Searching for Similarity: Classification

Isabelle Kirby, Bridgette Bryant, Rikita Patangay, Zuhayr Ali

Classification for Korea's top high elo teams in League of Legends

I utilized logistic regression and bayes to predict if the League of Legends team won or lost based on their stats for kills, deaths, gold, and vision scores. Our dataset has all these values in two datasets lose_team_stats.csv and win_team_stats.csv. Each game also has a gameId which is unique to that specific game. This ID is identical in each dataset. The win_team_stats.csv file contains the winning teams kills, deaths, gold earned, damage dealt, crowd control, and vision scores for each player. The lose_team_stats is the same but for the losing teams. These datasets are great because they contain nearly 100k games and have no NA/Null values.

Cleaning Data

First we have to unzip the archive (they are large data sets nearly 100k games). Then we will save the

```
win_stats_df <- read.csv((unz("League_Data/league_korea_high_elo_team_stats.zip", "win_team_stats.csv"))
lose_stats_df <- read.csv((unz("League_Data/league_korea_high_elo_team_stats.zip", "lose_team_stats.csv

str(win_stats_df)
```

```
## 'data.frame':    90500 obs. of  31 variables:
##  $ win_kills1                      : num  10 3 7 11 0 7 9 8 18 4 ...
##  $ win_kills2                      : num  4 7 3 4 4 1 10 10 5 2 ...
##  $ win_kills3                      : num  4 0 5 7 2 11 9 5 2 2 ...
##  $ win_kills4                      : num  6 7 9 2 11 10 9 10 2 6 ...
##  $ win_kills5                      : num  7 2 4 3 8 8 2 0 11 4 ...
##  $ win_deaths1                     : num  4 5 5 2 1 3 2 5 2 3 ...
##  $ win_deaths2                     : num  1 0 1 2 4 0 2 5 6 0 ...
##  $ win_deaths3                     : num  4 0 2 2 4 3 3 4 8 1 ...
##  $ win_deaths4                     : num  4 0 1 3 3 6 1 7 4 0 ...
##  $ win_deaths5                     : num  2 3 2 4 4 5 5 6 8 1 ...
##  $ win_totalDamageDealtToChampions1: num  17898 16662 16241 12111 10900 ...
##  $ win_totalDamageDealtToChampions2: num  15800 11674 7572 5510 11362 ...
##  $ win_totalDamageDealtToChampions3: num  10786 7498 10024 8440 14494 ...
##  $ win_totalDamageDealtToChampions4: num  16964 13016 15115 5373 27391 ...
##  $ win_totalDamageDealtToChampions5: num  11568 11393 12395 11038 22399 ...
##  $ win_goldEarned1                 : num  9802 8452 9029 10175 7217 ...
##  $ win_goldEarned2                 : num  9203 9069 6921 5552 10497 ...
##  $ win_goldEarned3                 : num  11127 6023 8331 7439 10323 ...
##  $ win_goldEarned4                 : num  9286 9868 11860 5873 13499 ...
##  $ win_goldEarned5                 : num  10414 7660 8589 7033 12720 ...
##  $ win_visionScore1                : num  28 14 11 17 42 41 19 23 72 9 ...
##  $ win_visionScore2                : num  16 27 35 25 38 41 46 55 50 21 ...
##  $ win_visionScore3                : num  23 46 25 17 30 21 25 47 29 13 ...
##  $ win_visionScore4                : num  17 16 15 9 18 39 31 25 80 19 ...
##  $ win_visionScore5                : num  36 22 21 19 26 19 86 70 22 10 ...
##  $ win_totalTimeCrowdControlDealt1 : num  183 33 178 134 69 310 168 279 365 15 ...
```

```
##  $ win_totalTimeCrowdControlDealt2 : num  92 291 82 61 503 45 291 493 119 62 ...
##  $ win_totalTimeCrowdControlDealt3 : num  231 31 371 332 562 133 102 287 456 126 ...
##  $ win_totalTimeCrowdControlDealt4 : num  54 235 140 274 79 78 444 501 215 209 ...
##  $ win_totalTimeCrowdControlDealt5 : num  281 407 122 163 69 73 92 193 300 168 ...
##  $ gameId                          : num  4.25e+09 4.25e+09 4.26e+09 4.26e+09 4.26e+09 ...
```

```
str(lose_stats_df)
```

```
## 'data.frame':    90500 obs. of  31 variables:
##  $ lose_kills1                     : num  3 0 3 1 4 7 0 11 3 0 ...
##  $ lose_kills2                     : num  0 2 5 6 2 1 3 5 10 1 ...
##  $ lose_kills3                     : num  4 3 1 2 4 4 4 3 7 2 ...
##  $ lose_kills4                     : num  4 3 1 1 1 5 2 7 2 2 ...
##  $ lose_kills5                     : num  4 0 1 3 5 0 4 1 6 0 ...
##  $ lose_deaths1                    : num  6 4 7 6 7 7 10 10 7 5 ...
##  $ lose_deaths2                    : num  6 6 4 3 6 9 5 4 6 2 ...
##  $ lose_deaths3                    : num  5 3 7 7 1 11 8 6 10 2 ...
##  $ lose_deaths4                    : num  7 4 5 4 7 4 9 7 10 3 ...
##  $ lose_deaths5                    : num  7 2 5 7 4 6 7 6 5 6 ...
##  $ lose_totalDamageDealtToChampions1: num  10844 4618 7096 9492 9686 ...
##  $ lose_totalDamageDealtToChampions2: num  7095 14837 17030 8557 14045 ...
##  $ lose_totalDamageDealtToChampions3: num  13458 9197 8735 6679 24086 ...
##  $ lose_totalDamageDealtToChampions4: num  9670 10035 7849 4058 2959 ...
##  $ lose_totalDamageDealtToChampions5: num  14972 5531 5815 6912 16719 ...
##  $ lose_goldEarned1                : num  6844 4524 6551 5442 7928 ...
##  $ lose_goldEarned2                : num  5205 8823 7562 7846 8042 ...
##  $ lose_goldEarned3                : num  8226 7788 5346 5092 12502 ...
##  $ lose_goldEarned4                : num  7911 9008 5004 3931 6185 ...
##  $ lose_goldEarned5                : num  8815 6993 5931 5071 10729 ...
##  $ lose_visionScore1               : num  14 30 14 1 25 11 17 46 30 13 ...
##  $ lose_visionScore2               : num  34 16 13 27 10 48 30 34 25 7 ...
##  $ lose_visionScore3               : num  8 27 40 9 29 14 38 19 39 13 ...
##  $ lose_visionScore4               : num  21 28 15 26 65 15 81 44 84 24 ...
##  $ lose_visionScore5               : num  14 10 9 5 28 13 13 68 45 8 ...
##  $ lose_totalTimeCrowdControlDealt1 : num  19 80 133 71 422 188 97 71 246 20 ...
##  $ lose_totalTimeCrowdControlDealt2 : num  38 67 249 476 151 77 36 327 98 102 ...
##  $ lose_totalTimeCrowdControlDealt3 : num  173 491 135 13 161 109 347 433 36 169 ...
##  $ lose_totalTimeCrowdControlDealt4 : num  305 50 125 52 63 204 208 44 95 47 ...
##  $ lose_totalTimeCrowdControlDealt5 : num  237 0 226 88 97 101 81 242 447 182 ...
##  $ gameId                          : num  4.25e+09 4.25e+09 4.26e+09 4.26e+09 4.26e+09 ...
```

Now we will merge the two data sets together based on their matching gameId column. This way our model can catagorize our data by team 0 or team 1 winning based on both teams contrasting stats.

```
# Replacing column names for rbind
colnames(win_stats_df) <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3',
colnames(lose_stats_df) <- c('kill1', 'kill2', 'kill3', 'kill4', 'kill5', 'death1', 'death2', 'death3',

# Adding column based on dataset it is in
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
```

```
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
win_stats_df <- win_stats_df %>%
  mutate(won=1)

lose_stats_df <- lose_stats_df %>%
  mutate(won =0)

#full_stats_df <- merge(win_stats_df, lose_stats_df, by = "gameId")
full_stats_df <- rbind(win_stats_df, lose_stats_df)
drop <- c("gameId")
full_stats_df <- full_stats_df[,!(names(full_stats_df) %in% drop)]
str(full_stats_df)
```

```
## 'data.frame':    181000 obs. of  31 variables:
##  $ kill1                         : num  10 3 7 11 0 7 9 8 18 4 ...
##  $ kill2                         : num  4 7 3 4 4 1 10 10 5 2 ...
##  $ kill3                         : num  4 0 5 7 2 11 9 5 2 2 ...
##  $ kill4                         : num  6 7 9 2 11 10 9 10 2 6 ...
##  $ kill5                         : num  7 2 4 3 8 8 2 0 11 4 ...
##  $ death1                        : num  4 5 5 2 1 3 2 5 2 3 ...
##  $ death2                        : num  1 0 1 2 4 0 2 5 6 0 ...
##  $ death3                        : num  4 0 2 2 4 3 3 4 8 1 ...
##  $ death4                        : num  4 0 1 3 3 6 1 7 4 0 ...
##  $ death5                        : num  2 3 2 4 4 5 5 6 8 1 ...
##  $ totalDamageDealtToChampions1: num  17898 16662 16241 12111 10900 ...
##  $ totalDamageDealtToChampions2: num  15800 11674 7572 5510 11362 ...
##  $ totalDamageDealtToChampions3: num  10786 7498 10024 8440 14494 ...
##  $ totalDamageDealtToChampions4: num  16964 13016 15115 5373 27391 ...
##  $ totalDamageDealtToChampions5: num  11568 11393 12395 11038 22399 ...
##  $ goldEarned1                   : num  9802 8452 9029 10175 7217 ...
##  $ goldEarned2                   : num  9203 9069 6921 5552 10497 ...
##  $ goldEarned3                   : num  11127 6023 8331 7439 10323 ...
##  $ goldEarned4                   : num  9286 9868 11860 5873 13499 ...
##  $ goldEarned5                   : num  10414 7660 8589 7033 12720 ...
##  $ visionScore1                  : num  28 14 11 17 42 41 19 23 72 9 ...
##  $ visionScore2                  : num  16 27 35 25 38 41 46 55 50 21 ...
##  $ visionScore3                  : num  23 46 25 17 30 21 25 47 29 13 ...
##  $ visionScore4                  : num  17 16 15 9 18 39 31 25 80 19 ...
##  $ visionScore5                  : num  36 22 21 19 26 19 86 70 22 10 ...
##  $ totalTimeCrowdControlDealt1 : num  183 33 178 134 69 310 168 279 365 15 ...
##  $ totalTimeCrowdControlDealt2 : num  92 291 82 61 503 45 291 493 119 62 ...
##  $ totalTimeCrowdControlDealt3 : num  231 31 371 332 562 133 102 287 456 126 ...
##  $ totalTimeCrowdControlDealt4 : num  54 235 140 274 79 78 444 501 215 209 ...
##  $ totalTimeCrowdControlDealt5 : num  281 407 122 163 69 73 92 193 300 168 ...
##  $ won                           : num  1 1 1 1 1 1 1 1 1 1 ...
```

Next let's randomly divide the data into train and test:

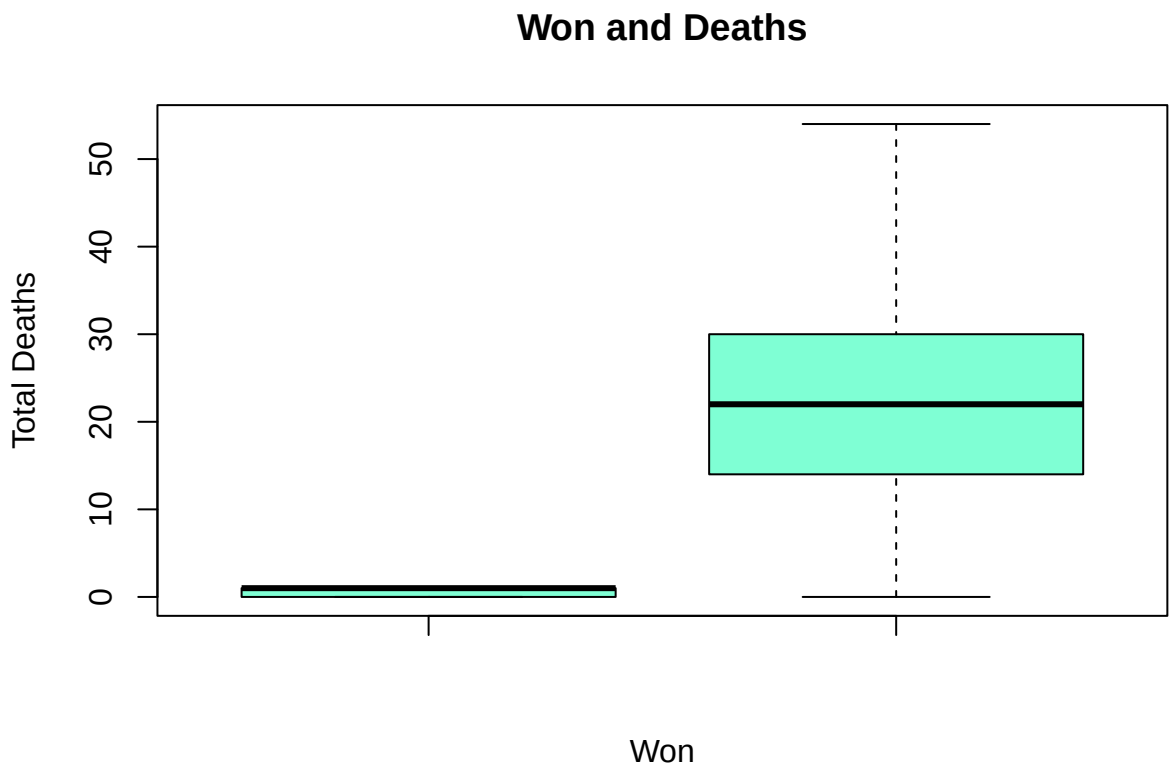
```
set.seed(1010)
i <- sample(1:nrow(full_stats_df), .75*nrow(full_stats_df), replace=FALSE)
full_stats_train <- full_stats_df[i,]
```

```
full_stats_test <- full_stats_df[-i,]
```

Data Exploration

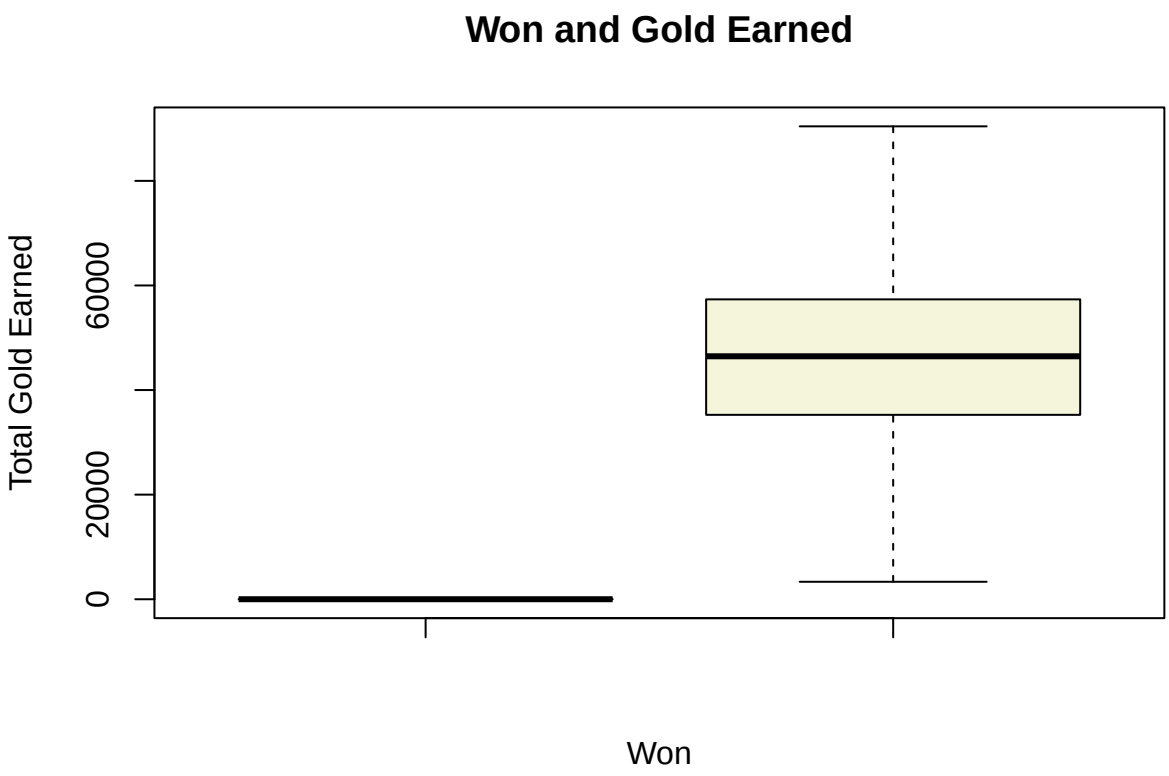
Next we will plot some of our data to see possible differences/correlations. Time to do some data exploration.

```
boxplot(full_stats_train$won, full_stats_train$death1+full_stats_train$death2+full_stats_train$death3+fu
```

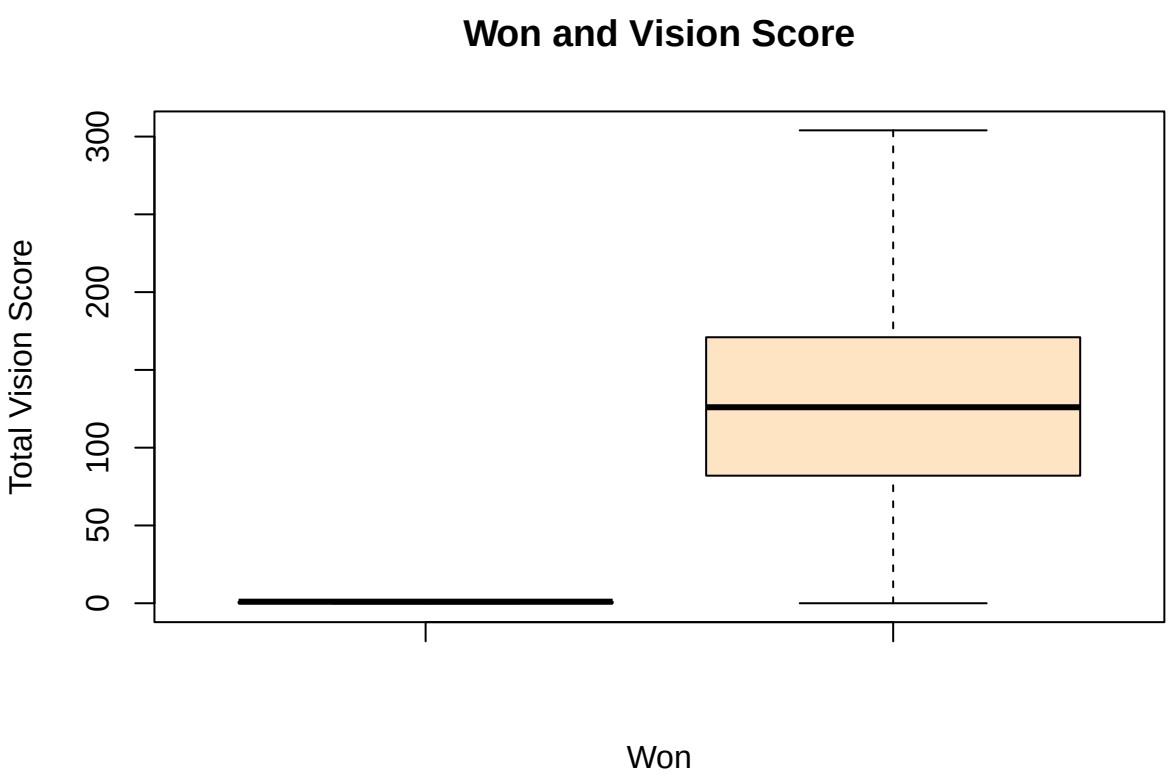



```
boxplot(full_stats_train$won, full_stats_train$kill1+full_stats_train$kill2+full_stats_train$kill3+full_
```

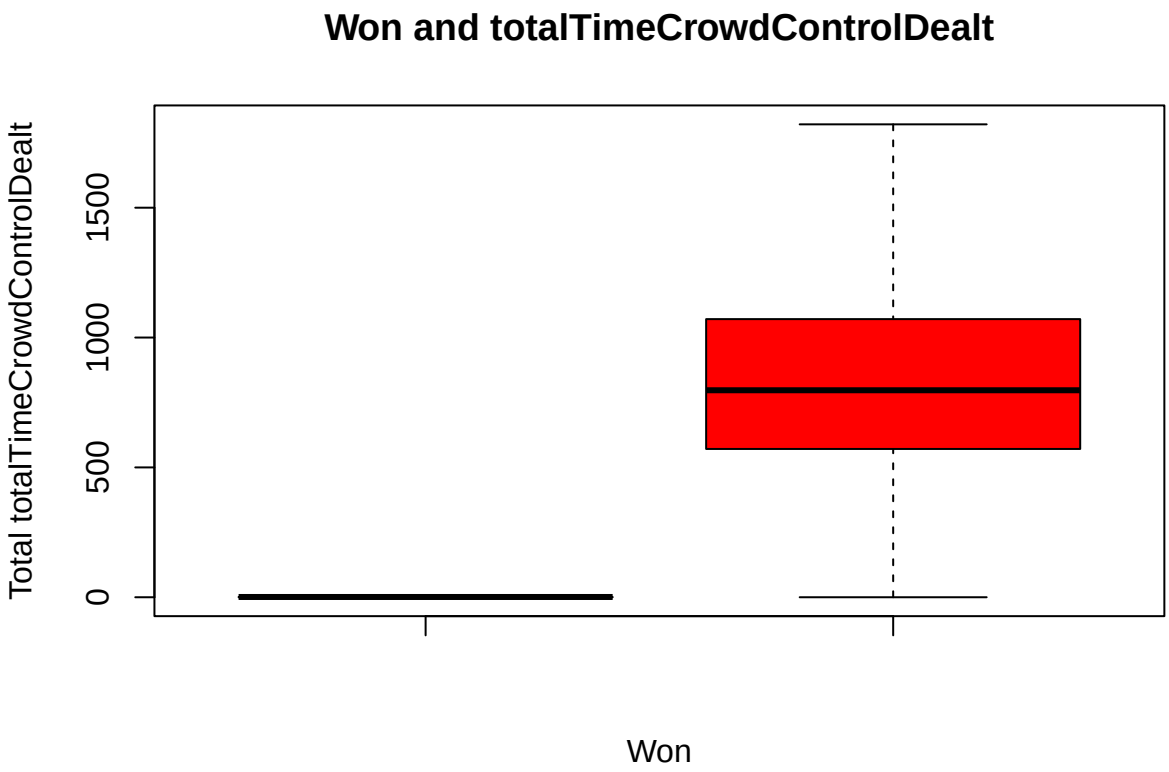

```
boxplot(full_stats_train$won, full_stats_train$goldEarned1+full_stats_train$goldEarned2+full_stats_trai
```



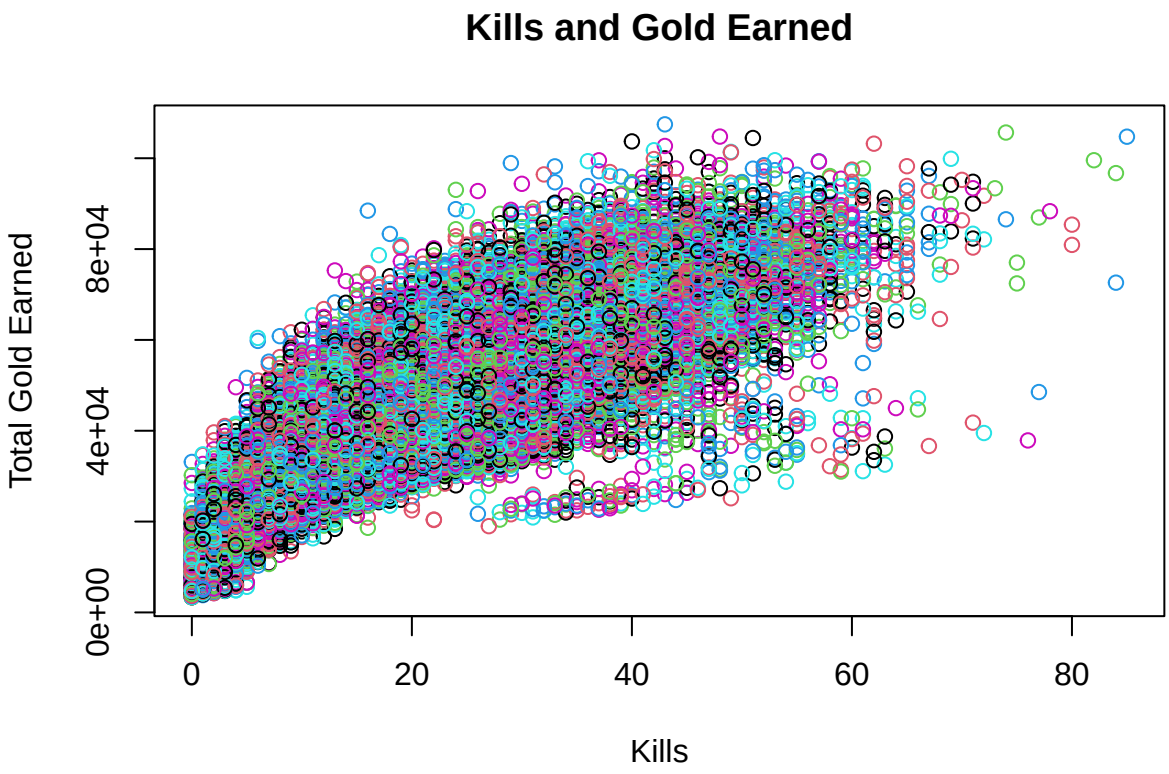

```
boxplot(full_stats_train$won, full_stats_train$visionScore1+full_stats_train$visionScore2+full_stats_tra
```




```
boxplot(full_stats_train$won, full_stats_train$totalTimeCrowdControlDealt1+full_stats_train$totalTimeCr
```

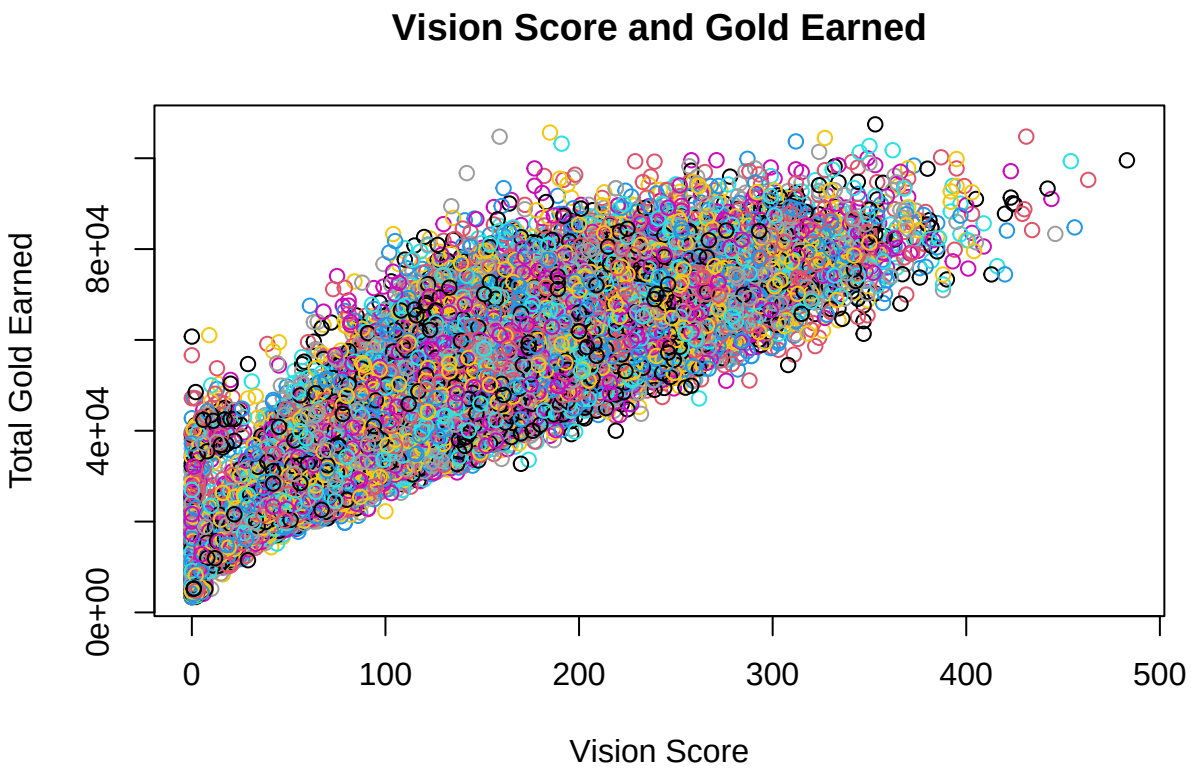



```
plot(full_stats_train$kill1+full_stats_train$kill2+full_stats_train$kill3+full_stats_train$kill4+full_s
```

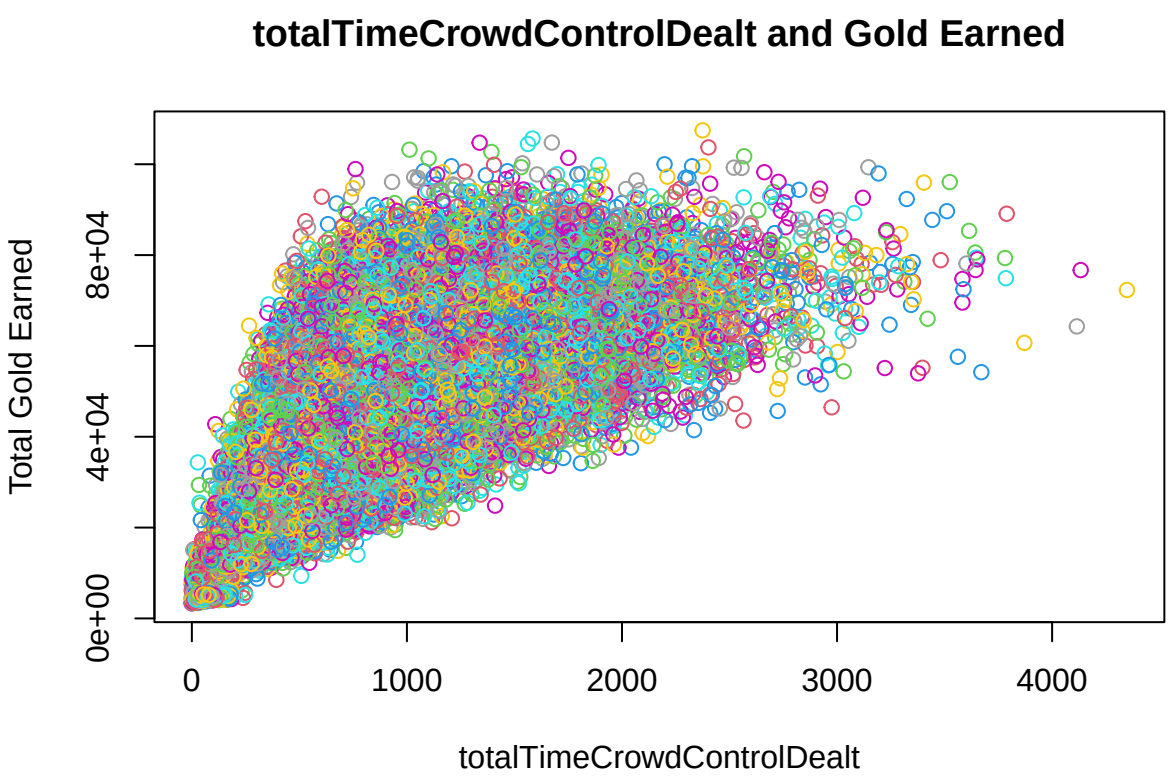



```
plot(full_stats_train$death1+full_stats_train$death2+full_stats_train$death3+full_stats_train$death4+ful
```


```
plot(full_stats_train$visionScore1+full_stats_train$visionScore2+full_stats_train$visionScore3+full_sta
```




```
plot(full_stats_train$totalTimeCrowdControlDealt1+full_stats_train$totalTimeCrowdControlDealt2+full_sta
```



Logistic Regression

Training

Now we will build our logistic regression model

```
glm_won <- glm(won~., data=full_stats_train, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm_won)
```

```
##
## Call:
## glm(formula = won ~ ., family = "binomial", data = full_stats_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.3264  -0.1052   0.0003   0.1033   5.6621
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -1.499e+00  4.975e-02 -30.128  < 2e-16 ***
## kill1                        3.049e-01  7.629e-03  39.970  < 2e-16 ***
## kill2                        3.166e-01  7.711e-03  41.066  < 2e-16 ***
## kill3                        3.097e-01  7.679e-03  40.334  < 2e-16 ***
## kill4                        2.936e-01  7.589e-03  38.681  < 2e-16 ***
## kill5                        3.099e-01  7.706e-03  40.218  < 2e-16 ***
```

```
## death1                        -4.803e-01  7.664e-03 -62.665  < 2e-16 ***
## death2                        -4.942e-01  7.757e-03 -63.719  < 2e-16 ***
## death3                        -4.705e-01  7.587e-03 -62.016  < 2e-16 ***
## death4                        -4.829e-01  7.719e-03 -62.557  < 2e-16 ***
## death5                        -4.963e-01  7.691e-03 -64.526  < 2e-16 ***
## totalDamageDealtToChampions1  9.772e-06  3.141e-06   3.111 0.001865 **
## totalDamageDealtToChampions2  1.199e-05  3.107e-06   3.859 0.000114 ***
## totalDamageDealtToChampions3  1.489e-05  3.109e-06   4.789 1.67e-06 ***
## totalDamageDealtToChampions4  1.769e-05  3.148e-06   5.619 1.92e-08 ***
## totalDamageDealtToChampions5  5.626e-06  3.184e-06   1.767 0.077264 .
## goldEarned1                   1.844e-04  1.315e-05  14.031  < 2e-16 ***
## goldEarned2                   1.702e-04  1.320e-05  12.894  < 2e-16 ***
## goldEarned3                   1.838e-04  1.312e-05  14.015  < 2e-16 ***
## goldEarned4                   1.845e-04  1.323e-05  13.944  < 2e-16 ***
## goldEarned5                   1.913e-04  1.313e-05  14.567  < 2e-16 ***
## visionScore1                  -2.464e-02  9.945e-04 -24.776  < 2e-16 ***
## visionScore2                  -2.507e-02  9.797e-04 -25.594  < 2e-16 ***
## visionScore3                  -2.409e-02  9.837e-04 -24.488  < 2e-16 ***
## visionScore4                  -2.368e-02  9.726e-04 -24.343  < 2e-16 ***
## visionScore5                  -2.523e-02  9.830e-04 -25.665  < 2e-16 ***
## totalTimeCrowdControlDealt1   -6.336e-04  9.615e-05  -6.590 4.41e-11 ***
## totalTimeCrowdControlDealt2   -7.739e-04  9.444e-05  -8.194 2.52e-16 ***
## totalTimeCrowdControlDealt3   -4.803e-04  9.594e-05  -5.006 5.55e-07 ***
## totalTimeCrowdControlDealt4   -7.796e-04  9.705e-05  -8.033 9.54e-16 ***
## totalTimeCrowdControlDealt5   -6.206e-04  9.666e-05  -6.421 1.35e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 188189  on 135749  degrees of freedom
## Residual deviance:  32753  on 135719  degrees of freedom
## AIC: 32815
##
## Number of Fisher Scoring iterations: 8
```

Testing & Evaluation

Now we can evaluate on the test set:

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
glm_probs <- predict(glm_won, newdata = full_stats_test, type = "response")
glm_pred <- ifelse(glm_probs > 0.5, 1, 0)
glm_acc <- mean(glm_pred == full_stats_test$won)

confusionMatrix(as.factor(glm_pred), reference = as.factor(full_stats_test$won))
```

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

```
##          0 21820  1101
##          1   922 21407
##
##                Accuracy : 0.9553
##                  95% CI : (0.9533, 0.9572)
##     No Information Rate : 0.5026
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9106
##
##  Mcnemar's Test P-Value : 7.574e-05
##
##             Sensitivity : 0.9595
##             Specificity : 0.9511
##          Pos Pred Value : 0.9520
##          Neg Pred Value : 0.9587
##              Prevalence : 0.5026
##          Detection Rate : 0.4822
##    Detection Prevalence : 0.5065
##       Balanced Accuracy : 0.9553
##
##        'Positive' Class : 0
##
```

kNN Classification

Normalization, Training, and Testing

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
full_stats_df_normalized <- as.data.frame(lapply(full_stats_df[, 1:30], normalize))
head(full_stats_df_normalized)
```

```
##        kill1      kill2 kill3      kill4 kill5     death1     death2     death3
## 1 0.21276596 0.07272727  0.08 0.15384615 0.175 0.17391304 0.04347826 0.16666667
## 2 0.06382979 0.12727273  0.00 0.17948718 0.050 0.21739130 0.00000000 0.00000000
## 3 0.14893617 0.05454545  0.10 0.23076923 0.100 0.21739130 0.04347826 0.08333333
## 4 0.23404255 0.07272727  0.14 0.05128205 0.075 0.08695652 0.08695652 0.08333333
## 5 0.00000000 0.07272727  0.04 0.28205128 0.200 0.04347826 0.17391304 0.16666667
## 6 0.14893617 0.01818182  0.22 0.25641026 0.200 0.13043478 0.00000000 0.12500000
##       death4     death5 totalDamageDealtToChampions1
## 1 0.18181818 0.08695652                   0.1943766
## 2 0.00000000 0.13043478                   0.1809533
## 3 0.04545455 0.08695652                   0.1763812
## 4 0.13636364 0.17391304                   0.1315284
## 5 0.13636364 0.17391304                   0.1183766
## 6 0.27272727 0.21739130                   0.1618610
##   totalDamageDealtToChampions2 totalDamageDealtToChampions3
## 1                   0.13785281                   0.11348309
## 2                   0.10185403                   0.07888895
## 3                   0.06606465                   0.10546583
## 4                   0.04807399                   0.08880004
## 5                   0.09913188                   0.15249619
```

```
## 6                    0.09735200                    0.17509601
##   totalDamageDealtToChampions4 totalDamageDealtToChampions5 goldEarned1
## 1                    0.17375095                    0.1128464   0.3128167
## 2                    0.13331421                    0.1111393   0.2666028
## 3                    0.15481287                    0.1209139   0.2863549
## 4                    0.05503206                    0.1076762   0.3255854
## 5                    0.28054776                    0.2185034   0.2243256
## 6                    0.15555032                    0.2140258   0.3220594
##   goldEarned2 goldEarned3 goldEarned4 goldEarned5 visionScore1 visionScore2
## 1   0.3081670   0.3421517   0.2755249   0.3519601   0.12280702   0.05818182
## 2   0.3033310   0.1752453   0.2941233   0.2525449   0.06140351   0.09818182
## 3   0.2258111   0.2507194   0.3577797   0.2860804   0.04824561   0.12727273
## 4   0.1764048   0.2215500   0.1664590   0.2299112   0.07456140   0.09090909
## 5   0.3548666   0.3158600   0.4101556   0.4352032   0.18421053   0.13818182
## 6   0.2706702   0.4290386   0.3284760   0.3877698   0.17982456   0.14909091
##   visionScore3 visionScore4 visionScore5 totalTimeCrowdControlDealt1
## 1   0.11386139   0.07083333   0.16744186                   0.08068783
## 2   0.22772277   0.06666667   0.10232558                   0.01455026
## 3   0.12376238   0.06250000   0.09767442                   0.07848325
## 4   0.08415842   0.03750000   0.08837209                   0.05908289
## 5   0.14851485   0.07500000   0.12093023                   0.03042328
## 6   0.10396040   0.16250000   0.08837209                   0.13668430
##   totalTimeCrowdControlDealt2 totalTimeCrowdControlDealt3
## 1                   0.03237157                   0.08571429
## 2                   0.10239268                   0.01150278
## 3                   0.02885292                   0.13766234
## 4                   0.02146376                   0.12319109
## 5                   0.17698804                   0.20853432
## 6                   0.01583392                   0.04935065
##   totalTimeCrowdControlDealt4 totalTimeCrowdControlDealt5
## 1                   0.02283298                   0.11408851
## 2                   0.09936575                   0.16524564
## 3                   0.05919662                   0.04953309
## 4                   0.11585624                   0.06617946
## 5                   0.03340381                   0.02801462
## 6                   0.03298097                   0.02963865
```

```
library(class)
set.seed(1010)
ind <- sample(1:nrow(full_stats_df_normalized), size=nrow(full_stats_df_normalized) * .70, replace = FA
k_train <- full_stats_df[ind,]
k_test <- full_stats_df[-ind,]
k_trainLabels <- full_stats_df[ind, 31]
k_testLabels <- full_stats_df[-ind, 31]
knn_pred <- knn(train = k_train, test = k_test, cl = k_trainLabels, k = 425) # I choose k = 425 because
```

Evaluating

```
confusionMatrix(as.factor(knn_pred), reference = as.factor(k_test$won))
```

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

```
##          0 15129  6021
##          1 12037 21114
##
##                Accuracy : 0.6674
##                  95% CI : (0.6635, 0.6714)
##     No Information Rate : 0.5003
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.335
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.5569
##             Specificity : 0.7781
##          Pos Pred Value : 0.7153
##          Neg Pred Value : 0.6369
##              Prevalence : 0.5003
##          Detection Rate : 0.2786
##    Detection Prevalence : 0.3895
##       Balanced Accuracy : 0.6675
##
##        'Positive' Class : 0
##
```

Decision Trees for Classification

Display the tree

```
library(tree)
stats_tree <- tree(won~., data = full_stats_df)
stats_tree
```

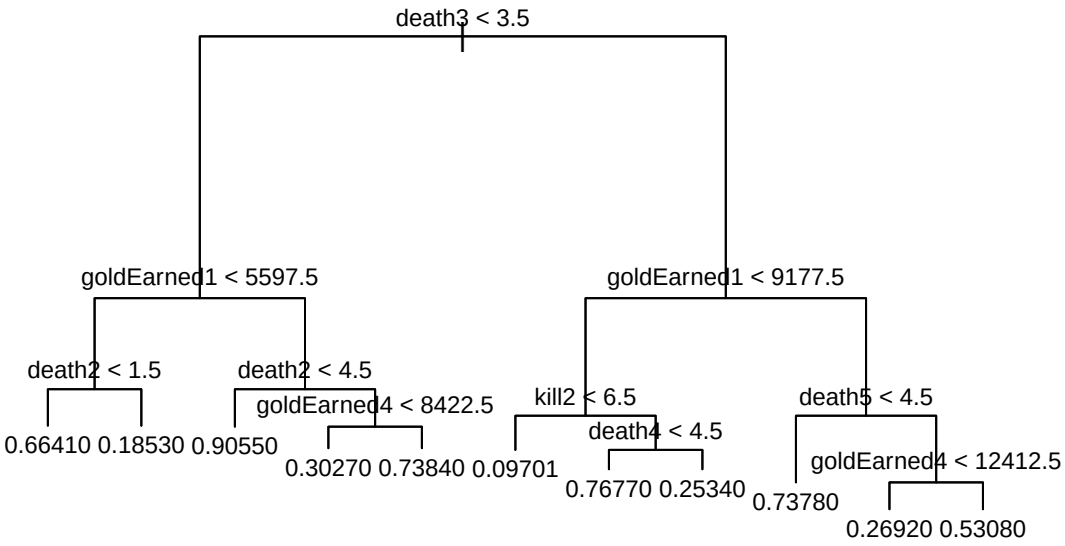
```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 181000 45250.0 0.50000
##    2) death3 < 3.5 72390 14080.0 0.73550
##      4) goldEarned1 < 5597.5 16391  3947.0 0.40410
##        8) death2 < 1.5 7491  1671.0 0.66410 *
##        9) death2 > 1.5 8900  1343.0 0.18530 *
##      5) goldEarned1 > 5597.5 55999  7807.0 0.83260
##       10) death2 < 4.5 42729  3658.0 0.90550 *
##       11) death2 > 4.5 13270  3191.0 0.59780
##         22) goldEarned4 < 8422.5 4282   903.7 0.30270 *
##         23) goldEarned4 > 8422.5 8988  1736.0 0.73840 *
##    3) death3 > 3.5 108610 24480.0 0.34300
##      6) goldEarned1 < 9177.5 48031  6330.0 0.15620
##       12) kill2 < 6.5 40274  3528.0 0.09701 *
##       13) kill2 > 6.5 7757  1929.0 0.46350
##         26) death4 < 4.5 3168   565.0 0.76770 *
##         27) death4 > 4.5 4589   868.3 0.25340 *
##      7) goldEarned1 > 9177.5 60579 15140.0 0.49110
##       14) death5 < 4.5 19150  3705.0 0.73780 *
##       15) death5 > 4.5 41429  9732.0 0.37710
```

```
##         30) goldEarned4 < 12412.5 24335  4787.0 0.26920 *
##         31) goldEarned4 > 12412.5 17094  4257.0 0.53080 *
```

```
summary(stats_tree)
```

```
##
## Regression tree:
## tree(formula = won ~ ., data = full_stats_df)
## Variables actually used in tree construction:
## [1] "death3"      "goldEarned1" "death2"      "goldEarned4" "kill2"
## [6] "death4"      "death5"
## Number of terminal nodes:  11
## Residual mean deviance:  0.1493 = 27020 / 181000
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.905500 -0.253400 -0.001231  0.000000  0.261600  0.903000
```

```
plot(stats_tree)
text(stats_tree, cex = .75, pretty = 0)
```



Train & Test

```
set.seed(1010)
i <- sample(181000, 45250, replace=FALSE)
tree_train <- full_stats_df[i,]
tree_test <- full_stats_df[-i,]
stats_train_tree <- tree(as.factor(won)~., data = tree_train)
```

```
print(stats_train_tree)
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 45250 62730 1 ( 0.49852 0.50148 )
##    2) death3 < 3.5 18038 20680 1 ( 0.26028 0.73972 )
##      4) goldEarned4 < 5968.5 4680  6423 0 ( 0.55897 0.44103 )
##        8) death1 < 2.5 2767  3593 1 ( 0.35309 0.64691 )
##         16) goldEarned1 < 4664 1309  1774 0 ( 0.58824 0.41176 ) *
##         17) goldEarned1 > 4664 1458  1191 1 ( 0.14198 0.85802 ) *
##        9) death1 > 2.5 1913  1572 0 ( 0.85677 0.14323 ) *
##      5) goldEarned4 > 5968.5 13358 11550 1 ( 0.15564 0.84436 )
##       10) death2 < 3.5 8325  3721 1 ( 0.05874 0.94126 ) *
##       11) death2 > 3.5 5033  6279 1 ( 0.31591 0.68409 )
##         22) kill3 < 4.5 2382  3302 0 ( 0.50462 0.49538 ) *
##         23) kill3 > 4.5 2651  2207 1 ( 0.14636 0.85364 ) *
##    3) death3 > 3.5 27212 35010 0 ( 0.65644 0.34356 )
##      6) goldEarned2 < 7840.5 8166  5350 0 ( 0.89885 0.10115 )
##       12) kill5 < 4.5 6232  2376 0 ( 0.95266 0.04734 ) *
##       13) kill5 > 4.5 1934  2273 0 ( 0.72544 0.27456 ) *
##      7) goldEarned2 > 7840.5 19046 26190 0 ( 0.55250 0.44750 )
##       14) death1 < 4.5 6314  7883 1 ( 0.31660 0.68340 )
##         28) death4 < 4.5 3294  3047 1 ( 0.17426 0.82574 ) *
##         29) death4 > 4.5 3020  4177 1 ( 0.47185 0.52815 ) *
##       15) death1 > 4.5 12732 16160 0 ( 0.66949 0.33051 )
##         30) goldEarned5 < 12172.5 7735  7966 0 ( 0.78927 0.21073 )
##           60) goldEarned4 < 11708.5 5112  3562 0 ( 0.88908 0.11092 ) *
##           61) goldEarned4 > 11708.5 2623  3542 0 ( 0.59474 0.40526 ) *
##         31) goldEarned5 > 12172.5 4997  6922 1 ( 0.48409 0.51591 ) *
```

```
tree_pred <- predict(stats_train_tree, newdata = tree_test, type = "class")
```

Evaluation

```
confusionMatrix(as.factor(tree_pred), reference = as.factor(tree_test$won))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 50820 13566
##          1 17122 54242
##
##                Accuracy : 0.7739
##                  95% CI : (0.7717, 0.7762)
##     No Information Rate : 0.5005
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5479
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7480
```

```
##             Specificity : 0.7999
##          Pos Pred Value : 0.7893
##          Neg Pred Value : 0.7601
##              Prevalence : 0.5005
##          Detection Rate : 0.3744
##    Detection Prevalence : 0.4743
##       Balanced Accuracy : 0.7740
##
##        'Positive' Class : 0
##
```

Logistic Regression vs kNN vs Decision Trees

Analyzing the results of each model based on the algorithms.

Logistic Regression

The Logistic Regression model had a high accuracy of about 96%, a Kappa value of about .91, a p-value of 7.574e-05, a sensitivity value of about .96, and a specificity of about .95. These are all very good metrics for a model. I believe the logistic regression model performed so well because it is a linear classifier and the data is very linear. Therefore computing log odds and probabilities linearly for the data was very effective.

kNN Classification

The kNN Classification model had an accuracy of about 67%, a Kappa value of about .34, a p-value of 2.2e-16, a sensitivity value of about .56, and a specificity value of about .78. These metrics aren't nearly as impressive looking compared to the logistic regression. However, you have to take into consideration how much variance there is in the data and that looking at nearest neighbors might not be as accurate as comparing the stats linearly. For this algorithm I had to normalize the data and use the square root of the size of the data in order to achieve the highest accuracy of 67%, before this it was around 50%. We also have many predictors in this data set where kNN prefers fewer predictors, if I were to limit the predictors/combine some of them it would likely perform better.

Decision Trees

The Decision Trees model had an accuracy of about 77%, a Kappa value of about .55, a p-value of 2.2e-16, a sensitivity value of about .75, and a specificity value of about .8. These are some pretty fair metrics overall. The tree itself is actually quite simple if you understand the data and what the predictors mean. It is clear which ones cause splits between likely to win or lose, I think it is fairly accurate because it was easy for the tree to make splits in the linear data based on the many predictors.