

PUBG Top 10% Placement Analysis

Chance Robinson, Allison Roderick and William Arnost

Master of Science in Data Science, Southern Methodist University, USA

1 Introduction

[Intro]

2 Data Description

The source data is available on Kaggle.com under the competition PUBG Finish Placement Prediction. The files used in our analysis were transformed to fit the requirements of a binomial logistic regression classifier. The data has also been pre-split into training and test files for consistency when comparing our different model types. Additionally, as the percentage-based nature of the top 10% of players is inherently unbalanced, we've also downsampled the higher frequency data to match that over the lower frequency outcome of the top 10% of players.

`pubg_solo_game_types.csv`

- Filtered for solo only game types

`pubg_solo_game_types_test_full.csv`

- Pre-split for test data

`pubg_solo_game_types_train_full.csv`

- Pre-split for train data without downsampling for the unbalanced response variable

`pubg_solo_game_types_train_downsampled.csv`

- Pre-split for train data with downsampling for the unbalanced response variable

2.1 Data Dictionary

Column Name	Description
DBNOs	Number of enemy players knocked.
assists	Number of enemy players this player damaged that were killed by teammates.
boosts	Number of boost items used.

Column Name	Description
damageDealt	Total damage dealt. Note: Self inflicted damage is subtracted.
headshotKills	Number of enemy players killed with headshots.
heals	Number of healing items used.
Id	Players Id
killPlace	Ranking in match of number of enemy players killed.
killPoints	Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.) If there is a value other than -1 in rankPoints, then any 0 in killPoints should be treated as a None.
killStreaks	Max number of enemy players killed in a short amount of time.
kills	Number of enemy players killed.
longestKill	Longest distance between player and player killed at time of death.
matchDuration	Duration of match in seconds.
matchId	ID to identify match. There are no matches that are in both the training and testing set.
matchType	String identifying the game mode that the data comes from.
rankPoints	Elo-like ranking of player.
revives	Number of times this player revived teammates.
rideDistance	Total distance traveled in vehicles measured in meters.
roadKills	Number of kills while in a vehicle.
swimDistance	Total distance traveled by swimming measured in meters.
teamKills	Number of times this player killed a teammate.
vehicleDestroys	Number of vehicles destroyed.
walkDistance	Total distance traveled on foot measured in meters.
weaponsAcquired	Number of weapons picked up.
winPoints	Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.) If there is a value other than -1 in rankPoints, then any 0 in winPoints should be treated as a None.
groupId	ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time.
numGroups	Number of groups we have data for in the match.
maxPlace	Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
winPlacePerc	This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match. (to be removed from our binomial classifier so as not to influence our predictive results)
top.10	The target of prediction. This is a percentile winning placement, where 1 corresponds to a top 10% placement a 0 in the lower 90%.

2.2 Exploratory Data Analysis

3 Objective I Analysis

3.1 Problem Statement

We would like to predict if a player finished in the top 10 of a PUBG match based on their stats during the match. Our response variable is top.10, which is 1 if the player finished in the top 10 and 0 if they did not. We will start with a logistic regression model before moving into more complex methods.

3.2 Establishing the Initial Model

During EDA, we discovered a number of variables that would not be useful for prediction, either because they were ID columns or match characteristics that were not relevant to individual player performance. Removing those, our first model contains all the remaining variables.

```
##
## Call:
## glm(formula = top.10 ~ assists + boosts + heals + teamKills +
##      weaponsAcquired + damageDealt + headshotKills + kills + killStreaks +
##      roadKills + vehicleDestroys + killPlace + killPoints + rankPoints +
##      winPoints + longestKill + swimDistance + rideDistance + walkDistance,
##      family = binomial(link = "logit"), data = train_obj1)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -3.5064  -0.2621   0.0017   0.4644   5.1751
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.335e+00  4.679e-01 -11.402  < 2e-16 ***
## assists      6.767e-01  7.031e-02   9.624  < 2e-16 ***
## boosts       3.279e-01  1.265e-02  25.930  < 2e-16 ***
## heals       -3.521e-02  6.748e-03  -5.217  1.82e-07 ***
## teamKills    -2.027e+00  2.315e-01  -8.757  < 2e-16 ***
## weaponsAcquired 4.750e-02  8.215e-03   5.782  7.36e-09 ***
## damageDealt   6.655e-05  2.758e-04   0.241  0.80935
## headshotKills 4.061e-02  3.300e-02   1.231  0.21841
## kills       -5.179e-02  3.254e-02  -1.591  0.11151
## killStreaks  -1.210e+00  5.375e-02 -22.509  < 2e-16 ***
## roadKills    -2.286e-01  1.223e-01  -1.870  0.06148 .
## vehicleDestroys -4.657e-01  1.674e-01  -2.783  0.00539 **
## killPlace    -9.337e-02  1.817e-03 -51.380  < 2e-16 ***
```

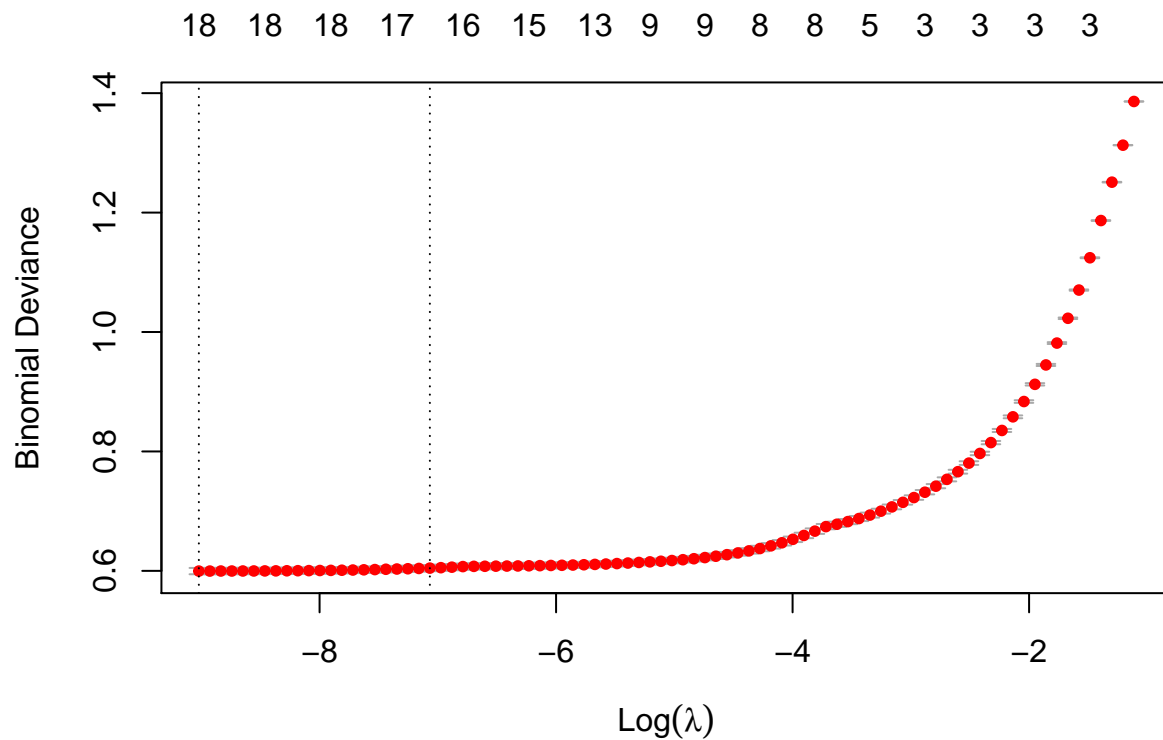
```
## killPoints      -1.578e-03  2.606e-04  -6.055  1.41e-09 ***
## rankPoints      3.995e-03  3.056e-04  13.074  < 2e-16 ***
## winPoints       5.282e-03  3.949e-04  13.376  < 2e-16 ***
## longestKill     -1.896e-03  4.007e-04  -4.732  2.22e-06 ***
## swimDistance    1.503e-03  3.705e-04   4.056  5.00e-05 ***
## rideDistance    1.316e-04  1.011e-05  13.026  < 2e-16 ***
## walkDistance    8.866e-04  2.384e-05  37.197  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 36895  on 26613  degrees of freedom
## Residual deviance: 15916  on 26594  degrees of freedom
## AIC: 15956
##
## Number of Fisher Scoring iterations: 6
```

We can see some insignificant variables, and the vif (Appendix Ref) will show some correlation. This model performance of this model is as follows:

Model	AUC	Sensitivity	Specificity	Error Rate (1 - Accuracy)
Initial	0.941	0.891	0.843	15.2%

3.3 Variable Selection

Next, we use lasso to narrow these down to the most important variables. First, we used `cv.glmnet` to search for the ideal value of `lambda` for the lasso regression (Appendix Reference). We selected the `lambda.1se` value, which will hopefully eliminate the most variables. Using that value for `lambda`, lasso produced this output:



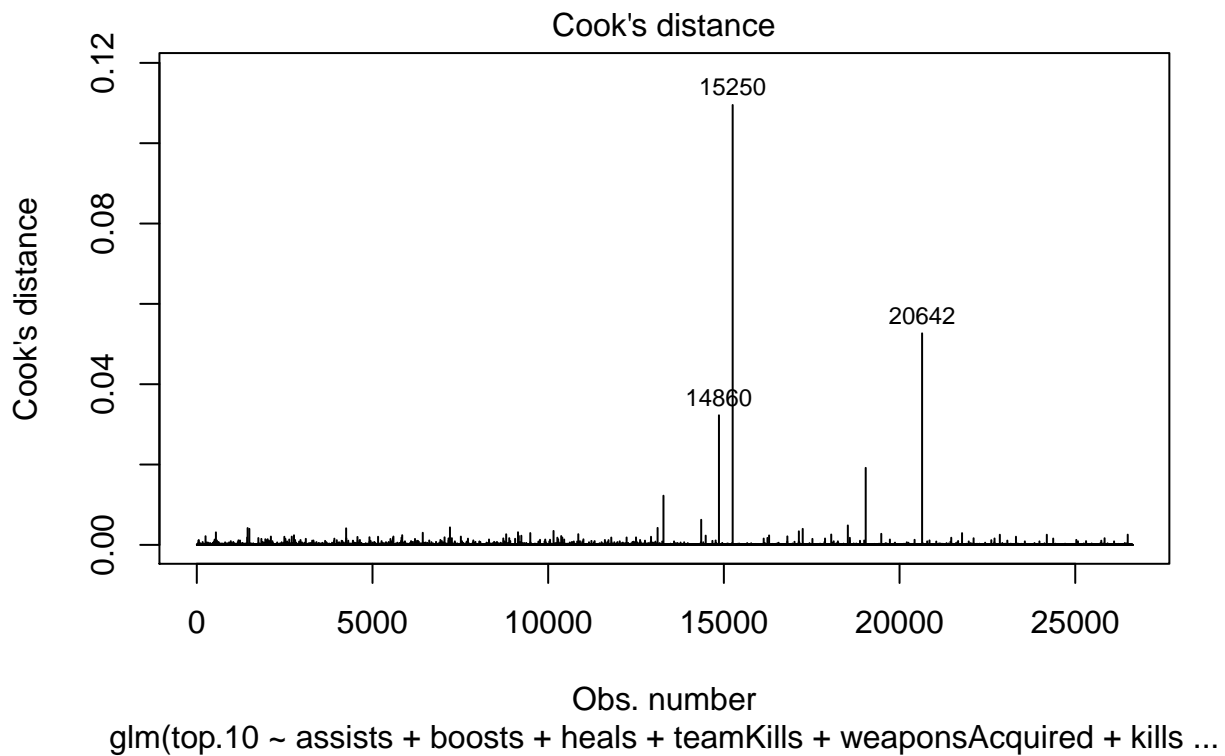
```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                               s0
## (Intercept)    -0.6733424558
## (Intercept)      .
## assists         0.6465602204
## boosts          0.3247910863
## heals          -0.0291791887
## teamKills       -1.9070440684
## weaponsAcquired 0.0445136944
## damageDealt     .
## headshotKills   .
## kills          -0.0262597549
## killStreaks     -1.1450786717
## roadKills       -0.1971398791
## vehicleDestroys -0.4170561352
## killPlace       -0.0876942534
## killPoints      .
## rankPoints      0.0008663913
## winPoints       0.0008998702
## longestKill     -0.0014955118
## swimDistance    0.0013710884
```

```
## rideDistance      0.0001344144
## walkDistance      0.0008663011
```

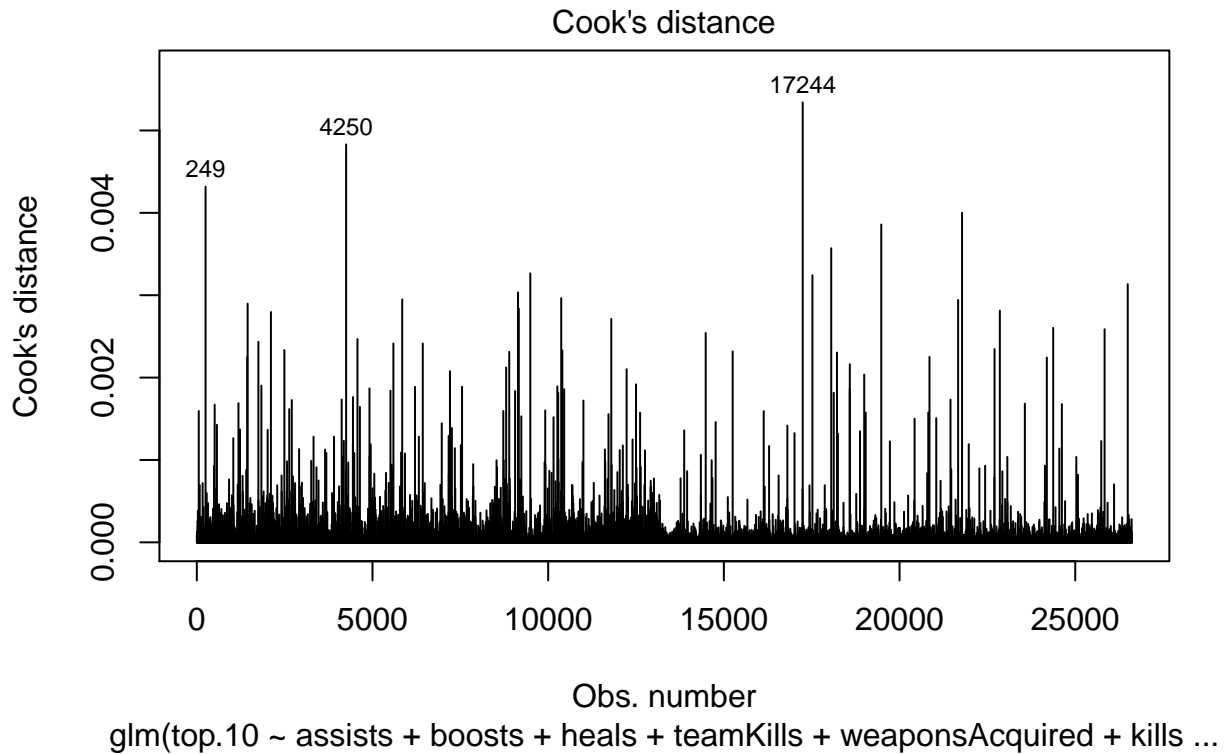
3.4 Final Iterations

The selection eliminated `damageDealt`, `headshotKills`, and `killPoints`. We created a model using just the variables from lasso (ref) and found two issues. First, `winPoints` and `rankPoints` have high VIFs. Removing them individually found that when one is missing, the other is no longer significant. We decided to remove them both. Second, the `cooksD` plot shows 3 potentially influential points. All three points have high values ($> p99$ threshold) for `roadKills` and `killStreaks`, and I also found through adding and removing variables that `swimDistance` contributes influential points. We chose to remove these variables, as a very small part of the population has high values in these metrics, and it negatively effects the model.

Here we can see influential points from the Cook's D chart.



After removing roadKills , killStreaks, and swimDistance



Final Model Coefficients

```
##
## Call:
## glm(formula = top.10 ~ assists + boosts + heals + weaponsAcquired +
##      killPlace + walkDistance, family = binomial(link = "logit"),
##      data = train_obj1)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -3.3664  -0.3759  -0.0174   0.4865   2.6092
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.934e-01  5.982e-02  -9.919  < 2e-16 ***
## assists       6.225e-01  6.666e-02   9.339  < 2e-16 ***
## boosts       3.067e-01  1.158e-02  26.493  < 2e-16 ***
## heals       -2.276e-02  6.449e-03  -3.529  0.000418 ***
## weaponsAcquired 7.900e-02  7.742e-03  10.205  < 2e-16 ***
## killPlace    -6.216e-02  1.192e-03 -52.172  < 2e-16 ***
```

```
## walkDistance      7.415e-04  2.073e-05  35.775  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 36895  on 26613  degrees of freedom
## Residual deviance: 17365  on 26607  degrees of freedom
## AIC: 17379
##
## Number of Fisher Scoring iterations: 6
```

3.5 Comparing Competing Models

Model	AUC	Sensitivity	Specificity	Error Rate (1 - Accuracy)
Initial	0.941	0.891	0.843	15.2%
Lasso	0.941	0.892	0.843	15.2%
Inf. Points	0.936	0.892	0.842	15.3%
Final	0.932	0.863	0.841	15.6%

3.6 Model Interpretation

We prefer the final model because it simplifies the previous versions without significant loss in accuracy. The variable interpretations are found in the table below.

Variable	Interpretation
assists	The log-odds of placing in the top 10 increase by 0.6225 for each additional assist
boosts	The log-odds of placing in the top 10 increase by 0.3067 for each additional boost
heals	The log-odds of placing in the top 10 decrease by -0.0228 for each additional heal
weaponsAcquired	The log-odds of placing in the top 10 increase by 0.0790 for each additional weapon acquired
killPlace	The log-odds of placing in the top 10 increase by 0.0622 for each additional rank in kills (lower is better for killPlace)

Variable	Interpretation
walkDistance	The log-odds of placing in the top 10 increase by 0.0007 for each additional meter walked

3.7 Conclusion

Players who do well in kill placement and who travel far tend to win games. Travel distance probably is a proxy for time alive, as players eliminated early would probably not travel far. Acquiring in game items like boosts or weapons also help. The negative coefficient on heals is odd, perhaps indicating that taking more damage is detrimental to placement. We will try to improve on the final model using more complex methods.

4 Objective II Analysis

4.1 Question of Interest

As we stated in the Problem Statement, we would like to predict if a player finished in the top 10 of a PUBG match based on their stats during the match. The next three methods of prediction we will present are Logistic Regression using more complicated terms than what were presented in Objective 1, Linear Discriminant Analysis (LDA), and Random Forest. These different techniques are all going to help us answer the question: What model is has the highest predictive accuracy for classifying players as top 10 finishers?

4.2 Model Selection

4.2.1 Linear Discriminant Analysis

Our next prediction tool is Linear Discriminant Analysis (LDA) for classifying the player as Top 10 or not. We have taken a subset of the continuous variables from our EDA to build the LDA off of. Before running the LDA, we will cover two things: a LASSO call to eliminate less important variables and assumption checking.

4.2.1.1 LASSO

The LASSO call plus manual variable selection reduced the predictors considered for the LDA model to: boosts, heals, killPlace, killStreaks, longestKill, matchDuration, rideDistance, swimDistance, teamKills, walkDistance, weaponsAcquired. See the Appendix - LDA for the output of the LASSO call.

4.2.1.2 Assumption Checking

LDA performs optimally when the assumptions of MANOVA are met. That is,

1. The predictors are normally distributed for each class of the response.
2. The covariance matrices for each class of the response are homogeneous.

When we check the first assumption for the predictors that are to be included in the LDA model, we see that the assumption is not met, as shown in the histograms of the variables as shown in the Appendix - LDA. Most of the predictors are right skewed. The variable matchDuration is bimodal. To remedy this, we tried transforming the variables, but it did not help our overall prediction accuracy. However, because issues of normality exist, we will explore QDA as well as LDA.

We also checked the homogeneity of correlation matrices, shown in the Appendix - LDA. We find that, overall, there are no major departures from homogeneity. The variables walkDistance and killPlace show the greatest deviances from homogenous correlations between bottom 90 and top 10 placements. Consequently, we tried removing those variables from the model. However, removing those variables reduced our prediction accuracy.

Thus, we will proceed with the variables selected and see if LDA or QDA performs better.

4.2.1.3 LDA Results

LDA has a prediction accuracy of 0.9206, with a sensitivity of 0.57479 and a specificity of 0.96099. The area under the ROC curve is 0.941. Output of the LDA call and the ROC curve are shown in Appendix - LDA. The confusion matrix is shown below for comparison with QDA.

```
## Confusion Matrix and Statistics
##
##
##      0      1
## 0 46972 2425
## 1  1907 3278
##
##              Accuracy : 0.9206
##              95% CI : (0.9183, 0.9229)
##      No Information Rate : 0.8955
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.5582
##
##  Mcnemar's Test P-Value : 3.998e-15
##
##              Sensitivity : 0.57479
##              Specificity : 0.96099
##              Pos Pred Value : 0.63221
##              Neg Pred Value : 0.95091
##              Prevalence : 0.10448
##              Detection Rate : 0.06006
##      Detection Prevalence : 0.09499
##      Balanced Accuracy : 0.76789
##
##      'Positive' Class : 1
##
```

4.2.1.4 QDA Results

QDA has a prediction accuracy of 0.8779, with a sensitivity of 0.68262 and a specificity of 0.90071. The area under the ROC curve is 0.912. Output of the QDA call and the ROC curve are shown in Appendix - LDA. The confusion matrix is shown below for comparison with LDA.

```
## Confusion Matrix and Statistics
##
##
##      0      1
```

```

## 0 46972 2425
## 1 1907 3278
##
##          Accuracy : 0.9206
##          95% CI : (0.9183, 0.9229)
##    No Information Rate : 0.8955
##    P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.5582
##
## Mcnemar's Test P-Value : 3.998e-15
##
##          Sensitivity : 0.57479
##          Specificity : 0.96099
##          Pos Pred Value : 0.63221
##          Neg Pred Value : 0.95091
##          Prevalence : 0.10448
##          Detection Rate : 0.06006
##    Detection Prevalence : 0.09499
##          Balanced Accuracy : 0.76789
##
##          'Positive' Class : 1
##

```

4.2.1.5 LDA Conclusion

Although the QDA is better at predicting the top 10 placements that were true top 10 than LDA (QDA sensitivity of 0.68 > LDA sensitivity of 0.57), QDA predicts many more incorrect top 10 placements than LDA does (1907 LDA false positives < 4853 QDA false positives). Because LDA has a better overall accuracy (0.9206 for LDA > 0.8779 for QDA), we think the LDA is a stronger model for prediction than QDA.

4.2.2 Random Forest

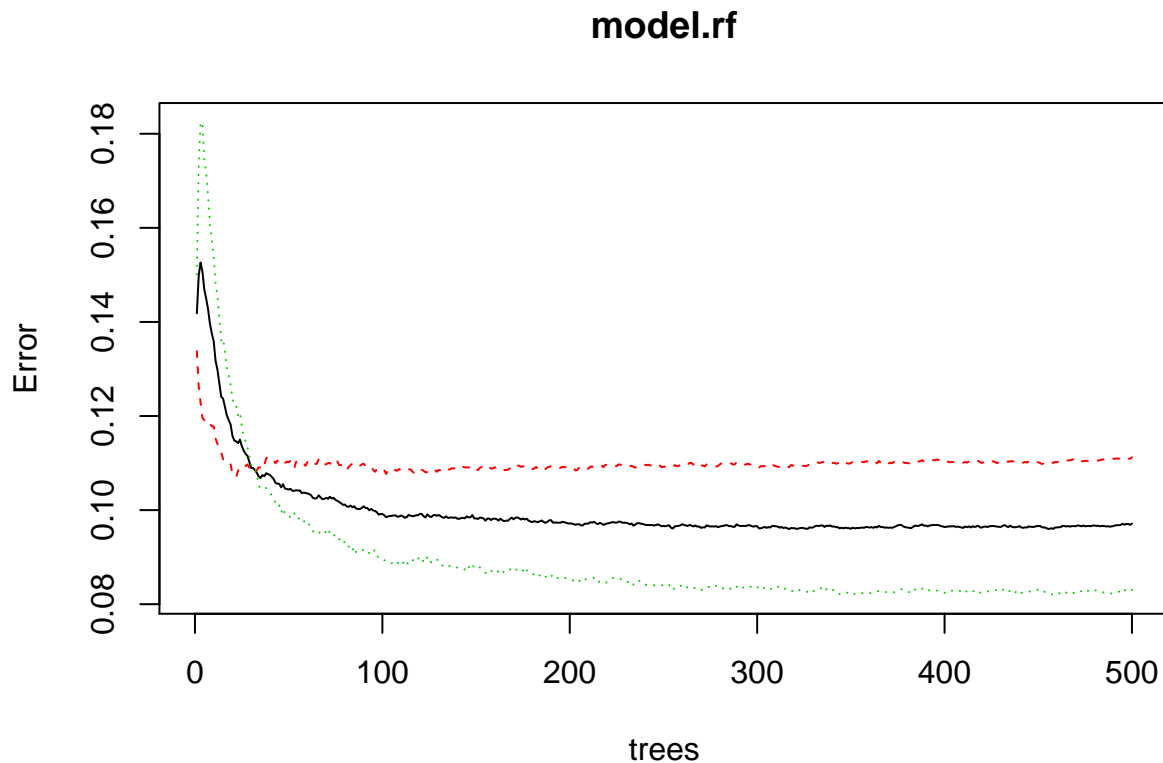
We then tried our first non-parametrical model with Random Forest, which averages out the results of many Decision Trees to provide the lowest error rates across all of the permutations attempted.

4.2.2.1 Assumption Checking

Random Forest methods do not have the same level of assumption restrictions as many of the parametrical models we reviewed. Fortunately, our data set had no null values to speak of which may have posed more of a problem.

4.2.2.2 Error Rates by No. of Trees

The model could likely be tuned further as you can see there's no additional payoff from having the `ntree` value set so high. In fact, it could be harming the performance by setting it to such a high value as you can see the error rates starting to trend upwards at the far-right edge of the graph.



4.2.2.3 Variable Importance

Similar features were confirmed as having the best predictive outcomes from the variable importance output. (i.e. . . Kill Place, Walking Distance, etc..) edge of the graph.

Column.Name	Overall.Score
assists	43.62334
boosts	1157.86767
damageDealt	587.86497
headshotKills	61.26146
heals	209.00441
killPlace	5172.69584
killPoints	155.16722
kills	635.76974
killStreaks	159.01334
longestKill	386.08015
matchDuration	105.38803
maxPlace	326.37764
numGroups	367.23582
rankPoints	289.94210
rideDistance	270.77805
roadKills	13.37229
swimDistance	96.74467
teamKills	22.79068
vehicleDestroys	13.58180
walkDistance	2802.62385
weaponsAcquired	261.50104
winPoints	155.96105

4.2.2.4 Random Forest Results

The out-of-bag estimate of the error rate was 9.71% which was consistent with our finding from classifying the test data not used to train the model with. This should allow us to better gauge what real-world performance would be like with new data to apply the model to.

When running our model against the hold out test data set, we received the following performance metrics.

Model	AUC	Sensitivity	Specificity	Error Rate (1 - Accuracy)
RF	0.906	0.918	0.894	10.4%

4.2.2.5 Random Forest Conclusion

The Random Forest approach required that the model be tuned with parameters better suited to the provided data set.

The following arguments were important to address bias/ variance issues that were observed when using the default settings. Trial and error techniques along with tuning libraries in `caret` and native to the `randomForest` library allowed us to better optimize the results. We were able to get a roughly 90/90 split between our sensitivity and specificity performance measures. Or rather, when it was in the top 10%, the model was able to accurately predict that it was roughly 90% of the time. The same could be said for the specificity metric.

From the library documentation:

- `ntree` : Number of trees to grow
- `mtry` : Number of variables randomly sampled as candidates at each split.
- `cutoff` : A vector of length equal to number of classes.

4.3 Comparing Competing Models

4.4 Model Interpretation

4.5 Conclusion

5 Appendix

5.1 Exploratory Data Analysis

5.2 Code

5.2.1 LDA

5.2.1.1 LASSO

```
# Reduce variables to relevant continuous variables
```

```
train1<-train[,c(5:6,8:10,12:15,19,21:27,29:30)]
```

```
test1<-test[,c(5:6,8:10,12:15,19,21:27,29:30)]
```

```
# train2 <- train1[,c(1,4:19)]
```

```
# test2 <- test1[,c(1,4:19)]
```

```
train2_alt <- train1[,c(1,4:17,19)]
```

```
test2_alt <- test1[,c(1,4:17,19)]
```

```
# Get data in format for LASSO
```

```
x=model.matrix(top.10~.,train2_alt)[,-1]
```

```
y=as.numeric(train2_alt$top.10)
```

```
xtest<-model.matrix(top.10~.,test2_alt)[,-1]
```

```
ytest<-as.numeric(test2_alt$top.10)
```

```
grid=10^seq(10,-2, length =100)
```

```
lasso.mod=glmnet(x,y,alpha=1, lambda =grid)
```

```
set.seed(23) #removes kills roadKills vehicleDestroys
```

```
lda.cv.out=cv.glmnet(x,y,alpha=1,family="binomial") #alpha=1 performs LASSO
```

```
# plot(lda.cv.out)
```

```
# simplest model
```

```
lda.lasso.model.coef<-coef(cv.out, cv.out$lambda.1se)
```

```
# lda.lasso.model.coef
```

```
lda.lasso.model.coef
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
```

```
##              1
```

```
## (Intercept)  3.091783e+00
```

```
## boosts      2.672920e-01
```

```
## heals       -2.784586e-03
```

```
## killPlace   -8.033586e-02
```



```
## kills      .
## killStreaks -1.117635e+00
## longestKill -5.733176e-05
## matchDuration -1.609689e-03
## rankPoints 1.034136e-05
## rideDistance 1.948066e-04
## roadKills .
## swimDistance 5.967117e-04
## teamKills -1.509184e+00
## vehicleDestroys .
## walkDistance 8.511812e-04
## weaponsAcquired 5.746021e-02
```

```
# Removing the kills, roadKills, and vehicleDestroys as shown above
# Also removing rankPoints because
# (a) the majority of the players in these data aren't ranked and
# (b) the Kaggle description says "This ranking is inconsistent and is being deprecated in the API's new
train_final <- train2_alt[,c(-4,-8,-10,-13)]
```

5.2.1.2 Assumption Checking

```
train_final_no <- train_final[which(train_final$top.10==0),]
train_final_yes <- train_final[which(train_final$top.10==1),]

# nrow(train_final)
# nrow(train_final_no)
# nrow(train_final_yes)

max_cols<-ncol(train_final)
no_matrix<-as.matrix(train_final_no[, -max_cols])
yes_matrix<-as.matrix(train_final_yes[, -max_cols])

no_hist<-multi.hist(no_matrix)
yes_hist<-multi.hist(yes_matrix)
# par(mfrow=c(1,1))
```

```
no_hist
```

```
## NULL
```

```
yes_hist
```

```
## NULL
```

<http://www.sthda.com/english/wiki/ggplot2-quick-correlation-matrix-heatmap-r-software-and-data-visuali>

```
get_upper_tri <- function(cormat){
  cormat[lower.tri(cormat)]<- NA
  return(cormat)
}

custom_corr_plot <- function(cormat){

  upper_tri <- get_upper_tri(cormat)
  # Melt the correlation matrix
  melted_cormat <- melt(upper_tri, na.rm = TRUE)
  # Create a ggheatmap
  ggheatmap <- ggplot(melted_cormat, aes(Var2, Var1, fill = value))+
    geom_tile(color = "white")+
    scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                        midpoint = 0, limit = c(-1,1), space = "Lab",
                        name="Pearson\nCorrelation") +
    theme_minimal()+ # minimal theme
    theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                      size = 12, hjust = 1))+
    coord_fixed()

  p<-ggheatmap +
    geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
    theme(
      axis.title.x = element_blank(),
      axis.title.y = element_blank(),
      panel.grid.major = element_blank(),
      panel.border = element_blank(),
      panel.background = element_blank(),
      axis.ticks = element_blank(),
      legend.justification = c(1, 0),
      legend.position = c(0.6, 0.7),
      legend.direction = "horizontal")+
    guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                                title.position = "top", title.hjust = 0.5))

  return(p)
}

cormat <- round(cor(no_matrix),2)
lda.no <- custom_corr_plot(cormat)
```

```
cormat <- round(cor(yes_matrix),2)
lda.yes <- custom_corr_plot(cormat)
```

```
lda.no
```

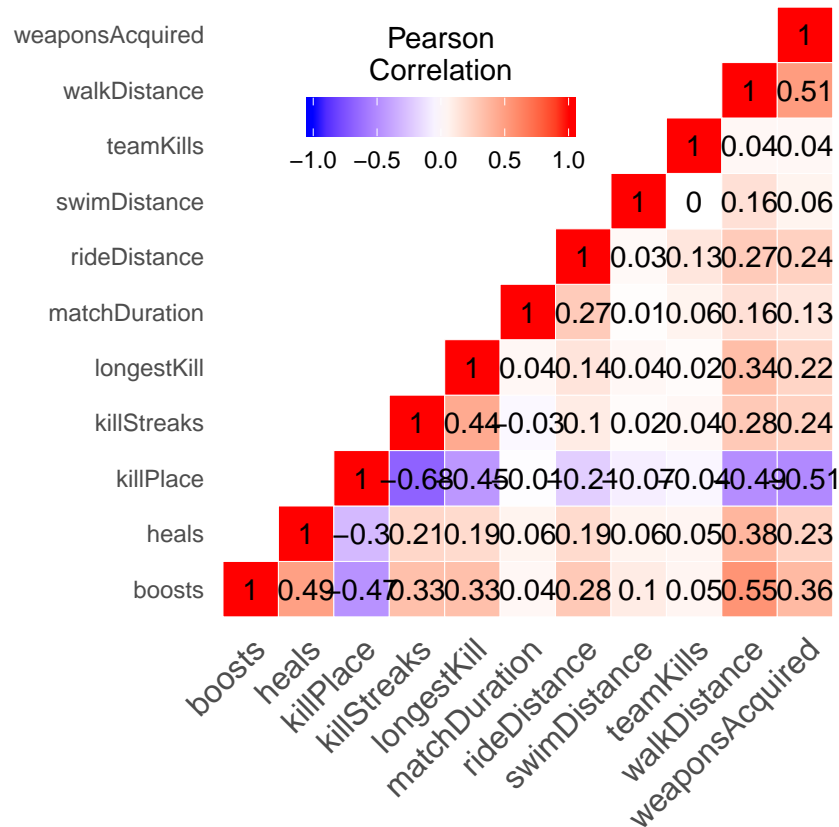


Figure 1: (#fig:code.lda.no)Correlation Matrix for Bottom 90 of Training Dataset

```
lda.yes
```

5.2.1.3 LDA Results

```
# Run LDA
lda <- lda(top.10 ~ . , data=train_final, prior=c(.9,.1))

lda
```

```
## Call:
## lda(top.10 ~ . , data = train_final, prior = c(0.9, 0.1))
```

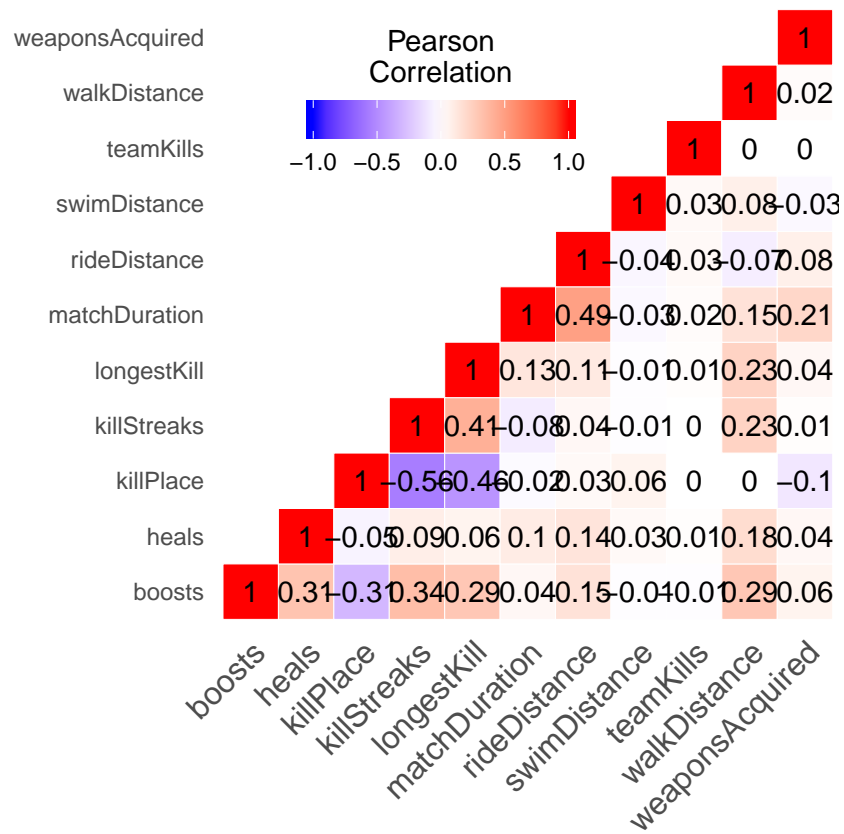


Figure 2: (#fig:code.lda.yes)Correlation Matrix for Top 90 of Training Dataset

```
##
## Prior probabilities of groups:
## 0 1
## 0.9 0.1
##
## Group means:
##      boosts      heals killPlace killStreaks longestKill matchDuration
## 0 0.7381078 0.8007815 50.25445 0.3915984 15.00272 1681.215
## 1 3.9244758 2.7710228 13.42301 0.8962200 70.77070 1680.553
##      rideDistance swimDistance teamKills walkDistance weaponsAcquired
## 0 515.6142 5.365372 0.016908394 823.7333 3.517848
## 1 1666.6767 16.136750 0.003005937 2351.6425 5.710528
##
## Coefficients of linear discriminants:
##                      LD1
## boosts      0.1636559657
## heals      -0.0071779101
## killPlace   -0.0313509817
## killStreaks -0.4645084746
## longestKill 0.0003963294
## matchDuration -0.0007443145
## rideDistance 0.0001033605
## swimDistance 0.0006812726
## teamKills    -1.0147862026
## walkDistance 0.0004217550
## weaponsAcquired 0.0155413057
```

```
predict <- predict(lda,newdata=test)

test_final <- test
test_final$predcited_place <- as.vector(predict$class)
test_final$top.10<-as.factor(test_final$top.10)
test_final$predcited_place<-as.factor(test_final$predcited_place)

xtab<-table(test_final$predcited_place,test_final$top.10)
confusionMatrix(xtab, positive="1")
```

```
## Confusion Matrix and Statistics
##
##
##      0      1
## 0 46972 2425
## 1 1907 3278
```

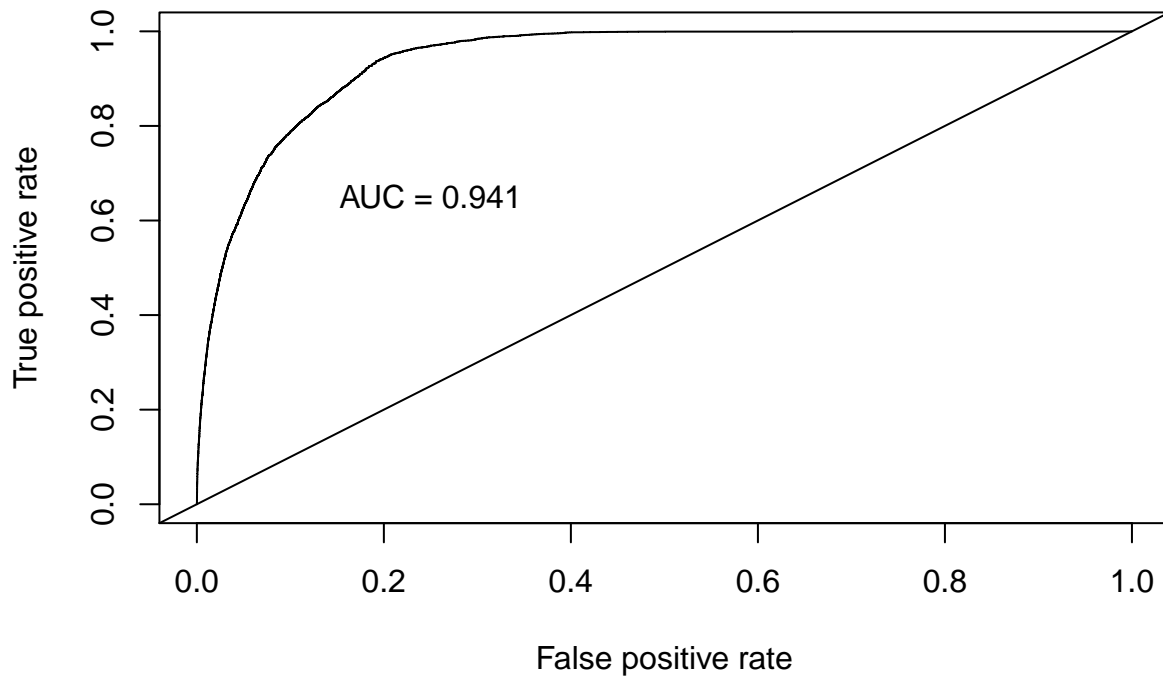
```
##
##           Accuracy : 0.9206
##           95% CI : (0.9183, 0.9229)
##      No Information Rate : 0.8955
##      P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.5582
##
##      McNemar's Test P-Value : 3.998e-15
##
##           Sensitivity : 0.57479
##           Specificity : 0.96099
##      Pos Pred Value : 0.63221
##      Neg Pred Value : 0.95091
##           Prevalence : 0.10448
##      Detection Rate : 0.06006
##      Detection Prevalence : 0.09499
##      Balanced Accuracy : 0.76789
##
##      'Positive' Class : 1
##
```

```
predict.posterior <- as.data.frame(predict$posterior)

# Evaluate the model
pred <- prediction(predict.posterior[,2], test$top.10)
roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
auc.train <- performance(pred, measure = "auc")
auc.train <- auc.train@y.values

# Plot
plot(roc.perf, main="ROC Curve - LDA")
abline(a=0, b= 1)
text(x = .25, y = .65 ,paste("AUC = ", round(auc.train[[1]],3), sep = ""))
```

ROC Curve – LDA



5.2.1.4 QDA Results

```
qda <- qda(top.10 ~ . , data=train_final, prior=c(.9,.1))
```

```
qda
```

```
## Call:
## qda(top.10 ~ . , data = train_final, prior = c(0.9, 0.1))
##
## Prior probabilities of groups:
##  0  1
## 0.9 0.1
##
## Group means:
##      boosts      heals killPlace killStreaks longestKill matchDuration
## 0 0.7381078 0.8007815 50.25445 0.3915984 15.00272 1681.215
## 1 3.9244758 2.7710228 13.42301 0.8962200 70.77070 1680.553
##  rideDistance swimDistance  teamKills walkDistance weaponsAcquired
## 0      515.6142      5.365372 0.016908394      823.7333      3.517848
## 1     1666.6767     16.136750 0.003005937     2351.6425     5.710528
```

```

predict.qda <- predict(qda,newdata=test)

# test_final <- test
test_final$predcited_place.qda <- as.vector(predict$class)
# test_final$top.10<-as.factor(test_final$top.10)
test_final$predcited_place.qda<-as.factor(test_final$predcited_place.qda)
# test_final$predcited_place<-as.factor(test_final$predcited_place)
# str(test_final)

xtab.qda<-table(test_final$predcited_place.qda,test_final$top.10)
confusionMatrix(xtab.qda, positive="1")

```

```

## Confusion Matrix and Statistics
##
##
##           0      1
##  0 46972  2425
##  1  1907  3278
##
##               Accuracy : 0.9206
##               95% CI : (0.9183, 0.9229)
##      No Information Rate : 0.8955
##      P-Value [Acc > NIR] : < 2.2e-16
##
##               Kappa : 0.5582
##
##  Mcnemar's Test P-Value : 3.998e-15
##
##               Sensitivity : 0.57479
##               Specificity : 0.96099
##      Pos Pred Value : 0.63221
##      Neg Pred Value : 0.95091
##      Prevalence : 0.10448
##      Detection Rate : 0.06006
##      Detection Prevalence : 0.09499
##      Balanced Accuracy : 0.76789
##
##      'Positive' Class : 1
##

```

```

predict.posterior.qda <- as.data.frame(predict.qda$posterior)

# Evaluate the model

```

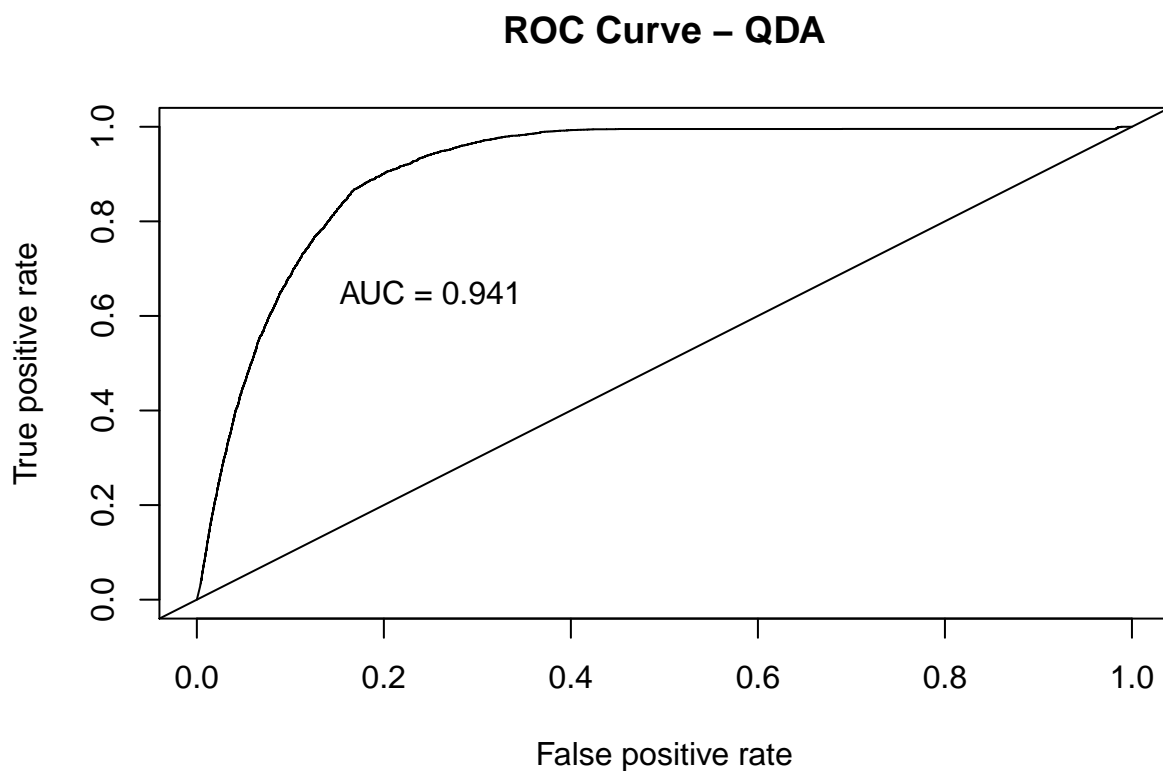


```

pred.qda <- prediction(predict.posterior.qda[,2], test$top.10)
roc.perf.qda = performance(pred.qda, measure = "tpr", x.measure = "fpr")
auc.train.qda <- performance(pred.qda, measure = "auc")
auc.train.qda <- auc.train.qda@y.values

# Plot
plot(roc.perf.qda, main="ROC Curve - QDA")
abline(a=0, b= 1)
text(x = .25, y = .65 ,paste("AUC = ", round(auc.train[[1]],3), sep = ""))

```



5.2.2 Random Forest

5.2.2.1 Model

```

# results='hide', message=FALSE, include=FALSE, echo=FALSE
rf_cols_to_remove = c("Id", "groupId", "matchId", "matchType", "DBNOs", "revives", "winPlacePerc")

train.rf <- train %>%
  dplyr::select(-rf_cols_to_remove) %>%

```

```

mutate(kills = kills * 100 / numGroups) %>% # Normalized Kills
mutate(matchDuration = as.factor(ifelse(matchDuration < mean(matchDuration), "Low", "High"))) %>%
mutate(top.10 = factor(top.10, labels = c("No", "Yes")))

test.rf <- test %>%
  dplyr::select(-rf_cols_to_remove) %>%
  mutate(kills = kills * 100 / numGroups) %>% # Normalized Kills
  mutate(matchDuration = as.factor(ifelse(matchDuration < mean(matchDuration), "Low", "High"))) %>%
  mutate(top.10 = factor(top.10, labels = c("No", "Yes")))

set.seed(1234)
model.rf <- randomForest(as.factor(top.10) ~ .,
                        data = train.rf,
                        ntree = 500,
                        mtry = 12,
                        cutoff = c(0.4, 1-0.40))

model.rf

```

5.2.2.2 Tuning Plot

```
plot(model.rf)
```

5.2.2.3 Variable Importance

```

# varImp(model.rf)

rf.imp.variables <- data.frame(varImp(model.rf))

# rf.imp.variables[order(rf.imp.variables$Overall, decreasing = T),]

kable(data.frame('Column Name' = rownames(rf.imp.variables),
                  'Overall Score' = rf.imp.variables$Overall,
                  "latex", booktabs = T) %>%
  kable_styling(position = "center")

```

5.2.2.4 Predict

```
prd.rf.train <- predict(model.rf, train.rf)
prd.rf.test <- predict(model.rf, test.rf)
```

5.2.2.5 Confusion Matrix

```
confusionMatrix(data=prd.rf.test,
                 reference=test.rf$top.10, "Yes")
```

5.2.2.6 ROC Curve

```
plotRoc <- function(preds, truth) {
  pred <- prediction(preds, truth)
  roc.perf = performance(pred, measure = "tpr", x.measure = "fpr")
  auc.train <- performance(pred, measure = "auc")
  auc.train <- auc.train@y.values
  #Plot ROC
  par(mar=c(4,4,4,4))
  plot(roc.perf,main="Random Forest")
  abline(a=0, b= 1) #Ref line indicating poor performance
  text(x = .40, y = .6,paste("AUC = ", round(auc.train[[1]],3), sep = ""))
  table(test$top.10, useNA = "ifany")
}

plotRoc(as.integer(prd.rf.test),as.integer(test.rf$top.10))

# ?randomForest
```