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	Type	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.
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

Column Name	Type	Description
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 classfier 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 Question of Interest

3.2 Model Selection

[Logistic Regression, Ridge, Lasso and Elastic Net]

3.3 Comparing Competing Models

1) AUC
2) Sen
3) Spec
4) Err Rate (1 - Accuracy)

3.4 Model Interpretation

3.5 Conclusion

4 Objective II Analysis

4.1 Question of Interest

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.

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. 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. 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.

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.

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 permuations 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. Fotunately, our data set had no null values to speak of which may have posed more of a problem.

4.2.2.2 Random Forest Results

The out-of-bag estimate of the error rate was 9.71% which seems to be consistent with our results against the test data not used to train the model with.

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

- Accuracy : 0.8964

- Sensitivity : 0.91811
- Specificity : 0.89384

- AUC : 0.90598

4.2.2.3 Random Forest Conclusion

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

The following arguments were important to address bias/ variance issues that we observed when using the default settings. Trial and error 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. Or rather, when it was in the top 10%, how often did we predict that it was and when it was in the bottom 90%, how often did we predict that it was respectively

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

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

```
# 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 descrption says "This ranking is inconsistent and is being deprecated in the API's nex
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),]
```

```r
# 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))
```

```r
#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)
no <- custom_corr_plot(cormat)

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

5.2.1.3 LDA Results

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

lda

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")

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

# Evaluate the model
pred <- prediction(predict.posteriors[,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
```

```r
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 = ""))
```

5.2.1.4 QDA Results

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

qda

predict <- predict(qda,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)
# test_final$predcited_place<-as.factor(test_final$predcited_place)
str(test_final)

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

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

# Evaluate the model
pred <- prediction(predict.posteriors[,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 - QDA")
abline(a=0, b= 1)
text(x = .25, y = .65 ,paste("AUC = ", round(auc.train[[1]],3), sep = ""))
```