Yankees Analyst Assessment

Kenny Miller

3/16/2022

Question 1

Part A

```
binom.test(8,10,p=0.5)
```

```
##
##  Exact binomial test
##
## data:  8 and 10
## number of successes = 8, number of trials = 10, p-value = 0.1094
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.4439045 0.9747893
## sample estimates:
## probability of success
##                    0.8
```

Response: I would expect to see 5 heads on the next 10 flips because we have not done enough tests/trials to determine the coin is unfair, and I have no reason to doubt it is not a fair coin. In these 10 flips I do not have evidence to assume that the coin I found is not fair; the binomial test above shows that even though we had 8 heads on the previous 10 flips, we fail to reject the null hypothesis that the probability of heads is different than 50% because it is contained within the confidence interval.

Part B

```
binom.test(800,1000,p=0.5)
```

```
##
##  Exact binomial test
##
## data:  800 and 1000
```

```
## number of successes = 800, number of trials = 1000, p-value < 2.2e-16
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.7738406 0.8243794
## sample estimates:
## probability of success
##                    0.8
```

Response: In completing 1000 trials, I see that 800 heads come up. I would expect to see 800 heads on the next 1000 trials because we have enough evidence, from the binomial test above, to reject the null hypothesis that the probability of heads is 50%, and determine the coin is not fair. After completing the additional 1000 trials, I need to adjust my perception of the coin and it is clearly not fair, with heads coming up about 80% of the time.

Question 2

Overview

Below are the steps I used to generate projections for the average outcome for each batter next season. I used the following steps to create a simulation of plate appearances to generate my projections:

1. Load in the data and check summary for any potential missing information (in this case there is not any). Then check the histogram of the outcome variable to see its distribution, since it is normal I felt comfortable using averages for all previous pitcher/batter interactions regardless of the number of previous matchups.
2. Next I worked on a single simulation for a batter to determine the proper logic to use when building my projection system.
    - I randomly generated 300 pitchers for the batter to face, assuming the probability of seeing each pitcher was equal.
    - I aggregated the data by pitcherID to find the average outcome each pitcher had, in case the randomized pitcher matchup had not occurred in the past.
    - I then subset the historical matchups by batter and aggregate that to find the average outcome in previous matchups with a pitcher.
    - A data frame of all potential pitcher matchups with this batter is made.
    - A for loop is then used to loop over all 300 randomly selected matchups, updating a data frame of outcomes in these 300 PAs.
    - Finally, the average of all 300 outcomes is taken to project the average outcome for this batter next season.
3. The last step in this projection is to run it with all batters!
    - The aggregated data by pitcherID is generated to ensure no changes from previous
    - A data frame for projections is made to store final values
    - A number of trials is set (1,000), this will provide me with multiple iterations of the random 300 matchups which will improve the accuracy of the simulation by helping to get my projections closer to the truth in the long-run (I tried trials at 10,000 but the run time was way too high and the change in the compared projections of 1,000 trials was minimal)

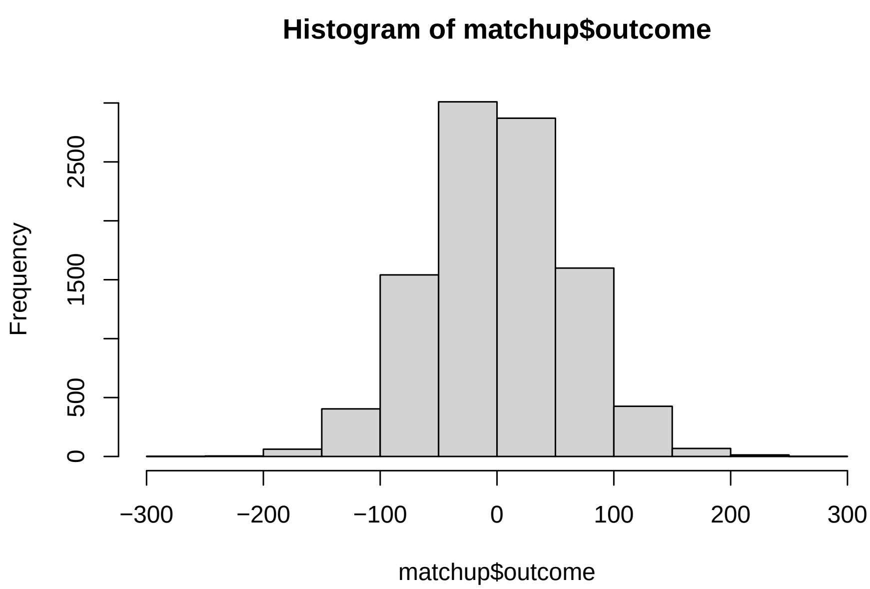
- The same logic for a single batter is now applied to all 100 for 1,000 trials.

4. Finally, the projections are reviewed via a summary and a histogram and the results are written to a csv.

```
matchup <- read.csv("matchupdata.csv")
summary(matchup) # no missing values
```

```
##     batterID        pitcherID        outcome
##  Min.   :  1.00   Min.   :101.0   Min.   :-266.0314
##  1st Qu.: 26.00   1st Qu.:126.0   1st Qu.: -40.3459
##  Median : 50.00   Median :150.0   Median :  -0.2034
##  Mean   : 50.44   Mean   :150.2   Mean   :   0.3944
##  3rd Qu.: 75.00   3rd Qu.:175.0   3rd Qu.:  41.3573
##  Max.   :100.00   Max.   :200.0   Max.   : 255.8625
```

```
hist(matchup$outcome) # since the distribution is normal, I feel comfortable using
```




```
# averages for pitcher/batter interactions
```

```
batter <- 1
pitchers_faced <- sample(101:200,300, replace = TRUE)
average_pitcher <- aggregate(outcome~pitcherID, data = matchup, FUN=mean)
# assuming each pitcher is equally likely to be faced
sub <- subset(matchup, batterID == batter)
batter_faced <- aggregate(outcome~pitcherID, data = sub, FUN=mean)
batter_faced <- rbind(batter_faced,
                      average_pitcher[which(average_pitcher$pitcherID %in%
                                                  c(setdiff(pitchers_faced, sub$pitcherID))),])
# average_pitcher[which(average_pitcher$pitcherID %in% c(setdiff(pitchers_faced, sub$pitcherID))),]
at_bats <- data.frame(pitcher=pitchers_faced, outcome = rep(0,300))
```

```r
for (i in 1:nrow(at_bats)) {
  at_bats$outcome[i] <- batter_faced$outcome[which(batter_faced$pitcherID == at_bats$pitcher[i])]
}
mean(at_bats$outcome)
```

```
## [1] -8.974832
```

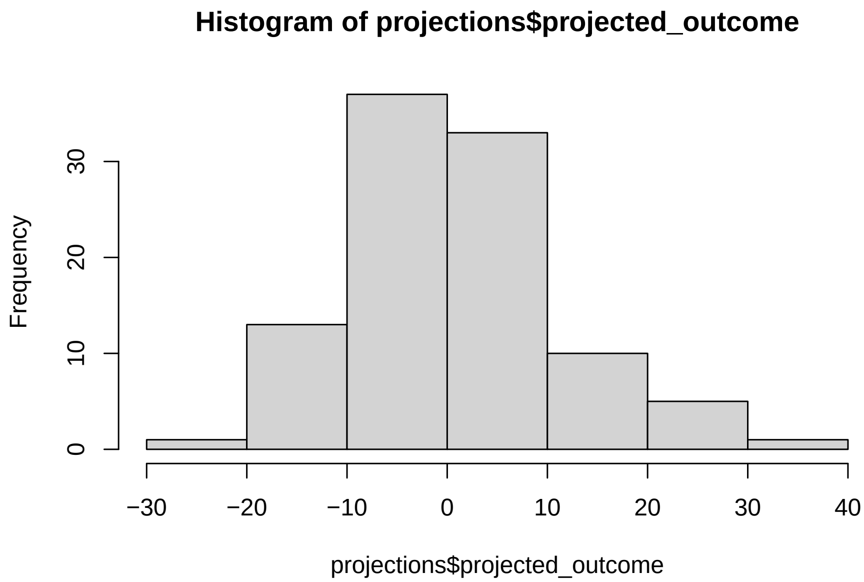
```r
average_pitcher <- aggregate(outcome~pitcherID, data = matchup, FUN=mean) # average pitcher result
projections <- data.frame(batter_ID=1:100, projected_outcome=rep(0,100))
trials <- 1000 # increases run time but helps to improve projection accuracy
set.seed(23); for (j in 1:100) {
  results <- c()
  # print(paste("working on batter: ",j))
  for (k in 1:trials) {
    pitchers_faced <- sample(101:200,300, replace = TRUE) # simulate pitchers faced
    sub <- subset(matchup, batterID == j) # utilize previous matchups when available
    batter_faced <- aggregate(outcome~pitcherID, data = sub, FUN=mean)
    batter_faced <- rbind(batter_faced,
                          average_pitcher[which(average_pitcher$pitcherID %in%
                                                  c(setdiff(pitchers_faced, sub$pitcherID))),])
    at_bats <- data.frame(pitcher=pitchers_faced, outcome = rep(0,300))
    for (i in 1:nrow(at_bats)) { # 300 plate appearances simulation
      at_bats$outcome[i] <- batter_faced$outcome[which(batter_faced$pitcherID == at_bats$pitcher[i])]
    }
    results[k] <- mean(at_bats$outcome)
  }
  projections$projected_outcome[j] <- mean(results)
}
head(projections)
```

```
##   batter_ID projected_outcome
## 1         1         -9.986501
## 2         2         -2.614995
## 3         3         16.847672
## 4         4         -1.083186
## 5         5         -5.137217
## 6         6         -9.948230
```

```r
summary(projections)
```

```
##    batter_ID      projected_outcome
##  Min.   :  1.00   Min.   :-24.1849
##  1st Qu.: 25.75   1st Qu.: -6.8148
##  Median : 50.50   Median : -0.2515
##  Mean   : 50.50   Mean   :  0.3693
##  3rd Qu.: 75.25   3rd Qu.:  5.9625
##  Max.   :100.00   Max.   : 35.0571
```

```r
hist(projections$projected_outcome)
```




```
write.csv(projections, "matchup_projections.csv", row.names = F)
```

Question 3

Summary

I was tasked with classifying pitch types based on the pitched ball's trajectories and some of the pitcher's characteristics. The model I developed is a random forest model which is able to accurately predict pitch type at a 93.55% rate on the training data and a 92.27% rate on the holdout data. Most of the models I tested had strong performances but the random forest edged ahead in the end.

Cleaning

- Checked the summaries of the Training and Test data and both looked good to go; nothing missing and all reasonable values.
- Utilizing the make.names function I converted the pitch type column to a character, then factor, that would allow it to be passed to the models properly
- Checked for near zero or zero variance predictors (none appeared)
- Checked for highly correlated predictors (none found)
- Split the training data into train and holdout samples, a 70/30 split respectively

Model Development

- Models will be compared using accuracy with 5 fold cross-validation
- First, I started with a vanilla partition model

- Stepped up modeling to random forest, boosted tree, k-nearest neighbors, and a single layer neural network. All of these models performed better than the vanilla partition in their accuracies on both the training data and holdout
- After running through these more advanced models, I wanted to compare them so I used the accuracy metric on the training data and the accuracy standard deviation (SD). Using the one-SD rule to compare models I found that the best model was the neural network (NN) but it also had the largest SD which kept all of the complex models, non-vanilla, in the running.
- Since the NN and random forest both performed very well and were close in accuracy and Kappa (model success in dealing with potential imbalances in class, pitch type, distributions) measures, I felt comfortable selecting the random forest as the final model due to its quicker run time
- Finally, predictions were made on the Test data frame using the random forest model, whose parameters were identified in training and the model was fit using all of the Training data frame.

Conclusions

In this process I was able to identify many models that could predict pitch type at a 90% or better accuracy. The model I chose to make final pitch type predictions was a random forest model. This model performed very well in classifying pitch types.

```
TRAIN.Original <- read.csv("pitchclassificationtrain.csv")
TEST <- read.csv("pitchclassificationtest.csv")
```

```
DATA <- TRAIN.Original
summary(DATA) # no missing values, let's go!
```

```
##     pitchid        pitcherid         yearid          height     
##  Min.   :    1   Min.   :1.000   Min.   :1.000   Min.   :72.00  
##  1st Qu.: 2662   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:72.00  
##  Median : 5324   Median :5.000   Median :2.000   Median :72.00  
##  Mean   : 5324   Mean   :3.829   Mean   :1.502   Mean   :74.01  
##  3rd Qu.: 7986   3rd Qu.:5.000   3rd Qu.:2.000   3rd Qu.:76.00  
##  Max.   :10647   Max.   :5.000   Max.   :2.000   Max.   :80.00  
##    ballSpeed         curve_X             curve_Z        releasePoint_X 
##  Min.   : 67.67   Min.   :-14.3862   Min.   :-13.819   Min.   :-3.074  
##  1st Qu.: 83.89   1st Qu.: -6.4124   1st Qu.:  2.312   1st Qu.:-2.066  
##  Median : 89.68   Median : -3.7987   Median :  6.264   Median :-1.709  
##  Mean   : 87.69   Mean   : -3.0704   Mean   :  4.653   Mean   :-1.674  
##  3rd Qu.: 91.73   3rd Qu.: -0.6754   3rd Qu.:  8.847   3rd Qu.:-1.268  
##  Max.   :100.07   Max.   : 12.4708   Max.   : 14.407   Max.   : 4.338  
##  releasePoint_Y  releasePoint_Z     ballSpin           type       
##  Min.   :5.435   Min.   :5.237   Min.   :  91.5   Min.   : 2.000  
##  1st Qu.:6.063   1st Qu.:5.767   1st Qu.:1841.8   1st Qu.: 4.000  
##  Median :6.200   Median :5.885   Median :2051.6   Median : 9.000  
##  Mean   :6.199   Mean   :5.906   Mean   :1982.2   Mean   : 7.344  
##  3rd Qu.:6.332   3rd Qu.:6.000   3rd Qu.:2289.3   3rd Qu.:10.000  
##  Max.   :6.886   Max.   :7.088   Max.   :3241.0   Max.   :10.000
```

```
summary(TEST) # no missing values either, let's start model development
```

```
##     pitchid        pitcherid         yearid      height        ballSpeed    
##  Min.   :10648   Min.   :1.000   Min.   :3   Min.   :72.0   Min.   :67.48  
##  1st Qu.:13736   1st Qu.:2.000   1st Qu.:3   1st Qu.:72.0   1st Qu.:81.98  
##  Median :16825   Median :3.000   Median :3   Median :76.0   Median :86.56
```

```
##  Mean   :16825   Mean   :3.026   Mean   :3   Mean   :75.7   Mean   :85.45
##  3rd Qu.:19913   3rd Qu.:4.000   3rd Qu.:3   3rd Qu.:77.0   3rd Qu.:89.49
##  Max.   :23001   Max.   :6.000   Max.   :3   Max.   :80.0   Max.   :95.89
##     curve_X            curve_Z        releasePoint_X    releasePoint_Y
##  Min.   :-13.1461   Min.   :-14.964   Min.   :-3.1767   Min.   :5.463
##  1st Qu.: -5.9927   1st Qu.:  1.758   1st Qu.:-2.1422   1st Qu.:6.065
##  Median : -1.7793   Median :  5.830   Median :-1.8896   Median :6.203
##  Mean   : -0.8292   Mean   :  4.487   Mean   :-0.5832   Mean   :6.203
##  3rd Qu.:  4.2463   3rd Qu.:  8.698   3rd Qu.: 1.9094   3rd Qu.:6.336
##  Max.   : 13.8256   Max.   : 14.966   Max.   : 4.7798   Max.   :6.930
##  releasePoint_Z     ballSpin
##  Min.   :5.096   Min.   : 114.6
##  1st Qu.:5.826   1st Qu.:2024.4
##  Median :5.985   Median :2196.2
##  Mean   :6.062   Mean   :2160.3
##  3rd Qu.:6.349   3rd Qu.:2357.5
##  Max.   :7.118   Max.   :3312.2
```

```
# converting pitch type to factor to better model classification
DATA <- DATA[,-c(1:3)]
DATA$type <- as.factor(make.names(DATA$type))
summary(DATA)
```

```
##      height        ballSpeed         curve_X            curve_Z
##  Min.   :72.00   Min.   : 67.67   Min.   :-14.3862   Min.   :-13.819
##  1st Qu.:72.00   1st Qu.: 83.89   1st Qu.: -6.4124   1st Qu.:  2.312
##  Median :72.00   Median : 89.68   Median : -3.7987   Median :  6.264
##  Mean   :74.01   Mean   : 87.69   Mean   : -3.0704   Mean   :  4.653
##  3rd Qu.:76.00   3rd Qu.: 91.73   3rd Qu.: -0.6754   3rd Qu.:  8.847
##  Max.   :80.00   Max.   :100.07   Max.   : 12.4708   Max.   : 14.407
##
##  releasePoint_X   releasePoint_Y  releasePoint_Z     ballSpin       type
##  Min.   :-3.074   Min.   :5.435   Min.   :5.237   Min.   :  91.5   X10:3102
##  1st Qu.:-2.066   1st Qu.:6.063   1st Qu.:5.767   1st Qu.:1841.8   X2 :1483
##  Median :-1.709   Median :6.200   Median :5.885   Median :2051.6   X3 : 205
##  Mean   :-1.674   Mean   :6.199   Mean   :5.906   Mean   :1982.2   X4 :1330
##  3rd Qu.:-1.268   3rd Qu.:6.332   3rd Qu.:6.000   3rd Qu.:2289.3   X7 : 901
##  Max.   : 4.338   Max.   :6.886   Max.   :7.088   Max.   :3241.0   X8 : 674
##                                                                    X9 :2952
```

```
table(DATA$type)
```

```
##
##  X10   X2   X3   X4   X7   X8   X9
## 3102 1483  205 1330  901  674 2952
```

```
#Looking for zero variance or near zero variance predictors
infodensity <- nearZeroVar(DATA, saveMetrics= TRUE)
infodensity[infodensity$nzv,][1:5,]
```

```
##      freqRatio percentUnique zeroVar nzv
## NA          NA            NA      NA  NA
```

```
## NA.1        NA              NA      NA  NA
## NA.2        NA              NA      NA  NA
## NA.3        NA              NA      NA  NA
## NA.4        NA              NA      NA  NA
```

```
DATA.NOnzv <- DATA[,-nearZeroVar(DATA)]
# no variables have zero variance or near zero variance

#Explore highly correlated or redundant predictors
highlycorrelated <- findCorrelation( cor(DATA.NOnzv) , cutoff = .90)
highlycorrelated
```

```
## integer(0)
```

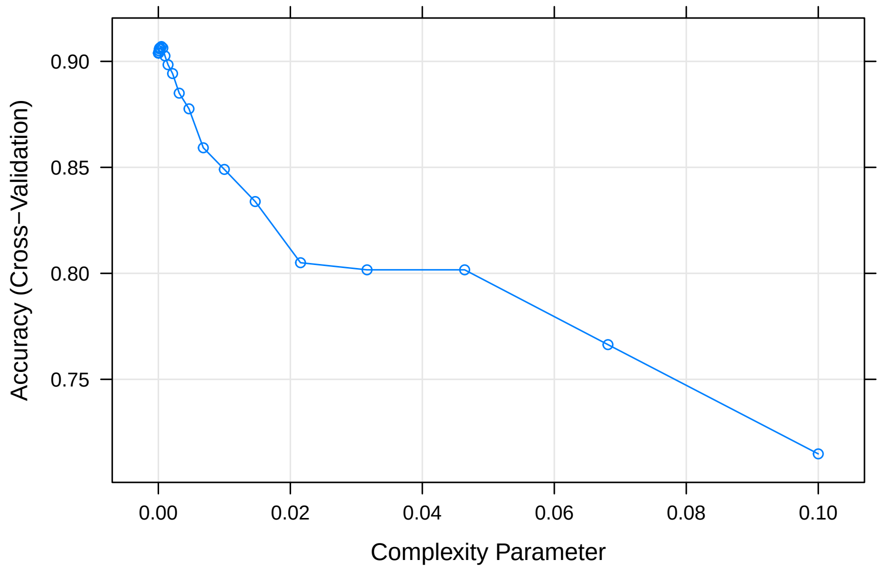
```
# no variables to worry about here either! Let's start model building
```

```
# y = type
#1) split training data
set.seed(23); train.rows <- sample(1:nrow(DATA), 0.7*nrow(DATA))
TRAIN <- DATA[train.rows,]
HOLDOUT <- DATA[-train.rows,]

#2) Set up how generalization error is to be estimated (5-fold crossvalidation shown here)
## USing Accuracy
fitControl <- trainControl(method="cv",number=5, classProbs=TRUE, allowParallel = TRUE)

#3) Fitting models
# Vanilla Partition
treeGrid <- expand.grid(cp=10^seq(-5,-1,length=25))

set.seed(23); TREE <- train(type~.,data=TRAIN,method='rpart', tuneGrid=treeGrid,
                            trControl=fitControl, preProc = c("center", "scale"))
# TREE  #Look at details of all fits
plot(TREE) #See how error changes with choices
```




```r
TREE$bestTune #Gives best parameters
```

```
##              cp
## 11 0.0004641589
```

```r
# TREE$results #Look at output in more detail (lets you see SDs)
TREE$results[rownames(TREE$bestTune),] # accuracy: 0.9070044, SD: 0.005983118
```

```
##              cp  Accuracy     Kappa  AccuracySD     KappaSD
## 11 0.0004641589 0.9070044 0.8822215 0.005983118 0.007552962
```

```r
varImp(TREE)
```

```
## rpart variable importance
##
##                  Overall
## curve_X          100.000
## curve_Z           91.043
## ballSpin          77.833
## ballSpeed         77.791
## releasePoint_X    34.180
## releasePoint_Z    27.447
## height             9.518
## releasePoint_Y     0.000
```

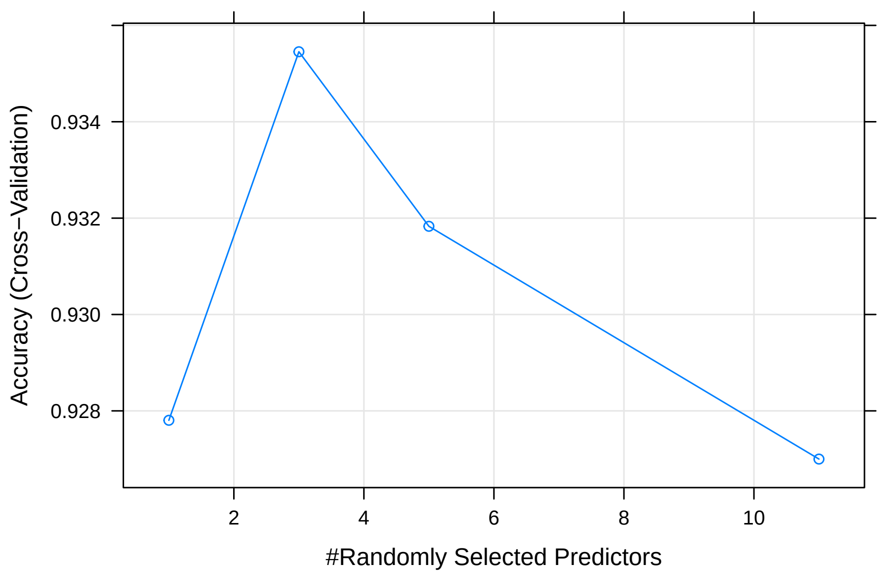
```r
postResample(predict(TREE,newdata=HOLDOUT),HOLDOUT$type) # accuracy: 0.8985915
```

```
##  Accuracy     Kappa
## 0.8985915 0.8720044
```

```r
# Random Forest
forestGrid <- expand.grid(mtry=c(1,3,5,11))

cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(23); FOREST <- train(type~.,data=TRAIN,method='rf',tuneGrid=forestGrid,
                              trControl=fitControl, preProc = c("center", "scale"))
stopCluster(cluster)
registerDoSEQ()

# FOREST
plot(FOREST)
```




```r
FOREST$bestTune #Gives best parameters
```

```
##   mtry
## 2    3
```

```r
# FOREST$results #Look at output in more detail (lets you see SDs)
FOREST$results[rownames(FOREST$bestTune),] # accuracy: 0.9354534, SD: 0.0058972
```

```
##   mtry  Accuracy     Kappa AccuracySD     KappaSD
## 2    3 0.9354534 0.9182707  0.0058972 0.007468678
```

```r
varImp(FOREST)
```

```
## rf variable importance
##
##                  Overall
## curve_X          100.000
```

```
## ballSpeed       84.888
## curve_Z         83.338
## ballSpin        80.813
## releasePoint_X  24.245
## releasePoint_Z  17.335
## height           2.801
## releasePoint_Y   0.000
```

```
postResample(predict(FOREST,newdata=HOLDOUT),HOLDOUT$type) # accuracy: 0.9226917
```

```
##  Accuracy     Kappa
## 0.9226917 0.9024461
```

```
confusionMatrix(data=predict(FOREST,HOLDOUT),reference=HOLDOUT$type)
```

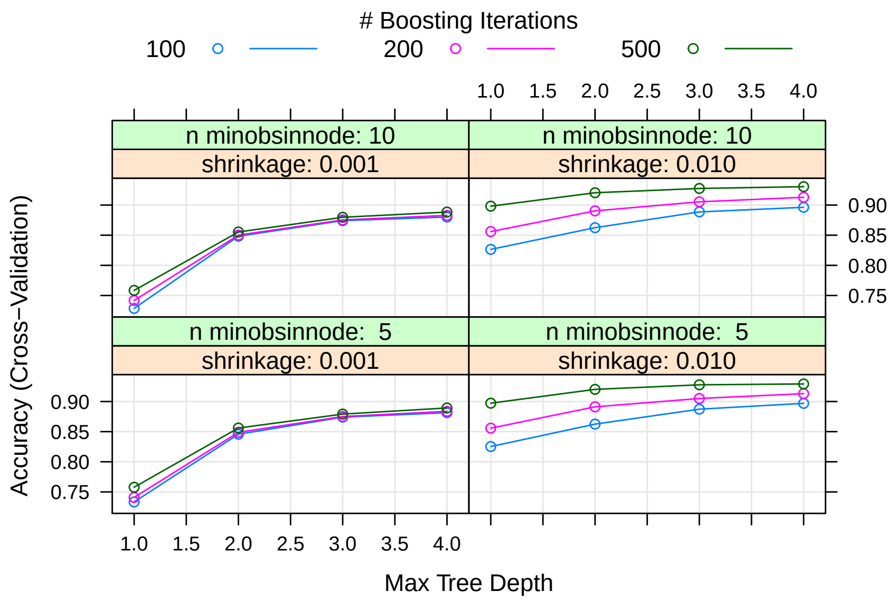
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction X10  X2  X3  X4  X7  X8  X9
##        X10 841   0   0   0   0   6  88
##        X2    0 461   4   0   0   0   0
##        X3    0   0  49   0   0   0   0
##        X4    0   0  12 367  20   0   0
##        X7    0   0   7  11 248   0   0
##        X8    6   0   1   0   0 198   0
##        X9   83   0   0   7   0   2 784
##
## Overall Statistics
##
##                Accuracy : 0.9227
##                  95% CI : (0.9129, 0.9317)
##     No Information Rate : 0.2911
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9024
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: X10 Class: X2 Class: X3 Class: X4 Class: X7
## Sensitivity              0.9043    1.0000   0.67123    0.9532   0.92537
## Specificity              0.9585    0.9985   1.00000    0.9886   0.99385
## Pos Pred Value           0.8995    0.9914   1.00000    0.9198   0.93233
## Neg Pred Value           0.9606    1.0000   0.99237    0.9936   0.99317
## Prevalence               0.2911    0.1443   0.02285    0.1205   0.08388
## Detection Rate           0.2632    0.1443   0.01534    0.1149   0.07762
## Detection Prevalence     0.2926    0.1455   0.01534    0.1249   0.08326
## Balanced Accuracy        0.9314    0.9993   0.83562    0.9709   0.95961
##                      Class: X8 Class: X9
## Sensitivity            0.96117    0.8991
## Specificity            0.99766    0.9604
## Pos Pred Value         0.96585    0.8950
```

```
## Neg Pred Value          0.99732    0.9621
## Prevalence              0.06448    0.2729
## Detection Rate          0.06197    0.2454
## Detection Prevalence    0.06416    0.2742
## Balanced Accuracy       0.97941    0.9297
```

```
# Boosted Tree
gbmGrid <- expand.grid(n.trees=c(100,200,500),interaction.depth=1:4,shrinkage=c(.01,.001),n.minobsinnode

cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(23); GBM <- train(type~.,data=TRAIN, method='gbm',tuneGrid=gbmGrid,verbose=FALSE,
                    trControl=fitControl, preProc = c("center", "scale"))
stopCluster(cluster)
registerDoSEQ()

# GBM  #Look at details of all fits
plot(GBM) #See how error changes with choices
```




```
GBM$bestTune #Gives best parameters
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 48     500                 4      0.01             10
```

```
# GBM$results #Look at output in more detail (lets you see SDs)
GBM$results[rownames(GBM$bestTune),] # accuracy: 0.9304887, SD: 0.006441617
```

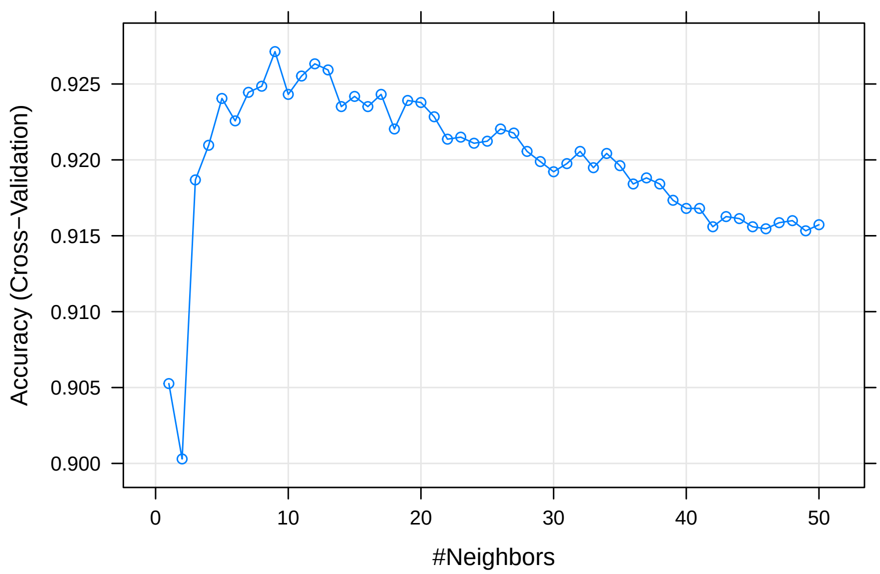
```
##    shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 48      0.01                 4             10     500 0.9304887 0.9119698
##     AccuracySD     KappaSD
## 48 0.006441617 0.008186315
```

```r
# significant run time increase against previous two
postResample(predict(GBM,newdata=HOLDOUT,n.trees=500),HOLDOUT$type) # accuracy: 0.9242567
```

```
##  Accuracy     Kappa
## 0.9242567 0.9043694
```

```r
# K-nearest neighbors
knnGrid <- expand.grid(k=1:50)
set.seed(23);  KNN <- train(type~.,data=TRAIN, method='knn', trControl=fitControl,tuneGrid=knnGrid,
                             preProc = c("center", "scale"))

# KNN  #Look at details of all fits
plot(KNN) #See how error changes with choices
```




```r
KNN$bestTune #Gives best parameters
```

```
##   k
## 9 9
```

```r
# KNN$results #Look at output in more detail (lets you see SDs)
KNN$results[rownames(KNN$bestTune),] # accuracy: 0.9271343, SD: 0.00581194
```

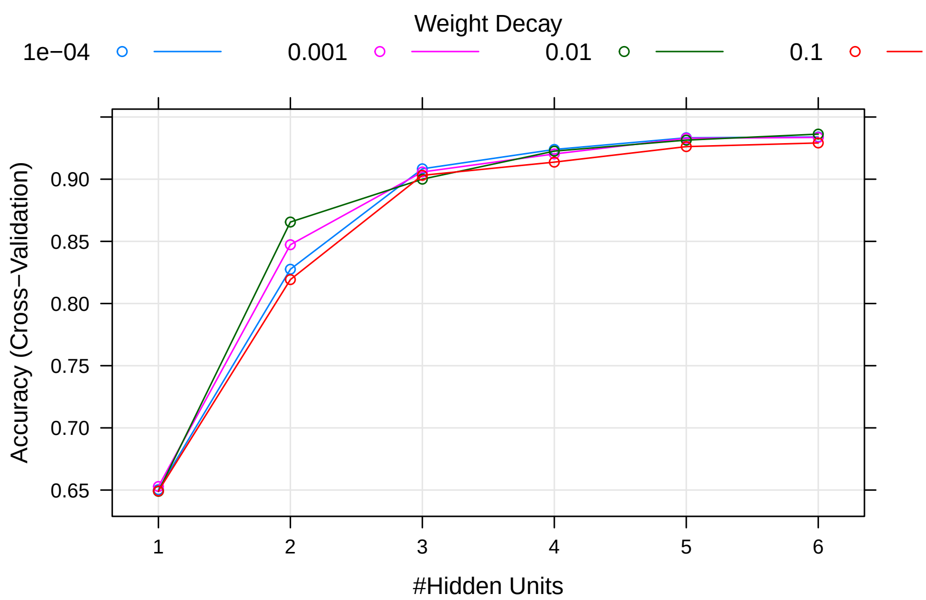
```
##   k  Accuracy     Kappa AccuracySD     KappaSD
## 9 9 0.9271343 0.9076466 0.00581194 0.007421194
```

```r
postResample(predict(KNN,newdata=HOLDOUT),HOLDOUT$type) # accuracy: 0.9126761
```

```
##  Accuracy     Kappa
## 0.9126761 0.8896666
```

```r
# Single layer Neural Net
nnetGrid <- expand.grid(size=1:6,decay=c(0.1,0.01,0.001,0.0001) )
set.seed(23);  NNET <- train(type~.,data=TRAIN,method='nnet',trControl=fitControl,tuneGrid=nnetGrid,
              trace=FALSE,linout=FALSE,preProc = c("center", "scale"))

# NNET  #Look at details of all fits
plot(NNET) #See how error changes with choices
```




```r
NNET$bestTune #Gives best parameters
```

```
##    size decay
## 23    6  0.01
```

```r
# NNET$results #Look at output in more detail (lets you see SDs)
NNET$results[rownames(NNET$bestTune),] # accuracy: 0.9362594 , SD: 0.009229208
```

```
##    size decay  Accuracy     Kappa  AccuracySD    KappaSD
## 23    6  0.01 0.9362594 0.9193189 0.009229208 0.01175718
```

```r
postResample(predict(NNET,newdata=HOLDOUT),HOLDOUT$type) # accuracy: 0.9242567
```

```
##  Accuracy     Kappa
## 0.9242567 0.9044476
```

```r
confusionMatrix(predict(NNET,newdata=HOLDOUT),HOLDOUT$type)
```

```
## Confusion Matrix and Statistics
##
##             Reference
```

```
## Prediction X10  X2  X3  X4  X7  X8  X9
##        X10 847   0   0   0   0   6  72
##        X2    0 460   3   0   0   1   0
##        X3    0   1  48   0   2   0   0
##        X4    1   0  10 365  28   0   0
##        X7    0   0  11  16 238   0   0
##        X8    3   0   1   0   0 197   2
##        X9   79   0   0   4   0   2 798
##
## Overall Statistics
##
##                Accuracy : 0.9243
##                  95% CI : (0.9145, 0.9332)
##     No Information Rate : 0.2911
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9044
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: X10 Class: X2 Class: X3 Class: X4 Class: X7
## Sensitivity              0.9108    0.9978   0.65753    0.9481   0.88806
## Specificity              0.9656    0.9985   0.99904    0.9861   0.99078
## Pos Pred Value           0.9157    0.9914   0.94118    0.9035   0.89811
## Neg Pred Value           0.9634    0.9996   0.99205    0.9928   0.98976
## Prevalence               0.2911    0.1443   0.02285    0.1205   0.08388
## Detection Rate           0.2651    0.1440   0.01502    0.1142   0.07449
## Detection Prevalence     0.2895    0.1452   0.01596    0.1264   0.08294
## Balanced Accuracy        0.9382    0.9982   0.82829    0.9671   0.93942
##                      Class: X8 Class: X9
## Sensitivity            0.95631    0.9151
## Specificity            0.99799    0.9634
## Pos Pred Value         0.97044    0.9037
## Neg Pred Value         0.99699    0.9680
## Prevalence             0.06448    0.2729
## Detection Rate         0.06166    0.2498
## Detection Prevalence   0.06354    0.2764
## Balanced Accuracy      0.97715    0.9393
```

```
# 4) final model with all training data
# going to go with a random forest, it had the second best accuracy of the models and was very close
# to the best. Run time also factored into this decision
set.seed(23); FINAL <- train(type~., data=DATA, method="rf",
                             tuneGrid=expand.grid(mtry=3),
                             trControl=trainControl(method="none"))
```

```
predictions <- predict(FINAL,TEST)
submit <- data.frame(pitchID=TEST$pitchid,
                     prediction_type=as.integer(substring(as.character(predictions),2)))
summary(submit)
```

```
##     pitchID      prediction_type
```

```
##  Min.   :10648   Min.   : 2.000
##  1st Qu.:13736   1st Qu.: 4.000
##  Median :16825   Median : 9.000
##  Mean   :16825   Mean   : 7.108
##  3rd Qu.:19913   3rd Qu.:10.000
##  Max.   :23001   Max.   :10.000
```

```r
write.csv(submit,"pitch_type_predictions.csv", row.names = FALSE)
```

Question 4

Part A

Response: There are several components to be considered before making a recommendation on the sensors. From a data analyst perspective, I would want to know "what will this data actually tell us?", "how will we use it?", and "what is the value of the sensor information?" and we could then compare this with the price of the sensors. I would want to see how well the sensor tracks player movements and captures distance run, speed, and exertion. Specifically, I would like to see how the sensor tracks "ground covered" in order to help us generate a range metric to gauge how well our athletes can "go get a ball." By having this information, I could provide updates and recommendations to player positioning on defense beyond current approaches. Finally, I would want to know if the GPS sensors also can provide us with exertion/biometric measures to help with additional uses such as recovery plans and performance. If I am provided with this information and answers to my questions, I would be comfortable providing a recommendation on whether or not we should purchase the sensors.

Part B

Response: *Assuming all of the raw location data includes time stamps* we can calculate the maximum acceleration using the following steps:

1. Using the haversine formula (would utilize r formula and google) I would determine the distance covered in between each of the time stamps
2. Once we have the distance covered, I would calculate the velocity of our runner between each of the time stamps by $\Delta d/\Delta t$
3. Finally, I would either (depending on the end user's preference):
   - calculate the average acceleration between each of our time stamps by $\Delta v/\Delta t$, or
   - using a velocity vs time graph, calculate instantaneous accelerations using calculus and limits
4. Report maximum acceleration of our player