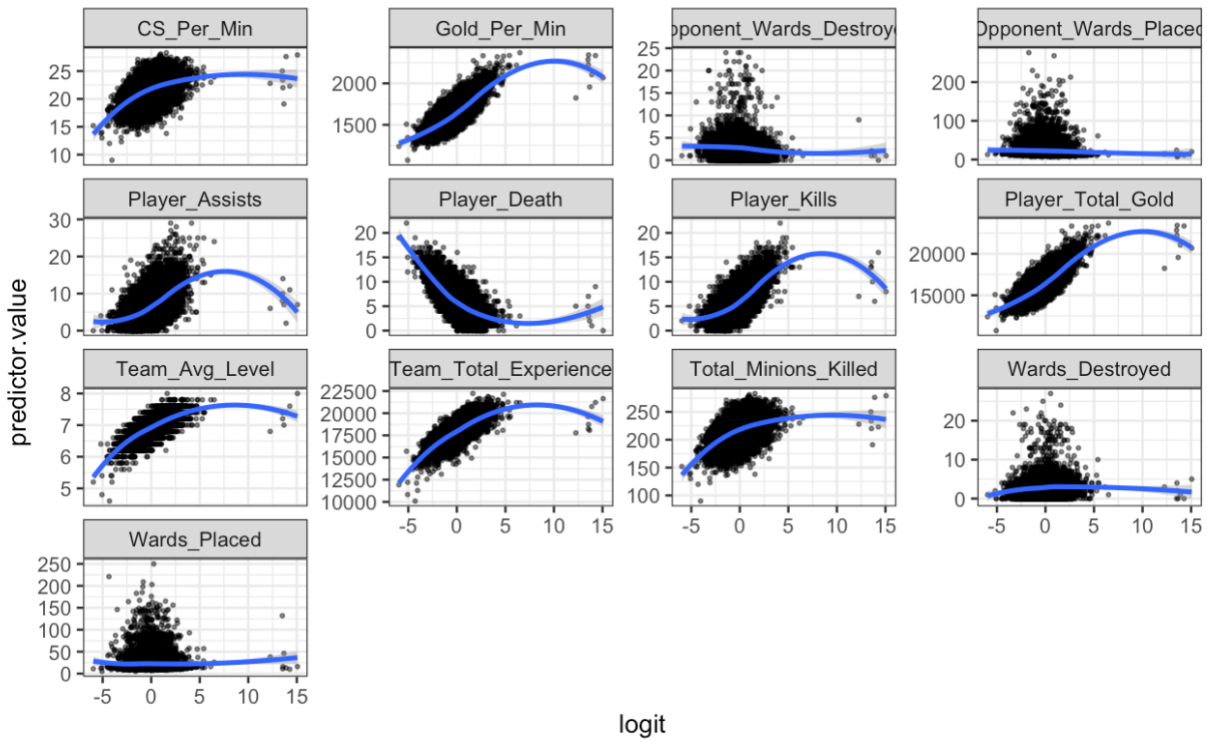
Then, we tried to identify extreme outliers or influential observations in the dataset using cook's distance for each observation. The problem here is that there is no hard cut off rate. When using 4/n as the cut off, we can see many observations are near the line, but those above the line are not necessarily to be considered influential as the beauty of LoL is that it has many different influencing factors and many uncertainties. We want to keep those data as long as they are not too abnormal. We also checked the Residuals and Leverage plot and concluded that there are no extreme outliers or influential points in our dataset.


Cook's Distance for Influential Obs
cooks
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0
2000
4000
6000
8000
10000
Index


Residuals vs Leverage
Std. Pearson resid.
5
0
-5
9604
8833
3349
1
0.5
0.5
1
Cook's distance
0.0
0.1
0.2
0.3
0.4
Leverage
glm(Win ~ . - CS_Per_Min - Gold_Per_Min - First_Rift_Heralds - Player_Total ...

2.4 Identify Non-linear Relationship

To identify any non-linear relationship, we checked the relationships between the predictor variables and the logit of the outcome "Win". The scatter plots show that all the variables are quite linearly associated with the "Win" outcome in logit scale. If the variable does not demonstrate a linear relationship, it might need some transformations like using polynomial terms in the model, but in our case, we do not see a particularly big concern of it.



2.5 New Model Performance

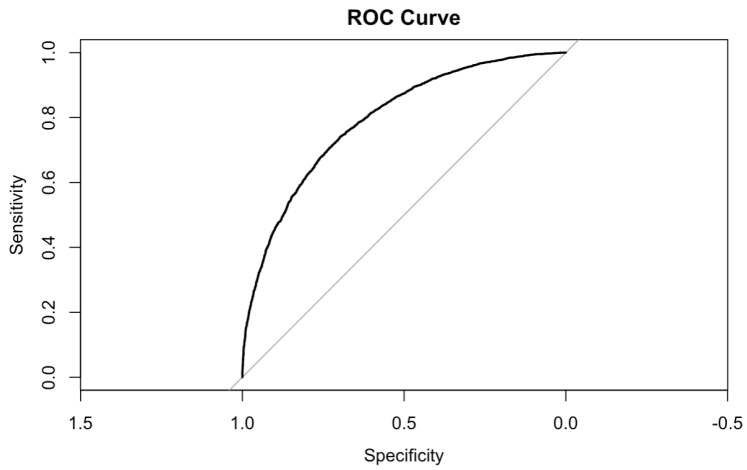
After the diagnostic of potential problems in the data and model, we got a logistic regression model, which provides better interpretation. In the model summary, *First_Blood, Player_Kills, Player_Death, Killed_Elite_Monsters_As_A_Team, First_Dragon,*

Player_Towers_Destroyed, Team_Avg_Level, and *Total_Minions_Killed* seem to be associated with the probability of winning the game.

```
##
## Call:
## glm(formula = Win ~ . - CS_Per_Min - Gold_Per_Min - First_Rift_Heralds -
##     Player_Total_Gold - Team_Total_Experience, family = "binomial",
##     data = lol)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7494  -0.9105  -0.1602   0.9144   2.8027
##
## Coefficients:
##                                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       -6.784312   0.701820  -9.667  < 2e-16 ***
## Wards_Placed                      -0.001089   0.001310  -0.832  0.40560
## Wards_Destroyed                    0.004820   0.011104   0.434  0.66426
## First_Blood1                       0.107938   0.049145   2.196  0.02807 *
## Player_Kills                       0.251708   0.016020  15.712  < 2e-16 ***
## Player_Death                      -0.233547   0.010526 -22.187  < 2e-16 ***
## Player_Assists                    -0.001661   0.010309  -0.161  0.87197
## Killed_Elite_Monsters_As_A_Team1   0.075174   0.076270   0.986  0.32432
## Killed_Elite_Monsters_As_A_Team2   0.328546   0.130215   2.523  0.01163 *
## First_Dragon1                      0.397091   0.081401   4.878 1.07e-06 ***
## Player_Towers_Destroyed1           0.349876   0.130358   2.684  0.00728 **
## Player_Towers_Destroyed2           2.186267   1.053487   2.075  0.03796 *
## Player_Towers_Destroyed3          11.330232 187.811300   0.060  0.95189
## Player_Towers_Destroyed4          10.640348 535.411177   0.020  0.98414
## Team_Avg_Level                     0.696114   0.112381   6.194 5.86e-10 ***
## Total_Minions_Killed               0.007752   0.001390   5.578 2.44e-08 ***
## Opponent_Wards_Placed             -0.001745   0.001287  -1.355  0.17536
## Opponent_Wards_Destroyed          -0.017279   0.011211  -1.541  0.12327
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13695  on 9878  degrees of freedom
## Residual deviance: 10816  on 9861  degrees of freedom
## AIC: 10852
##
## Number of Fisher Scoring iterations: 12
```

Through the confusion matrix, we did the following calculations: the overall misclassification rate is (1370+1413)/9879 = 28.17%, sensitivity is 3517/(1370+3517) = 71.97%, and specificity is 3579/(3579+1413) = 71.69%. Therefore, the fitted logistic regression model has both relatively high levels of sensitivity and specificity, the model performance is pretty acceptable, but there is still space to improve. In addition, the area under the curve is 0.7939 for the model.

```
          Reference
Prediction    0    1
         0 3579 1370
         1 1413 3517
```



3 Model Selection

To simplify the model, we have tried to perform stepwise selection using AIC and BIC.

3.1 Perform Stepwise Selection Using AIC

Compared with the previous logistic regression model, *Wards_Placed, Wards_Destroyed, Player_Assists,* and *Opponent_Wards_Placed* are removed.

```
glmod.aic <- stepAIC(glmod3,direction="both",k=2,trace=FALSE)
summary(glmod.aic)
```

```
##
## Call:
## glm(formula = Win ~ First_Blood + Player_Kills + Player_Death +
##     Killed_Elite_Monsters_As_A_Team + First_Dragon + Player_Towers_Destroyed +
##     Team_Avg_Level + Total_Minions_Killed + Opponent_Wards_Destroyed,
```

```
##     family = "binomial", data = lol)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7409  -0.9091  -0.1625   0.9158   2.7977
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -6.80773    0.69246  -9.831  < 2e-16 ***
## First_Blood1                      0.10773    0.04913   2.193   0.0283 *
## Player_Kills                      0.25045    0.01058  23.680  < 2e-16 ***
## Player_Death                     -0.23358    0.01045 -22.349  < 2e-16 ***
## Killed_Elite_Monsters_As_A_Team1  0.07534    0.07621   0.989   0.3229
## Killed_Elite_Monsters_As_A_Team2  0.32906    0.13011   2.529   0.0114 *
## First_Dragon1                     0.39754    0.08123   4.894 9.88e-07 ***
## Player_Towers_Destroyed1          0.34554    0.13029   2.652   0.0080 **
## Player_Towers_Destroyed2          2.19502    1.05393   2.083   0.0373 *
## Player_Towers_Destroyed3         11.32543  188.11662   0.060   0.9520
## Player_Towers_Destroyed4         10.67068  535.41118   0.020   0.9841
## Team_Avg_Level                    0.68817    0.11100   6.200 5.66e-10 ***
## Total_Minions_Killed              0.00788    0.00138   5.709 1.13e-08 ***
## Opponent_Wards_Destroyed         -0.01805    0.01104  -1.635   0.1021
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13695  on 9878  degrees of freedom
## Residual deviance: 10819  on 9865  degrees of freedom
## AIC: 10847
##
## Number of Fisher Scoring iterations: 12
```

3.2 Perform Stepwise Selection Using BIC

Compared with the AIC-selected model, *First_Blood, Killed_Elite_Monsters_As_A_Team, Player_Towers_Destroyed*, and *Opponent_Wards_Destroyed* are removed from the model. The predictors left are *Player_Kills, Player_Death, First_Dragon, Team_Avg_Level,* and *Total_Minions_Killed*, and they are all significant.

It is reasonable for the AIC model to include those predictors because the first blood would give the player more money, so he could buy equipment and improve himself earlier than others. killing more elite monsters as a team would give the team greater buff, thus helping the team to win. For player towers destroyed, the only way to win in this game is to destroy the last tower, and destroying towers would give the player a fixed amount of money to be stronger. For

the last variable, destroying the opponent's wards gives the team more chance to do a sneak attack, thus assisting them to win the game.

However, the model selected by BIC provides us a view of which variables are the most relevant with the "Win" outcome, and it is logical when analyzing the variables selected. More kills and less death are very basic and essential to win in this game as everyone should know. For the first dragon, it would give the team a great buff, and the team would get a super powerful buff when killing four dragons. Buffs definitely help the player to win.

```
n <- dim(lol)[1]
glmod.bic <- stepAIC(glmod3,direction="both",k=log(n),trace=FALSE)
summary(glmod.bic)
```

```
##
## Call:
## glm(formula = Win ~ Player_Kills + Player_Death + First_Dragon +
##     Team_Avg_Level + Total_Minions_Killed, family = "binomial",
##     data = lol)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6316  -0.9143  -0.1590   0.9147   2.7871
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -6.842422   0.685953  -9.975  < 2e-16 ***
## Player_Kills         0.261789   0.010170  25.740  < 2e-16 ***
## Player_Death        -0.240831   0.010197 -23.617  < 2e-16 ***
## First_Dragon1        0.505479   0.049171  10.280  < 2e-16 ***


## Team_Avg_Level       0.684118   0.110164   6.210 5.30e-10 ***
## Total_Minions_Killed 0.008212   0.001371   5.989 2.11e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13695  on 9878  degrees of freedom
## Residual deviance: 10854  on 9873  degrees of freedom
## AIC: 10866
##
## Number of Fisher Scoring iterations: 4
```

3.3 Training Misclassification Rates & Training AUC values

Since the model selected by AIC is more complicated and bigger, fitting the training data better, it has smaller training error and also larger training area under the curve. Choosing between these two models, since our goal is to identify how to increase the probability of winning the game, we want to know the most relevant variables which players could focus more on later, so we further investigated the BIC model.

```
train.err.aic

## [1] 0.2819111
train.err.bic

## [1] 0.2856564
train.auc.aic

## Area under the curve: 0.7937
train.auc.bic

## Area under the curve: 0.7921
```

3.4 Split Data into Training Data + Test Data

```
set.seed(1)
train <- sample(1:nrow(lol), size = 2500)
lol.train <- lol[train,]
lol.test <- lol[-train,]
```

We splitted the data into training and test data, 2500 and 7379 observations respectively, and computed the test error. The test error is greater than the training error as expected.

```
c(test.err,train.err.bic)
```

```
## [1] 0.3054614 0.2856564
```

```
c(test.auc,train.auc.bic)
```

```
## [1] 0.7908401 0.7921491
```

3.5 Cross Validation

The cross validated test error is also around 30%. There is still space to improve.

```
c(mean(test.err),train.err.bic)
```

```
## [1] 0.3021846 0.2856564
```

```
c(mean(test.auc),train.auc.bic)
```

```
## [1] 0.7910619 0.7921491
```

4.0 Principal Component Analysis

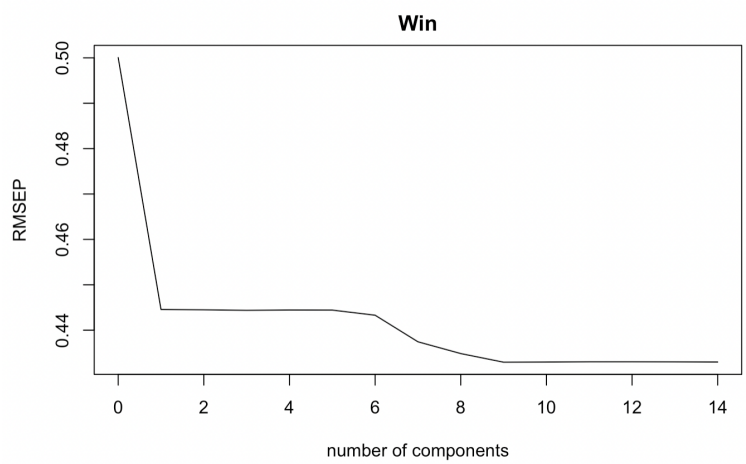
4.1 Standarlization

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7    PC8     PC9    PC10    PC11      PC12      PC13
Standard deviation     2.2048 1.5881 1.10380 1.03139 0.95073 0.90460 0.82427 0.7485 0.46045 0.29160 0.27527 1.379e-14 1.551e-15
Proportion of Variance 0.3739 0.1940 0.09372 0.08183 0.06953 0.06295 0.05226 0.0431 0.01631 0.00654 0.00583 0.000e+00 0.000e+00
Cumulative Proportion  0.3739 0.5679 0.66165 0.74348 0.81301 0.87596 0.92822 0.9713 0.98763 0.99417 1.00000 1.000e+00 1.000e+00
                                PC1         PC2          PC3          PC4         PC5         PC6          PC7          PC8
Wards_Placed             -0.009895538  0.02605849  0.377911998 -0.628201284  0.62476350  0.16221134 -0.195126783  0.081569506
Wards_Destroyed          -0.050042385 -0.06389289  0.605199987  0.292102776 -0.24126299  0.67340354 -0.052418856 -0.162120754
Player_Kills             -0.339439549  0.38311565  0.021359626  0.015655568 -0.04236110 -0.02496702 -0.003394061  0.053559613
Player_Death              0.185338556  0.33097226  0.067940135  0.029534467 -0.12916448 -0.23282835 -0.739127609 -0.450383771
Player_Assists           -0.286924760  0.38518007  0.069618272  0.018270690 -0.07557774  0.05313084  0.042721568  0.395347805
Player_Total_Gold        -0.416277583  0.20970342  0.011331812  0.012737045 -0.03746800 -0.05026395 -0.100040927  0.067546142
Team_Avg_Level           -0.371600171 -0.14659430 -0.036189525 -0.050640160  0.11927337 -0.04832546  0.194503196 -0.557039710
Team_Total_Experience    -0.393875569 -0.16024805 -0.039153651 -0.047505740  0.09670617 -0.04067038  0.149276970 -0.424256082
Total_Minions_Killed     -0.249090657 -0.47413161 -0.007835084  0.009163476 -0.09650658 -0.06592845 -0.363175635  0.222505108
CS_Per_Min               -0.249090657 -0.47413161 -0.007835084  0.009163476 -0.09650658 -0.06592845 -0.363175635  0.222505108
Gold_Per_Min             -0.416277583  0.20970342  0.011331812  0.012737045 -0.03746800 -0.05026395 -0.100040927  0.067546142
Opponent_Wards_Placed     0.007296640 -0.03572623  0.392639387  0.629911198  0.51016927 -0.42292866  0.043119172  0.081248690
Opponent_Wards_Destroyed  0.039473770 -0.06750134  0.568866172 -0.341300710 -0.47115515 -0.51570897  0.255436292  0.004404044
```

The principal component analysis is one of the powerful non-linear methodologies that would potentially contribute to model optimization. There are 13 principal components in the

output after standardization. We can conclude that 8 principal components would explain 97.13% of cumulative variances among all numeric variables. Since our predicted variable is binary or logistic, we decide to create new independent variables by using principal components to experiment whether that would potentially improve our model; in other words, decrease the AIC or test error.

4.2 PCs Selection



The plot above indicates we require nine principal components for the minimum cross validation error.

4.3 Test Error Comparison

1	Player_Kills
2	Player_Death
3	First_Dragon
4	Team_Avg_Level
5	Total_Minions_Killed
6	PC1
7	PC2

							test.auc	test.err	train.auc.bic	train.err.bic
1	2	3	4	5	6	7	0.797833	0.300989	0.792149	0.285656
1	2	3	4	5	6	7	0.79084	0.305461		
1	2	3	4	5	6	7	0.796908	0.303564		
1	2	3	4	5	6	7	0.793796	0.307494		
1	2	3	4	5	6	7	0.78285	0.318607		
1	2	3	4	5	6	7	0.79637	0.301938		
1	2	3	4	5	6	7	0.797969	0.30126		
1	2	3	4	5	6	7	0.785824	0.316032		
1	2	3	4	5	6	7	0.755368	0.336767		

```
                     Pr(>|z|)
(Intercept)           < 2e-16 ***
Player_Kills          < 2e-16 ***
Player_Death          < 2e-16 ***
First_Dragon1         < 2e-16 ***
Team_Avg_Level       5.30e-10 ***
Total_Minions_Killed 2.11e-09 ***
```

We have introduced the best independent variables to conduct a logistic regression with minimum test error from our previous step. Then, combining principal components one and two as new independent variables with them and fining the lowest test error. The lowest test error is from combining two principal components with all five significant independent variables, which is 30.0989%. We also tried to combine PC3 to PC8 to the model, but the test error runs a diminishing return. Thus, we think maybe a decision tree would be a more powerful non-linear methodology helping us further improve our model.

5 Tree Models

In this part of the project, we would like to find the tree model with the least error rate and largest area under the curve number. To reach this goal, we fit the League of Legend training data set to four different tree models: single tree model, pruned tree model, bagging tree model, and random forest tree model. The three models are as follows.

5.1 Single Tree Model

The single tree model for the legal of legend training data set uses two variables: Player total gold and player death. The model contains 6 terminal nodes, and the misclassification error rate is 29.88%.

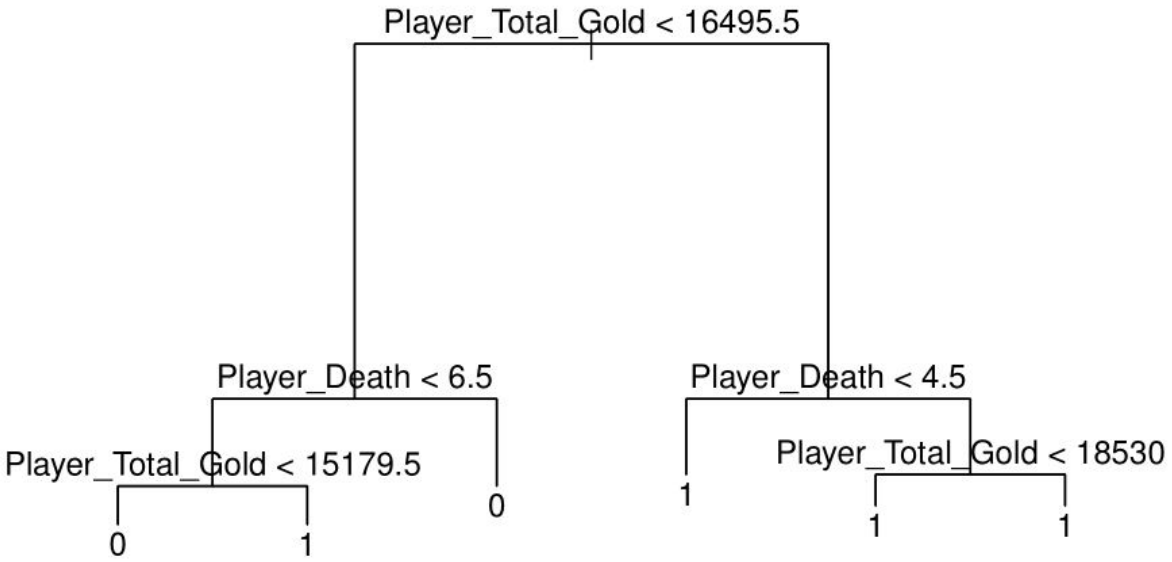


Figure 5.1.1 The single tree model

The predicted test error rate for the single tree model is 31.98%

pred.test	0	1
0	1878	579
1	1781	3141

Figure 5.1.2 Single tree confusion matrix

5.2 Pruned Tree Model

According to the cross-validated error for the legal of legend training data set, the best tree size would be 6.

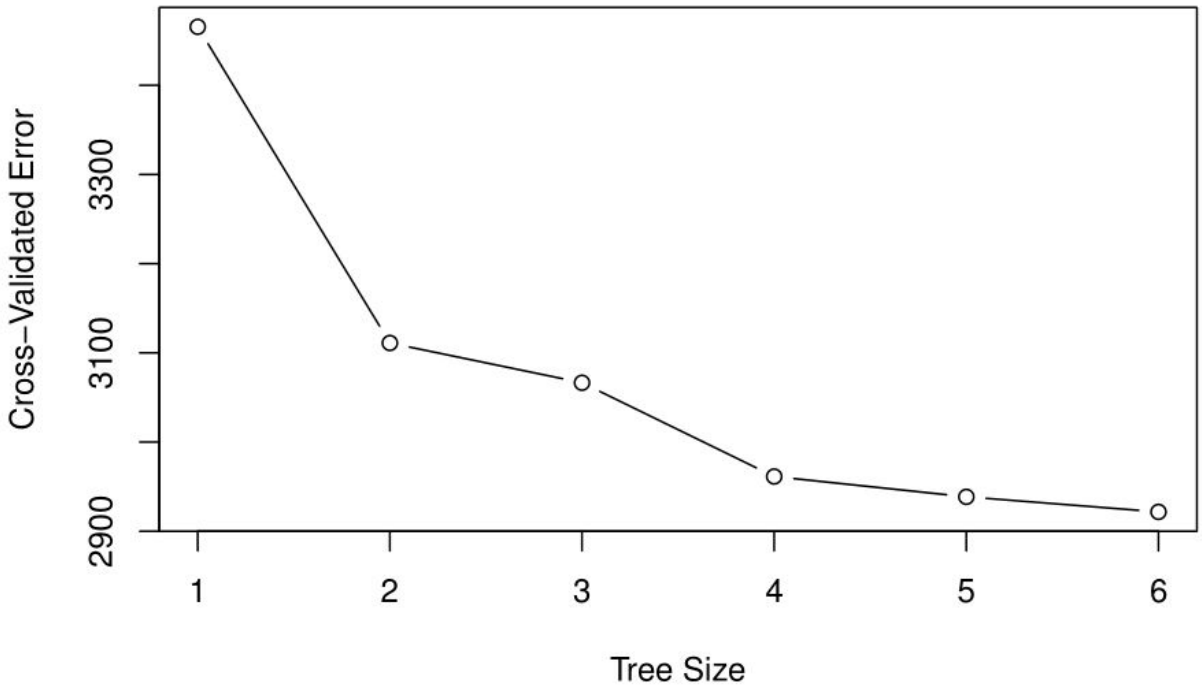


Figure 5.2.1 Cross-Validation Error to Tree Size

By using a tree size equal to 6 as the turning value, there is no difference between the pruned and single tree models. The pruned tree model also used Player total gold and player death variables. The model contains 6 terminal nodes, and the misclassification error rate is 29.88%. The test error rate for the pruned tree is 31.98%.

5.3 Bagging tree

To find the bagging tree model for the data, we used the randomForest function in the randomForest package. For the bagging tree, we set the turning value to 18. The bagging tree model is as follows.

```
##
## Call:
##  randomForest(formula = Win ~ ., data = lol.train, mtry = 18,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 18
##
##         OOB estimate of  error rate: 30.8%
## Confusion matrix:
##     0   1 class.error
## 0 906 384   0.2976744
## 1 386 824   0.3190083
```

Figure 5.3.1 Bagging tree model

The training error rate for the bagging tree model is 30.8%, and according to the bagging tree confusion matrix, its test error rate is 30.87%.

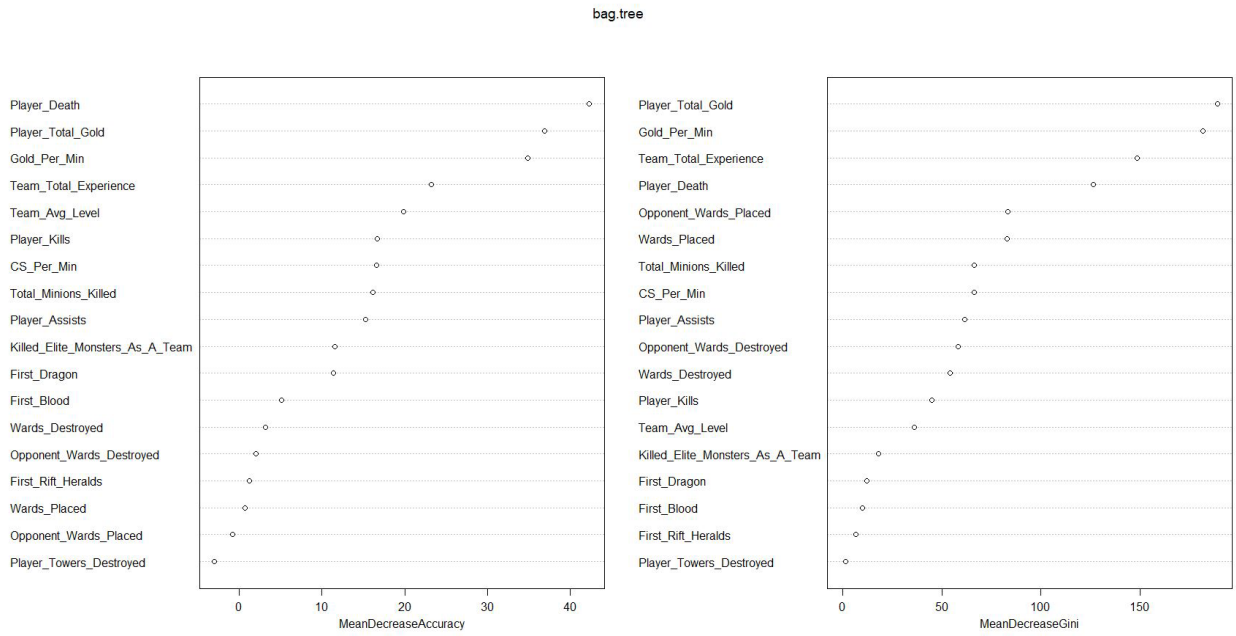


Figure 5.3.2 Important Variables for Bagging Tree Model

To find the important variables for the bagging tree model, we used %IncMSE and %IncNodePurity index. By comparing the two indexes, the important variables for the

bagging tree model are Player Death, Player Total Gold, Gold per Min, and Team Total Experience.

5.4 Random Forest Tree Model

To find the random forest tree model for the data, we used the randomForest function in the randomForest package. For the random forest tree, we set the turning value to 4 and 5 since the square root of the total number of variables (18) is around 4 and 5. The random forest tree model is as follows.

```
##
## Call:
##  randomForest(formula = Win ~ ., data = lol.train, mtry = 4, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 29.68%
## Confusion matrix:
##     0   1 class.error
## 0 905 385   0.2984496
## 1 357 853   0.2950413
```

Figure 5.4.1 Random Forest Tree Model Turning Value = 4

```
##
## Call:
##  randomForest(formula = Win ~ ., data = lol.train, mtry = 5, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 29.36%
## Confusion matrix:
##     0   1 class.error
## 0 914 376   0.2914729
## 1 358 852   0.2958678
```

Figure 5.4.2 Random Forest Tree Model Turning Value = 5

By comparing the training test error for the two random forest tree models, we choose to stick with the random forest tree model with a turning value equal to 5. The random forest tree model's training error rate is 29.36%, and according to its confusion matrix, its test error rate is 29.99%

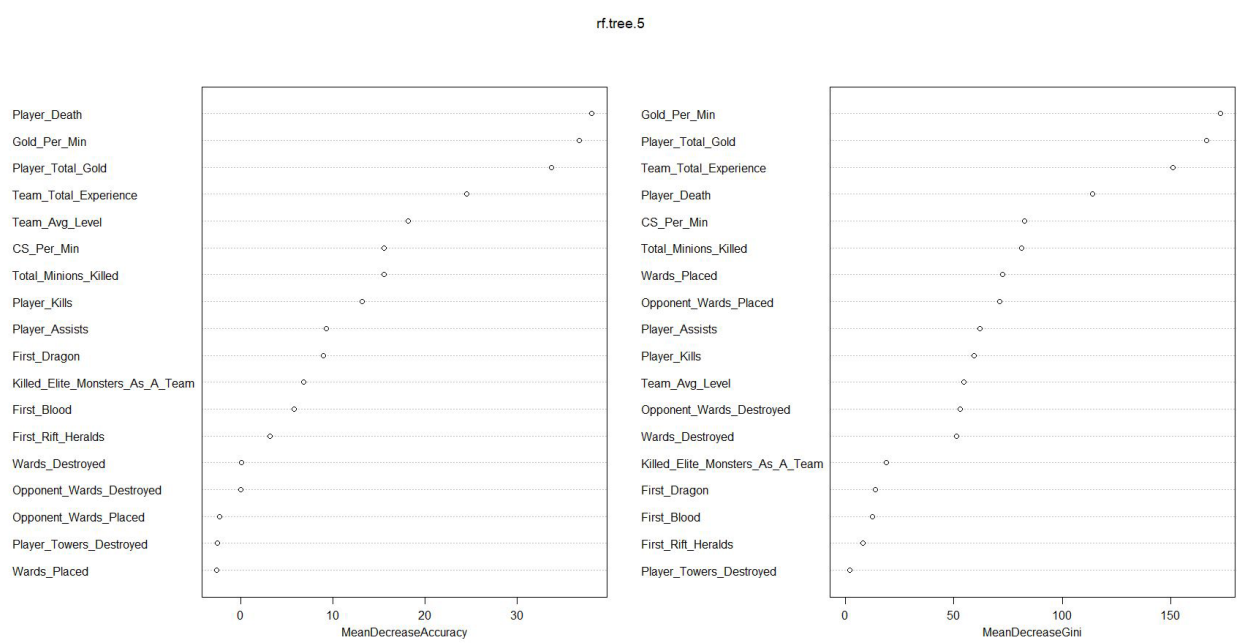


Figure 5.4.3 Important Variables for Random Forest Tree Model Turning Value = 5

By comparing the two indexes, the important variables for the random forest tree model are Player Death, Player Total Gold, Gold per Min, and Team Total Experience. The bagging tree model and the random forest tree model share the same important variables.

5.5 Final Tree Model

To find the best tree model, we compared the training error and area under the curve value for each tree model. By comparing the training error for each tree model, the best tree model is the random forest tree model. We compared the roc curve for every tree model to test this result. The random forest tree model is better than other tree models in every aspect. The ROC curve is as follows. Blue: random forest tree model (AUC = 77.38%); Red: bagging tree model (AUC = 76.5%); Black: single tree model (AUC = 74.57%)

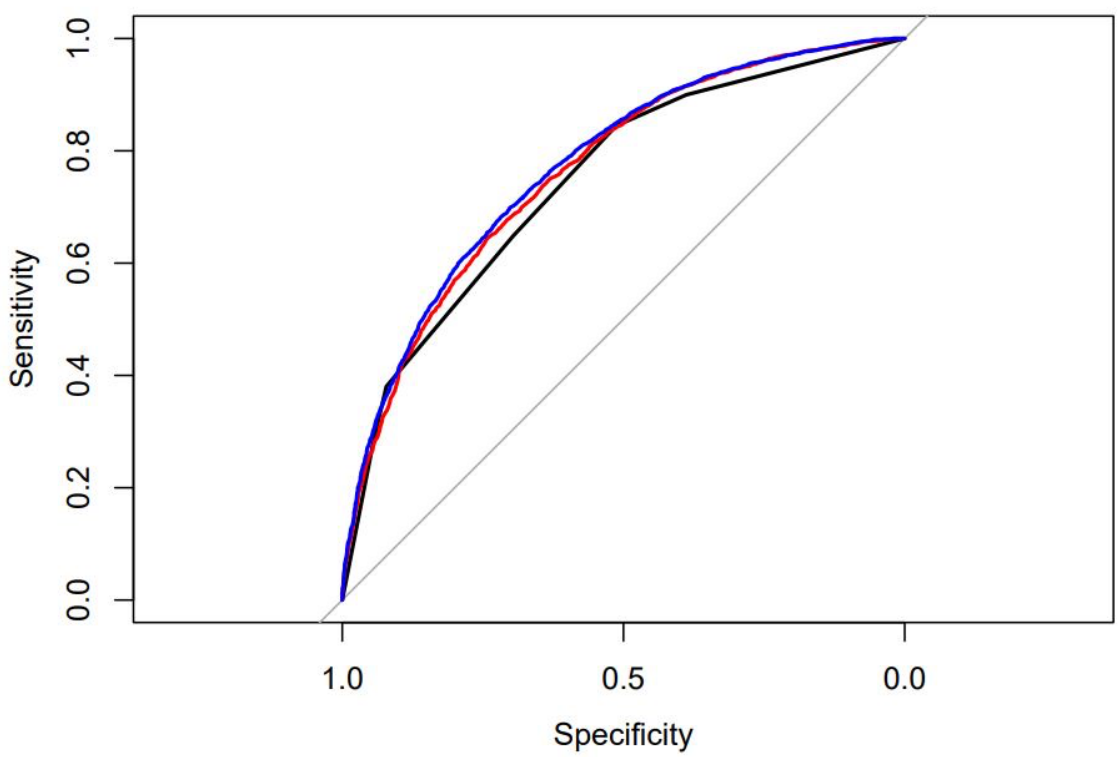


Figure 5.5.1 ROC Curve for Tree Models

According to the training error rate and the roc curve, the final tree model for the legal of legend data set is the random forest tree model. The random forest tree model's test error rate is 29.99%, and its AUC is 77.38%.

6 Summary

Logistic Model

Test error: 30.13%

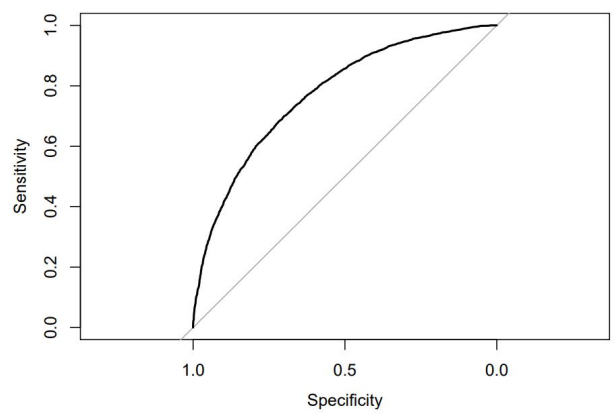
AUC: 79.80%

1	Player_Kills
2	Player_Death
3	First_Dragon
4	Team_Avg_Level
5	Total_Minions_Killed
6	PC1
7	PC2

Tree Model (Randomforest mtry = 5)

Test error: 29.99%

AUC: 77.38%



Although the AUC in logistic model is higher than tree model, but the test error in tree model is lower. So finally we choose the tree model as our final model.

According to the random forest tree model. There are four attribute that are crucial to the result of the game. Those are player death and player total gold and gold per min and team total experience.

In short, when next time we play league of legend. We should try to keep ourself alive and have more total gold during the match,

References

Buteikis, Andrius. "Practical Econometrics and Data Science." *4.5 Multicollinearity*, 13 Oct. 2020, web.vu.lt/mif/a.buteikis/wp-content/uploads/PE_Book/4-5-Multiple-collinearity.html.

Contribution

Tim Yu: 25%

Jenny Tu: 25%

Jiahao Ye: 25%

Jiahao Chang: 25%

Appendix

Initial Data Analysis Code

```
## 3.3 Boxplot
name <- names(LOL)
for(i in c(1:3)){
  par(mfrow=c(3,3))
}


##3.4 Histogram
ggplot(lol, aes(x = Player_Kills, fill = Win)) +
  geom_bar()+
  geom_vline(aes(xintercept=mean(Player_Kills)), color="purple", linetype="dashed", size=1)+
  ggtitle("Figure 6.03 Player Kills Histogram") +
  xlab("Number of Player Kills") + ylab("Frequency")+
  theme(
plot.title = element_text(color="#658531", size=16, face="bold.italic"),
axis.title.x = element_text(color="#993333", size=14, face="bold"),
axis.title.y = element_text(color="#993333", size=14, face="bold")
)





## 3.5Barplot
for(i in c(2:20)){
 plot(LOL[,i]~LOL$Win,xlab=name[i],main=name[i] )
  }
##Create contingency table
my_table <- table(LOL$Win,LOL$First_Blood)

name <- names(LOL)
par(mfrow=c(2,2))
for(i in c(4,9,10)){


  my_table <- table(LOL$Win,LOL[,i])
  prob_table <- prop.table(my_table,margin = 2)
  barplot(prob_table,beside=TRUE, xlab=name[i],ylab="number of win",col
=c("darkgreen",col="orange"))
  if(i==7){
```

```
  legend  (x=2.1,y=1,legend=c("Not left","Left"), fill = c("darkgreen",col="orange"),bty='n')
  }

}
```

Model Code

Full Logistic Regression Model

```
## full model
lol <- read.csv("high_diamond_ranked.csv", header = T)
lol$Win <- as.factor(lol$Win)
lol$First_Blood <- as.factor(lol$First_Blood)
lol$First_Dragon <- as.factor(lol$First_Dragon)
lol$First_Rift_Heralds <- as.factor(lol$First_Rift_Heralds)
lol$Killed_Elite_Monsters_As_A_Team <- as.factor(lol$Killed_Elite_Monsters_As_A_Team)
lol$Player_Towers_Destroyed <- as.factor(lol$Player_Towers_Destroyed)
glmod1 <- glm(Win~.,data=lol,family="binomial")
summary(glmod1)

## fix singularities
#check correlation
library(corrplot)
LOL <- read.csv("high_diamond_ranked.csv")
M <-cor(LOL)
round(cor(LOL), digits = 2)
#corrplot(M, main="Figure 4.1:Correlation between Attributes",type="upper", order="hclust")

glmod2 <- glm(Win~.-CS_Per_Min-Gold_Per_Min-
First_Rift_Heralds,data=lol,family="binomial")
summary(glmod2)

## Check Multicollinearity
library(car)
vif(glmod2)
vif(glmod2)[,3]^2
```

```
glmod3 <- glm(Win~.-CS_Per_Min-Gold_Per_Min-First_Rift_Heralds-Player_Total_Gold-
Team_Total_Experience,data=lol,family="binomial")
summary(glmod3)
vif(glmod3)
vif(glmod3)[,3]^2

glmod4 <- glm(Win~.-CS_Per_Min-Gold_Per_Min-First_Rift_Heralds-
Player_Total_Gold,data=lol,family="binomial")
summary(glmod4)
vif(glmod4)
vif(glmod4)[,3]^2

## identify outliers
n <- nrow(lol)
plot(glmod3, which=5)
#plot(glmod3, which = 4, id.n = 3)
#abline(h = 4/n, lty = 2, col = "steelblue")
cooks <- cooks.distance(glmod3)
plot(cooks, main = "Cook's Distance for Influential Obs")
abline(h = 4/n, lty = 2, col = "red")

#identify influential points
influential_obs <- as.numeric(names(cooks)[(cooks > (4/n))])
lol$cooks <- cooks
lol_removed <- subset(lol, cooks < (4/n))

## identify non-linear relationship
probabilities <- predict(glmod3, type = "response")
predicted.classes <- ifelse(probabilities > 0.5,1,0)
library(tidyverse)
library(broom)
theme_set(theme_classic())
# Select only numeric predictors
mydata <- lol %>%
  dplyr::select_if(is.numeric)
predictors <- colnames(mydata)
# Bind the logit and tidying the data for plot
mydata <- mydata %>%
  mutate(logit = log(probabilities/(1-probabilities))) %>%
  gather(key = "predictors", value = "predictor.value", -logit)
# Create the scatter plots
ggplot(mydata, aes(logit, predictor.value))+
  geom_point(size = 0.5, alpha = 0.5) +
  geom_smooth(method = "loess") +
  theme_bw() +
  facet_wrap(~predictors, scales = "free_y")
```

```
## Transformation
lol$logOpponent_Wards_Placed <- log(lol$Opponent_Wards_Placed)
lol$logOpponent_Wards_Destroyed <- log(lol$Opponent_Wards_Destroyed+1)
lol$logWards_Placed <- log(lol$Wards_Placed)
lol$logWards_Destroyed <- log(lol$Wards_Destroyed+1)
lol$logPlayer_Assists <- log(lol$Player_Assists+1)

glmod5 <- glm(Win~.-Opponent_Wards_Placed-Opponent_Wards_Destroyed-Wards_Placed-
Wards_Destroyed-Player_Assists-CS_Per_Min-Gold_Per_Min-
First_Rift_Heralds,data=lol,family="binomial")
summary(glmod5)

## Confusion Matrix
est.prob <- predict(glmod3,type = "response")
pred.win <- as.factor(ifelse(est.prob>0.5,1,0))
#create the confusion matrix
library(caret)
c.matrix <- confusionMatrix(lol$Win,pred.win)
c.matrix$table

library(pROC)
###Create the ROC curve
est.prob <- predict(glmod3,type = "response")
r1<- roc(lol$Win ~ est.prob)
plot(r1,main = "ROC Curve")
###Compute AUC values for two models
auc(r1)
```

Model Selection

```
# Perform stepwise selection using AIC
library(MASS)
glmod.aic <- stepAIC(glmod3,direction="both",k=2,trace=FALSE)
summary(glmod.aic)

#Stepwise selection using BIC
n <- dim(lol)[1]
glmod.bic <- stepAIC(glmod3,direction="both",k=log(n),trace=FALSE)
summary(glmod.bic)

# Calculate the training miss-classification rates (using 0.5 as cutoff) and the traning AUC value
of models selected by AIC and BIC
## Find estimated probabilities for all training data
train.prob.aic <- predict(glmod.aic,type="response")
```

```
train.prob.bic <- predict(glmod.bic,type="response")

## find the predicted classes
train.pred.aic <- ifelse(train.prob.aic>0.5,"1","0")
train.pred.bic <- ifelse(train.prob.bic>0.5,"1","0")

### Find the miss-classification rates
train.err.aic <- mean(train.pred.aic!=lol$Win)
train.err.bic <- mean(train.pred.bic!=lol$Win)
train.err.aic
train.err.bic

library(pROC)
###Compute AUC values for two models
r.aic <- roc(lol$Win,train.prob.aic)
train.auc.aic <- auc(r.aic)
r.bic <- roc(lol$Win,train.prob.bic)
train.auc.bic <- auc(r.bic)
train.auc.aic
train.auc.bic

# Split the data into training data + test data. Use the training data to fit the model, and use the
test data to re-calculate the misclassification rates and auc values.
set.seed(1)
train <- sample(1:nrow(lol), size = 2500)
lol.train <- lol[train,]
lol.test <- lol[-train,]

###Fit the model using training data
glmod.bic <- glm(Win ~ Player_Kills + Player_Death + First_Dragon +
   Team_Avg_Level + Total_Minions_Killed,data=lol.train,family="binomial")

## Find estimated probablities for all test data
test.prob <- predict(glmod.bic,newdata=lol.test)

## find the predicted classes
test.pred <- ifelse(test.prob>0.5,"1","0")

### Find the miss-classification rates
test.err <- mean(test.pred!=lol.test$Win)

library(pROC)
###Compute AUC values for two models
r <- roc(lol.test$Win,test.prob)
test.auc <- auc(r)
c(test.err,train.err.bic)
```

```
c(test.auc,train.auc.bic)

# Cross-validation
##create vectors to store results
test.err <- test.auc <- numeric()
for (i in 1:100) {
set.seed(i)
train <- sample(1:nrow(lol), size = 2500)
lol.train <- lol[train,]
lol.test <- lol[-train,]
###Fit the model using training data
glmod.bic <- glm(Win ~ Player_Kills + Player_Death + First_Dragon +
   Team_Avg_Level + Total_Minions_Killed,data=lol.train,family="binomial")
## Find estimated probablities for all test data
test.prob <- predict(glmod.bic,newdata=lol.test)
## find the predicted classes
test.pred <- ifelse(test.prob>0.5,"1","0")
### Find the miss-classification rates
test.err[i] <- mean(test.pred!=lol.test$Win)

library(pROC)
###Compute AUC values for two models
r <- roc(lol.test$Win,test.prob,quiet = T)
test.auc[i] <- auc(r)
}
c(mean(test.err),train.err.bic)
c(mean(test.auc),train.auc.bic)
```

Tree Code

```
# Single tree

tree.train <- tree(Win~., data = lol.train)
summary(tree.train)
plot(tree.train)
text(tree.train)
pred.test <- predict(tree.train, newdata = lol.test, type = "class")
table(pred.test, lol.test$Win)
test.err.tree <- mean(pred.test != lol.test$Win)
Test.err.tree


# Pruned Tree

set.seed(100)
```

```
cv.train <- cv.tree(tree.train)
cv.train
plot(cv.train$size, cv.train$dev, type = "b", xlab = "Tree Size", ylab = "Cross-Validated Error")
cv.train$size[which.min(cv.train$dev)]
tree.train.prune <- prune.tree(tree.train, best = 6)
plot(tree.train.prune)
text(tree.train.prune)
summary(tree.train.prune)


# Bagging Tree

library(randomForest)
set.seed(1)
bag.tree <- randomForest(Win~., data = lol.train, mtry = 18, importance = TRUE)
bag.tree
pred.test.bag <- predict(bag.tree, newdata = lol.test, type = "class")
test.err.bag <- mean(pred.test.bag != lol.test$Win)
test.err.bag
varImpPlot(bag.tree)


# Random Forest Tree

rf.tree.5 <- randomForest(Win~., data = lol.train, mtry = 5, importance = TRUE)
rf.tree.5
pred.test.rf.5 <- predict(rf.tree.5, newdata = lol.test, type = "class")
test.err.rf.5 <- mean(pred.test.rf.5 != lol.test$Win)
test.err.rf.5
varImpPlot(rf.tree.5)


# Comparison ROC Curve

library(pROC)
prob.tree.test <- predict(tree.train, newdata = lol.test, type = "vector")
tree.roc <- roc(lol.test$Win, prob.tree.test[,2])
tree.auc <- auc(tree.roc)
prob.bag.test <- predict(bag.tree, newdata = lol.test, type = "prob")
bag.roc <- roc(lol.test$Win, prob.bag.test[,2])
bag.auc <- auc(bag.roc)
prob.rf.test <- predict(rf.tree.5, newdata = lol.test, type = "prob")
rf.roc <- roc(lol.test$Win, prob.rf.test[,2])
rf.auc <- auc(rf.roc)
plot(tree.roc, type = "l", lwd = 2)
lines(bag.roc,col = "red")
```

```
lines(rf.roc, col = "blue")
plot(rf.roc,type = "l", lwd = 2)
c(tree.auc,bag.auc,rf.auc)
c(test.err.tree,test.err.bag,test.err.rf.5)
```