Machine Learning Project

Ruiming Yu

2022-04-25

Introduction

LOL games!

Related work

Most players would focus only on one of the features to predict the winner of the game, which is the "Kills" difference. However, as the result of the game is related to multiple factors, and players in higher rankings can figure out more details in the game,

Methods

Data and experiments setup

Data preparation and cleaning

```
library(tidyverse) # for data manipulation
library(faraway) # to perform `cor()`
library(reshape2) # for creating heatmap of variables
library(pROC) # computing AUC for models
library(caret) # for KNN model fitting
library(MASS) # use of `step()` and other function
library(tree) # tree models
library(randomForest) # random forest
library(mlbench) # validation
```

Package loading

```
# setwd("/Users/jordanbelfort/Desktop/Academic/mL/project")
data <- read_csv("lol_10min.csv")
colnames(data)
```

Data import and explanation

```
##  [1] "gameId"                          "blueWins"
##  [3] "blueWardsPlaced"                 "blueWardsDestroyed"
##  [5] "blueFirstBlood"                  "blueKills"
##  [7] "blueDeaths"                      "blueAssists"
##  [9] "blueEliteMonsters"               "blueDragons"
## [11] "blueHeralds"                     "blueTowersDestroyed"
## [13] "blueTotalGold"                   "blueAvgLevel"
## [15] "blueTotalExperience"             "blueTotalMinionsKilled"
## [17] "blueTotalJungleMinionsKilled" "blueGoldDiff"
## [19] "blueExperienceDiff"              "blueCSPerMin"
## [21] "blueGoldPerMin"                  "redWardsPlaced"
## [23] "redWardsDestroyed"               "redFirstBlood"
## [25] "redKills"                        "redDeaths"
## [27] "redAssists"                      "redEliteMonsters"
## [29] "redDragons"                      "redHeralds"
## [31] "redTowersDestroyed"              "redTotalGold"
## [33] "redAvgLevel"                     "redTotalExperience"
## [35] "redTotalMinionsKilled"           "redTotalJungleMinionsKilled"
## [37] "redGoldDiff"                     "redExperienceDiff"
## [39] "redCSPerMin"                     "redGoldPerMin"
```

(We can make a brief explanation of the variables here, maybe in the final report. In the video I guess we can use the picture you included in the sildes)

We can see that almost all the variables except the `gameId` and `blueWins` are continuous variables, and are related to one of the features of the game when it lasts 10 minutes. The `blueWins` will be used as the response for the model while other variables will work as the predictors in the model.

However, we do notice that there are **collinearty** problems in the data set when some of the columns are completely negatively related such as `redGoldDiff` and `blueGoldDiff`. We will leave only one of the variables in each pair. And `gameId` is irrelevant to the research so we'd also remove it.

```
data <- data %>%
  dplyr::select(-"gameId", -"redGoldDiff", -"redExperienceDiff", -"redFirstBlood", -"blueDeaths", -"redD
```

```
# use `is.na()` to detect marked NA values
sum(is.na(data))
```

Missing values detect

```
## [1] 0
```

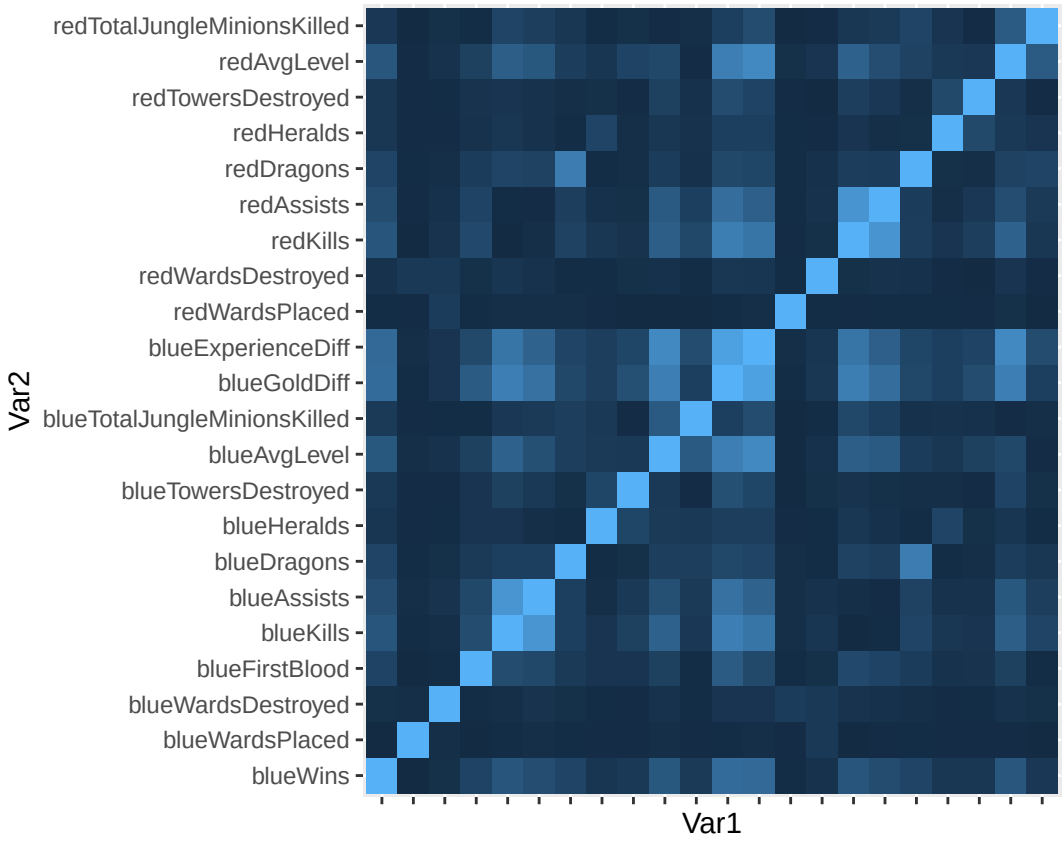
```
# use `range()` to check if there are abnormal values in the data set
data.frame(range = c("Minimum", "Maximum"), map(data, range))
```

```
##     range blueWins blueWardsPlaced blueWardsDestroyed blueFirstBlood blueKills
## 1 Minimum        0               5                  0              0         0
## 2 Maximum        1             250                 27              1        22
##   blueAssists blueDragons blueHeralds blueTowersDestroyed blueAvgLevel
## 1           0           0           0                   0          4.6
```

```
## 2          29           1           1                    4         8.0
##   blueTotalJungleMinionsKilled blueGoldDiff blueExperienceDiff redWardsPlaced
## 1                            0       -10830              -9333              6
## 2                           92        11467               8348            276
##   redWardsDestroyed redKills redAssists redDragons redHeralds
## 1                 0        0          0          0          0
## 2                24       22         28          1          1
##   redTowersDestroyed redAvgLevel redTotalJungleMinionsKilled
## 1                  0         4.8                           4
## 2                  2         8.2                          92
```

The results do show that there are obvious missing values problem.

```
# use `abs()` to better detect those variables with low correlation scores
corr_mat <- abs(round(cor(data),2))
melted_corr_mat <- melt(corr_mat)

ggplot(data = melted_corr_mat, aes(x = Var1, y = Var2, fill = value)) +
geom_tile() +
  theme(axis.text.x = element_blank())
```



Correlation to response

Response variable factorization As we will be using `blueWins` as the response for this research, and it stands for classification results but not merely 1 and 0, we need to factorize the response.

```
data$blueWins <- as.factor(data$blueWins)
```

Till now we have finished the part of data preparation, and the cleaned data set will have 9879 observations and 22 columns. We will conduct our research in the following.

Data set splitting In this part we will split the data set into training and test set so that in the following parts we can better measure efficiency of a model. We will follow the **80:20** ratio to split the data set. And we'd also split the training set into k-fold cross-validation set. Here we choose $k = 5$.

```
# select approximate 0.8 of the observations as the training set
set.seed(0)
train_index <- sample(nrow(data), round(0.8 * nrow(data)), replace = FALSE, prob = NULL)

data_train <- data[train_index, ]
data_test <- data[-train_index, ]

# use `trainControl()` for cross-validation
cv <- trainControl(method = "cv", number = 5)
```

Naive prediction In the previous part we have stated that for many of the players trying to predict the winner, they will focus only on the differences of "kills" and the side with more "kills" are more likely to win the game. Therefore, we are using this idea in our naive model part.

(classification method would change no matter data set we have, so no cv is needed)

```
data_naive_train <- data_train %>%
  dplyr::select(blueWins, blueKills, redKills) %>%
  mutate(blueWinsPredict = ifelse(blueKills - redKills >= 0, 1, 0))

train_error_naive <- mean(data_naive_train$blueWins != data_naive_train$blueWinsPredict)

data_naive_test <- data_test %>%
  dplyr::select(blueWins, blueKills, redKills) %>%
  mutate(blueWinsPredict = ifelse(blueKills - redKills >= 0, 1, 0))

test_error_naive <- mean(data_naive_test$blueWins != data_naive_test$blueWinsPredict)

# output training and test error, roc of the model on test set
train_error_naive
```

```
## [1] 0.2967228
```

```
test_error_naive
```

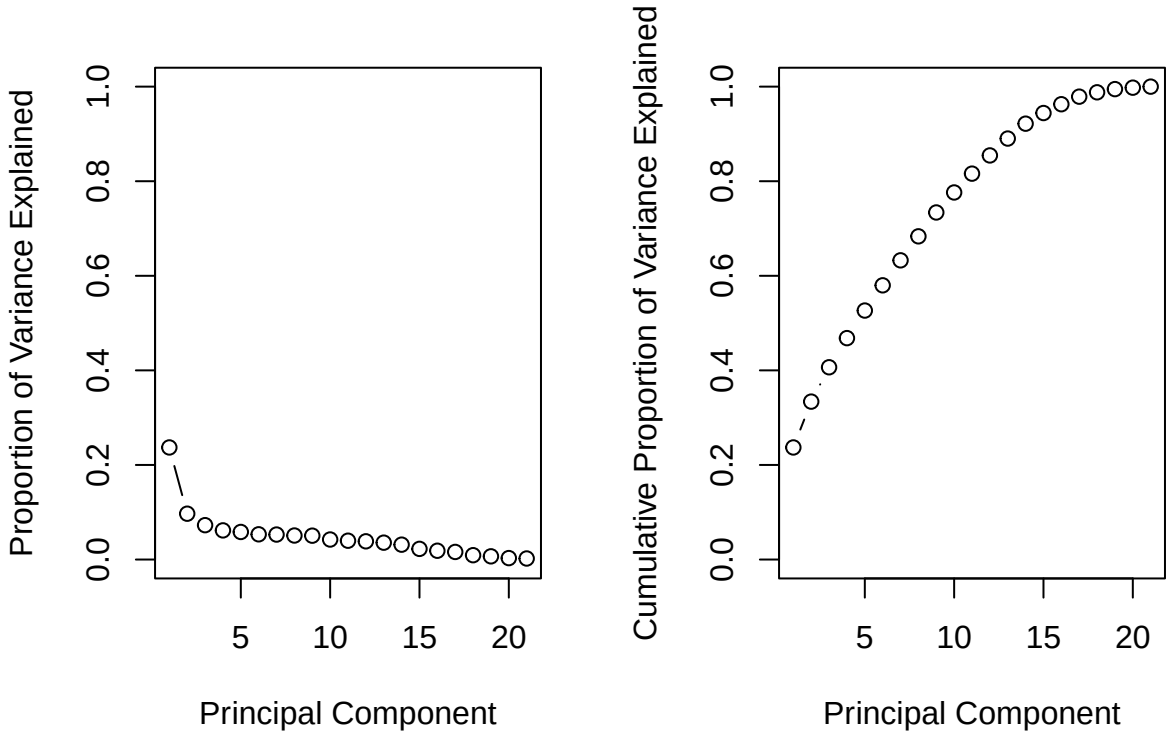
```
## [1] 0.2899798
```

```
roc_naive <- roc(data_naive_test$blueWins, data_naive_test$blueWinsPredict, levels = c(0, 1), direction
```

(we can use `table` to present the confusion matrix)

From above we know that the training and test error are both around 0.3, which makes this model a reasonable but not excellent predictor. In the following part, we will conduct different models to improve the prediction efficiency.

```
set.seed(0)
pr.out <- prcomp(data[-1], scale = TRUE)
# biplot(pr.out, scale = TRUE)

pr.var <- pr.out$sdev^2
pve <- pr.var / sum(pr.var)

par(mfrow = c(1, 2))
plot(pve, xlab = "Principal Component",
ylab = "Proportion of Variance Explained", ylim = c(0, 1),
type = "b")
plot(cumsum(pve), xlab = "Principal Component",
ylab = "Cumulative Proportion of Variance Explained", ylim = c(0, 1), type = "b")
```



PCA

The results show that except for the first primary component, there is no significant drop in the proportion of variance explained in the following components, so PCA is not a good method in this study.

Logistic regression Since the response for this model is binary, it is straightforward that we can think of fitting a logistic regression model and judge if the outcome is greater than 0.5 or not to define the prediction.

```
model_logistic <- train(blueWins ~., data = data_train, method = "glm", trControl = cv, family = "binom

prob_train_logistic <- predict(model_logistic, newdata = data_train, type = "prob")
label_train_logistic <- ifelse(prob_train_logistic[, 2] >= 0.5, 1, 0)
train_error_logistic <- mean(label_train_logistic != data_train$blueWins)

prob_test_logistic <- predict(model_logistic, newdata = data_test, type = "prob")
label_test_logistic <- ifelse(prob_test_logistic[, 2] >= 0.5, 1, 0)
test_error_logistic <- mean(label_test_logistic != data_test$blueWins)

train_error_logistic
```

```
## [1] 0.2667342
```

```
test_error_logistic
```

```
## [1] 0.2732794
```

```
roc_logistic <- roc(data_test$blueWins, prob_test_logistic[, 2], levels = c(0, 1), direction = "auto")
```

```
model_lda <- train(blueWins ~., data = data_train, method = "lda", trControl = cv, family = "binomial")

prob_train_lda <- predict(model_lda, newdata = data_train, type = "prob")
label_train_lda <- ifelse(prob_train_logistic[, 2] >= 0.5, 1, 0)
train_error_lda <- mean(label_train_logistic != data_train$blueWins)

prob_test_lda <- predict(model_lda, newdata = data_test, type = "prob")
label_test_lda <- ifelse(prob_test_lda[, 2] >= 0.5, 1, 0)
test_error_lda <- mean(label_test_lda != data_test$blueWins)

train_error_lda
```

LDA

```
## [1] 0.2667342
```

```
test_error_lda
```

```
## [1] 0.2773279
```

```
roc_lda <- roc(data_test$blueWins, prob_test_lda[, 2], levels = c(0, 1), direction = "auto")
```

```
model_qda <- train(blueWins ~., data = data_train, method = "qda", trControl = cv, family = "binomial")

prob_train_qda <- predict(model_qda, newdata = data_train, type = "prob")
label_train_qda <- ifelse(prob_train_qda[, 2] >= 0.5, 1, 0)
train_error_qda <- mean(label_train_qda != data_train$blueWins)

prob_test_qda <- predict(model_qda, newdata = data_test, type = "prob")
label_test_qda <- ifelse(prob_test_qda[, 2] >= 0.5, 1, 0)
test_error_qda <- mean(label_test_qda != data_test$blueWins)

train_error_qda
```

QDA

```
## [1] 0.2883715
```

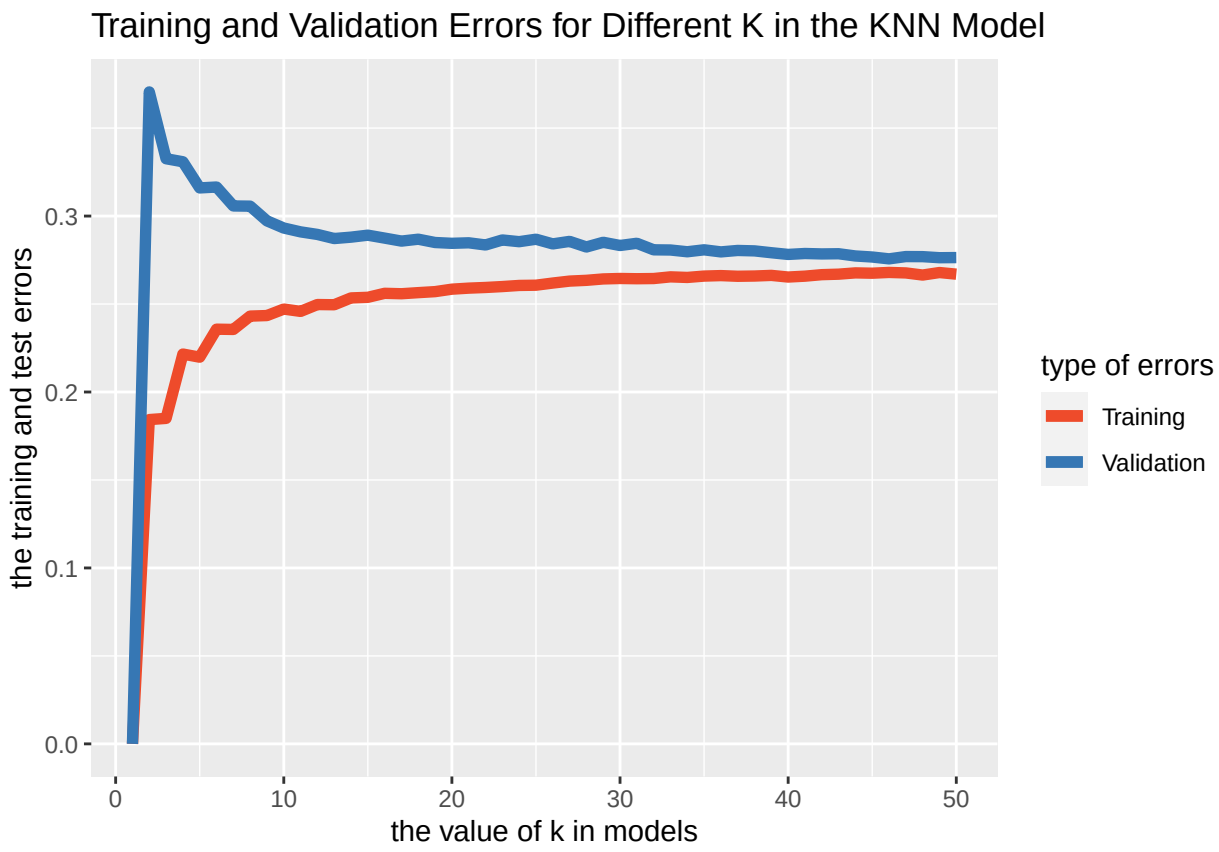
```
test_error_lda
```

```
## [1] 0.2773279
```

```
roc_qda <- roc(data_test$blueWins, prob_test_qda[, 2], levels = c(0, 1), direction = "auto")
```

```
set.seed(0)

# save training and test errors
train_error_knn <- rep(0, 50)
test_error_knn <- rep(0, 50)

fold_index <- sample(1:5, nrow(data_train), replace = TRUE)

# run for loops to store the errors for each cv set and k
for (i in 1:5) {
  cv_train <- data_train[fold_index != i, ]
  cv_test <- data_train[fold_index == i, ]

  for (j in 2:50) {
    knn_model <- knn3(blueWins ~ ., data = cv_train, k = j)

    prob_train_knn <- predict(knn_model, newdata = cv_train)
    label_train_knn <- ifelse(prob_train_knn[, 2] >= 0.5, 1, 0)
    train_error_knn[j] = train_error_knn[j] + mean(label_train_knn != cv_train$blueWins)

    prob_test_knn <- predict(knn_model, newdata = cv_test)
    label_test_knn <- ifelse(prob_test_knn[, 2] >= 0.5, 1, 0)
    test_error_knn[j] = test_error_knn[j] + mean(label_test_knn != cv_test$blueWins)
  }
}
```

```
# calculate the mean of training and test errors
train_error_knn = train_error_knn / 5
test_error_knn = test_error_knn / 5

error_knn <- data.frame(k = 1:50,
           train_error_knn = train_error_knn,
           validation_error_knn = test_error_knn)

colors = c("Training" = "#EE4B2B", "Validation" = "#3677B4")
ggplot(data = error_knn) +
  geom_line(aes(x = k, y = train_error_knn, color = "Training"), lwd = 2) +
  geom_line(aes(x = k, y = validation_error_knn, color = "Validation"), lwd = 2) +
  scale_color_manual(values = colors) +
  labs(x = "the value of k in models",
       y = "the training and test errors",
       color = "type of errors") +
  ggtitle("Training and Validation Errors for Different K in the KNN Model")
```



KNN

```
best_k <- which.min(test_error_knn[-1])
```

With the best k found, train the final knn model and compute the training and test error.

```
model_knn_final <- knn3(blueWins ~ ., data = data_train, k = best_k)

prob_train_knn <- predict(model_knn_final, newdata = data_train)
```

```
label_train_knn <- ifelse(prob_train_knn[, 2] >= 0.5, 1, 0)
train_error_knn <- mean(label_train_knn != data_train$blueWins)

prob_test_knn <- predict(model_knn_final, newdata = data_test)
label_test_knn <- ifelse(prob_test_knn[, 2] >= 0.5, 1, 0)
test_error_knn <- mean(label_test_knn != data_test$blueWins)

train_error_knn
```

```
## [1] 0.2683791
```

```
test_error_knn
```

```
## [1] 0.2869433
```

```
roc_knn <- roc(data_test$blueWins, prob_test_knn[, 2], levels = c(0, 1), direction = "auto")
```

```
set.seed(0)
model_tree_origin <- tree(blueWins ~ blueGoldDiff, data = data_train)
cv_tree <- cv.tree(model_tree_origin)
best_size <- cv_tree$size[which.min(cv_tree$dev)]
model_tree <- prune.tree(model_tree_origin, best = best_size)

prob_train_tree <- predict(model_tree, newdata = data_train)
label_train_tree <- ifelse(prob_train_tree[, 2] >= 0.5, 1, 0)
train_error_tree <- mean(label_train_tree != data_train$blueWins)

prob_test_tree <- predict(model_tree, newdata = data_test)
label_test_tree <- ifelse(prob_test_tree[, 2] >= 0.5, 1, 0)
test_error_tree <- mean(label_test_tree != data_test$blueWins)

train_error_tree
```

Decision Tree

```
## [1] 0.2741997
```

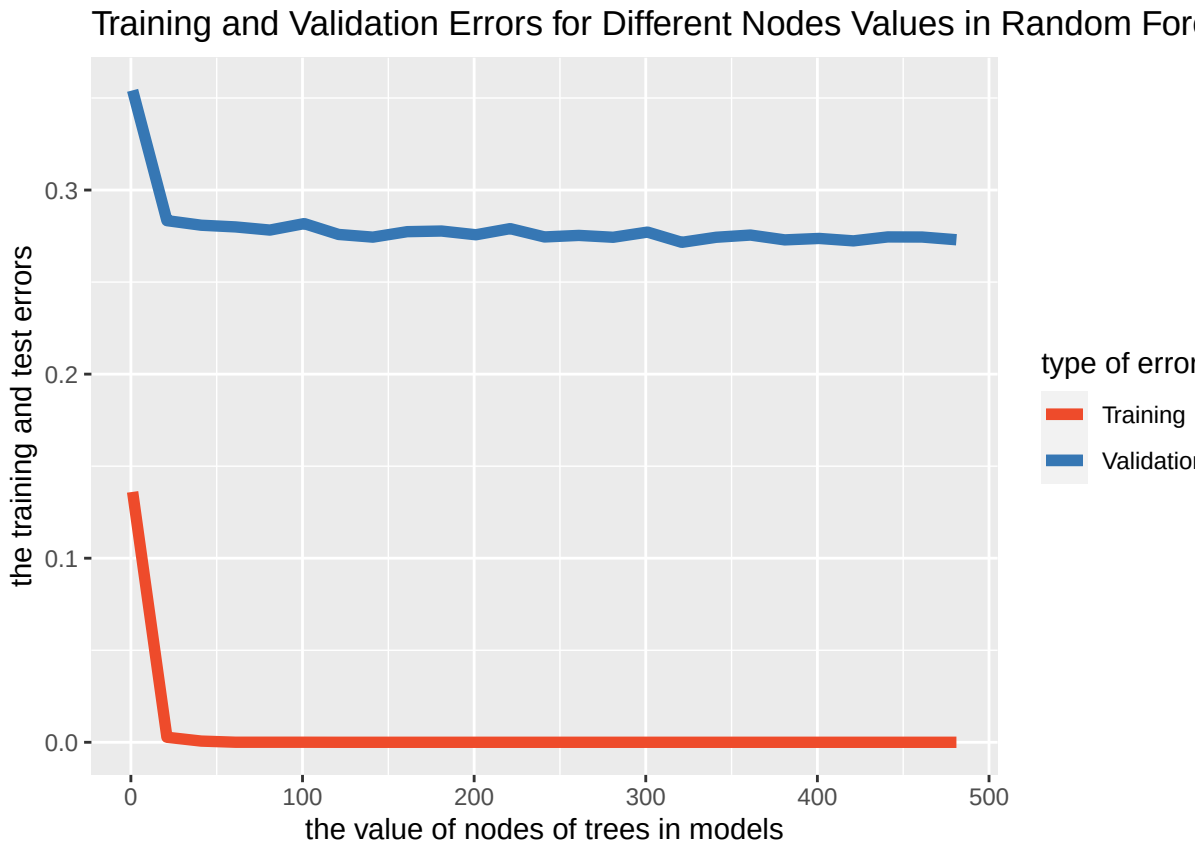
```
test_error_tree
```

```
## [1] 0.2763158
```

```
roc_tree <- roc(data_test$blueWins, label_test_tree, levels = c(0, 1), direction = "auto")
```

Random forest

```r
set.seed(0)

tree_num <- seq(1, 500, 20)

validation_train_error <- rep(0, 25)
validation_test_error <- rep(0, 25)

fold_index <- sample(1:5, nrow(data_train), replace = TRUE)

for (i in 1:5) {
  cv_train <- data_train[fold_index != i, ]
  cv_test <- data_train[fold_index == i, ]

  for (j in 1:25) {
    model_rf <- randomForest(blueWins ~ ., data = cv_train, importance = TRUE, ntree = tree_num[j])

    label_train <- predict(model_rf, newdata = cv_train, type = "response")
    train_error <- mean(label_train != cv_train$blueWins)
    validation_train_error[j] = validation_train_error[j] + train_error

    label_test <- predict(model_rf, newdata = cv_test, type = "response")
    test_error <- mean(label_test != cv_test$blueWins)
    validation_test_error[j] = validation_test_error[j] + test_error
  }
}

# calculate the mean of training and test errors
validation_train_error = validation_train_error / 5
validation_test_error = validation_test_error / 5

error_rf <- data.frame(nodes = tree_num,
           train_error_rf = validation_train_error,
           validation_error_rf = validation_test_error)

colors = c("Training" = "#EE4B2B", "Validation" = "#3677B4")
ggplot(data = error_rf) +
  geom_line(aes(x = nodes, y = train_error_rf, color = "Training"), lwd = 2) +
  geom_line(aes(x = nodes, y = validation_error_rf, color = "Validation"), lwd = 2) +
  scale_color_manual(values = colors) +
  labs(x = "the value of nodes of trees in models",
       y = "the training and test errors",
       color = "type of errors") +
  ggtitle("Training and Validation Errors for Different Nodes Values in Random Forest")
```



Tuning for `ntree`

```
rf_ntree_index <- which.min(validation_test_error)
rf_ntree <- tree_num[rf_ntree_index]
```

From above, $ntree = 321$ is the best fitting.

```
# Random Search
control <- trainControl(method = "cv", number = 5, search = "random")

set.seed(0)

mtry <- sqrt(ncol(data_train))

rf_random <- train(blueWins ~ ., data = data_train, method = "rf", metric = "Accuracy", tuneLength = 10

print(rf_random)
```

Tuning for `mtry`

```
## Random Forest
##
## 7903 samples
##   21 predictor
```

```
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6323, 6322, 6322, 6322, 6323
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.7298476  0.4596151
##    2    0.7283304  0.4566089
##    4    0.7244073  0.4487731
##    7    0.7247871  0.4495372
##   11    0.7197259  0.4394010
##   14    0.7189676  0.4379057
##   18    0.7144124  0.4288008
##   19    0.7142863  0.4285369
##   21    0.7146661  0.4292983
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 1.
```

To choose $mtry = 1$ is the best fitting.

```
set.seed(0)

model_randomforest <- randomForest(blueWins ~ ., data = data_train, mtry = 1, ntree = 321)

prob_train_rf <- predict(model_randomforest, newdata = data_train, type = "prob")
label_train_rf <- ifelse(prob_train_rf[, 2] >= 0.5, 1, 0)
train_error_rf <- mean(label_train_rf != data_train$blueWins)

prob_test_rf <- predict(model_randomforest, newdata = data_test, type = "prob")
label_test_rf <- ifelse(prob_test_rf[, 2] >= 0.5, 1, 0)
test_error_rf <- mean(label_test_rf != data_test$blueWins)

train_error_rf
```

Model Fitting

```
## [1] 0.2033405
```

```
test_error_rf
```

```
## [1] 0.2788462
```

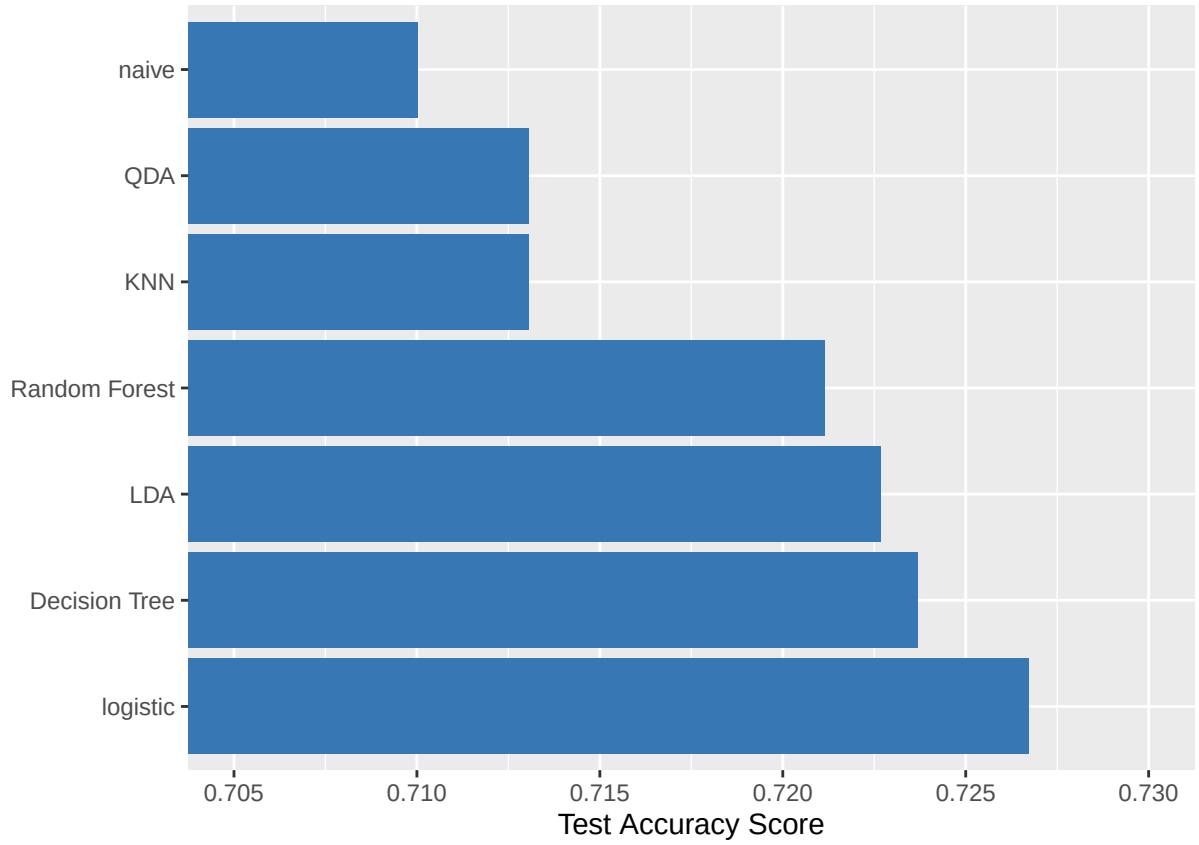
```
roc_rf <- roc(data_test$blueWins, prob_test_rf[, 2], levels = c(0, 1), direction = "auto")
```

Results

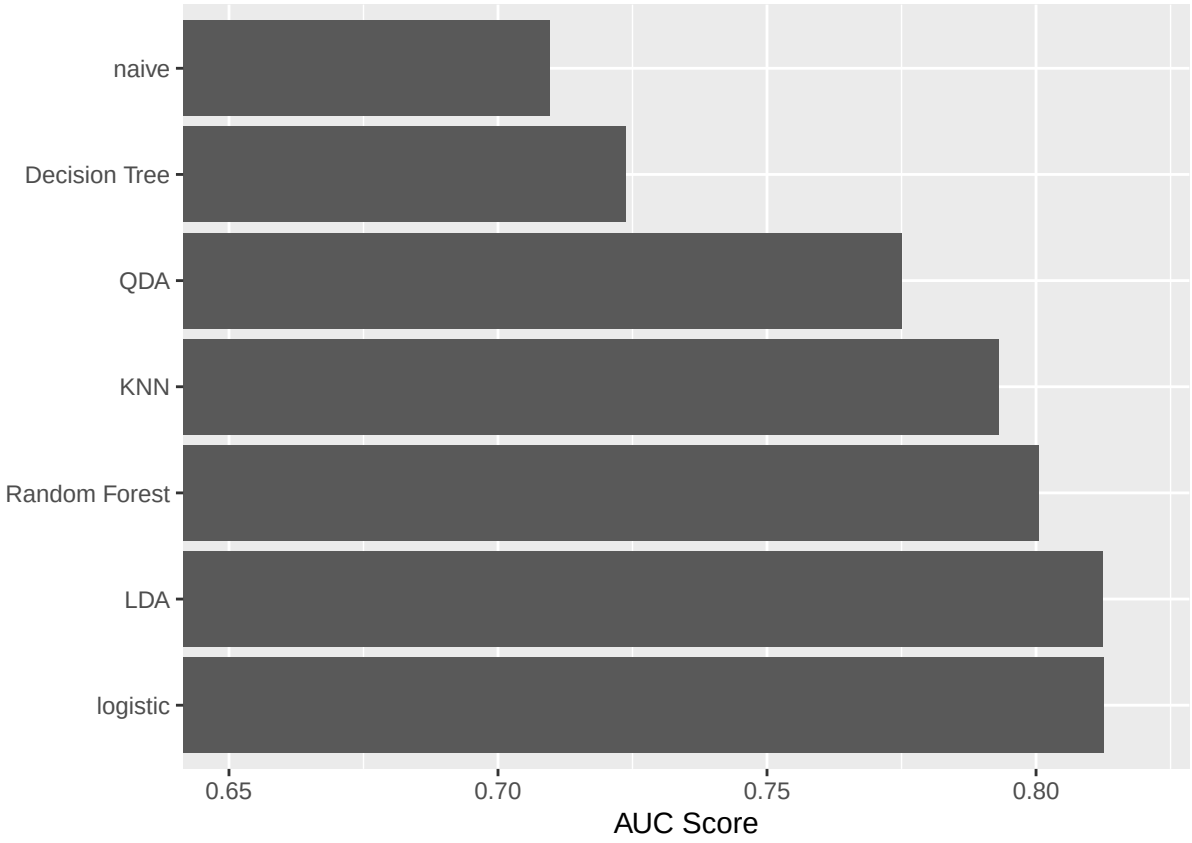
```
evaluation <- data.frame(label = c("naive", "logistic", "LDA", "QDA", "KNN", "Decision Tree", "Random Fo
                         train_error = c(train_error_naive, train_error_logistic, train_error_lda, train
                         test_error = c(test_error_naive, test_error_logistic, test_error_lda, test_erro
                         auc_score = c(auc(roc_naive), auc(roc_logistic), auc(roc_lda), auc(roc_qda), au

train_error_plot <- evaluation %>%
  ggplot() +
  geom_bar(aes(y = reorder(label, train_error), x = 1 - train_error), stat = "identity", fill = "#EE4B2B
  coord_cartesian(xlim = c(0.65, 1)) +
  labs(x = "Training Accuracy Score", y = "")

test_error_plot <- evaluation %>%
  ggplot() +
  geom_bar(aes(y = reorder(label, test_error), x = 1 - test_error), stat = "identity", fill = "#3677B4")
  coord_cartesian(xlim = c(0.705, 0.73)) +
  labs(x = "Test Accuracy Score", y = "")

auc_score_plot <- evaluation %>%
  ggplot() +
  geom_bar(aes(y = reorder(label, -auc_score), x = auc_score), stat = "identity") +
  coord_cartesian(xlim = c(0.65, 0.82)) +
  labs(x = "AUC Score", y = "")

train_error_plot
```

Evaluation of errors

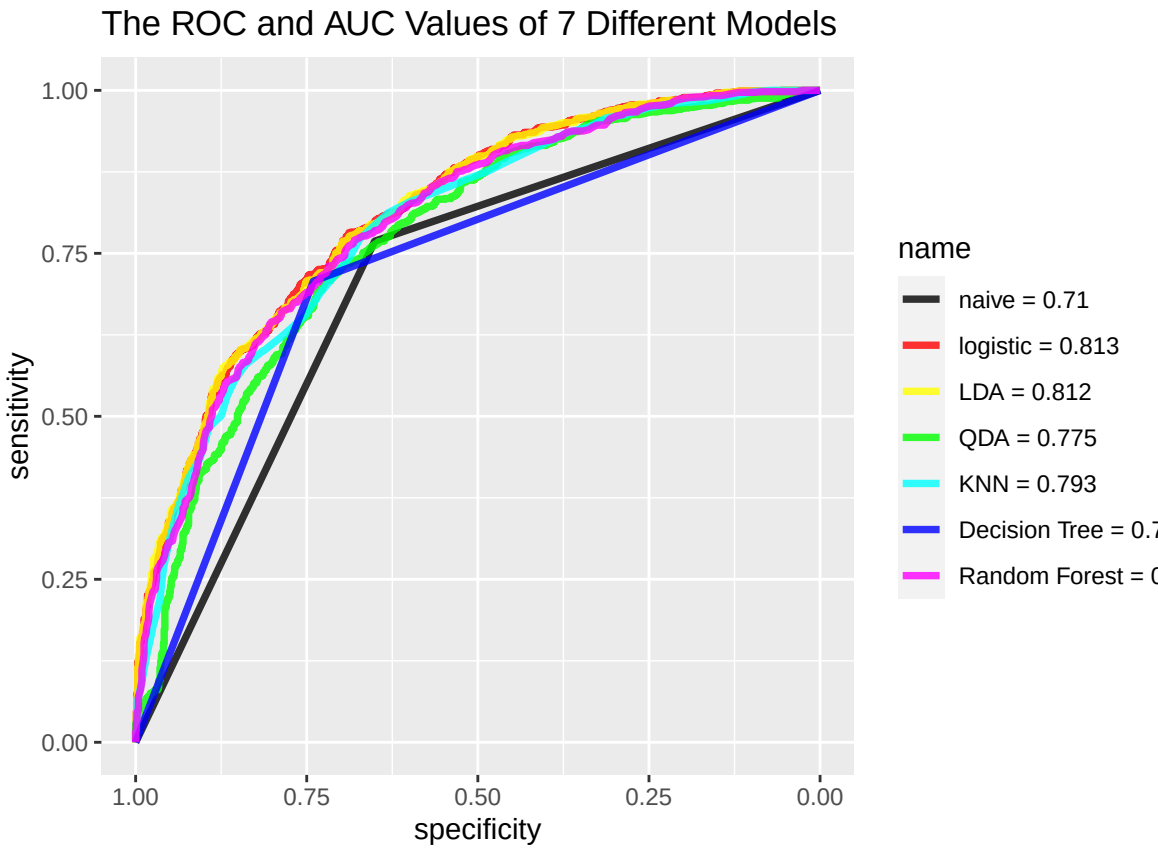
```
test_error_plot
```




```
auc_score_plot
```




```
list_roc <- list(roc_naive, roc_logistic, roc_lda, roc_qda, roc_knn, roc_tree, roc_rf)

model_type <- c("naive", "logistic", "LDA", "QDA", "KNN", "Decision Tree", "Random Forest")
model_auc <- round(c(auc(roc_naive), auc(roc_logistic), auc(roc_lda), auc(roc_qda), auc(roc_knn), auc(r
model_legend <- paste(model_type, "=", model_auc)


ggroc(list_roc, size = 1.3, alpha = 0.8) +
  scale_color_manual(labels = model_legend,
                     values = c("black", rainbow(6))) +
  ggtitle("The ROC and AUC Values of 7 Different Models")
```



Evaluation of ROC

Naive model is marked in black color for better comparison.

Combining all models *(Model assembling)*

As we have found that no models have improved the

Main idea is to combine all 6 models and judge the results from the decision of each model.

```
model_train_combine <-(prob_train_logistic[, 2] + prob_train_lda[, 2] + prob_train_qda[, 2] + prob_train
label_combine_train <- ifelse(model_train_combine >= 0.5, 1, 0)
train_error_combine <- mean(label_combine_train != data_train$blueWins)

model_test_combine <- (prob_test_logistic[, 2] + prob_test_lda[, 2] + prob_test_qda[, 2] + prob_test_kn
label_combine_test <- ifelse(model_test_combine >= 0.5, 1, 0)
test_error_combine <- mean(label_combine_test != data_test$blueWins)

roc_combine <- roc(data_test$blueWins, model_test_combine, levels = c(0, 1), direction = "auto")

train_error_combine
```

```
## [1] 0.2547134
```

```
test_error_combine
```

```
## [1] 0.2722672
```

```
roc_combine
```

```
##
## Call:
## roc.default(response = data_test$blueWins, predictor = model_test_combine,     levels = c(0, 1), dir
##
## Data: model_test_combine in 982 controls (data_test$blueWins 0) < 994 cases (data_test$blueWins 1).
## Area under the curve: 0.8092
```

Not much change in the combination of model, which means there is so other factors that are not considered in this study. blablabla.

```
data.frame(summary(model_logistic)$coefficients) %>%
  dplyr::select(Estimate, Pr...z..) %>%
  rename(coef = Estimate, pvalue = Pr...z..) %>%
  filter(pvalue < 0.01) %>%
  arrange(pvalue)
```

Analyze the logistic model (best model)

```
##                             coef       pvalue
## blueGoldDiff        0.0004170976 2.222750e-22
## blueDragons         0.3714623295 3.838848e-07
## blueExperienceDiff  0.0002218953 5.558907e-06
## redDragons         -0.2650010719 2.120126e-04
```

Consider red team and blue are interchangeable, so the most important features in winning the games are: **Gold Diff**, **Dragon Slaying** and **Experience Diff**