ML Project: Predicting *Dota 2* Game Outcome

Richard Zhou 4627865

2/11/2020

```r
library(tidyverse)
library(dplyr)
library(caret)
library(ggplot2)
library(ROCR)
library(e1071)
library(liquidSVM)
library(rpart.plot)
library(randomForest)
library(gbm)
library(knitr)
library(kableExtra)
library(ggpubr)
```

Introduction

Dota 2 is a MOBA (Multiple Online Battle Arena) game developed by Valve which is a sequel to the popular game *Defense Of the Ancients (DotA)*. There are two teams in the game- Radiant and Dire where each team contains 5 players. For each game every player will pick a hero. The map with three lanes is separated diagonally into two parts by a river. The goal of the game is to defend the ancient and take down enemy's ancient.

DIRE
RIVER
RADIANT
Fountain
Tower
Rune Spot
Side Shop
Ancient
Barracks
Roshan
Secret Shop

Dota 2 is a famous video game with a large esports scene in the world. *Dota 2* has an annual esports world championship tournament with over $34 million US dollars prize pool in the recent one. The ability to predict the out come of a specific game is extremely crucial not only in the sports tournament but also in regular game pubs. Knowing the winning factor of game will help player to analyze the game and strategy.

Data

Data overview

The data set comes from a public *kaggle competition*. This competition is organized by *mlcourse.ai* in collaboration with *GOSU.AI* which is a company specialized using artificial intelligence to help player improve skills and strategies. The publisher already provides `train_features.csv` and `test_features.csv` sets as training and test data sets. However since this is a kaggle competition, the true result of the `test_features.csv` set is not released at this point. Therefore I will create my own test sets from the `train_features.csv` set. The `train_targes.csv` data set is the results for games from the training set, *i.e.* whether team Radiant eventually won. I will combine two data sets for easier data manipulation.

```
#Load the Data
features <- read.csv('train_features.csv')
targets <- read.csv('train_targets.csv')
```

The descriptive variable we are interested in is `radiant_win` which indicates `True` or `False`. Now we create our own dataframe by combining data sets.

```
#Combine two data sets
win <- targets %>% dplyr::select(match_id_hash, radiant_win)
df <- left_join(features, win, by = 'match_id_hash')
```

```r
#Rearrange
df <- df %>%
  dplyr::select(match_id_hash, radiant_win, everything())
head(df)[1:10]
```

```
##                      match_id_hash radiant_win game_time game_mode
## 1 a400b8f29dece5f4d266f49f1ae2e98a       False       155        22
## 2 b9c57c450ce74a2af79c9ce96fac144d        True       658         4
## 3 6db558535151ea18ca70a6892197db41        True        21        23
## 4 46a0ddce8f7ed2a8d9bd5edcbb925682        True       576        22
## 5 b1b35ff97723d9b7ade1c9c3cf48f770       False       453        22
## 6 19c39fe2af2b547e48708ca005c6ae74       False       160        22
##   lobby_type objectives_len chat_len r1_hero_id r1_kills r1_deaths
## 1          7              1       11         11        0         0
## 2          0              3       10         15        7         2
## 3          0              0        0        101        0         0
## 4          7              1        4         14        1         0
## 5          7              1        3         42        0         1
## 6          7              0        0         57        0         0
```

Now check the dimension and existence of missing values. The data set has 39675 rows and 247 columns. The data sets have 437 missing values. I will drop missing values first because it will interfere with my logistic regression model.

```r
#check dimension
dim(df)
```

```
## [1] 39675   247
```

```r
#check missing value
table(is.na(df))
```

```
## 
##   FALSE    TRUE 
## 9799288     437
```

```r
df = drop_na(df)
```

Before splitting my dataset into training and test sets, I will first do some exploratory analysis. Lets first see the distribution of our response variable `radiant_win`. Surprisingly, Radiant have a slightly higher win rate at 52.5% than Dire at 47.5%.

```r
rad_win_rate <- 100*table(df$radiant_win)['True']/length(df$radiant_win)
dire_win_rate <- 100*table(df$radiant_win)['False']/length(df$radiant_win)
rad_win_rate
```

```
##     True 
## 52.49248
```

```r
dire_win_rate
```

```
##    False 
## 47.50752
```

Here is the short list of definitions of explanatory variables that I am going to use in my analysis:

- `towers_killed`: number of enemy team's towers killed
- `K/D/A`: kills, deaths, assists
- `lh`: last hits, number of enemy creeps killed
- `denies`: number of friendly creeps killed to deny enemy's gold and experience

- `gold`: gold player earned
- `xp`: player's experience
- `level`: player's level
- `stuns`: total duration of stun, which immobilize enemy players
- `firstblood`: first player to complete a kill, which has better gold rewards
- `obs_placed`, `sen_placed`: the number of observation and sentry wards placed by a player

Data Preparation

Since there are total 246 variables in the data set, I will select and combine some columns to reduce the number of predictors. For example, each row has data for 10 players in the game. I will take sum of the same stat as a team to reduce size. First I will separate data sets into two subsets: `rad` and `dire`, which stands for two teams. Then I will sum up each player's stats to team stats. Then I will drop some variables irrelevant to the analysis and combine data sets.

```
#Radiant data set
rad <- df[8:127 ]
names(rad) = paste('r', substring(names(rad), 4), sep='_')
rad_sum = as.data.frame(sapply(split.default(rad, colnames(rad)), rowSums))
#Dire data set
dire = df[128:247]
names(dire) = paste('d', substring(names(dire), 4), sep='_')
dire_sum = as.data.frame(sapply(split.default(dire, colnames(dire)), rowSums))
#Combine data set
df = cbind(df[1:7],rad_sum,dire_sum)
#Drop irrelevant variables
df = df %>%
  dplyr::select(-c(match_id_hash,game_mode, lobby_type,objectives_len,
            chat_len,r_health,r_hero_id,
            r_max_health,r_max_mana,r_x,r_y,d_health,d_hero_id,
            d_max_health,d_max_mana,d_x,d_y))

df$radiant_win = as.factor(ifelse(df$radiant_win == 'True', 'Yes','No'))
```

Split training and test sets.

```
set.seed(1)
train_samples <- df$radiant_win%>%
  createDataPartition(p = 0.6, list=FALSE)
train <- df[train_samples, ]
test <- df[-train_samples, ]
```

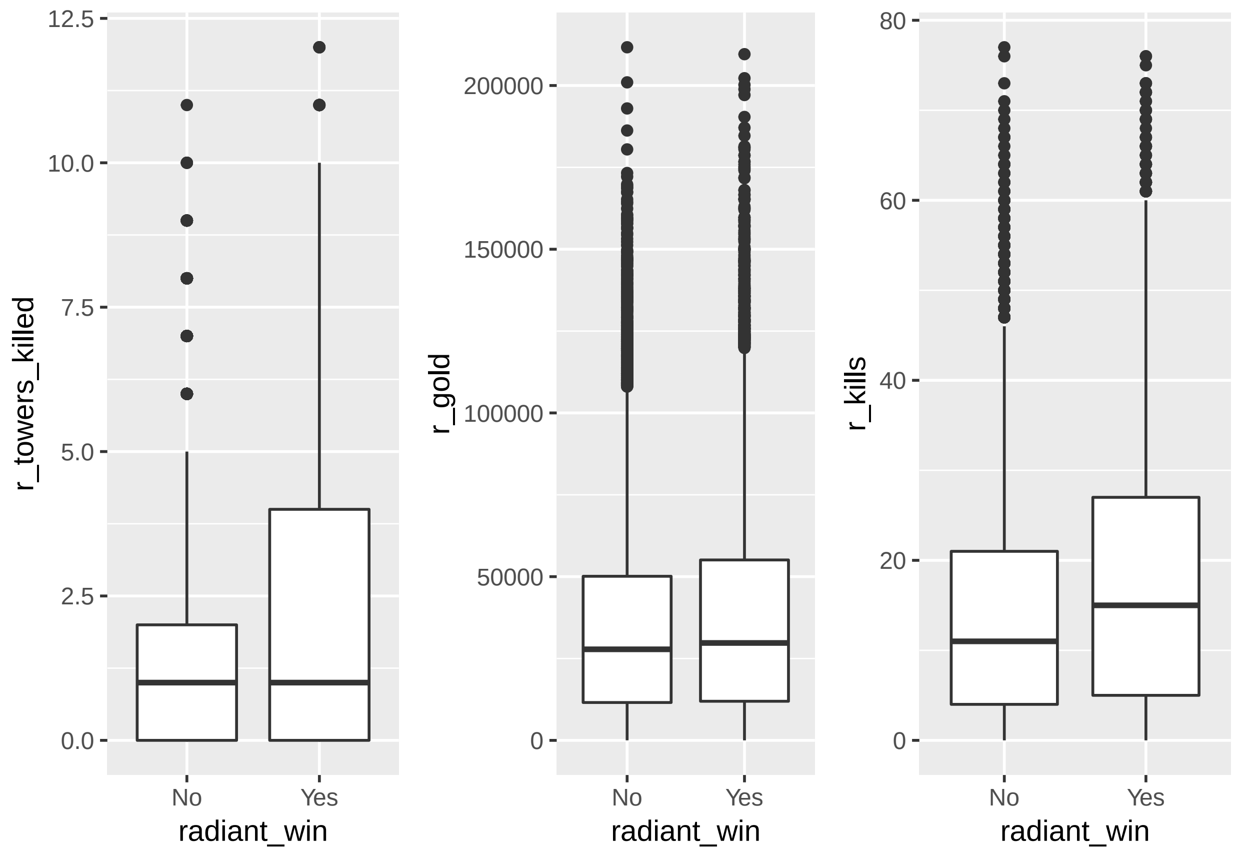
Check summary of data types

```
table(sapply(df, class))
```

```
##
##  factor integer numeric
##       1       1      36
```

Let's do some preliminary exploratory analysis plotting. I pick following three variables as I believe these are the important winning factors in *Dota* game.

```
p1= ggplot(df,aes(y=r_towers_killed,x=radiant_win))+geom_boxplot()
p2=ggplot(df,aes(y=r_gold,x=radiant_win))+geom_boxplot()
p3= ggplot(df,aes(y=r_kills,x=radiant_win))+geom_boxplot()
ggarrange(p1,p2,p3,ncol=3,nrow=1)
```



From the boxplots of `gold` and `kills` versus `radiant_win`, we can tell there are significant amount of outliers. We should be aware of these in future analysis.

Apply Machine Learning Models

Logistic Regression

First we construct a simple logistic regression model using default threshold 0.5 to get a preliminary confusion matrix and test error.

```
set.seed(2)
logit_fit = glm(radiant_win~., data=test, family="binomial")
logit_pred = predict(logit_fit, test, type="response")
predWin = as.factor(ifelse(logit_pred <= 0.5, 'No','Yes'))
#Confusion Matrix
logit_cm = table(predWin, true=test$radiant_win)
logit_cm
```

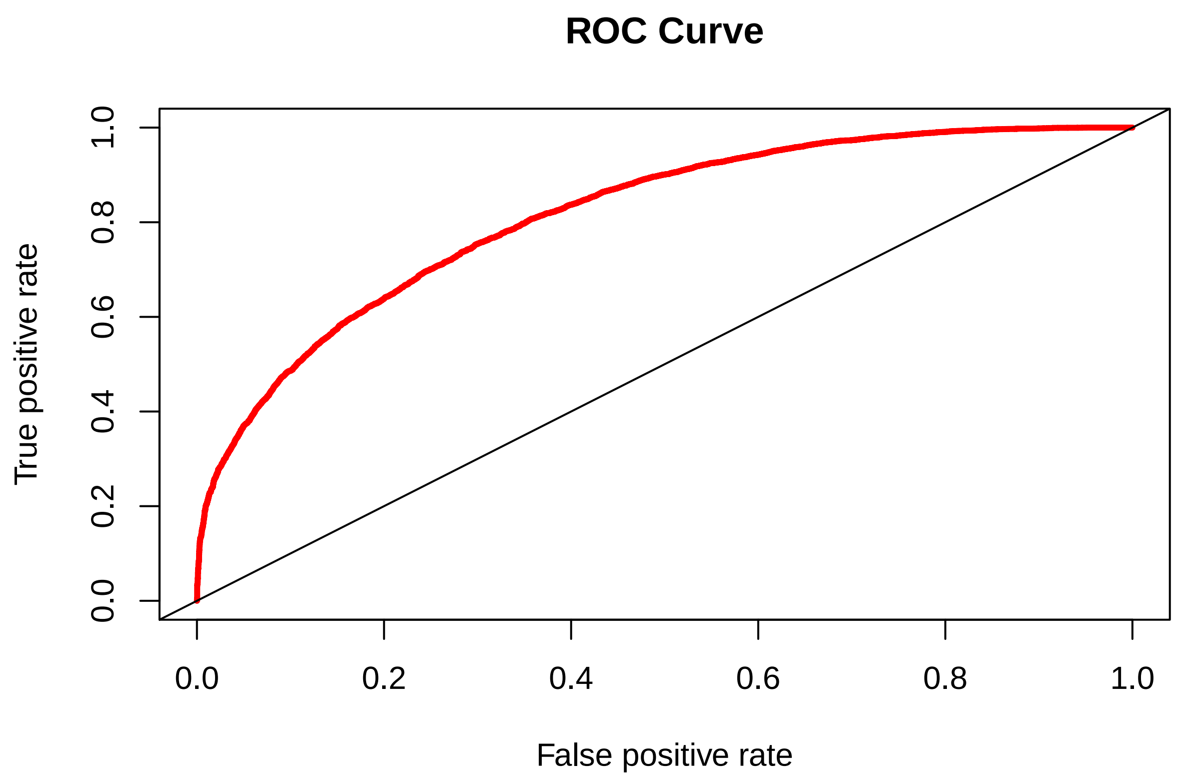
```
##        true
## predWin   No  Yes
##     No  4962 1788
##     Yes 2494 6450
```

```
#test error
logit_e = 1-sum(diag(logit_cm))/sum(logit_cm)
logit_e
```

```
## [1] 0.2728431
```

Next we will use ROC and AUC to determine best threshold value.

```
#Fit ROC
pred = prediction(logit_pred, ifelse(test$radiant_win == 'Yes',1,0))
perf = performance(pred, measure="tpr", x.measure="fpr")
plot(perf,col=2,lwd=3, main="ROC Curve")
abline(0,1)
```



```
auc = performance(pred, "auc")@y.values
auc
```

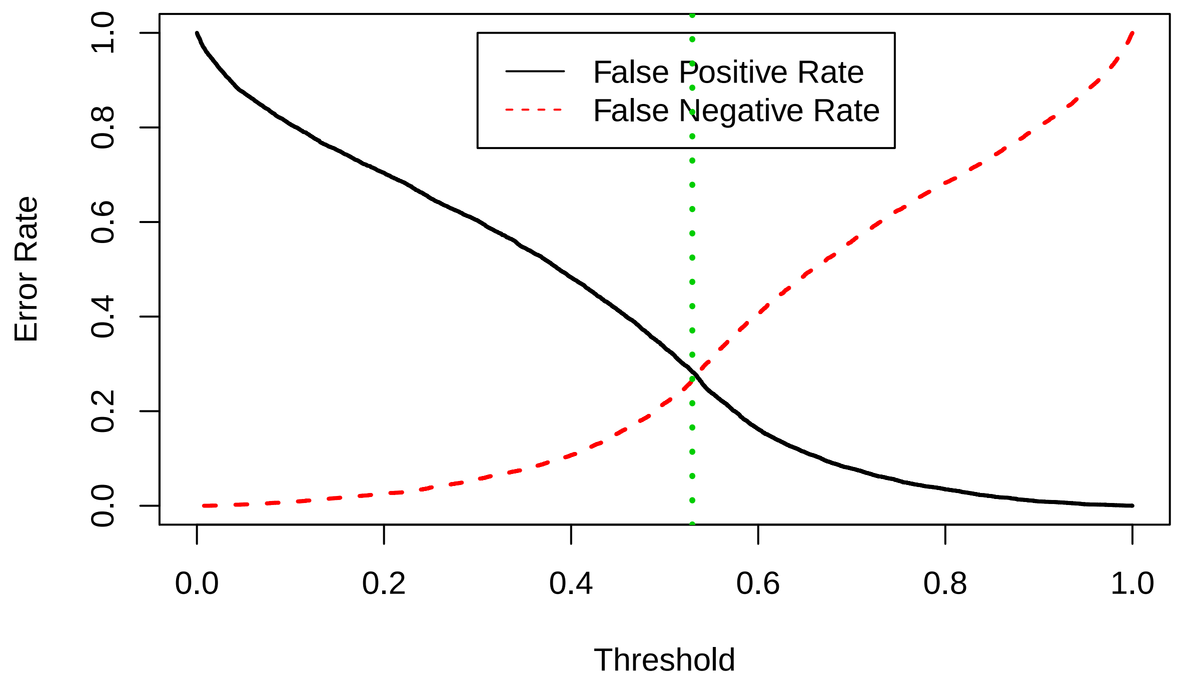
```
## [[1]]
## [1] 0.814713
```

```
# FPR
fpr = performance(pred, "fpr")@y.values[[1]]
cutoff = performance(pred, "fpr")@x.values[[1]]
# FNR
fnr = performance(pred,"fnr")@y.values[[1]]
# Plot
matplot(cutoff, cbind(fpr,fnr), type="l",lwd=2, xlab="Threshold",ylab="Error Rate")
# Add legend to the plot
legend(0.3, 1, legend=c("False Positive Rate","False Negative Rate"),
       col=c(1,2), lty=c(1,2))
rate = as.data.frame(cbind(Cutoff=cutoff, FPR=fpr, FNR=fnr))
rate$distance = sqrt((rate[,2])^2+(rate[,3])^2)
index = which.min(rate$distance)
best = rate$Cutoff[index]
```

```
#Best threshold value
best
```

```
## [1] 0.5295383
```

```
abline(v=best, col=3, lty=3, lwd=3)
```



Then we build a new logistic model with new threshold value and compare results with the previous model.

```
#Choose new threshold value
logit_pred_best = as.factor(ifelse(logit_pred <=best,'No','Yes'))
logit_cm_best = table(pred = logit_pred_best, test$radiant_win)
#Confusion Matrix
logit_cm_best
```

```
##
## pred    No  Yes
##   No  5346 2167
##   Yes 2110 6071
```

```
#Test error
logit_best_e = 1-sum(diag(logit_cm_best))/sum(logit_cm_best)
logit_best_e
```

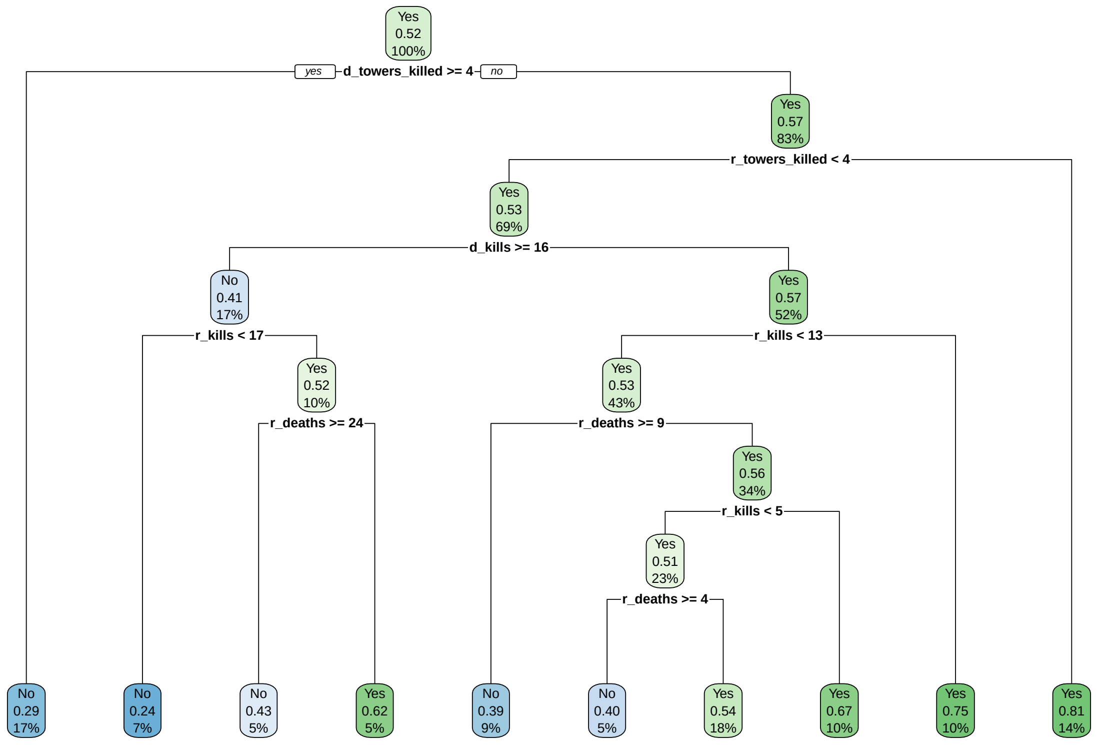
```
## [1] 0.2725245
```

The test error is slightly better than default `p=0.5` threshold value.

Random Forests

Before fitting Random Forests algorithm, we will first grow a single Random Tree. Random Tree is recognized as a classification algorithm that is easy to interprete.

```
set.seed(4)
tree = rpart(radiant_win~., data=train, method="class")
rpart.plot(tree)
```

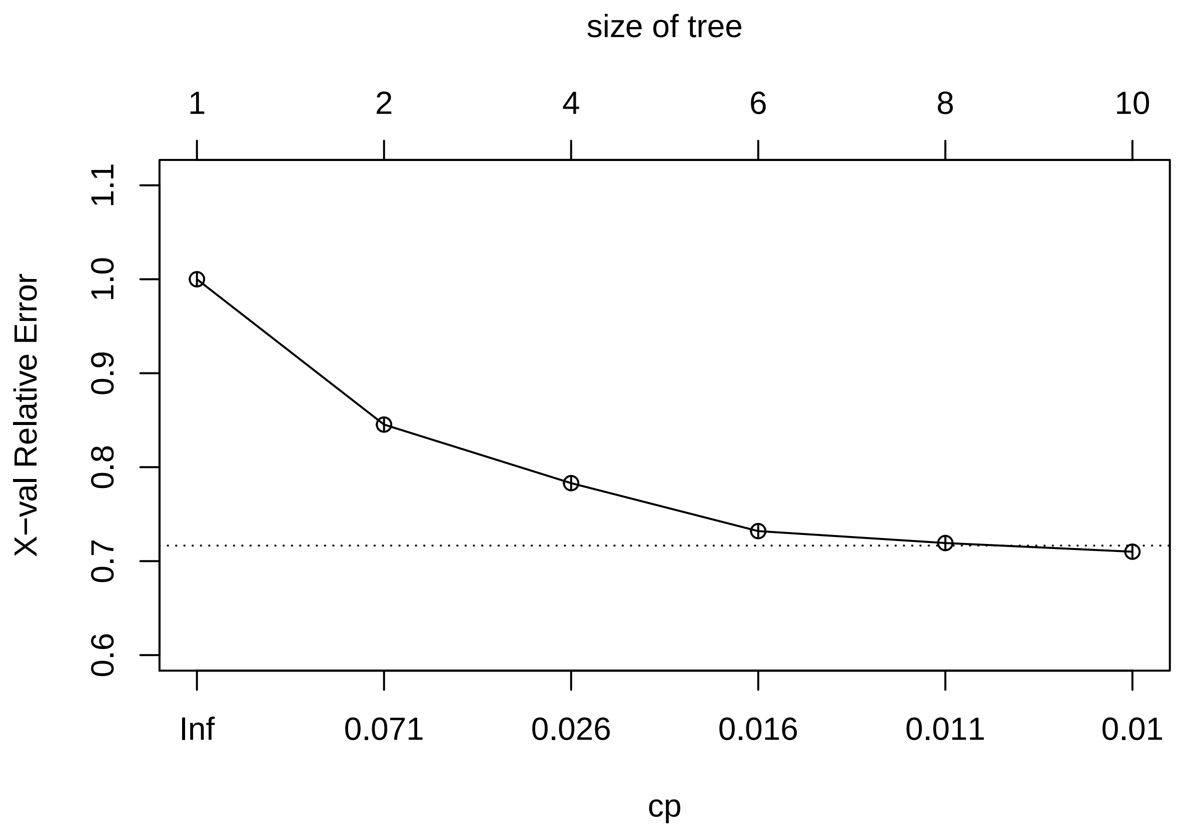


```
#Confusion Matrix
tree_pred = predict(tree, test, type="class")
tree_cm = table(tree_pred, test$radiant_win)
#Test error
tree_e = 1-sum(diag(tree_cm))/ sum(tree_cm)
tree_e
```

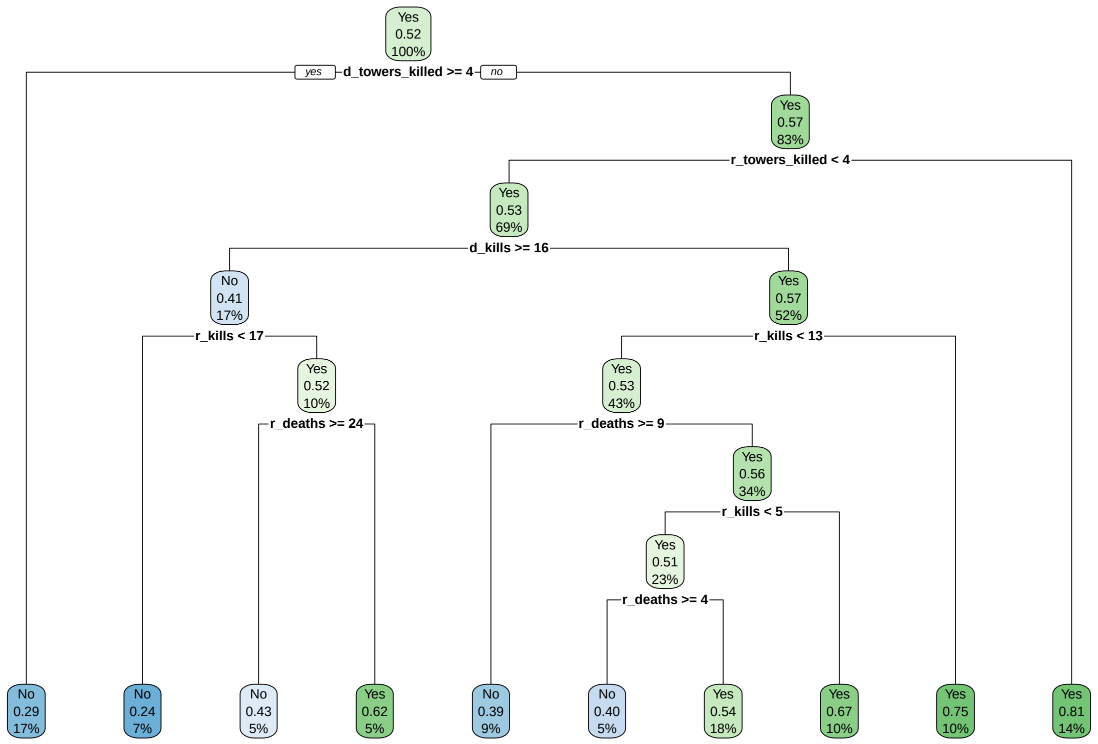
```
## [1] 0.3345227
```

As expected, random tree performs poorly with 33.4% error rate on test set. This is because Random Tree tends to overfit on training data therfore it will have a higher variance. However the graph of tree provides some useful insights. The tree splits at `towers_killed`, `kills`, `deaths` which are very important game factors. The tree splits first with number of towers killed making perfect sense because the team with no defending towers left are most likely losing. We will then perform cross validation to find the best `cp` value to prune the tree.

```
#CP Plot
plotcp(tree)
```



```r
bestcp = tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
pruned_tree = prune(tree,cp=bestcp)
rpart.plot(pruned_tree)
```



It turns out that the tree is the same before and after pruning. Next we will use bagging.

```
#m=p
set.seed(5)
bag = randomForest(radiant_win~., train, mtry =ncol(train)-1,importance=T)
bag_pred = predict(bag, test, type="response")
#Confusion Matrix
bag_cm = table(bag_pred, true = test$radiant_win)
bag_cm
```

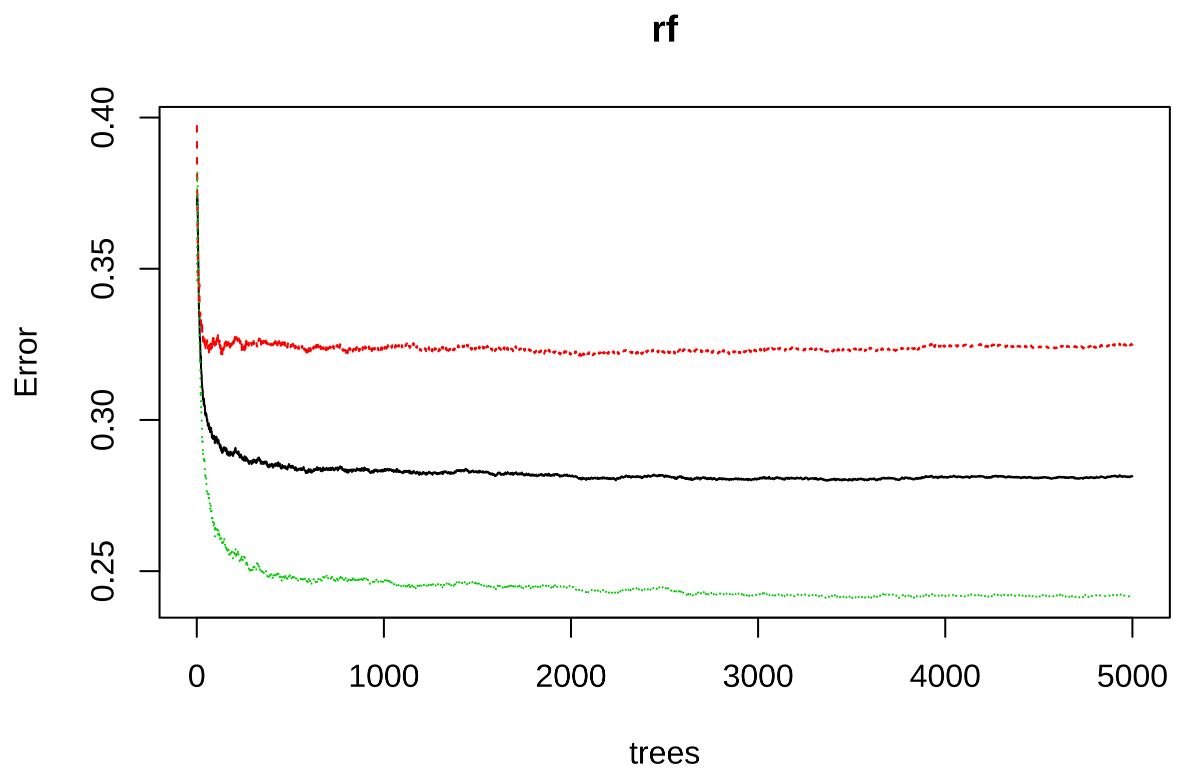
```
##          true
## bag_pred   No  Yes
##      No  4977 2079
##      Yes 2479 6159
```

```
#Test error
bag_e = 1-sum(diag(bag_cm))/sum(bag_cm)
bag_e
```

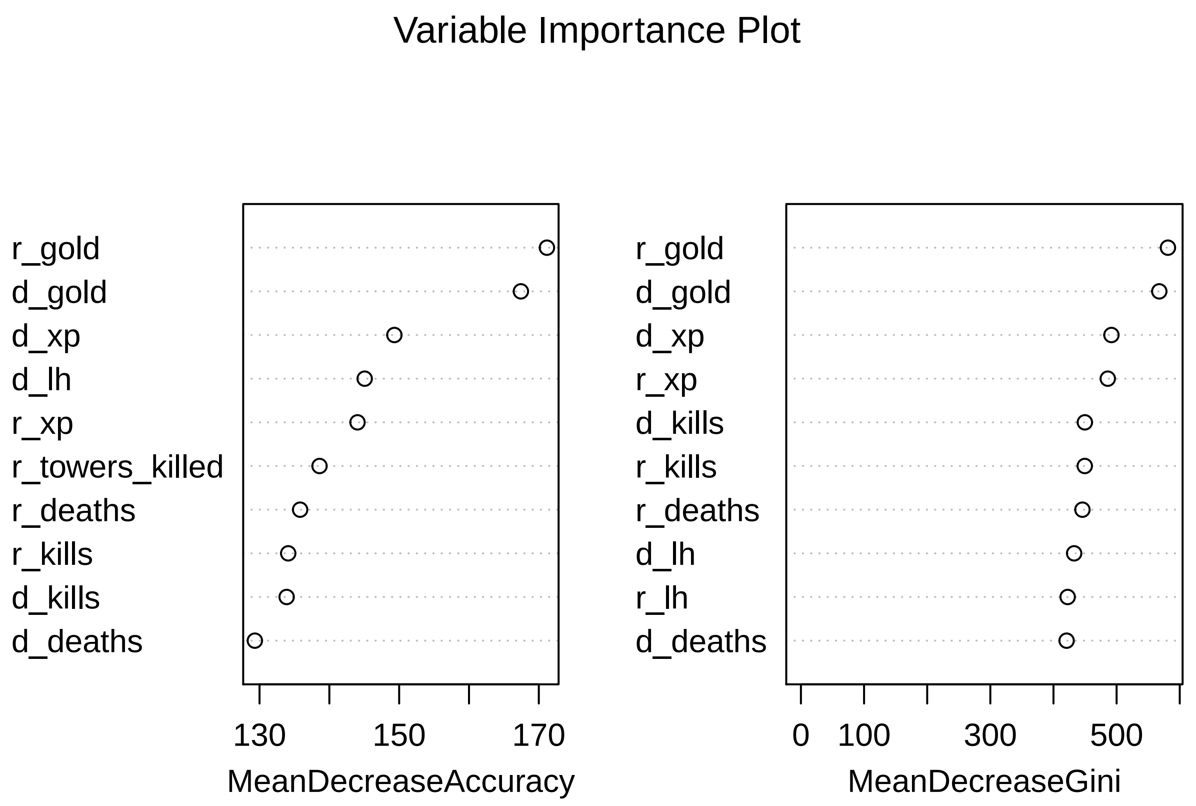
```
## [1] 0.2904295
```

Next we apply random forests to our data. We will use the default value $mtry = \sqrt{p}$ and grow 5000 trees.

```
#Random Forest model
set.seed(6)
rf = randomForest(radiant_win~., train, mtry = sqrt(ncol(train)-1),
                  importance=T,ntree=5000)
plot(rf)
```



```
varImpPlot(rf,n.var=10, main="Variable Importance Plot")
```



```
rf_pred = predict(rf, test, type="response")
rf_cm = table(pred = rf_pred, true=test$radiant_win)
rf_cm
```

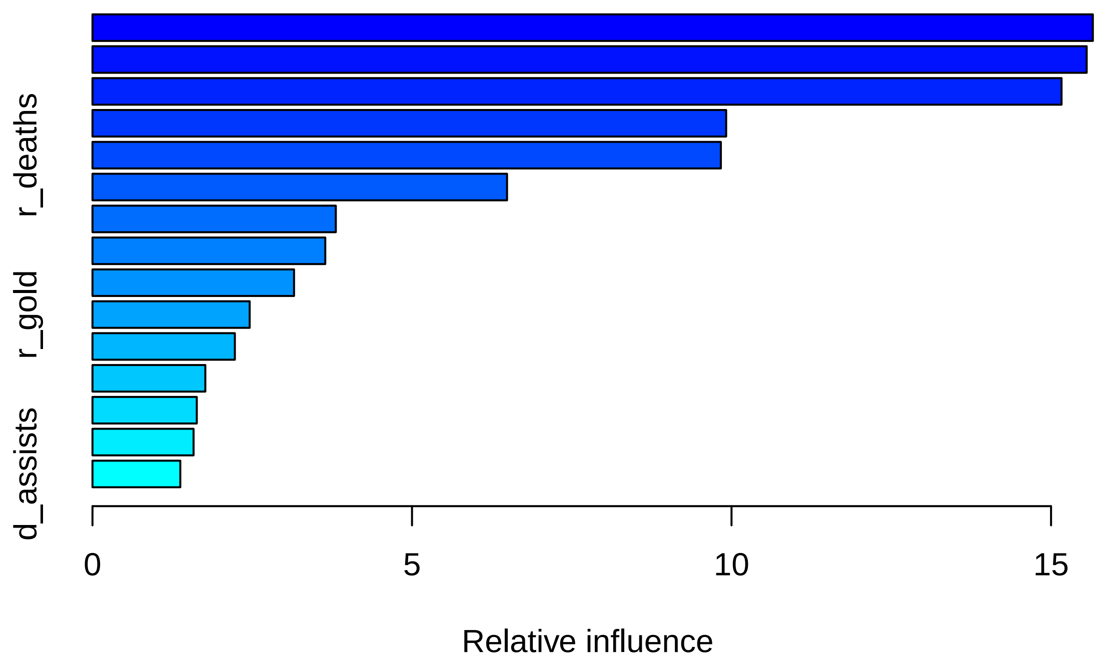
```
##      true
## pred    No  Yes
##   No  5014 2026
##   Yes 2442 6212
```

```
#Test error
rf_e = 1-sum(diag(rf_cm))/sum(rf_cm)
rf_e
```

```
## [1] 0.2846948
```

From the error vs trees plot we can see that the error decreases while number of trees grown increases. The variable importance plot clearly showed the most important 10 variables. The `gold` for both teams are listed as the top 2 important variables. The Random Forest classifier also performs significantly better than single Random Tree.*

```
set.seed(7)
boost = gbm(ifelse(radiant_win=='Yes',1,0)~., data=train,
                   distribution = "bernoulli",
            n.trees=5000, interaction.depth = 6)
summary(boost,cBars=15)
```



```
##                                                var      rel.inf
## d_towers_killed                    d_towers_killed 1.565419e+01
## r_towers_killed                    r_towers_killed 1.555762e+01
## r_kills                                    r_kills 1.516447e+01
## d_kills                                    d_kills 9.916129e+00
## r_deaths                                  r_deaths 9.833282e+00
## d_deaths                                  d_deaths 6.487173e+00
## d_denies                                  d_denies 3.807874e+00
## r_denies                                  r_denies 3.642838e+00
## r_rune_pickups                      r_rune_pickups 3.154137e+00
## r_gold                                      r_gold 2.459887e+00
## d_rune_pickups                      d_rune_pickups 2.228256e+00
## d_gold                                      d_gold 1.766100e+00
## d_lh                                          d_lh 1.632307e+00
## r_assists                                r_assists 1.581850e+00
## d_assists                                d_assists 1.374163e+00
## d_xp                                          d_xp 1.370787e+00
## r_lh                                          r_lh 1.113168e+00
## r_xp                                          r_xp 8.881097e-01
## d_roshans_killed                  d_roshans_killed 8.259788e-01
## r_roshans_killed                  r_roshans_killed 6.732495e-01
## d_firstblood_claimed          d_firstblood_claimed 1.979383e-01
## d_level                                    d_level 1.816701e-01
## r_level                                    r_level 1.558571e-01
## r_firstblood_claimed          r_firstblood_claimed 1.213217e-01
## d_teamfight_participation d_teamfight_participation 6.684301e-02
## r_teamfight_participation r_teamfight_participation 5.548767e-02
## r_stuns                                       r_stuns 3.106252e-02
## d_obs_placed                             d_obs_placed 2.080688e-02
## d_stuns                                       d_stuns 9.885187e-03
## game_time                                   game_time 9.147006e-03
## d_creeps_stacked                     d_creeps_stacked 5.817264e-03
## r_sen_placed                             r_sen_placed 3.986268e-03
## r_obs_placed                             r_obs_placed 3.699748e-03
## r_camps_stacked                       r_camps_stacked 2.417096e-03
## d_camps_stacked                       d_camps_stacked 1.141460e-03
## d_sen_placed                             d_sen_placed 7.717971e-04
## r_creeps_stacked                     r_creeps_stacked 5.760924e-04
```

```
boost_pred = ifelse(predict(boost, test, n.trees=5000) >=0.5,'Yes','No')
boost_cm = table(pred = boost_pred, true = test$radiant_win)
boost_cm
```

```
##      true
## pred    No  Yes
##   No  6521 4186
##   Yes  935 4052
```

```
#Test Error
boost_e = 1-sum(diag(boost_cm))/sum(boost_cm)
boost_e
```

```
## [1] 0.326303
```

The boosted model performs better than Random Tree but worse than Random Forest

Support Vector Machine

Lastly, I will apply the SVM model. I will compare results from `svm` functions from `e1071` and `liquidSVM` package. I will set `kernal="polynomial"` and `cost=1`.

```
set.seed(8)
#Using SVM from LiquidSVM package
svmfit = mcSVM(radiant_win~., train, scale=TRUE, gpus=1)
svm_pred = predict(svmfit,test, type="response")
svm_cm = table(pred=svm_pred, true=test$radiant_win)
svm_cm
```

```
##      true
## pred    No  Yes
##   No  4746 1858
##   Yes 2710 6380
```

```
svm_e = 1- sum(diag(svm_cm)) /sum(svm_cm)
svm_e
```

```
## [1] 0.2910666
```

```
#Using SVM from e1071 package
set.seed(9)
svmfit2 = e1071::svm(radiant_win~., train, kernal="polynomial",cost=1)
svm_pred2 = predict(svmfit2,test, type="response")
svm_cm2 = table(pred=svm_pred2, true=test$radiant_win)
svm_cm2
```

```
##      true
## pred    No  Yes
##   No  4822 1806
##   Yes 2634 6432
```

```
svm_e2 = 1- sum(diag(svm_cm2)) /sum(svm_cm2)
svm_e2
```

```
## [1] 0.2829107
```

The `svm` function in `e1071` gives lower test error than `mcSVM` in `liquidSVM` package. However, `liquidSVM` package is significantly faster than `e1071` package. This is crucial in production. Considering the size of the data set and limitation of computational power, it is not time efficient to run `tune` function to find the best `cost` value. Therefore we will keep using the default hyperparameters.

Conclusion

Final model

Comparing the test errors of all the models, the final model with lowest error is logistic regression with threshold set to 0.5325936. Here is a table of all the test errors:

```
test_error = data.frame(logistic = logit_e, logistic_best_cutoff = logit_best_e, random_tree = tree_e, 
test_error = t(test_error)
test_error = test_error[order(test_error),]
test_error %>%
  kable()%>%
  kable_styling()
```

	x
logistic_best_cutoff	0.2725245
logistic	0.2728431
svm_e1071	0.2829107
random_forest	0.2846948
bagging	0.2904295
svm_liquidSVM	0.2910666
boost	0.3263030
random_tree	0.3345227

Study limitations

First, since R stores all its objects in memory, the size of the dataset limits the training probabilities. Even with 16GB of RAM, training models still takes several minutes, especially for complex methods such as random forests and SVM. The execution speed of algorithms is crucial espesically in production. If applying those ML algorithms in a real-time winning predicting AI, the speed should be as fast as possible. Therefore the limitation of computational power prohibits better tuning and production ability of the model. With cloud computing service such as Google Cloud and AWS, I believe algorithms like SVM, neural networks and deep learning will perform better.

Second, this machine learning project did not focus too much on outliers. As the box plot showed before, there exists a significant amount of outliers. It makes sense because there are a lot of uncertainties in an online game. A team can win even with no towers standing and low kills. Turnovers are very common in *Dota2* games. It happens when enemy team making a fatal mistake or a player is throwing in the game. Methods like logistic regression are more sensitive to outliers while tree based methods are more robust. Therefore if outliers are dealt with properly, these machine learning algorithms will idealy perform better.

References

data source: https://www.kaggle.com/c/mlcourse-dota2-win-prediction/overview