MLB Pitcher’s Salary Prediction

James Colton Behannon

December 14, 2020

Table of Contents

Introduction

MLB players are some of the highest paid athletes of any sport with pitchers bringing home the most per position. Pitchers are expected to dictate the game and are therefore considered deserving of their premium salary. But how much of their salary is actually directly tied to their performance? In this paper I will attempt to predict pitcher salary within three quantiles (low, medium, and high pay) using their performance statistics for that season.

Q0:

The Data Set

The data sets being used come from Sean Lahman's baseball database. The database covers pitching, hitting, and fielding performance from 1871 to 2019. It has been integrated into R as a package and is what this paper will be utilizing. The specific data sets drawn from are "Pitching", "People", and "Salaries". Salaries only has salary information until 2016 and will therefore be the most recent year used in the analysis (years 2003 – 2016). The data set obtained from merging Pitching, People, and Salaries, using the selected features, results in 6104 observations (after removing three with NA values) and 28 features. These features will be broken down by the individual data sets they were drawn from.

Features: Pitching

From the Pitching data set a total of 28 variables are utilized. playerID and yearID are used to merge the table with People and Salaries. The remaining 26 variables are stint, W, L, G, GS, CG, SHO, SV, IPouts, H, ER, HR, BB, SO, BAOpp, ERA, IBB, WP, HBP, BK, BFP, GF, R, SH, SF, and GIDP.

Features: People

From the People data set a total of three variables are obtained. playerID is used as the key to merge with the other tables while weight and height are features.

Features: Salaries

From the Salaries data set only three variables are used. The first two, playerID and yearID, are once again used to merge this table with the other two. The remaining variable, salary, is what will be utilized to obtain the target variable of salary bracket.

Glossary:

playerID – Unique identifier for each player

yearID – Year the data was collected

weight – The player’s weight in pound

height – The player’s height in inches

debut – The date of the player’s first game

salary – The salary received by the player

stint – Order of appearances within a season

teamID – Team

lgID – The league they are in

W – Wins

L – Losses

G – Games

GS – Games started

CG – Complete games

SHO – Shutouts

SV – Saves

IPouts – Outs pitched

H – Hits

ER – Earned runs

HR – Homeruns

BB – Walks

SO – Strikeouts

BAOpp – Opponent’s batting average (number of hits against / number of at bats)

ERA – Earned run average

IBB – Intentional walks

WP – Wild pitches

HBP – Batters hit by pitch

BK – Balks

BFP – Batters faced by the pitcher

GF – Games finished

R – Runs allowed

SH – Sacrifices by opposing batters

SF – Sacrifice flies by opposing batters

GIDP – Grounded into double plays by batters

Preparation

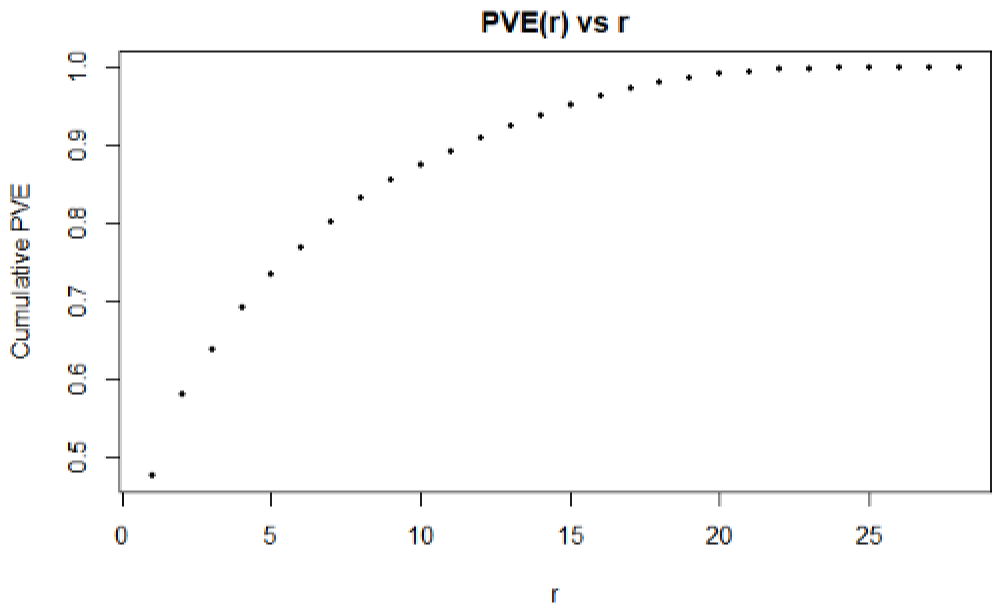
The data set containing all of the previously mentioned features contains salary as a numeric value. For the purposes of this project this must be transformed into three classes (class1, class2, and class3) which represent low, medium, and high salary, respectively. However, before this can be done the salaries must be normalized to account for inflation due to the different years in which the data comes from (2003 – 2016). In order to make this conversion the CPI (Consumer Price Index) values from these years are obtained from the BLS (U.S. Bureau of Labor and Statistics)[1]. The values will be converted to 2016 dollars. The formula used is as follows with CPI(j) referring to the CPI of each year that is referenced in the data set. The adjustment factor is then multiplied by each salary in order to obtain the inflation adjusted salary, or salary in 2016 dollars. The salaries are then ready to be divided into three quantiles and labeled into their corresponding classes.

$$\frac{CPI(2016) - CPI(j)}{CPI(j)} + 1 = Adjustment$$

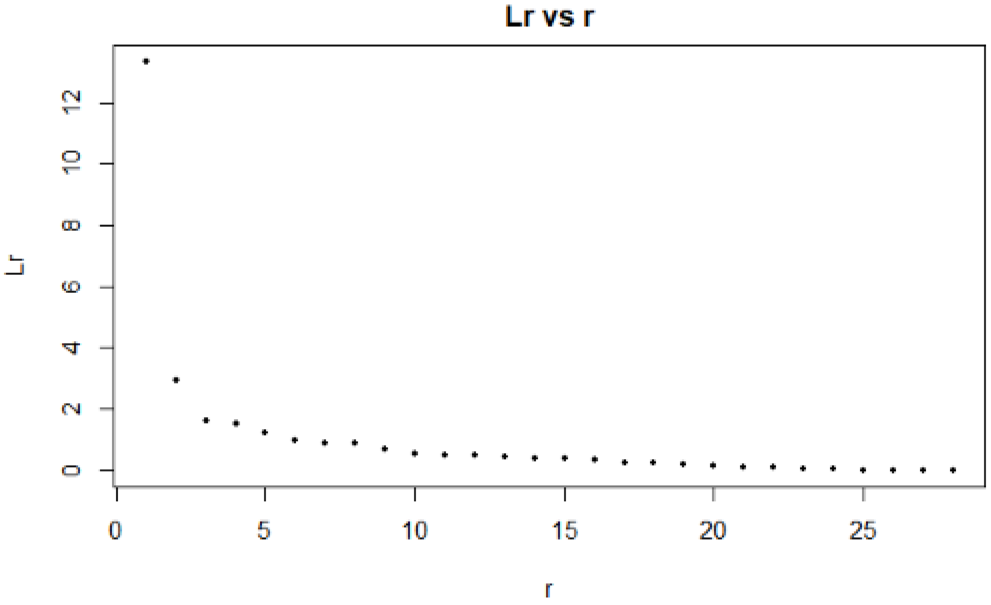
Q1:

Principle component analysis (PCA) is performed on the standardized features SDATA. PCA is a method of dimensionality-reduction that transfers large data sets into a more practical size with minimal information loss. To examine the results, we can take a look at the plot of proportion of variance explained vs r (number of principle components) and the eigenvalues vs r.

[1] Bureau of Labor and Statistics CPI data - (https://data.bls.gov/pdq/SurveyOutputServlet).

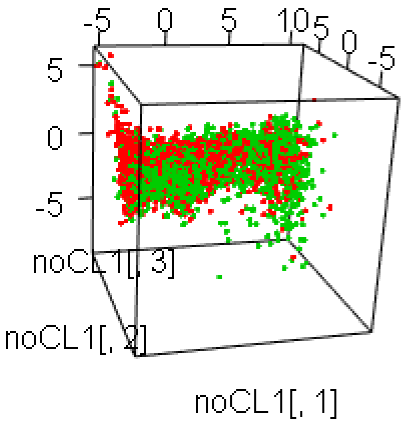


From here we can see that 95% of the variance is explained by the first 15 principle components.

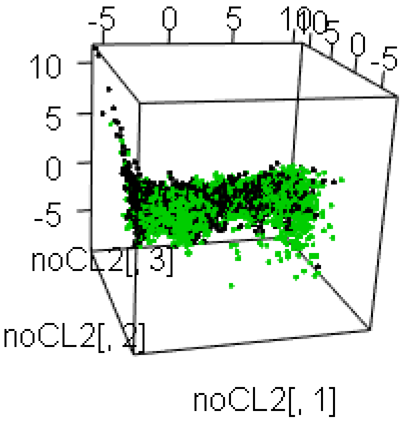


From this plot we can see the decrease in the eigenvalues as they progress. This plot is not cumulative such as the prior one as well.

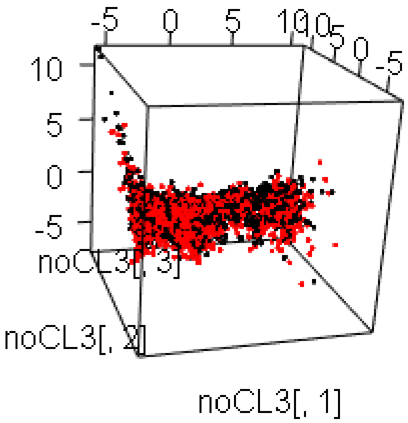
3D projection of class2(red) and class3(green)



3D projection of class1(black) and class3(green)



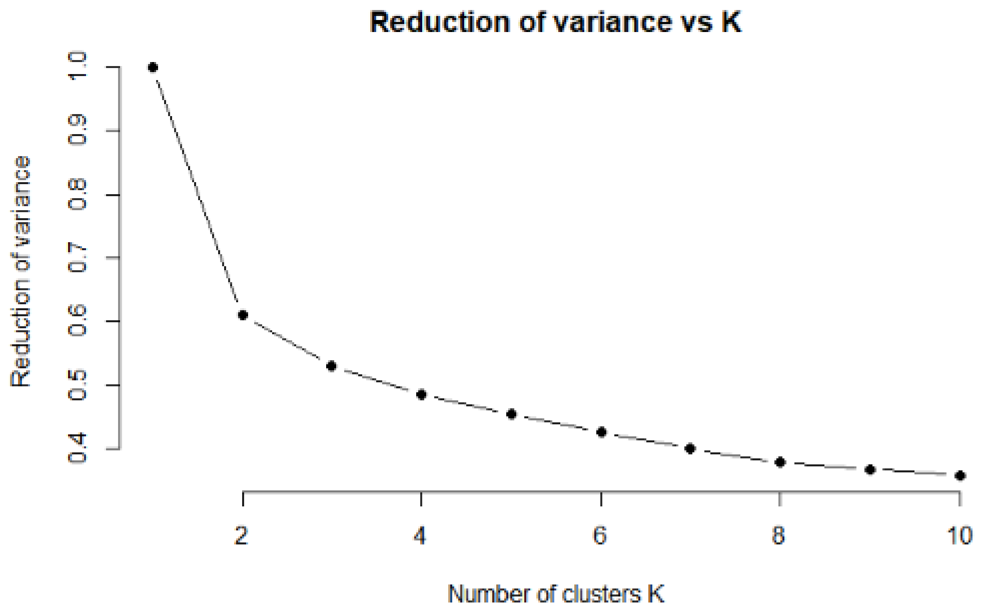
3D projection of class1(black) and class2(red)

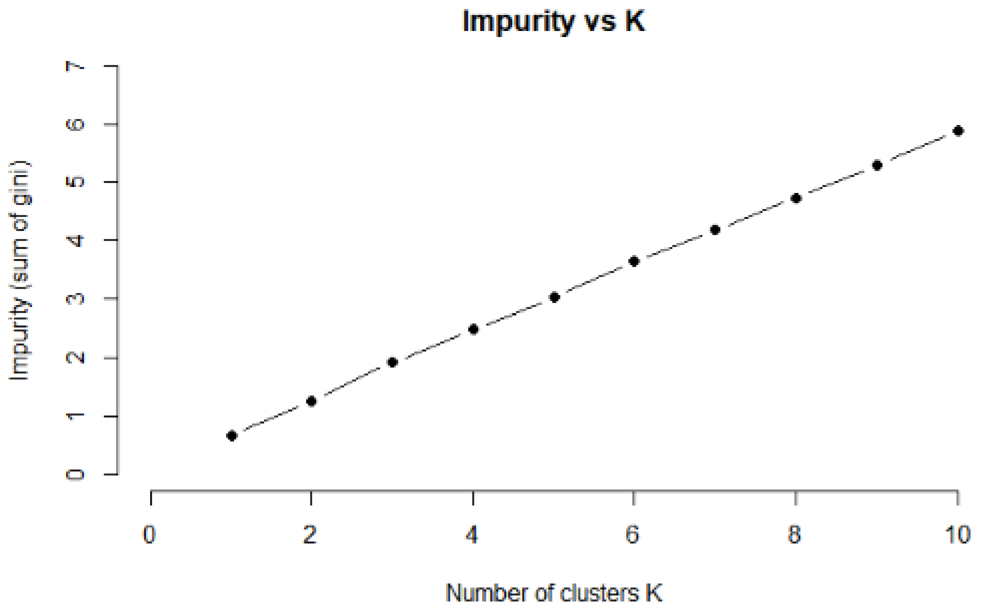


The pairs of classes are projected into three dimensions on the first three PCA eigenvectors. The resulting 3D plots present no easily distinguishable patterns with which to separate the classes.

Q2:

Kmeans clustering is an unsupervised algorithm for separating data into K clusters of “similar” standing. The algorithm identifies K centroids and then sorts the data into its nearest cluster. Kmeans was applied to the standardized data set with K = [1:10]. The following plots were obtained to display the reduction of variance and gini index vs K.





From examining the two plots we arrive at a best K value K* of 8. At K = 8 the impurity continues to increase at a steady rate while the reduction of variance slows down its reduction to a minimal amount.

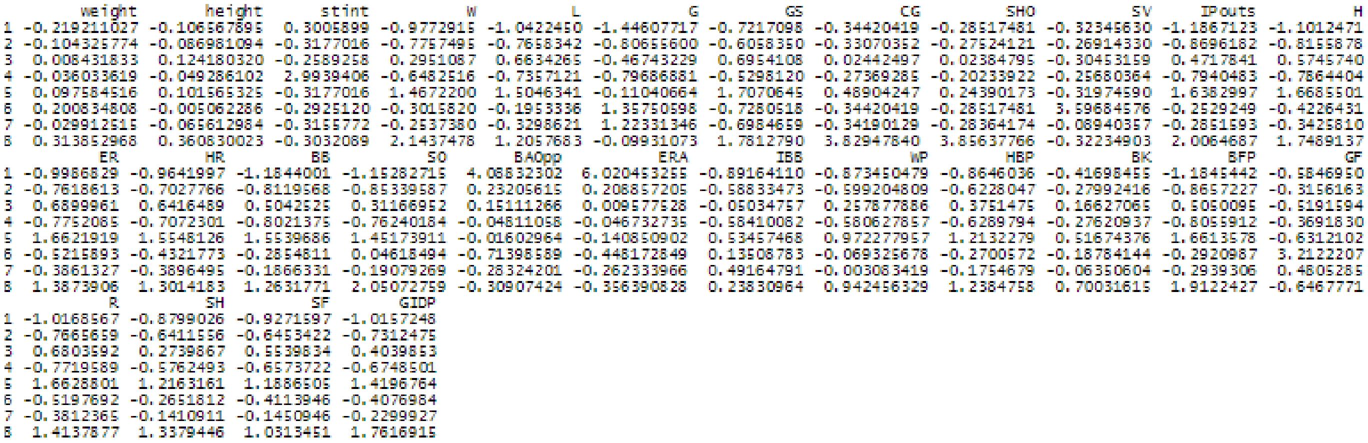
The computation time for each K was as follows:
K1: 0.521 seconds
K2: 0.669 seconds

K3: 0.846 seconds
K4: 1.028 seconds
K5: 1.331 seconds
K6: 1.563 seconds
K7: 1.873 seconds
K8: 2.574 seconds
K9: 2.381 seconds
K10: 2.578 seconds

Q3:

The following is obtained for kmeans with K=8

<u>Cluster Centers</u> –

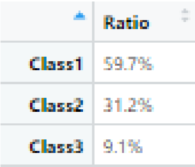


```
        weight       height      stint          W          L           G         GS          CG         SHO          SV     IPouts          H
1 -0.219211027 -0.106567895  0.3005899 -0.9772915 -1.0422450 -1.44607717 -0.7217098 -0.34420419 -0.28517481 -0.32345630 -1.1867123 -1.1012471
2 -0.104325774 -0.086981094 -0.3177016 -0.7757495 -0.7658342 -0.80655600 -0.6058350 -0.33070352 -0.27524121 -0.26914330 -0.8696182 -0.8155878
3  0.008431833  0.124180320 -0.2589258  0.2951087  0.6634265 -0.46743229  0.6954108  0.02442497  0.02384795 -0.30453159  0.4717841  0.5745740
4 -0.036033619 -0.049286102  2.9939406 -0.6482516 -0.7357121 -0.79686881 -0.5298120 -0.27369285 -0.20233922 -0.25680364 -0.7940483 -0.7864404
5  0.097584516  0.101565325 -0.3177016  1.4672200  1.5046341 -0.11040664  1.7070645  0.48904247  0.24390173 -0.31974590  1.6382997  1.6685501
6  0.200834808 -0.005062286 -0.2925120 -0.3015820 -0.1953336  1.35750598 -0.7280518 -0.34420419 -0.28517481  3.59684576 -0.2529249 -0.4226431
7 -0.029912515 -0.065612984 -0.3155772 -0.2537380 -0.3298621  1.22331346 -0.6984659 -0.34190129 -0.28364174 -0.08940357 -0.2851593 -0.3425810
8  0.313852968  0.360830023 -0.3032089  2.1437478  1.2057683 -0.09931073  1.7812790  3.82947840  3.85637766 -0.32234903  2.0064687  1.7489137
          ER         HR         BB          SO       BAOpp          ERA         IBB           WP        HBP          BK        BFP         GF
1 -0.9986829 -0.9641997 -1.1844001 -1.15282715  4.08832302  6.020453255 -0.89164110 -0.873450479 -0.8646036 -0.41698455 -1.1845442 -0.5846950
2 -0.7618613 -0.7027766 -0.8119568 -0.85339587  0.23205615  0.208857205 -0.58833473 -0.599204809 -0.6228047 -0.27992416 -0.8657227 -0.3156163
3  0.6899961  0.6416489  0.5042525  0.31166952  0.15111266  0.009577528 -0.05034757  0.257877886  0.3751475  0.16627065  0.5050095 -0.5191594
4 -0.7752085 -0.7072301 -0.8021375 -0.76240184 -0.04811058 -0.046732735 -0.58410082 -0.580627857 -0.6289794 -0.27620937 -0.8055912 -0.3691830
5  1.6621919  1.5548126  1.5539686  1.45173911 -0.01602964 -0.140850902  0.53457468  0.972277957  1.2132279  0.51674376  1.6613578 -0.6312102
6 -0.5215893 -0.4321773 -0.2854811  0.04618494 -0.71398589 -0.448172849  0.13508783 -0.069325678 -0.2700572 -0.18784144 -0.2920987  3.2122207
7 -0.3861327 -0.3896495 -0.1866331 -0.19079269 -0.28324201 -0.262333966  0.49164791 -0.003083419 -0.1754679 -0.06350604 -0.2939306  0.4805285
8  1.3873906  1.3014183  1.2631771  2.05072759 -0.30907424 -0.356390828  0.23830964  0.942456329  1.2384758  0.70031615  1.9122427 -0.6467771
           R         SH         SF       GIDP
1 -1.0168567 -0.8799026 -0.9271597 -1.0157248
2 -0.7665659 -0.6411556 -0.6453422 -0.7312475
3  0.6803592  0.2739867  0.5539834  0.4039853
4 -0.7719589 -0.5762493 -0.6573722 -0.6748501
5  1.6628801  1.2163161  1.1886505  1.4196764
6 -0.5197692 -0.2651812 -0.4113946 -0.4076984
7 -0.3812365 -0.1410911 -0.1450946 -0.2299927
8  1.4137877  1.3379446  1.0313451  1.7616915
```

Cluster	Sizes	Dispersion	Gini
1	77	1976.92	0.5377
2	1614	11243.80	0.6388
3	756	8866.34	0.6454
4	553	4379.70	0.6423
5	1013	17655.68	0.6072
6	378	3196.68	0.5798
7	1494	12014.42	0.6319
8	219	5345.67	0.4453

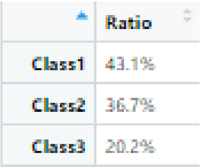
Frequencies –

Cluster 1:



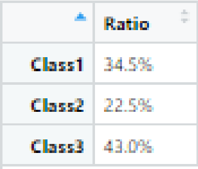
	Ratio
Class1	59.7%
Class2	31.2%
Class3	9.1%

Cluster 2:



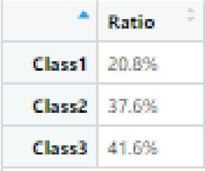
	Ratio
Class1	43.1%
Class2	36.7%
Class3	20.2%

Cluster 3:



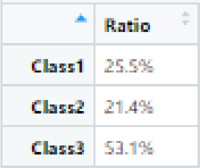
	Ratio
Class1	34.5%
Class2	22.5%
Class3	43.0%

Cluster 4:



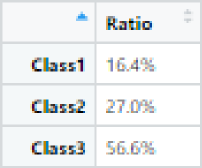
	Ratio
Class1	20.8%
Class2	37.6%
Class3	41.6%

Cluster 5:



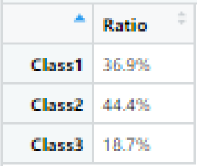
	Ratio
Class1	25.5%
Class2	21.4%
Class3	53.1%

Cluster 6:



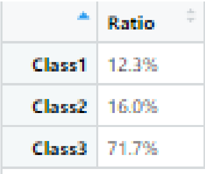
	Ratio
Class1	16.4%
Class2	27.0%
Class3	56.6%

Cluster 7:

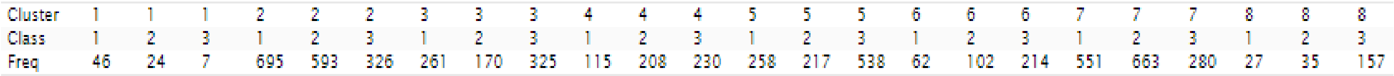


	Ratio
Class1	36.9%
Class2	44.4%
Class3	18.7%

Cluster 8:

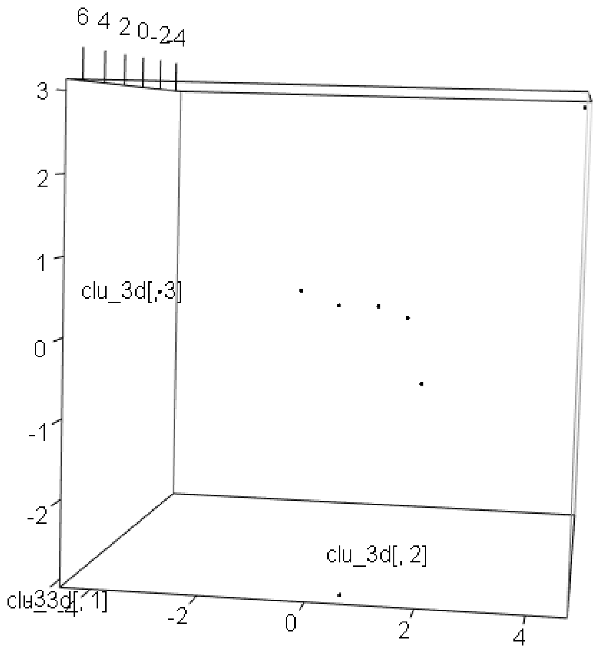


	Ratio
Class1	12.3%
Class2	16.0%
Class3	71.7%



Cluster	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8
Class	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Freq	46	24	7	695	593	326	261	170	325	115	208	230	258	217	538	62	102	214	551	663	280	27	35	157

3D Projection of Cluster Centers –

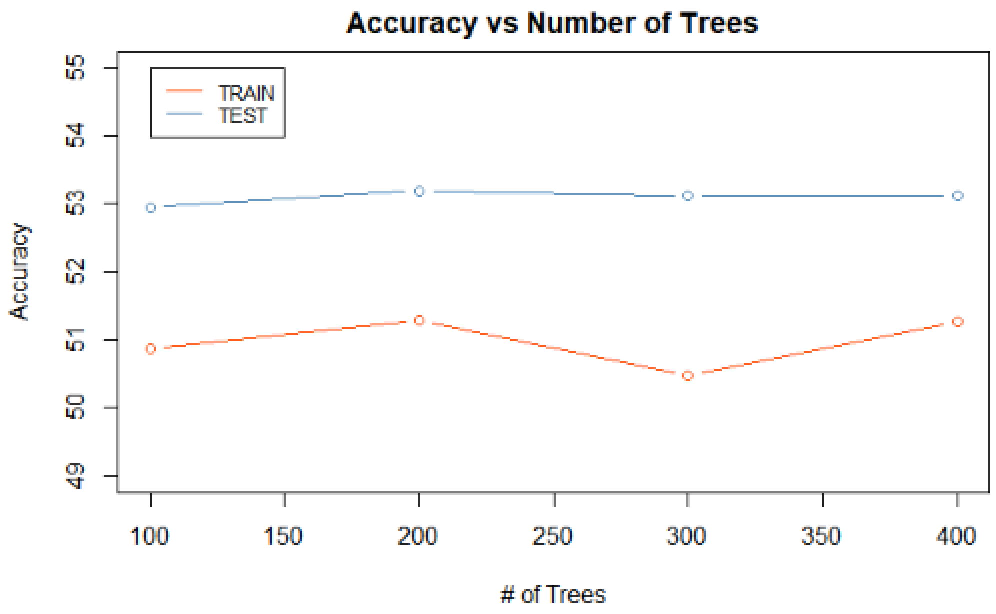


Q4:

The classes within the data set were built to be roughly 33% quantile increments and are therefore balanced enough to not require cloning. The train set TRAIN and test set TEST are created using an 80% vs 20% split.

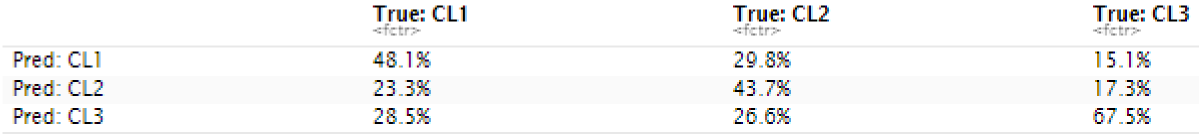
Q5:

A decision tree is a predictive modeling algorithm that uses basic “decisions” based on the data to lead to conclusions. A random forest is a supervised learning algorithm that makes uses of several decision trees in order to combat any overfitting that standard decision trees are likely to fall prey to. Random forest is applied to the train set with ntry = 5 (sqrt(29) = 5.385) and ntrees = {100, 200, 300, 400}. The resulting train and test accuracies can be seen in relation to each ntrees values with the plot.



The results within the test sets can be seen within these confusion matrices.

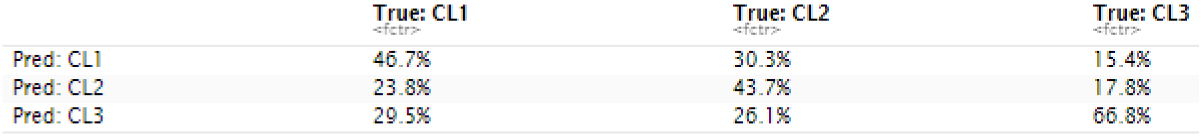
ntrees = 100



	True: CL1	True: CL2	True: CL3
	<fctr>	<fctr>	<fctr>
Pred: CL1	48.1%	29.8%	15.1%
Pred: CL2	23.3%	43.7%	17.3%
Pred: CL3	28.5%	26.6%	67.5%

The random forest with ntrees = 100 accurately predicted class1 48.1% of the time, class2 43.7% of the time, and class3 67.5% of the time with an overall classification accuracy of 52.95%. The computation time was 10.629 seconds.

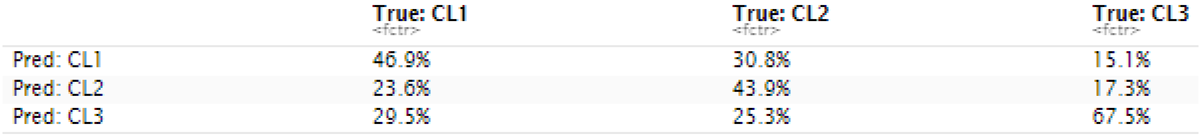
ntrees = 200



	True: CL1	True: CL2	True: CL3
	<fctr>	<fctr>	<fctr>
Pred: CL1	46.7%	30.3%	15.4%
Pred: CL2	23.8%	43.7%	17.8%
Pred: CL3	29.5%	26.1%	66.8%

The random forest with ntrees = 200 accurately predicted class1 46.7% of the time, class2 43.7% of the time, and class3 66.8% of the time with an overall classification accuracy of 53.19%. The computation times was 10.452 seconds.

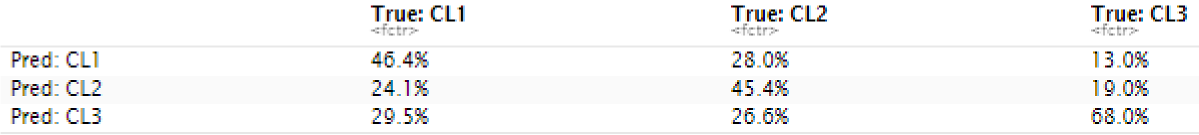
ntrees = 300



	True: CL1	True: CL2	True: CL3
	<fctr>	<fctr>	<fctr>
Pred: CL1	46.9%	30.8%	15.1%
Pred: CL2	23.6%	43.9%	17.3%
Pred: CL3	29.5%	25.3%	67.5%

The random forest with ntrees = 300 accurately predicted class1 46.9% of the time, class2 43.9% of the time, and class3 67.5% of the time with an overall classification accuracy of 53.11%. The computation time was 10.759 seconds.

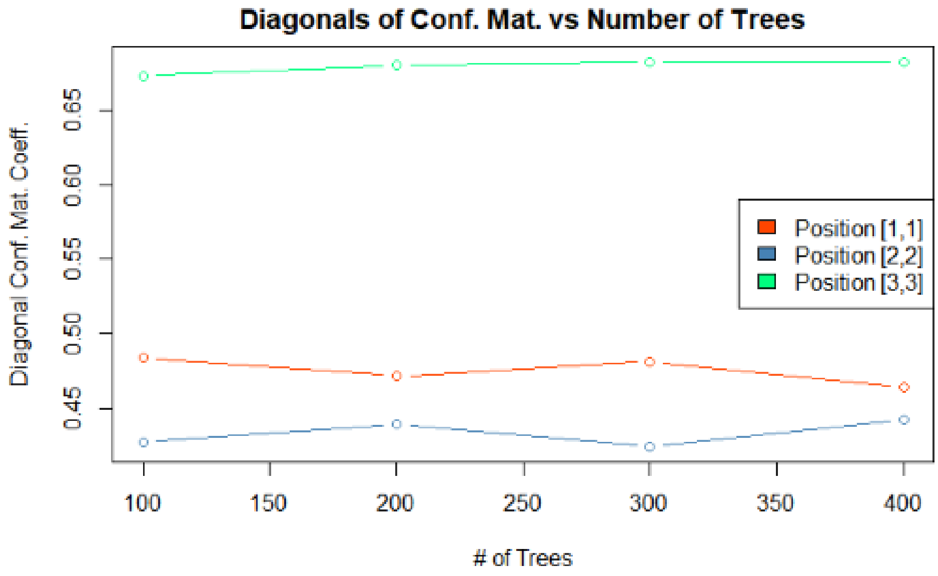
ntrees = 400



	True: CL1	True: CL2	True: CL3
	<fctr>	<fctr>	<fctr>
Pred: CL1	46.4%	28.0%	13.0%
Pred: CL2	24.1%	45.4%	19.0%
Pred: CL3	29.5%	26.6%	68.0%

The random forest with ntrees = 400 accurately predicted class1 46.4% of the time, class2 45.4% of the time, and class3 68% of the time with an overall classification accuracy of 53.11%. The computation time was 10.602 seconds.

Q6:



Position[3,3] (class3) consistently performs the best out of the three diagonals. Class3 has the most observations within TRAIN and therefore the model had more information to train off of. However, the difference in observations between class3, the largest class, and class2, the smallest class, is only 65 observations while the difference in accurate classification of the two is around 20%. If this trend held true, any extra observations that could be added might have significant results on the model's performance.

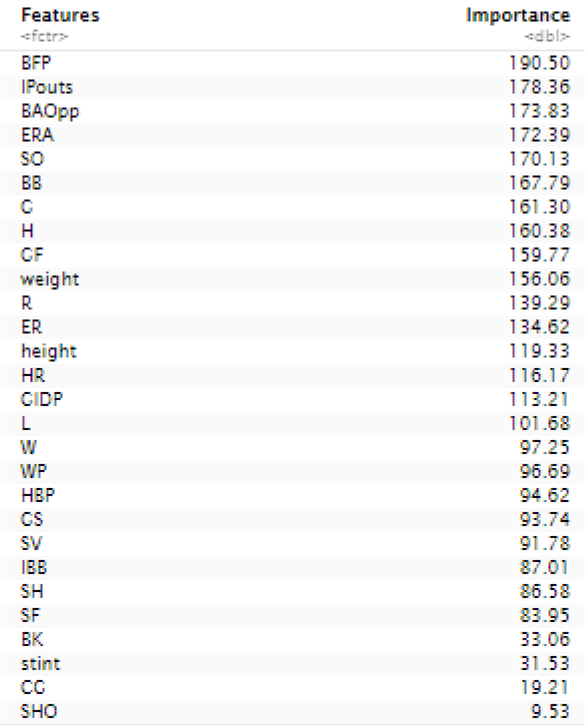
The best ntrees BNT is 200. The performance from position[1,1] and position[2,2] alternate between one increasing and the other decreasing with each new ntrees making it hard to choose the best option. Position[3,3] continues to increase in performance with more ntrees but only slightly. The difference between ntrees = 200 to 400 is minimal in position[3,3]'s performance so the smallest ntrees = 200 is chosen. It should be noted that performance for all of these options were very similar and the selected BNT could have potentially been any of the options with only slightly varied results.

From referencing the off-diagonal terms of the confusion matrices, we can see a similar trend in classification accuracy. When the true class is class1, class3 is incorrectly predicted less than class2 is. The same holds true when looking at predictions of class2. When the true class is class2, class3 is incorrectly predicted less frequently than class1.

Q7:

Variable importance is calculated from the mean decrease in gini for each feature from the random forest with ntry = 5 and ntrees = 200. These values were calculated and reordered to display their descending outputs.

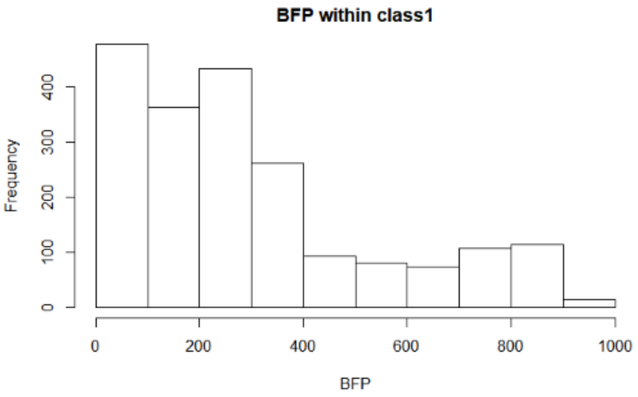
Features order by decreasing importance –

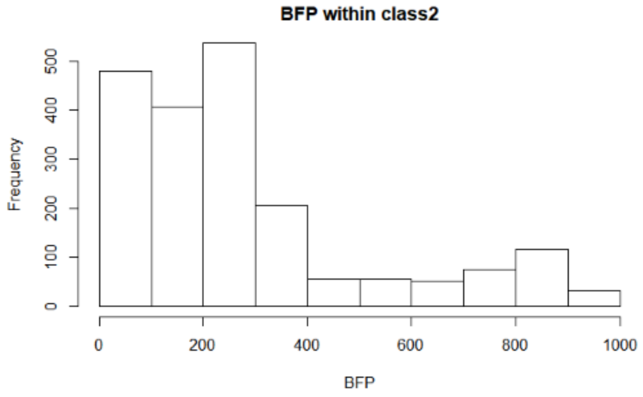


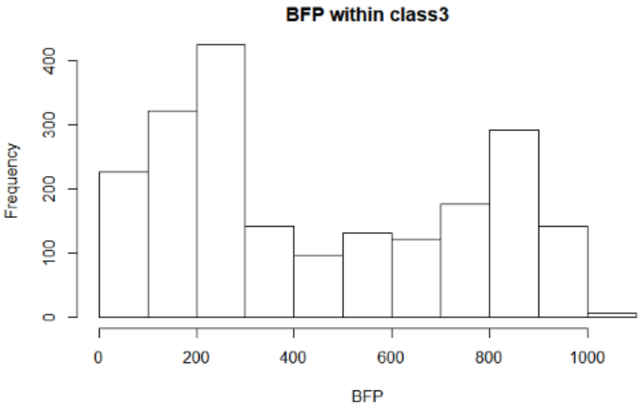
Features <fctr>	Importance <dbl>
BFP	190.50
IPouts	178.36
BAOpp	173.83
ERA	172.39
SO	170.13
BB	167.79
G	161.30
H	160.38
GF	159.77
weight	156.06
R	139.29
ER	134.62
height	119.33
HR	116.17
GIDP	113.21
L	101.68
W	97.25
WP	96.69
HBP	94.62
GS	93.74
SV	91.78
IBB	87.01
SH	86.58
SF	83.95
BK	33.06
stint	31.53
CG	19.21
SHO	9.53

Q8:

BFP (Most important feature) -



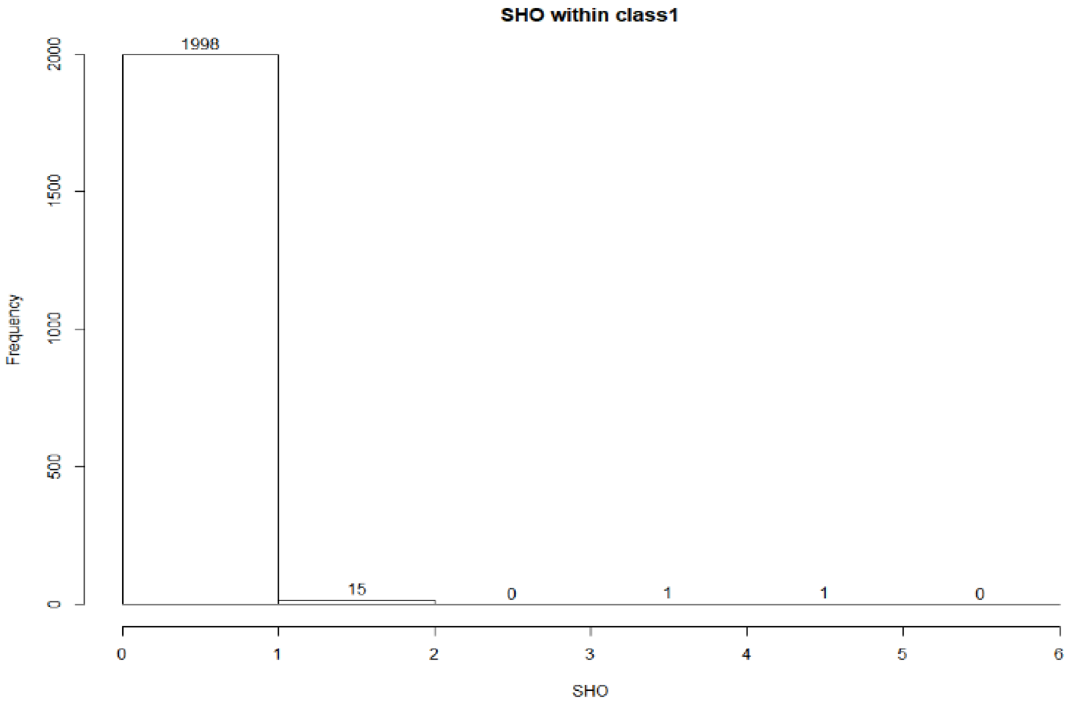


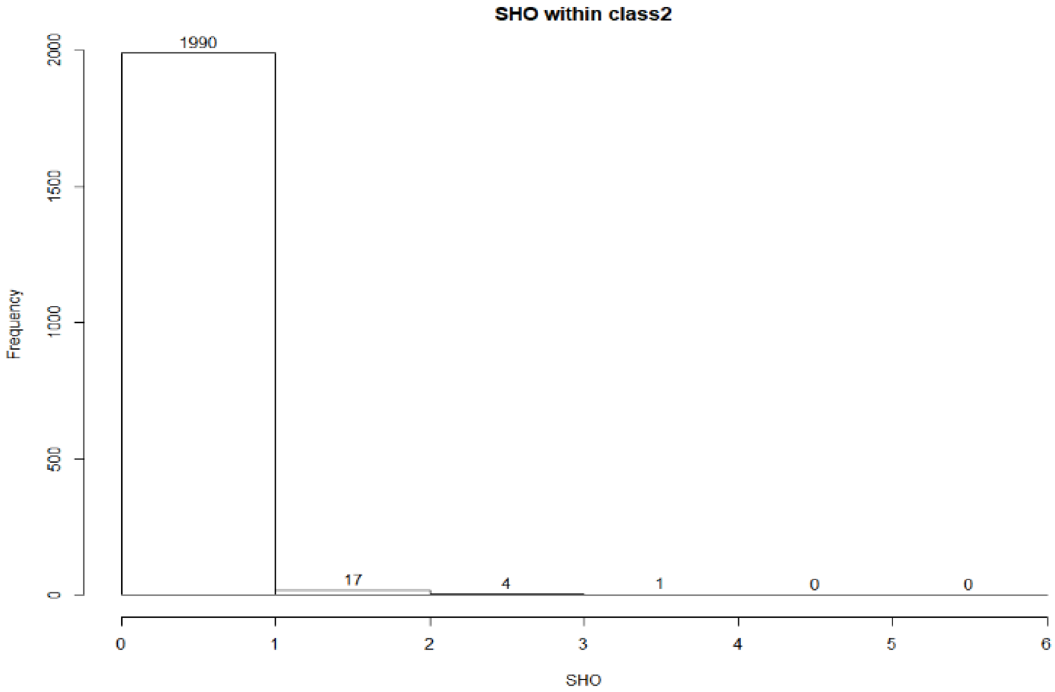


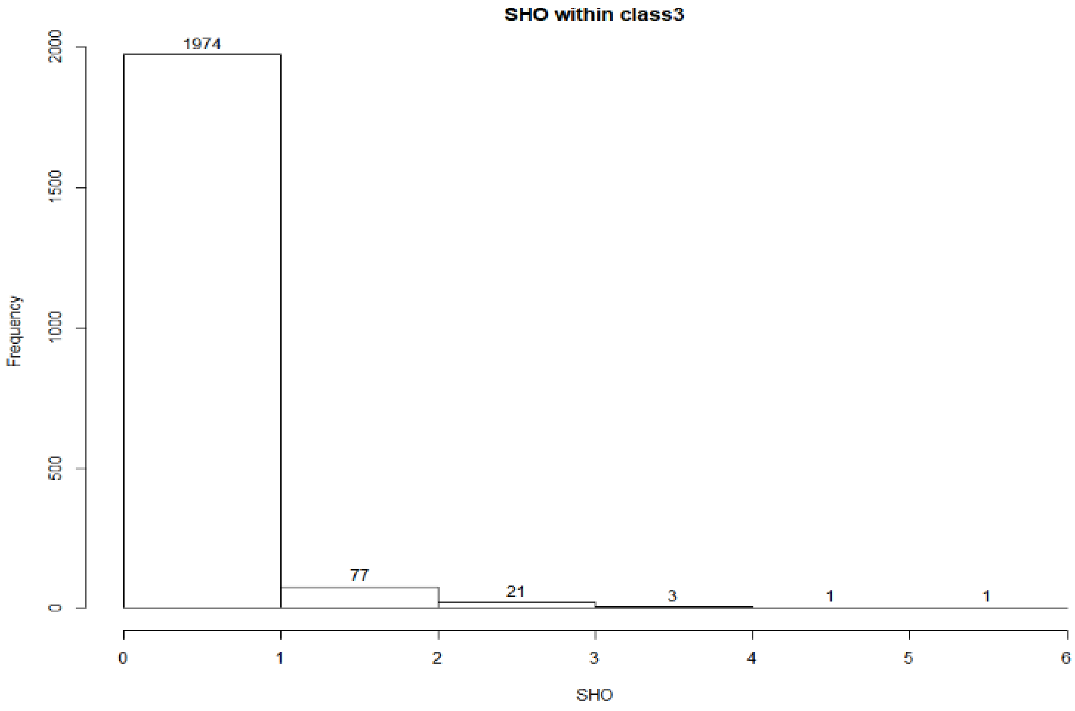
BFP represents the amount of batters faced by a pitcher. This is directly related to pitching experience and therefore makes sense that it would be important in determining a pitcher's salary. From looking at the histograms it is difficult to notice a difference between class1 and class2. However, class3 has noticeably more cases within the higher BFP values (specifically greater than 500) than class1 or class2.

The KS test to compare the histograms confirms the variables importance. The comparison of BFP within class1 and class2 received a p-value (P<.001). The comparisons between class1 and class3 as well as class2 and class3 each contained p-values of (P<.001) as well. It should be noted however that the p-value for both comparisons involving class3 were much smaller (P<2.2e-16) than the already small p-value for the comparison of class1 and class2 (P=1.783e-05).

SHO (Least important feature) –







SHO represents the number of shutouts that a pitcher has earned. This would seem like an important statistic for determining the prowess of a pitcher and therefore how much they should be earning. However, the nature of shutouts makes it a hard tool to use. They are extremely rare, and the maximum amount recorded within our data set is 6. From looking at the histograms there is a slight difference in the distributions as pitchers of class2 and then class3 obtain more shutouts than those within the previous class. The difference between class1 and class2 is only five more pitchers that have earned at least one shutout and is pretty insignificant. Class3 shows the most difference among the three with a higher prevalence of at least one shutout than the other two classes (though they are still extremely rare). Class3 contains 103 pitchers with one or more shutouts while class2 contains 22 and class1 contains 17.

The KS test reveals similar findings to what was seen within the histograms. Class1 and class2 are deemed not significantly different with a p-value of P(=.9995). However, the comparison of class1 and class3 as well as class2 and class3 obtained a significant p-value less than .001 (P=4.408e-09 and 3.549e-11, respectively).

Q9:

Cluster 8 from question 3 had the minimum gini. A new random forest with ntry = 5 and ntrees = 200 is trained based solely on the occurrences within this cluster. However, before this can be done, we need to clone classes 1 and 2 as there are far fewer cases of these two classes than class 3. There were only 15% as many cases of class1 as class3 so six copies of the class1 observations are made, each with minor perturbations to the feature values, and combined with the original observations. The same is done for class2 as there are only 26% as many cases of class2 as there are for class3. Three duplicates of class2, also with minor perturbations to each feature value, are combined with the original observations. The formula followed to create the perturbations within the feature values is as follows:

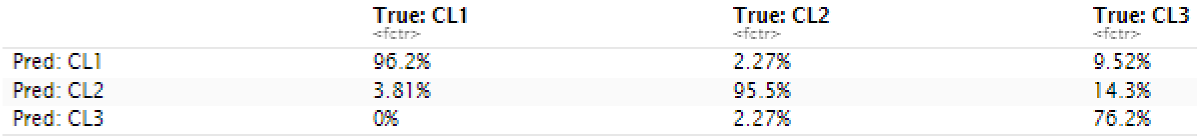
$$ClonedFeature = originalFeature \cdot (1 + (runif(1) - 0.5)/10000)$$

Here, runif(1) refers to a random decimal in the range [0, 1].

The final class sizes come out to be 119 observations from class1, 112 observations from class2, and 130 observations from class3. The same process is repeated for the test set and the final sizes come out to be 30 observations from class1, 28 observations from class2, and 27 observations from class3. A random forest can now be trained and tested on these new sets.

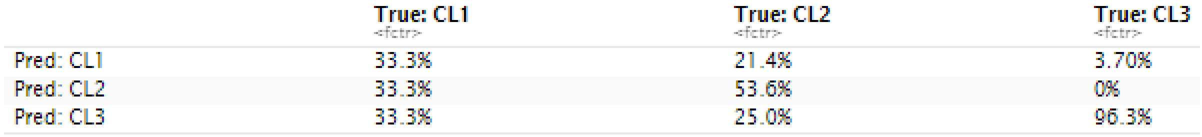
Q10:

The resulting model had a classification accuracy of 86.05% on the train set and 60% on the test set. The model performed well on the train set as would be expected. The computation time to train the model was 0.598 seconds.



	True: CL1 <fctr>	True: CL2 <fctr>	True: CL3 <fctr>
Pred: CL1	96.2%	2.27%	9.52%
Pred: CL2	3.81%	95.5%	14.3%
Pred: CL3	0%	2.27%	76.2%

However, the test set had interesting results.



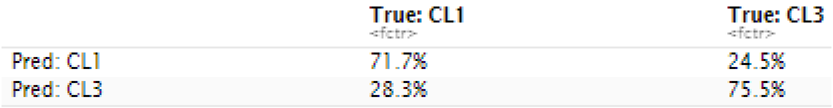
	True: CL1 <fctr>	True: CL2 <fctr>	True: CL3 <fctr>
Pred: CL1	33.3%	21.4%	3.70%
Pred: CL2	33.3%	53.6%	0%
Pred: CL3	33.3%	25.0%	96.3%

The model accurately predicted class3 96.3% of the time, class2 53.6% of the time, and class1 a mere 33.3% of the time. After repeated sampling and reruns of the random forest the results were all very similar. Class3 maintains a very high classification rate while 2 is smaller but still acceptable and class1 performs terribly. In a few iterations it would even accurately predict class1 less times than 33.3% meaning it performed worse than would be expected given a random selection. These results could potentially be explained by the class density that was originally present in the cluster. Class3 was by far the most prevalent with class1 and class2 requiring extensive cloning. Class1 originally only contained 15% as many cases as class3 and this could help explain its poor classification.

Using the clusters to retrain random forests could potentially obtain better results than one on the whole set. As seen here class3 achieved remarkable results in its predictions. However, class1 did very poorly and class2 did similarly to the original model. Retraining a random forest for each cluster would have differing success depending on the densities within each cluster but could have a positive outcome when taking the amount of classifications per class per cluster into account.

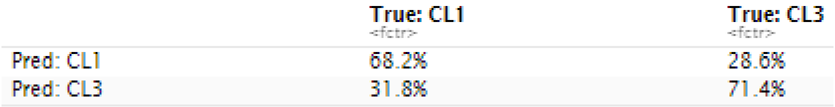
Q11:

A linear support vector machine (SVM) is an algorithm that separates data into distinct classes using a line or hyperplane and optimizes by finding the separator that has the largest margin (distance) between the nearest points to create a more robust classifier. A linear SVM was trained on a new training set of just class1 and class3 observations in order to classify between class1 and class3. The SVM obtained a 73.6% classification accuracy on the train set and 69.84% classification accuracy on the test set. The model took 3.862 seconds to train.



	True: CL1	True: CL3
	<fctr>	<fctr>
Pred: CL1	71.7%	24.5%
Pred: CL3	28.3%	75.5%

The SVM accurately predicted class1 71.7% of the time and class3 75.5% of the time on the train set.



	True: CL1	True: CL3
	<fctr>	<fctr>
Pred: CL1	68.2%	28.6%
Pred: CL3	31.8%	71.4%

The SVM accurately predicted class1 68.2% of the time and class3 71.4% of the time on the test set.

Code:

```
# install.packages("Lahman")

# Import the dataframes
data("People", package="Lahman")
data("Salaries", package="Lahman")
data("Pitching", package="Lahman")

# Merge the dataframes and filter it for years 2003-2016
peop <- People[c("playerID", "weight", "height", "debut")]
sal <- Salaries[c("playerID", "yearID", "salary")]
yearRange <- c(2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 20
13, 2014, 2015, 2016)

sal1016 <- sal[sal$"yearID" %in% yearRange,]

peopsal <- merge(peop, sal1016, by="playerID")
all <- merge(peopsal, Pitching, by=c("playerID", "yearID"))

# Normalize the salaries by accounting for inflation between years
# CPI 1982-84 = 100; data collected from https://data.bls.gov/pdq/SurveyOutpu
tServlet
# pct_change measures percent change in CPI between each year (2013-2016) and
2016
cpi = c(184.0, 188.9, 195.3, 201.6, 207.342, 215.303, 214.537, 218.056, 224.9
39, 229.594, 232.957, 236.736, 237.017, 240.007)
pct_change = c()
for (i in 1:length(cpi)) {
  pct_change = c(pct_change, (cpi[14] - cpi[i])/cpi[i]*100)
}
pct_change = (pct_change * 0.01) + 1

# Apply the percent changes in inflation to the player salaries
for (i in 1:nrow(all)) {
  if (all$yearID[i] == 2003) {
    all$salary_adj[i] = all$salary[i] * pct_change[1]
  }
  if (all$yearID[i] == 2004) {
    all$salary_adj[i] = all$salary[i] * pct_change[2]
  }
  if (all$yearID[i] == 2005) {
    all$salary_adj[i] = all$salary[i] * pct_change[3]
  }
  if (all$yearID[i] == 2006) {
    all$salary_adj[i] = all$salary[i] * pct_change[4]
  }
  if (all$yearID[i] == 2007) {
    all$salary_adj[i] = all$salary[i] * pct_change[5]
  }
  if (all$yearID[i] == 2008) {
    all$salary_adj[i] = all$salary[i] * pct_change[6]
  }
  if (all$yearID[i] == 2009) {
    all$salary_adj[i] = all$salary[i] * pct_change[7]
  }
  if (all$yearID[i] == 2010) {
    all$salary_adj[i] = all$salary[i] * pct_change[8]
  }
  if (all$yearID[i] == 2011) {
    all$salary_adj[i] = all$salary[i] * pct_change[9]
  }
  if (all$yearID[i] == 2012) {
    all$salary_adj[i] = all$salary[i] * pct_change[10]
  }
  if (all$yearID[i] == 2013) {
    all$salary_adj[i] = all$salary[i] * pct_change[11]
  }
  if (all$yearID[i] == 2014) {
    all$salary_adj[i] = all$salary[i] * pct_change[12]
  }
  if (all$yearID[i] == 2015) {
    all$salary_adj[i] = all$salary[i] * pct_change[13]
  }
  if (all$yearID[i] == 2016) {
    all$salary_adj[i] = all$salary[i] * pct_change[14]
  }
}
all$salary_adj = as.integer(all$salary_adj)

# Split the adjusted salaries into three quantiles
quantile(all$salary_adj, probs=c(0, 0.33, 0.66, 1))

##       0%      33%      66%     100%
##   305781   521496  2999630 33000000

quant33 <- nrow(all[all$salary_adj<=521496,])
quant66 <- nrow(all[all$salary_adj<=2999630 & all$salary_adj>521496,])
quant100 <- nrow(all[all$salary_adj<=33000000 & all$salary_adj>2999630,])

# Confirm we did not miss any of the observations
nrow(all[all$salary_adj<=521496,])+nrow(all[all$salary_adj<=2999630 & all$sal
ary_adj>521496,])+nrow(all[all$salary_adj<=33000000 & all$salary_adj>2999630,
])

## [1] 6107

# Create a new dataframe that contains the information classifying each salar
y into 1, 2, or 3 based on quantity
data <- as.data.frame(all)

for (i in 1:nrow(data)) {
  if (data$salary_adj[i] <= 521496) {
    data$class[i] = 1
  }
  if (data$salary_adj[i] <= 2999630 & data$salary_adj[i] > 521496) {
    data$class[i] = 2
  }
   if (data$salary_adj[i] <= 33000000 & data$salary_adj[i] > 2999630) {
    data$class[i] = 3
  }
}

# Check for missing values
apply(is.na(data), 2, which)
```

```
## $playerID
## integer(0)
##
## $yearID
## integer(0)
##
## $weight
## integer(0)
##
## $height
## integer(0)
##
## $debut
## integer(0)
##
## $salary
## integer(0)
##
## $stint
## integer(0)
##
## $teamID
## integer(0)
##
## $lgID
## integer(0)
##
## $W
## integer(0)
##
## $L
## integer(0)
##
## $G
## integer(0)
##
```

```
## $GS
## integer(0)
##
## $CG
## integer(0)
##
## $SHO
## integer(0)
##
## $SV
## integer(0)
##
## $IPouts
## integer(0)
##
## $H
## integer(0)
##
## $ER
## integer(0)
##
## $HR
## integer(0)
##
## $BB
## integer(0)
##
## $SO
## integer(0)
##
## $BAOpp
## [1] 5589
##
## $ERA
## [1] 4225 5494 5589
##
## $IBB
## integer(0)
##
## $WP
## integer(0)
##
## $HBP
## integer(0)
##
## $BK
## integer(0)
##
## $BFP
## integer(0)
```

```
##
## $GF
## integer(0)
##
## $R
## integer(0)
##
## $SH
## integer(0)
##
## $SF
## integer(0)
##
## $GIDP
## integer(0)
##
## $salary_adj
## integer(0)
##
## $class
## integer(0)

# Remove the 3 cases that contain missing values
data = na.omit(data)

# Check for missing values again
sum(is.na(data))

## [1] 0

# remove the salary and salary_adj variables and just leave the classes
salary = as.vector(data$salary)
data$salary = NULL
salary_adj = as.vector(data$salary_adj)
data$salary_adj = NULL

# Convert class to a factor
data$class <- as.factor(data$class)
class_vec = as.vector(data$class)

# Obtain a vector of playerID
playerID = as.vector(data$playerID)

# Create a list of the variables we want to use as features
vars <- c("weight", "height", "stint", "W", "L", "G", "GS", "CG", "SHO", "SV"
, "IPouts", "H", "ER", "HR", "BB", "SO", "BAOpp", "ERA", "IBB", "WP", "HBP",
"BK", "BFP", "GF", "R", "SH", "SF", "GIDP", "class")

# Filter the data to only hold the previously identified features
data <- data[,vars]
```

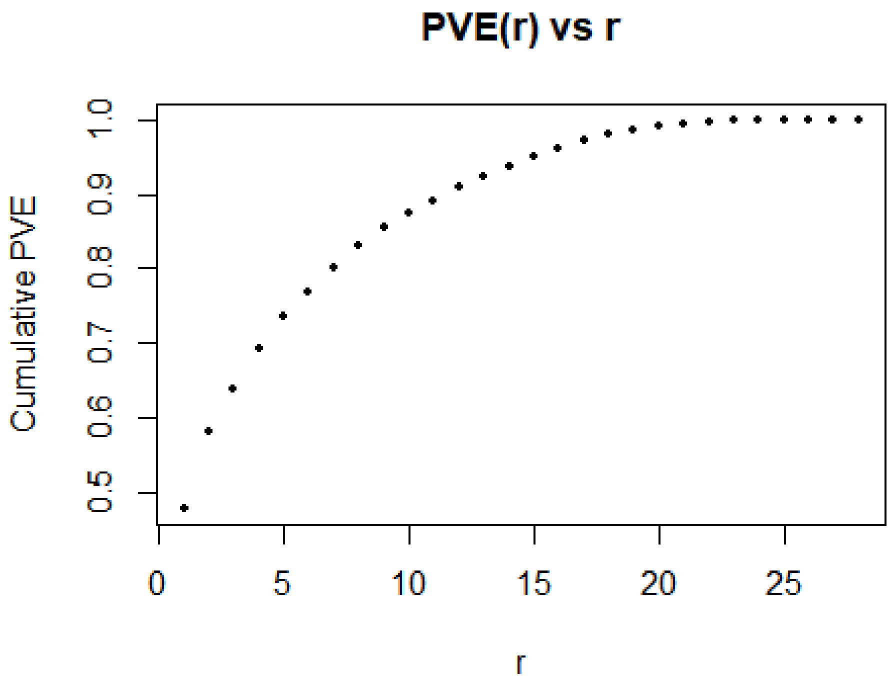
```
# Confirm we have all of correct vars
vars %in% colnames(data)
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [29] TRUE
```

There are now a total of 6104 observations. Time to standardize the data.

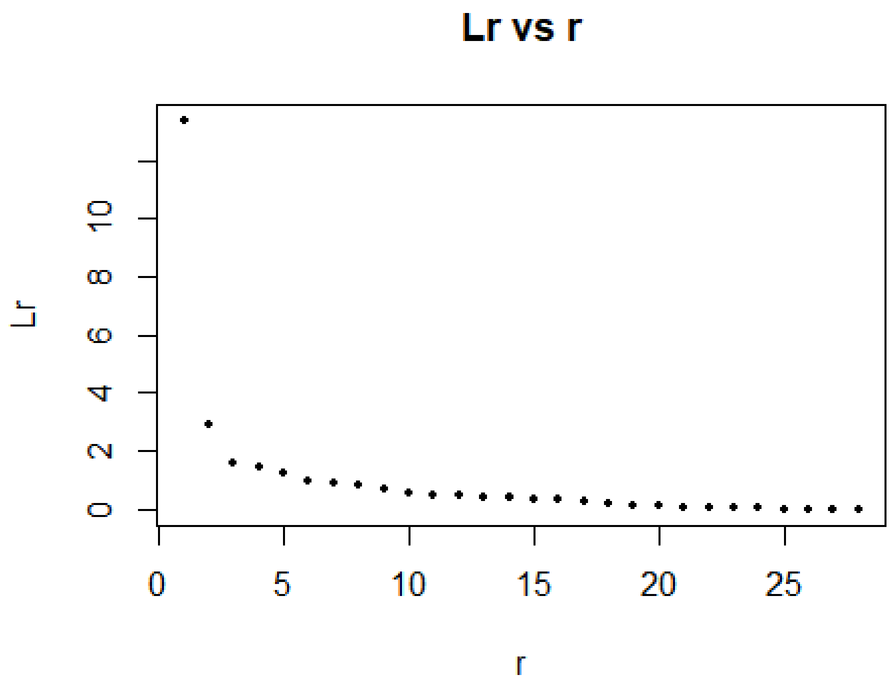
Q1

```
SDATA <- data.frame(scale(data[,-29]))
# SDATA$class <- class_vec

# PCA
SDATA.cor <- cor(SDATA)
SDATA.eig <- eigen(SDATA.cor)

# Variance explained
PVE <- SDATA.eig$values/sum(SDATA.eig$values)

# Plot PVE(r) vs r
it = 0
PVE.sum <- c()
for (i in 1:length(PVE)){
  PVE.sum <- sum(PVE[1:i])
  it = it + 1
  if (PVE.sum >= .95){
    break
  }
}

cumPVE <- plot(cumsum(PVE),pch=19, cex=.6, ylab = "Cumulative PVE", xlab = "r
", main = "PVE(r) vs r")
```



95% of the variance is explained by the first 15 principle componenets.

```
plot(SDATA.eig$values, pch=20, cex=.75, xlab="r", ylab="Lr", main="Lr vs r")
```



```
format(SDATA.eig$values[1:3], scientific=FALSE)

## [1] "13.371663" " 2.912144" " 1.594754"

PVE[1:3]

## [1] 0.47755940 0.10400516 0.05695552
```

The plot of eigenvalues as a function of r. The first three eigenvalues are 13.371663, 2.912144, and 1.594754. They explain 47.8%, 10.4%, and 5.7% of the the total variance respectively.

```
W <- SDATA.eig$vectors[,1:15]
PrinComp <- as.data.frame(as.matrix(SDATA) %*% W)
PrinComp$class <- class_vec

noCL1 <- PrinComp[PrinComp$class==2 | PrinComp$class==3,]
noCL2 <- PrinComp[PrinComp$class==1 | PrinComp$class==3,]
noCL3 <- PrinComp[PrinComp$class==1 | PrinComp$class==2,]

library(rgl)

## Warning: package 'rgl' was built under R version 3.6.3

plot3d(noCL1[,1], noCL1[,2], noCL1[,3], col=noCL1$class)
plot3d(noCL2[,1], noCL2[,2], noCL2[,3], col=noCL2$class)
plot3d(noCL3[,1], noCL3[,2], noCL3[,3], col=noCL3$class)
```

The three pairs of classes do not show any distinguishable evidence of seperability.

Q2

```
k1_start <- Sys.time()
k1 <- kmeans(SDATA, centers=1, nstart=50)
k1_end <- Sys.time()
k2_start <- Sys.time()
k2 <- kmeans(SDATA, centers=2, nstart=50)
k2_end <- Sys.time()
k3_start <- Sys.time()
k3 <- kmeans(SDATA, centers=3, nstart=50)
k3_end <- Sys.time()
k4_start <- Sys.time()
k4 <- kmeans(SDATA, centers=4, nstart=50)
k4_end <- Sys.time()
k5_start <- Sys.time()
k5 <- kmeans(SDATA, centers=5, nstart=50)
k5_end <- Sys.time()
k6_start <- Sys.time()
k6 <- kmeans(SDATA, centers=6, nstart=50)
k6_end <- Sys.time()
k7_start <- Sys.time()
k7 <- kmeans(SDATA, centers=7, nstart=50)
k7_end <- Sys.time()
```

```
k8_start <- Sys.time()
k8 <- kmeans(SDATA, centers=8, nstart=50)
k8_end <- Sys.time()
k9_start <- Sys.time()
k9 <- kmeans(SDATA, centers=9, nstart=50)
k9_end <- Sys.time()
k10_start <- Sys.time()
k10 <- kmeans(SDATA, centers=10, nstart=50)
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
k10_end <- Sys.time()

# Compute times
k1_end - k1_start
```

```
## Time difference of 0.1938782 secs
```

```
k2_end - k2_start
```

```
## Time difference of 0.397753 secs
```

```
k3_end - k3_start
```

```
## Time difference of 0.6256192 secs
```

```
k4_end - k4_start
```

```
## Time difference of 0.9484148 secs
```

```
k5_end - k5_start
```

```
## Time difference of 1.176275 secs
```

```
k6_end - k6_start
```

```
## Time difference of 1.365161 secs
```

```
k7_end - k7_start
```

```
## Time difference of 1.639989 secs
```

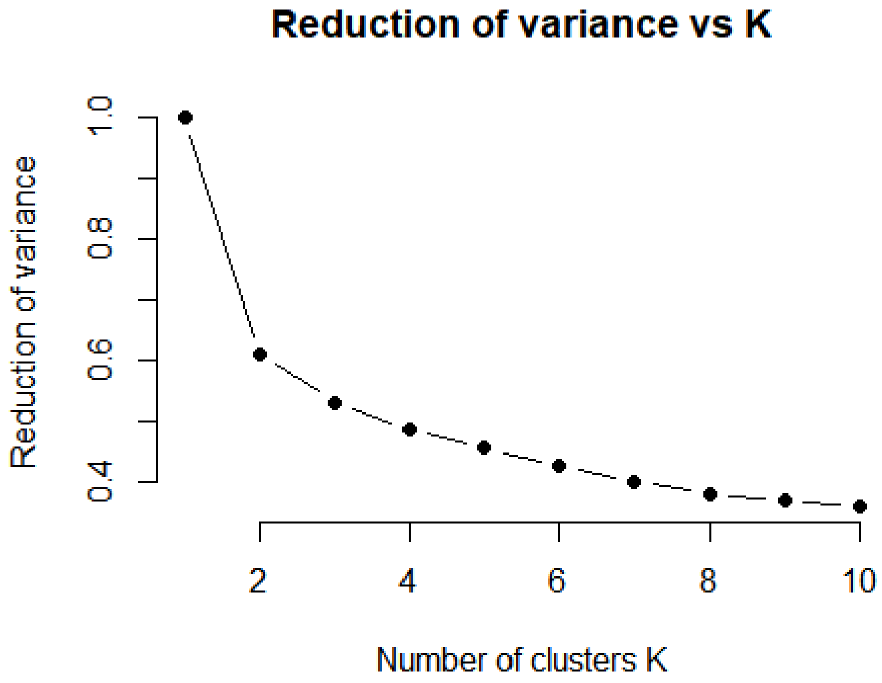
```
k8_end - k8_start
```

```
## Time difference of 2.055737 secs
```

```
k9_end - k9_start
```

```
## Time difference of 2.355561 secs
```

```
k10_end - k10_start

## Time difference of 2.593405 secs

var1 <- k1$betweenss/k1$totss
var2 <- k2$betweenss/k2$totss
var3 <- k3$betweenss/k3$totss
var4 <- k4$betweenss/k4$totss
var5 <- k5$betweenss/k5$totss
var6 <- k6$betweenss/k6$totss
var7 <- k7$betweenss/k7$totss
var8 <- k8$betweenss/k8$totss
var9 <- k9$betweenss/k9$totss
var10 <- k10$betweenss/k10$totss

var_values <- c(var1,var2,var3,var4,var5,var6,var7,var8,var9,var10)
k_values <- 1:10
plot(k_values, 1-var_values, type="b", pch = 19, frame = FALSE, xlab="Number
of clusters K", ylab="Reduction of variance", main = "Reduction of variance v
s K")
```



Best K = 8.

Q3

```
# Obtain cluster centers
k8$centers

##          weight       height      stint          W          L           G
## 1 -0.219211027 -0.106567895  0.3005899 -0.9772915 -1.0422450 -1.44607717
## 2  0.097584516  0.101565325 -0.3177016  1.4672200  1.5046341 -0.11040664
## 3 -0.036033619 -0.049286102  2.9939406 -0.6482516 -0.7357121 -0.79686881
## 4 -0.104325774 -0.086981094 -0.3177016 -0.7757495 -0.7658342 -0.80655600
## 5  0.008431833  0.124180320 -0.2589258  0.2951087  0.6634265 -0.46743229
## 6  0.200834808 -0.005062286 -0.2925120 -0.3015820 -0.1953336  1.35750598
## 7  0.313852968  0.360830023 -0.3032089  2.1437478  1.2057683 -0.09931073
## 8 -0.029912515 -0.065612984 -0.3155772 -0.2537380 -0.3298621  1.22331346
##           GS          CG         SHO          SV     IPouts          H
## 1 -0.7217098 -0.34420419 -0.28517481 -0.32345630 -1.1867123 -1.1012471
## 2  1.7070645  0.48904247  0.24390173 -0.31974590  1.6382997  1.6685501
## 3 -0.5298120 -0.27369285 -0.20233922 -0.25680364 -0.7940483 -0.7864404
## 4 -0.6058350 -0.33070352 -0.27524121 -0.26914330 -0.8696182 -0.8155878
## 5  0.6954108  0.02442497  0.02384795 -0.30453159  0.4717841  0.5745740
## 6 -0.7280518 -0.34420419 -0.28517481  3.59684576 -0.2529249 -0.4226431
## 7  1.7812790  3.82947840  3.85637766 -0.32234903  2.0064687  1.7489137
## 8 -0.6984659 -0.34190129 -0.28364174 -0.08940357 -0.2851593 -0.3425810
##           ER         HR         BB          SO       BAOpp          ERA
## 1 -0.9986829 -0.9641997 -1.1844001 -1.15282715  4.08832302  6.020453255
## 2  1.6621919  1.5548126  1.5539686  1.45173911 -0.01602964 -0.140850902
## 3 -0.7752085 -0.7072301 -0.8021375 -0.76240184 -0.04811058 -0.046732735
## 4 -0.7618613 -0.7027766 -0.8119568 -0.85339587  0.23205615  0.208857205
## 5  0.6899961  0.6416489  0.5042525  0.31166952  0.15111266  0.009577528
## 6 -0.5215893 -0.4321773 -0.2854811  0.04618494 -0.71398589 -0.448172849
## 7  1.3873906  1.3014183  1.2631771  2.05072759 -0.30907424 -0.356390828
## 8 -0.3861327 -0.3896495 -0.1866331 -0.19079269 -0.28324201 -0.262333966
##           IBB           WP        HBP          BK        BFP         GF
## 1 -0.89164110 -0.873450479 -0.8646036 -0.41698455 -1.1845442 -0.5846950
## 2  0.53457468  0.972277957  1.2132279  0.51674376  1.6613578 -0.6312102
## 3 -0.58410082 -0.580627857 -0.6289794 -0.27620937 -0.8055912 -0.3691830
## 4 -0.58833473 -0.599204809 -0.6228047 -0.27992416 -0.8657227 -0.3156163
## 5 -0.05034757  0.257877886  0.3751475  0.16627065  0.5050095 -0.5191594
## 6  0.13508783 -0.069325678 -0.2700572 -0.18784144 -0.2920987  3.2122207
## 7  0.23830964  0.942456329  1.2384758  0.70031615  1.9122427 -0.6467771
## 8  0.49164791 -0.003083419 -0.1754679 -0.06350604 -0.2939306  0.4805285
##            R         SH         SF       GIDP
## 1 -1.0168567 -0.8799026 -0.9271597 -1.0157248
## 2  1.6628801  1.2163161  1.1886505  1.4196764
## 3 -0.7719589 -0.5762493 -0.6573722 -0.6748501
## 4 -0.7665659 -0.6411556 -0.6453422 -0.7312475
## 5  0.6803592  0.2739867  0.5539834  0.4039853
## 6 -0.5197692 -0.2651812 -0.4113946 -0.4076984
## 7  1.4137877  1.3379446  1.0313451  1.7616915
## 8 -0.3812365 -0.1410911 -0.1450946 -0.2299927

# Obtain cluster sizes
k8$size

## [1]   77 1013  553 1614  756  378  219 1494
```

```
# Obtain dispersion
k8$withinss

## [1]  1976.924 17655.684  4379.702 11243.800  8866.344  3196.681  5345.665
## [8] 12014.417
```

Freqency table of classes within each cluster

```
# Create a frequency table of all clusters and classes
library(plyr)

cluster_vals = c()
for (i in 1:length(k8$cluster)) {
  cluster_vals <- c(cluster_vals, k8$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")

gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
```

Impurity calculation

```
gini_impurity <- c()

# k1 cluster values
cluster_vals = c()
for (i in 1:length(k1$cluster)) {
  cluster_vals <- c(cluster_vals, k1$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")
```

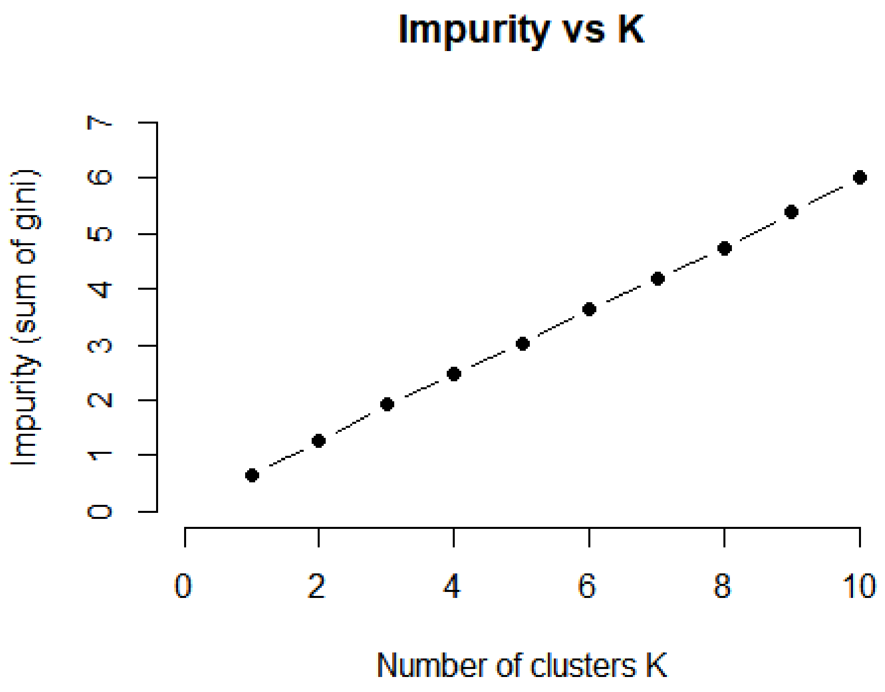
```
gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k2 cluster values
cluster_vals = c()
for (i in 1:length(k2$cluster)) {
  cluster_vals <- c(cluster_vals, k2$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")

gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k3 cluster values
cluster_vals = c()
for (i in 1:length(k3$cluster)) {
  cluster_vals <- c(cluster_vals, k3$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")

gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k4 cluster values
cluster_vals = c()
for (i in 1:length(k4$cluster)) {
  cluster_vals <- c(cluster_vals, k4$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")

gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
```

```
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k5 cluster values
cluster_vals = c()
for (i in 1:length(k5$cluster)) {
  cluster_vals <- c(cluster_vals, k5$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")

gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k6 cluster values
cluster_vals = c()
for (i in 1:length(k6$cluster)) {
  cluster_vals <- c(cluster_vals, k6$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")
gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k7 cluster values
cluster_vals = c()
for (i in 1:length(k7$cluster)) {
  cluster_vals <- c(cluster_vals, k7$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")
gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k8 cluster values
cluster_vals = c()
for (i in 1:length(k8$cluster)) {
  cluster_vals <- c(cluster_vals, k8$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")
gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k9 cluster values
cluster_vals = c()
for (i in 1:length(k9$cluster)) {
  cluster_vals <- c(cluster_vals, k9$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")
gini_per_clust <- c()
i=1
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))

#k10 cluster values
cluster_vals = c()
for (i in 1:length(k10$cluster)) {
  cluster_vals <- c(cluster_vals, k10$cluster[[i]])
}

clust_vs_class = cbind(cluster_vals, class_vec)

FREQ <- count(clust_vs_class, vars = c("cluster_vals", "class_vec"))
names(FREQ) <- c("Cluster", "Class", "Freq")
gini_per_clust <- c()
i=1
```

```
while (i <=  nrow(FREQ)) {
  class1 <- FREQ[i, 3]
  class2 <- FREQ[i+1, 3]
  class3 <- FREQ[i+2, 3]
  total <- class1+class2+class3
  class1_pct <- class1/total
  class2_pct <- class2/total
  class3_pct <- class3/total
  gini <- class1_pct*(1-class1_pct) + class2_pct*(1-class2_pct) + class3_pct*
(1 - class3_pct)
  gini_per_clust <- c(gini_per_clust, gini)
  i = i+3
}
gini_impurity <- c(gini_impurity, sum(gini_per_clust))
```

Plot impurity vs k

```
plot(k_values, gini_impurity, type="b", pch = 19, frame = FALSE, xlab="Number
of clusters K", ylab="Impurity (sum of gini)",xlim = c(0,10), ylim = c(0, 7),
main = "Impurity vs K")
```



```
library(scales)

class1 <- FREQ[1, 3]
class2 <- FREQ[2, 3]
class3 <- FREQ[3, 3]
total <- class1+class2+class3
```

```
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio1 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio1) <- "Ratio"

class1 <- FREQ[4, 3]
class2 <- FREQ[5, 3]
class3 <- FREQ[6, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio2 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio2) <- "Ratio"

class1 <- FREQ[7, 3]
class2 <- FREQ[8, 3]
class3 <- FREQ[9, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio3 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio3) <- "Ratio"

class1 <- FREQ[10, 3]
class2 <- FREQ[11, 3]
class3 <- FREQ[12, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio4 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio4) <- "Ratio"

class1 <- FREQ[13, 3]
class2 <- FREQ[14, 3]
class3 <- FREQ[15, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio5 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio5) <- "Ratio"
```

```
class1 <- FREQ[16, 3]
class2 <- FREQ[17, 3]
class3 <- FREQ[18, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio6 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio6) <- "Ratio"

class1 <- FREQ[19, 3]
class2 <- FREQ[20, 3]
class3 <- FREQ[21, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio7 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio7) <- "Ratio"

class1 <- FREQ[22, 3]
class2 <- FREQ[23, 3]
class3 <- FREQ[24, 3]
total <- class1+class2+class3
class1_pct <- class1/total
class2_pct <- class2/total
class3_pct <- class3/total
freq_ratio8 = data.frame(percent(c(class1_pct, class2_pct, class3_pct)), row.
names = c("Class1", "Class2", "Class3"))
names(freq_ratio8) <- "Ratio"
```

Create 3D plots of clusters on first 3 eigenvectors

```
library(rgl)
clu_3d <- as.data.frame(as.matrix(k8$centers) %*% W)
plot3d(clu_3d[,1], clu_3d[,2], clu_3d[,3])
```

Q4

```
# Obtain the training and testing data

# First split the data into classes
CL1 = as.data.frame(data[data$class == 1,])
CL2 = as.data.frame(data[data$class == 2,])
CL3 = as.data.frame(data[data$class == 3,])

# Obtain train/test split
trainCL1.index <- sample(1:nrow(CL1), 0.8 * nrow(CL1))
```

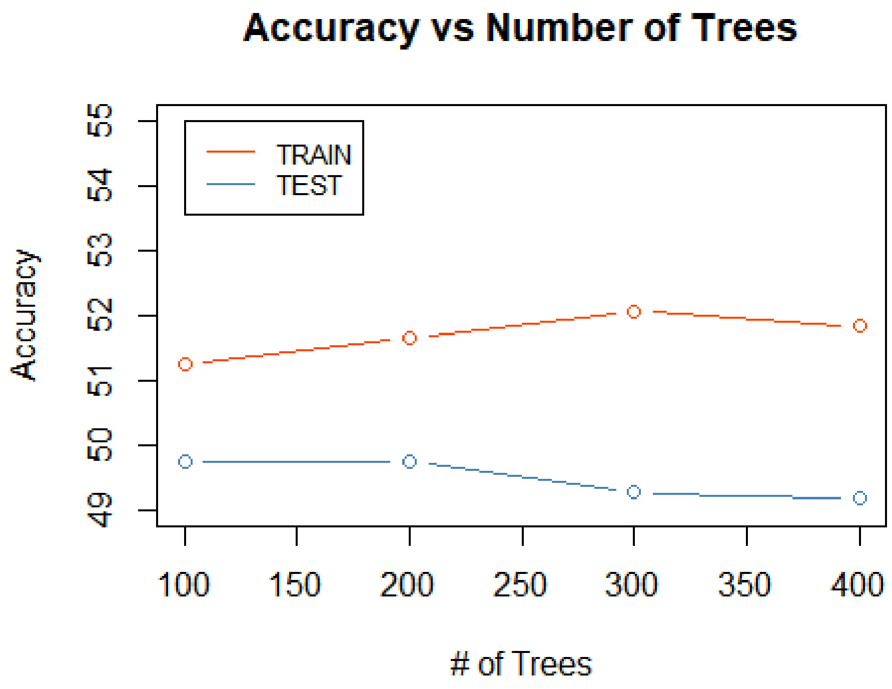
```r
testCL1.index <- setdiff(1:nrow(CL1), trainCL1.index)
trainCL1 <- CL1[trainCL1.index,]
testCL1 <- CL1[testCL1.index,]
# train.labels1 <- class_vec[trainCL1.index]
# test.labels1 <- class_vec[testCL1.index]
train.labels1 <- rep(1, nrow(trainCL1))
test.labels1 <- rep(2, nrow(testCL1))

trainCL2.index <- sample(1:nrow(CL2), 0.8 * nrow(CL2))
testCL2.index <- setdiff(1:nrow(CL2), trainCL2.index)
trainCL2 <- CL2[trainCL2.index,]
testCL2 <- CL2[testCL2.index,]
# train.labels2 <- class_vec[trainCL2.index]
# test.labels2 <- class_vec[testCL2.index]
train.labels2 <- rep(2, nrow(trainCL2))
test.labels2 <- rep(2, nrow(testCL2))

trainCL3.index <- sample(1:nrow(CL3), 0.8 * nrow(CL3))
testCL3.index <- setdiff(1:nrow(CL3), trainCL3.index)
trainCL3 <- CL3[trainCL3.index,]
testCL3 <- CL3[testCL3.index,]
# train.labels3 <- class_vec[trainCL3.index + nrow(CL2) + nrow(CL1)]
# test.labels3 <- class_vec[testCL3.index + nrow(CL2) + nrow(CL1)]
train.labels3 <- rep(3, nrow(trainCL3))
test.labels3 <- rep(3, nrow(testCL3))

TRAIN <- rbind(trainCL1, trainCL2, trainCL3)
TEST <- rbind(testCL1, testCL2, testCL3)
train.labels <- c(train.labels1, train.labels2, train.labels3)
test.labels <- c(test.labels1, test.labels2, test.labels3)

TRAIN_y <- TRAIN$class
TEST_y <- TEST$class

TRAIN <- scale(TRAIN[,-29])
TEST <- scale(TEST[,-29])

TRAIN <- as.data.frame(TRAIN)
TEST <- as.data.frame(TEST)

TRAIN$class <- TRAIN_y
TEST$class <- TEST_y
```

Q5

```r
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.6.3

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.
```

```
rf100_start <- Sys.time()
rf100 <- randomForest(TRAIN[,-29], TRAIN$class, ntry = 5, ntrees = 100)
rf100_end <- Sys.time()
rf200_start <- Sys.time()
rf200 <- randomForest(TRAIN[,-29], TRAIN$class, ntry = 5, ntrees = 200)
rf200_end <- Sys.time()
rf300_start <- Sys.time()
rf300 <- randomForest(TRAIN[,-29], TRAIN$class, ntry = 5, ntrees = 300)
rf300_end <- Sys.time()
rf400_start <- Sys.time()
rf400 <- randomForest(TRAIN[,-29], TRAIN$class, ntry = 5, ntrees = 400)
rf400_end <- Sys.time()

rf100_end - rf100_start

## Time difference of 11.09518 secs

rf200_end - rf200_start

## Time difference of 10.73739 secs

rf300_end - rf300_start

## Time difference of 10.64945 secs

rf400_end - rf400_start

## Time difference of 10.48955 secs

pred.test100 <- predict(rf100, TEST[,-29])
pred.test200 <- predict(rf200, TEST[,-29])
pred.test300 <- predict(rf300, TEST[,-29])
pred.test400 <- predict(rf400, TEST[,-29])

# Create function to obtain accuracy
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}

# Obtain the confusion matrices for predicting the train set using training d
ata
conf.mat.train100 <- rf100$confusion
conf.mat.train200 <- rf200$confusion
conf.mat.train300 <- rf300$confusion
conf.mat.train400 <- rf400$confusion

# Obtain the confusion matrices for predicting the test set using training da
ta
conf.mat.test100 <- table(pred.test100, TEST$class)
conf.mat.test200 <- table(pred.test200, TEST$class)
conf.mat.test300 <- table(pred.test300, TEST$class)
conf.mat.test400 <- table(pred.test400, TEST$class)
```

```
acc_train <- c(accuracy(conf.mat.train100), accuracy(conf.mat.train200), accu
racy(conf.mat.train300), accuracy(conf.mat.train400))
acc_test <- c(accuracy(conf.mat.test100), accuracy(conf.mat.test200), accurac
y(conf.mat.test300), accuracy(conf.mat.test400))
ntrees <- c(100, 200, 300, 400)

plot(ntrees, acc_train, type = "b", xlab = "# of Trees", ylab = "Accuracy", m
ain = "Accuracy vs Number of Trees", col = "orangered", ylim = c(49, 55))
lines(ntrees, acc_test, type = "b", col= "steelblue")
legend(100, 55, legend=c("TRAIN", "TEST"), col=c("orangered", "steelblue"), l
ty=1:1, cex=0.8)
```



```
# Display confusion matrices of the test set

# ntrees = 100
cl1pred1 <- percent(conf.mat.test100[1,1]/sum(conf.mat.test100[,1]))
cl1pred2 <- percent(conf.mat.test100[2,1]/sum(conf.mat.test100[,1]))
cl1pred3 <- percent(conf.mat.test100[3,1]/sum(conf.mat.test100[,1]))

cl2pred2 <- percent(conf.mat.test100[2,2]/sum(conf.mat.test100[,2]))
cl2pred1 <- percent(conf.mat.test100[1,2]/sum(conf.mat.test100[,2]))
cl2pred3 <- percent(conf.mat.test100[3,2]/sum(conf.mat.test100[,2]))

cl3pred3 <- percent(conf.mat.test100[3,3]/sum(conf.mat.test100[,3]))
cl3pred1 <- percent(conf.mat.test100[1,3]/sum(conf.mat.test100[,3]))
cl3pred2 <- percent(conf.mat.test100[2,3]/sum(conf.mat.test100[,3]))
```

```
row1 <- c(cl1pred1, cl2pred1, cl3pred1)
row2 <- c(cl1pred2, cl2pred2, cl3pred2)
row3 <- c(cl1pred3, cl2pred3, cl3pred3)


conf.matrix.percent.test100 <- rbind(row1, row2, row3)
rownames(conf.matrix.percent.test100) <- c("Pred: CL1", "Pred: CL2", "Pred: C
L3")
colnames(conf.matrix.percent.test100) <- c("True: CL1", "True: CL2", "True: C
L3")
as.data.frame(conf.matrix.percent.test100)
```

```
##           True: CL1 True: CL2 True: CL3
## Pred: CL1     37.5%     29.3%     15.9%
## Pred: CL2     30.3%     42.9%     15.9%
## Pred: CL3     32.3%     27.8%     68.3%
```

```
# ntrees = 200
cl1pred1 <- percent(conf.mat.test200[1,1]/sum(conf.mat.test200[,1]))
cl1pred2 <- percent(conf.mat.test200[2,1]/sum(conf.mat.test200[,1]))
cl1pred3 <- percent(conf.mat.test200[3,1]/sum(conf.mat.test200[,1]))

cl2pred2 <- percent(conf.mat.test200[2,2]/sum(conf.mat.test200[,2]))
cl2pred1 <- percent(conf.mat.test200[1,2]/sum(conf.mat.test200[,2]))
cl2pred3 <- percent(conf.mat.test200[3,2]/sum(conf.mat.test200[,2]))

cl3pred3 <- percent(conf.mat.test200[3,3]/sum(conf.mat.test200[,3]))
cl3pred1 <- percent(conf.mat.test200[1,3]/sum(conf.mat.test200[,3]))
cl3pred2 <- percent(conf.mat.test200[2,3]/sum(conf.mat.test200[,3]))

row1 <- c(cl1pred1, cl2pred1, cl3pred1)
row2 <- c(cl1pred2, cl2pred2, cl3pred2)
row3 <- c(cl1pred3, cl2pred3, cl3pred3)


conf.matrix.percent.test200 <- rbind(row1, row2, row3)
rownames(conf.matrix.percent.test200) <- c("Pred: CL1", "Pred: CL2", "Pred: C
L3")
colnames(conf.matrix.percent.test200) <- c("True: CL1", "True: CL2", "True: C
L3")
as.data.frame(conf.matrix.percent.test200)
```

```
##           True: CL1 True: CL2 True: CL3
## Pred: CL1     38.5%     30.0%     16.6%
## Pred: CL2     30.5%     43.4%     16.6%
## Pred: CL3     31.0%     26.6%     66.8%
```

```
# ntrees = 300
cl1pred1 <- percent(conf.mat.test300[1,1]/sum(conf.mat.test300[,1]))
cl1pred2 <- percent(conf.mat.test300[2,1]/sum(conf.mat.test300[,1]))
cl1pred3 <- percent(conf.mat.test300[3,1]/sum(conf.mat.test300[,1]))
```

```
cl2pred2 <- percent(conf.mat.test300[2,2]/sum(conf.mat.test300[,2]))
cl2pred1 <- percent(conf.mat.test300[1,2]/sum(conf.mat.test300[,2]))
cl2pred3 <- percent(conf.mat.test300[3,2]/sum(conf.mat.test300[,2]))

cl3pred3 <- percent(conf.mat.test300[3,3]/sum(conf.mat.test300[,3]))
cl3pred1 <- percent(conf.mat.test300[1,3]/sum(conf.mat.test300[,3]))
cl3pred2 <- percent(conf.mat.test300[2,3]/sum(conf.mat.test300[,3]))

row1 <- c(cl1pred1, cl2pred1, cl3pred1)
row2 <- c(cl1pred2, cl2pred2, cl3pred2)
row3 <- c(cl1pred3, cl2pred3, cl3pred3)


conf.matrix.percent.test300 <- rbind(row1, row2, row3)
rownames(conf.matrix.percent.test300) <- c("Pred: CL1", "Pred: CL2", "Pred: C
L3")
colnames(conf.matrix.percent.test300) <- c("True: CL1", "True: CL2", "True: C
L3")
as.data.frame(conf.matrix.percent.test300)
```

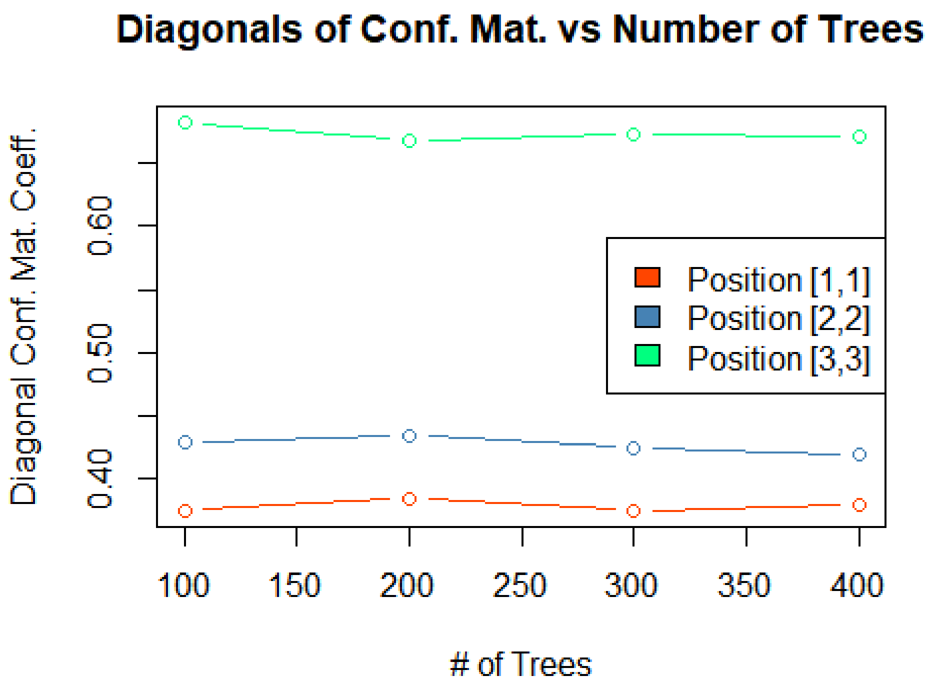
```
##           True: CL1 True: CL2 True: CL3
## Pred: CL1     37.5%     30.0%     16.3%
## Pred: CL2     31.3%     42.4%     16.3%
## Pred: CL3     31.3%     27.5%     67.3%
```

```
# ntrees = 400
cl1pred1 <- percent(conf.mat.test400[1,1]/sum(conf.mat.test400[,1]))
cl1pred2 <- percent(conf.mat.test400[2,1]/sum(conf.mat.test400[,1]))
cl1pred3 <- percent(conf.mat.test400[3,1]/sum(conf.mat.test400[,1]))

cl2pred2 <- percent(conf.mat.test400[2,2]/sum(conf.mat.test400[,2]))
cl2pred1 <- percent(conf.mat.test400[1,2]/sum(conf.mat.test400[,2]))
cl2pred3 <- percent(conf.mat.test400[3,2]/sum(conf.mat.test400[,2]))

cl3pred3 <- percent(conf.mat.test400[3,3]/sum(conf.mat.test400[,3]))
cl3pred1 <- percent(conf.mat.test400[1,3]/sum(conf.mat.test400[,3]))
cl3pred2 <- percent(conf.mat.test400[2,3]/sum(conf.mat.test400[,3]))

row1 <- c(cl1pred1, cl2pred1, cl3pred1)
row2 <- c(cl1pred2, cl2pred2, cl3pred2)
row3 <- c(cl1pred3, cl2pred3, cl3pred3)


conf.matrix.percent.test400 <- rbind(row1, row2, row3)
rownames(conf.matrix.percent.test400) <- c("Pred: CL1", "Pred: CL2", "Pred: C
L3")
colnames(conf.matrix.percent.test400) <- c("True: CL1", "True: CL2", "True: C
L3")
as.data.frame(conf.matrix.percent.test400)
```

```
##           True: CL1 True: CL2 True: CL3
## Pred: CL1     38.0%     30.0%     16.6%
## Pred: CL2     30.0%     41.9%     16.3%
## Pred: CL3     32.0%     28.0%     67.1%
```

Q6

```
diag1 <- c(conf.mat.test100[1,1]/sum(conf.mat.test100[,1]), conf.mat.test200[
1,1]/sum(conf.mat.test200[,1]), conf.mat.test300[1,1]/sum(conf.mat.test300[,1
]), conf.mat.test400[1,1]/sum(conf.mat.test400[,1]))
diag2 <- c(conf.mat.test100[2,2]/sum(conf.mat.test100[,2]), conf.mat.test200[
2,2]/sum(conf.mat.test200[,2]), conf.mat.test300[2,2]/sum(conf.mat.test300[,2
]), conf.mat.test400[2,2]/sum(conf.mat.test400[,2]))
diag3 <- c(conf.mat.test100[3,3]/sum(conf.mat.test100[,3]), conf.mat.test200[
3,3]/sum(conf.mat.test200[,3]), conf.mat.test300[3,3]/sum(conf.mat.test300[,3
]), conf.mat.test400[3,3]/sum(conf.mat.test400[,3]))


plot(ntrees, diag1, type = "b", ylim = range(c(diag1, diag2,diag3)), col = "o
rangered", xlab = "# of Trees", ylab = "Diagonal Conf. Mat. Coeff.", main = "
Diagonals of Conf. Mat. vs Number of Trees")
lines(ntrees, diag2, type = "b", col = "steelblue")
lines(ntrees, diag3, type="b", col = "springgreen")
legend("right", c("Position [1,1]", "Position [2,2]", "Position [3,3]"), fill
= c("orangered", "steelblue", "springgreen"))
```



Position[3,3] (class3) consistently performs the best out of the three diagonals. Class3 has the most observations within TRAIN and therefore the model had more information to train off of. However, the difference in observations between class3, the largest class, and class2, the smallest class, is only 65 observations while the difference in accurate classification of the two is around 10%. If this trend held true then any extra observations that could be added might have significant results on the model's performance.

The best ntrees BNT is 200. The performance from position[1,1] and position[2,2] decreases on the way from ntrees = 200 to 300 while position[3,3] decreases. These changes are minimal as are the changes to ntrees = 400. Therefore ntrees = 200 is selected as the best since there is not a large difference in accuracy but it is the simpler model than those with larger ntrees.

From referencing the confusion matrices we can see a similar trend in classification accuracy for class3. When the true class is class1, class3 is incorrectly predicted less than class2 is. The same holds true when looking at predictions of class2. When the true class is class2, class3 is incorrectly predicted less frequently than class1.

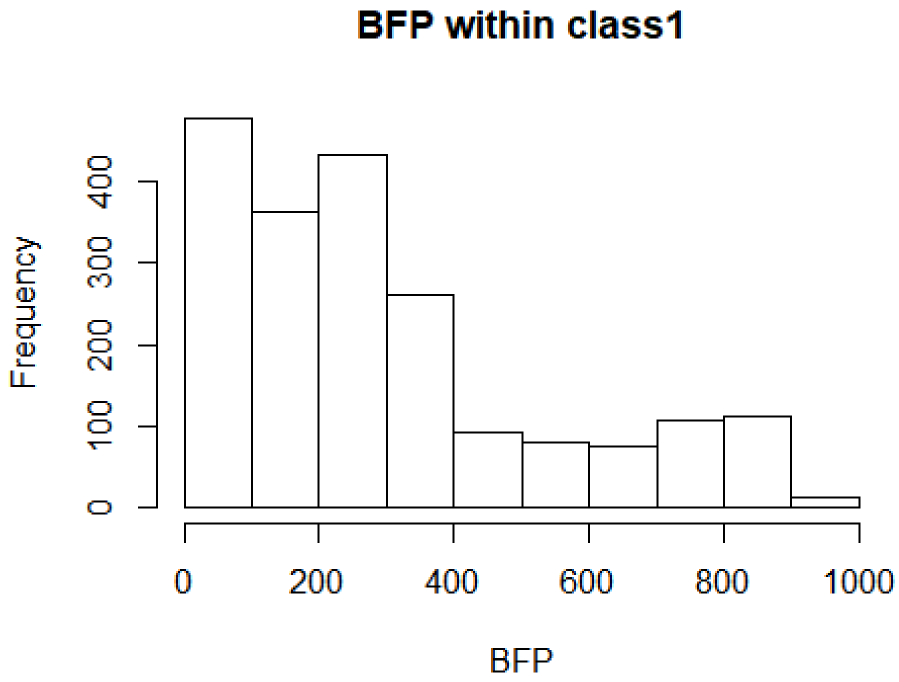
Q7

```r
# Display feature importances of bestRF (rf200)

imp <- as.data.frame(importance(rf200, type = 2))
imp <- data.frame(Features = rownames(imp), Importance = round(imp$MeanDecrea
seGini,2))
imp[order(imp$Importance,decreasing = TRUE),]
```

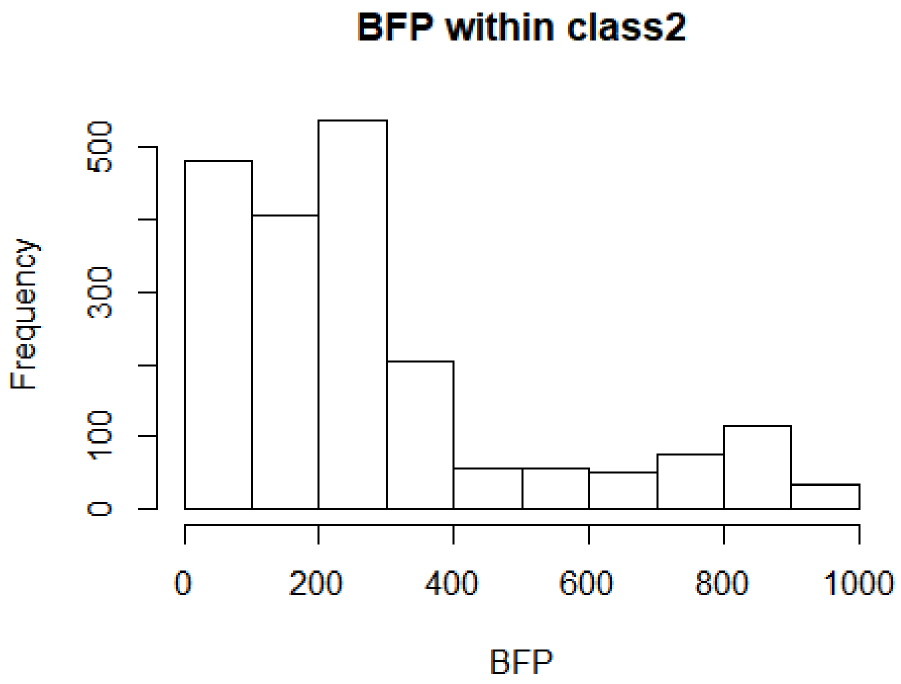
```
##    Features Importance
## 23      BFP     183.18
## 11   IPouts     182.33
## 17    BAOpp     173.40
## 18      ERA     172.27
## 16       SO     171.23
## 15       BB     168.26
## 6         G     164.43
## 24       GF     163.64
## 12        H     162.26
## 1    weight     154.52
## 25        R     141.34
## 13       ER     136.55
## 2    height     120.41
## 14       HR     115.79
## 28     GIDP     110.48
## 5         L     104.39
## 20       WP      97.15
## 10       SV      94.47
## 21      HBP      92.61
## 4         W      92.60
## 7        GS      91.18
## 26       SH      88.67
## 19      IBB      86.70
## 27       SF      81.32
## 22       BK      33.86
## 3      stint       32.60
## 8         CG       17.19
## 9        SHO        9.48
```

Q8

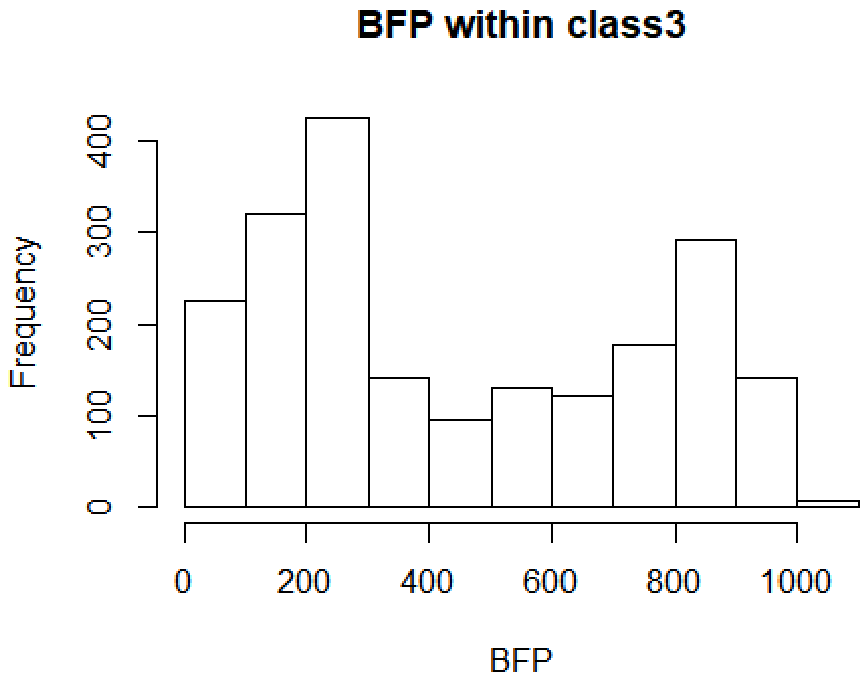
```
# Display histogram of most important feature Z (BFP) within each class
hist(data$BFP[data$class == 1], main = "BFP within class1", xlab = "BFP")
```



```
hist(data$BFP[data$class == 2], main = "BFP within class2", xlab = "BFP")
```



```r
hist(data$BFP[data$class == 3], main = "BFP within class3", xlab = "BFP")
```



```r
ks.test(data$BFP[data$class==1], data$BFP[data$class==2])
```

```
## Warning in ks.test(data$BFP[data$class == 1], data$BFP[data$class == 2]):
## p-value will be approximate in the presence of ties

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  data$BFP[data$class == 1] and data$BFP[data$class == 2]
## D = 0.075992, p-value = 1.783e-05
## alternative hypothesis: two-sided
```

```
ks.test(data$BFP[data$class==1], data$BFP[data$class==3])
```

```
## Warning in ks.test(data$BFP[data$class == 1], data$BFP[data$class == 3]):
## p-value will be approximate in the presence of ties

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  data$BFP[data$class == 1] and data$BFP[data$class == 3]
## D = 0.23198, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

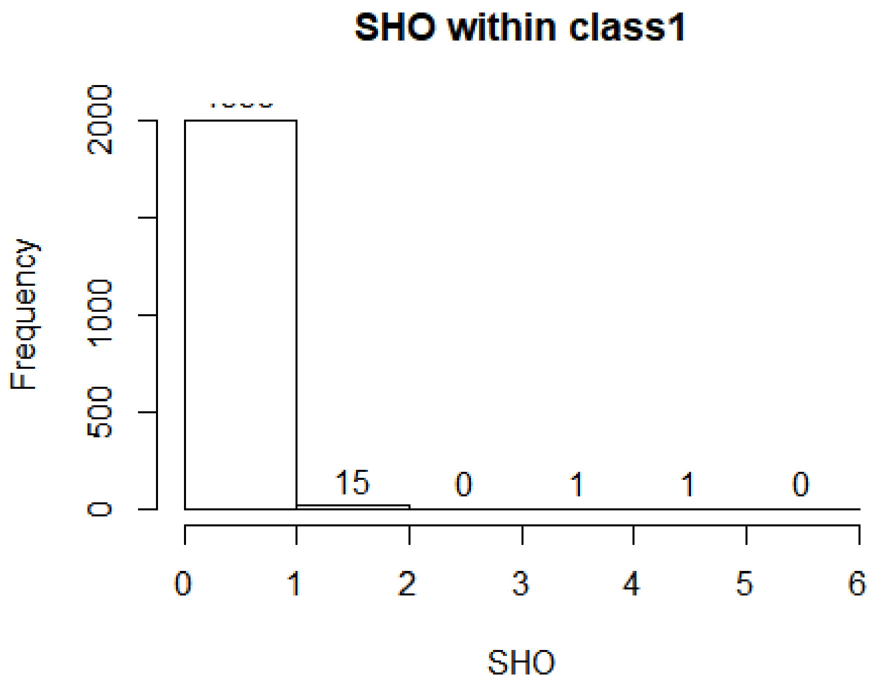
```
ks.test(data$BFP[data$class==2], data$BFP[data$class==3])
```

```
## Warning in ks.test(data$BFP[data$class == 2], data$BFP[data$class == 3]):
## p-value will be approximate in the presence of ties

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  data$BFP[data$class == 2] and data$BFP[data$class == 3]
## D = 0.27803, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

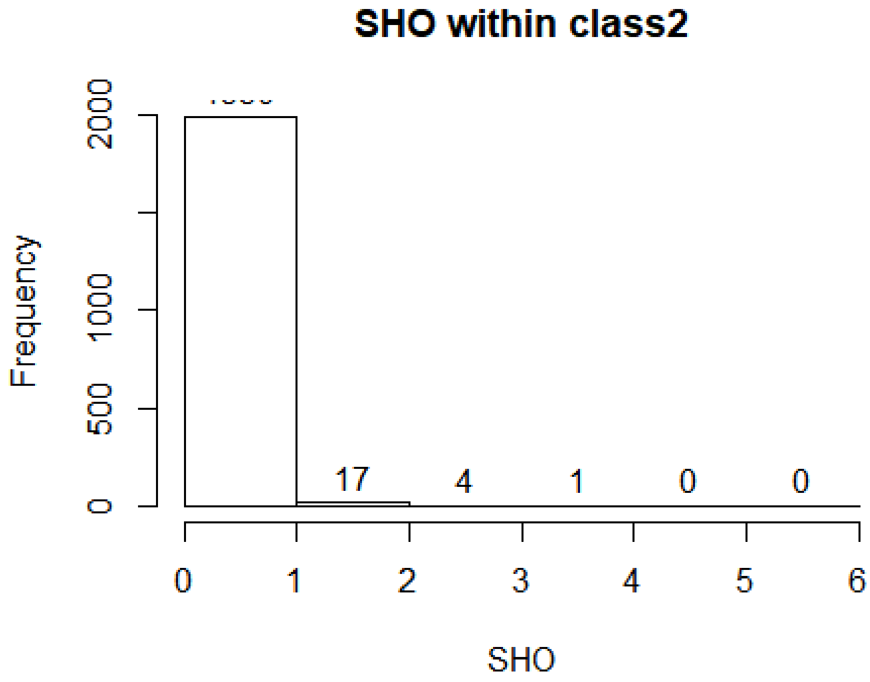
BFP represents the amount of batters faced by a pitcher. This is directly related to pitching experience and therefore makes sense that it would be important in determining a pitcher's salary. From looking at the histograms it is difficult to notice a difference between class1 and class2. However, class3 has noticably more cases within the higher BFP values (specifically greater than 500) than class1 or class2.

The KS test to compare the histograms confirms the variables importance. The comparison of BFP within class1 and class2 received a p-value (P<.001). The comparisons between class1 and class3 as well as class2 and class3 each contained p-values of (P<.001) as well. It should be noted however that the p-value for both comparisons involving class3 were much smaller (P<2.2e-16) than the already small p-value for the comparison of class1 and class2 (P=1.783e-05).

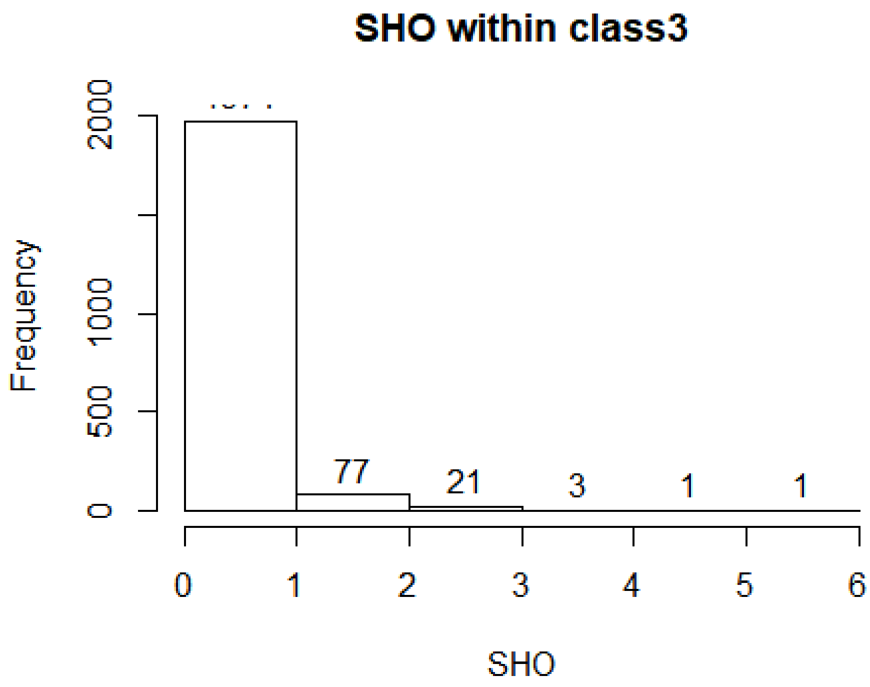
```
# Display histogram of least important feature (SHO) within each class
hist(data$SHO[data$class == 1], breaks = c(0,1,2,3,4,5,6), labels = TRUE, mai
n = "SHO within class1", xlab = "SHO")
```



```
hist(data$SHO[data$class == 2], breaks = c(0,1,2,3,4,5,6), labels = TRUE, mai
n = "SHO within class2", xlab = "SHO")
```



```
hist(data$SHO[data$class == 3], breaks = c(0,1,2,3,4,5,6), labels = TRUE, mai
n = "SHO within class3", xlab = "SHO")
```



```
ks.test(data$SHO[data$class==1], data$SHO[data$class==2])

## Warning in ks.test(data$SHO[data$class == 1], data$SHO[data$class == 2]):
## p-value will be approximate in the presence of ties

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  data$SHO[data$class == 1] and data$SHO[data$class == 2]
## D = 0.011336, p-value = 0.9995
## alternative hypothesis: two-sided

ks.test(data$SHO[data$class==1], data$SHO[data$class==3])

## Warning in ks.test(data$SHO[data$class == 1], data$SHO[data$class == 3]):
## p-value will be approximate in the presence of ties

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  data$SHO[data$class == 1] and data$SHO[data$class == 3]
## D = 0.098715, p-value = 4.408e-09
## alternative hypothesis: two-sided

ks.test(data$SHO[data$class==2], data$SHO[data$class==3])
```

```
## Warning in ks.test(data$SHO[data$class == 2], data$SHO[data$class == 3]):
## p-value will be approximate in the presence of ties

##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  data$SHO[data$class == 2] and data$SHO[data$class == 3]
## D = 0.11005, p-value = 3.549e-11
## alternative hypothesis: two-sided
```

SHO represents the amount of shutouts that a pitcher has earned. This would seem like an important statistic for determining the prowess of a pitcher and therefore how much they should be earning. However, the nature of shutouts makes it a hard tool to use. They are extremely rare and the maximum amount recorded within our data set is 6. From looking at the histograms there is a slight difference in the distributions as pitchers of class2 and then class3 obtain more shutouts than those within the previous class. The difference between class1 and class2 is only five more pitchers that have earned at least one shutout and is pretty insignificant. Class3 shows the most difference among the three with a higher prevalence of at least one shutout than the other two classes (though they are still extremely rare). Class3 contains 103 pitchers with one or more shutouts while class2 contains 22 and class1 contains 17.

The KS test reveals similar findings to what was seen within the histograms. Class1 and class2 are deemed not significantly different with a p-value of P(=.9995). However, the comparison of class1 and class3 as well as class2 and class3 obtained a significant p-value less than .001 (P=4.408e-09 and 3.549e-11, respectively).

Q9

Cluster #8 had the lowest gini

```
# Obtain data from cluster 8
CLj = SDATA[which(k8$cluster == 8),]
CLjnonSDATA = data[which(k8$cluster == 8),]
CLj$class = CLjnonSDATA$class

# Create train/test samples from CLj
set.seed(42)
trainCLj.index <- sample(1:nrow(CLj), 0.8 * nrow(CLj))
testCLj.index <- setdiff(1:nrow(CLj), trainCLj.index)
trainCLj <- CLj[trainCLj.index,]
testCLj <- CLj[testCLj.index,]
train.labels.CLj <- class_vec[trainCLj.index]
test.labels.CLj <- class_vec[testCLj.index]

# Check class balance in the train set
nrow(trainCLj[trainCLj['class'] == 1,])

## [1] 436
```

```r
nrow(trainCLj[trainCLj['class'] == 2,])
```

```
## [1] 537
```

```r
nrow(trainCLj[trainCLj['class'] == 3,])
```

```
## [1] 222
```

```r
# There are far more cases of class 3 than class 1 and 2 so we will use cloning to make them even
# There are only 15% as many cases of class 1 as class 3 so we will duplicate class 1 by 6
trainCLj.class1 <- trainCLj[trainCLj['class'] == 1,]
trainCLj.class1_copy1 <- as.data.frame(trainCLj.class1)
trainCLj.class1_copy2 <- as.data.frame(trainCLj.class1)
trainCLj.class1_copy3 <- as.data.frame(trainCLj.class1)
trainCLj.class1_copy4 <- as.data.frame(trainCLj.class1)
trainCLj.class1_copy5 <- as.data.frame(trainCLj.class1)
trainCLj.class1_copy6 <- as.data.frame(trainCLj.class1)

# There are only 26% as many cases of class 2 as class 3 so we will duplicate class 2 by 3
trainCLj.class2 <- trainCLj[trainCLj['class'] == 2,]
trainCLj.class2_copy1 <- as.data.frame(trainCLj.class2)
trainCLj.class2_copy2 <- as.data.frame(trainCLj.class2)
trainCLj.class2_copy3 <- as.data.frame(trainCLj.class2)

# Create clones of class1 from the training set with small perturbations
for (i in 1:17) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class1_copy1[i,-29] <- trainCLj.class1_copy1[i,-29]*perturb
}
for (i in 1:17) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class1_copy2[i,-29] <- trainCLj.class1_copy2[i,-29]*perturb
}
for (i in 1:17) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class1_copy3[i,-29] <- trainCLj.class1_copy3[i,-29]*perturb
}
for (i in 1:17) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class1_copy4[i,-29] <- trainCLj.class1_copy4[i,-29]*perturb
}
for (i in 1:17) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class1_copy5[i,-29] <- trainCLj.class1_copy5[i,-29]*perturb
}
for (i in 1:17) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class1_copy6[i,-29] <- trainCLj.class1_copy6[i,-29]*perturb
}

# Combine the new clones with the original class1 observations
newtrainCLj.class1 <- rbind(trainCLj.class1,trainCLj.class1_copy1, trainCLj.c
lass1_copy2, trainCLj.class1_copy3, trainCLj.class1_copy4, trainCLj.class1_co
py5, trainCLj.class1_copy6)

# Create clones of class2 from the training set with small perturbations
for (i in 1:28) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class2_copy1[i,-29] <- trainCLj.class2_copy1[i,-29]*perturb
}
for (i in 1:28) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class2_copy2[i,-29] <- trainCLj.class2_copy2[i,-29]*perturb
}
for (i in 1:28) {
  perturb = 1 + (runif(1) - 0.5)/10000
  trainCLj.class2_copy3[i,-29] <- trainCLj.class2_copy3[i,-29]*perturb
}


# Combine the new clones with the original class2 observations
newtrainCLj.class2 <- rbind(trainCLj.class2,trainCLj.class2_copy1, trainCLj.c
lass2_copy2, trainCLj.class2_copy3)

newTRAIN <- rbind(newtrainCLj.class1, newtrainCLj.class2, trainCLj[trainCLj['
class'] == 3,])

nrow(newTRAIN[newTRAIN['class'] == 1,])

## [1] 3052

nrow(newTRAIN[newTRAIN['class'] == 2,])

## [1] 2148

nrow(newTRAIN[newTRAIN['class'] == 3,])

## [1] 222
```

There are now 119 class1 observations, 112 class2, and still 130 class3.

The same process will be repeated for the test set.

```
# Check class balance in the test set
nrow(testCLj[testCLj['class'] == 1,])

## [1] 115

nrow(testCLj[testCLj['class'] == 2,])
```

```
## [1] 126
```

```
nrow(testCLj[testCLj['class'] == 3,])
```

```
## [1] 58
```

```
# There are still more cases of class 3 than class 1 and 2 so we will use clo
ning to make them more even
# There are only 10 class1 cases to the 27 class3 cases so we will triple the
size of class1
testCLj.class1 <- testCLj[testCLj['class'] == 1,]
testCLj.class1_copy1 <- as.data.frame(testCLj.class1)
testCLj.class1_copy2 <- as.data.frame(testCLj.class1)

# There are only 7 cases of class2 so we will clone it 3 times
testCLj.class2 <- testCLj[testCLj['class'] == 2,]
testCLj.class2_copy1 <- as.data.frame(testCLj.class2)
testCLj.class2_copy2 <- as.data.frame(testCLj.class2)
testCLj.class2_copy3 <- as.data.frame(testCLj.class2)

# Create clones of class1 from the test set with small perturbations
for (i in 1:10) {
  perturb = 1 + (runif(1) - 0.5)/10000
  testCLj.class1_copy1[i,-29] <- testCLj.class1_copy1[i,-29]*perturb
}
for (i in 1:10) {
  perturb = 1 + (runif(1) - 0.5)/10000
  testCLj.class1_copy2[i,-29] <- testCLj.class1_copy2[i,-29]*perturb
}

# Combine the new clones with the original class1 observations
newtestCLj.class1 <- rbind(testCLj.class1,testCLj.class1_copy1, testCLj.class
1_copy2)

# Create clones of class1 from the test set with small perturbations
for (i in 1:7) {
  perturb = 1 + (runif(1) - 0.5)/10000
  testCLj.class2_copy1[i,-29] <- testCLj.class2_copy1[i,-29]*perturb
}
for (i in 1:7) {
  perturb = 1 + (runif(1) - 0.5)/10000
  testCLj.class2_copy2[i,-29] <- testCLj.class2_copy2[i,-29]*perturb
}
for (i in 1:7) {
  perturb = 1 + (runif(1) - 0.5)/10000
  testCLj.class2_copy3[i,-29] <- testCLj.class2_copy3[i,-29]*perturb
}


# Combine the new clones with the original class1 observations
newtestCLj.class2 <- rbind(testCLj.class2,testCLj.class2_copy1, testCLj.class
2_copy2, testCLj.class2_copy3)

newTEST <- rbind(newtestCLj.class1, newtestCLj.class2, testCLj[testCLj['class
'] == 3,])

nrow(newTEST[newTEST['class'] == 1,])

## [1] 345

nrow(newTEST[newTEST['class'] == 2,])

## [1] 504

nrow(newTEST[newTEST['class'] == 3,])

## [1] 58
```

There are now 30 class1 observations, 28 class2, and still 27 class3.

```
rf.cluster8_start <- Sys.time()
rf.cluster8 <- randomForest(newTRAIN[,-29], y=newTRAIN$class, ntry=5, ntrees
= 200 )
rf.cluster8_end <- Sys.time()
rf.cluster8_end - rf.cluster8_start

## Time difference of 9.535136 secs

pred.rf.cluster8 <- predict(rf.cluster8, newTEST[,-29])
```

Q10

```
conf.mat.train.cluster8 <- rf.cluster8$confusion
conf.mat.test.cluster8 <- table(pred.rf.cluster8, newTEST$class)

conf.mat.test.cluster8

##
## pred.rf.cluster8   1   2   3
##                1 150  95  13
##                2 195 405  43
##                3   0   4   2

accuracy(conf.mat.train.cluster8)

## [1] 95.92492

accuracy(conf.mat.test.cluster8)

## [1] 61.41125
```

There appears there would be an advantage to training separate rf’s.

```
# Train set nice conf matrix
library(scales)
```

```r
cl1pred1 <- percent(conf.mat.train.cluster8[1,1]/sum(conf.mat.train.cluster8[
,1]))
cl1pred2 <- percent(conf.mat.train.cluster8[2,1]/sum(conf.mat.train.cluster8[
,1]))
cl1pred3 <- percent(conf.mat.train.cluster8[3,1]/sum(conf.mat.train.cluster8[
,1]))

cl2pred2 <- percent(conf.mat.train.cluster8[2,2]/sum(conf.mat.train.cluster8[
,2]))
cl2pred1 <- percent(conf.mat.train.cluster8[1,2]/sum(conf.mat.train.cluster8[
,2]))
cl2pred3 <- percent(conf.mat.train.cluster8[3,2]/sum(conf.mat.train.cluster8[
,2]))

cl3pred3 <- percent(conf.mat.train.cluster8[3,3]/sum(conf.mat.train.cluster8[
,3]))
cl3pred1 <- percent(conf.mat.train.cluster8[1,3]/sum(conf.mat.train.cluster8[
,3]))
cl3pred2 <- percent(conf.mat.train.cluster8[2,3]/sum(conf.mat.train.cluster8[
,3]))

row1 <- c(cl1pred1, cl2pred1, cl3pred1)
row2 <- c(cl1pred2, cl2pred2, cl3pred2)
row3 <- c(cl1pred3, cl2pred3, cl3pred3)


conf.matrix.percent.train.cluster8 <- rbind(row1, row2, row3)
rownames(conf.matrix.percent.train.cluster8) <- c("Pred: CL1", "Pred: CL2", "
Pred: CL3")
colnames(conf.matrix.percent.train.cluster8) <- c("True: CL1", "True: CL2", "
True: CL3")
as.data.frame(conf.matrix.percent.train.cluster8)

##             True: CL1 True: CL2 True: CL3
## Pred: CL1      98.6%        0%        0%
## Pred: CL2         0%     92.4%        0%
## Pred: CL3      1.42%     7.57%      100%

# Test set nice conf matrix
library(scales)
cl1pred1 <- percent(conf.mat.test.cluster8[1,1]/sum(conf.mat.test.cluster8[,1
]))
cl1pred2 <- percent(conf.mat.test.cluster8[2,1]/sum(conf.mat.test.cluster8[,1
]))
cl1pred3 <- percent(conf.mat.test.cluster8[3,1]/sum(conf.mat.test.cluster8[,1
]))

cl2pred2 <- percent(conf.mat.test.cluster8[2,2]/sum(conf.mat.test.cluster8[,2
]))
cl2pred1 <- percent(conf.mat.test.cluster8[1,2]/sum(conf.mat.test.cluster8[,2
]))
cl2pred3 <- percent(conf.mat.test.cluster8[3,2]/sum(conf.mat.test.cluster8[,2
]))

cl3pred3 <- percent(conf.mat.test.cluster8[3,3]/sum(conf.mat.test.cluster8[,3
]))
cl3pred1 <- percent(conf.mat.test.cluster8[1,3]/sum(conf.mat.test.cluster8[,3
]))
cl3pred2 <- percent(conf.mat.test.cluster8[2,3]/sum(conf.mat.test.cluster8[,3
]))

row1 <- c(cl1pred1, cl2pred1, cl3pred1)
row2 <- c(cl1pred2, cl2pred2, cl3pred2)
row3 <- c(cl1pred3, cl2pred3, cl3pred3)


conf.matrix.percent.test.cluster8 <- rbind(row1, row2, row3)
rownames(conf.matrix.percent.test.cluster8) <- c("Pred: CL1", "Pred: CL2", "P
red: CL3")
colnames(conf.matrix.percent.test.cluster8) <- c("True: CL1", "True: CL2", "T
rue: CL3")
as.data.frame(conf.matrix.percent.test.cluster8)

##             True: CL1 True: CL2 True: CL3
## Pred: CL1     43.5%     18.8%     22.4%
## Pred: CL2     56.5%     80.4%     74.1%
## Pred: CL3        0%    0.794%     3.45%
```

Q11

```
library(e1071)

## Warning: package 'e1071' was built under R version 3.6.3

# Create train/test samples from for class1 and class3
newTRAsvm <- rbind(trainCL1, trainCL3)
newTESTsvm <- rbind(testCL1, testCL3)

svm_start <- Sys.time()
svm_model <- svm(newTRAsvm[,-29], newTRAsvm$class)
svm_end <- Sys.time()
pred_svm <- predict(svm_model, newTESTsvm[,-29])
svm_end - svm_start

## Time difference of 3.466868 secs

conf.mat.svm.train <- table(svm_model$fitted, newTRAsvm$class)
conf.mat.svm.test <- table(pred_svm, newTESTsvm$class)

accuracy(conf.mat.svm.train)

## [1] 74.21326
```

```
accuracy(conf.mat.svm.test)

## [1] 68.49817

cl1pred1 <- percent(conf.mat.svm.train[1,1]/sum(conf.mat.svm.train[,1]))
cl1pred3 <- percent(conf.mat.svm.train[3,1]/sum(conf.mat.svm.train[,1]))

cl3pred3 <- percent(conf.mat.svm.train[3,3]/sum(conf.mat.svm.train[,3]))
cl3pred1 <- percent(conf.mat.svm.train[1,3]/sum(conf.mat.svm.train[,3]))

row1 <- c(cl1pred1, cl3pred1)
row3 <- c(cl1pred3, cl3pred3)


conf.matrix.percent.svm.train <- rbind(row1, row3)
rownames(conf.matrix.percent.svm.train) <- c("Pred: CL1", "Pred: CL3")
colnames(conf.matrix.percent.svm.train) <- c("True: CL1", "True: CL3")
as.data.frame(conf.matrix.percent.svm.train)

##           True: CL1 True: CL3
## Pred: CL1     72.7%     24.3%
## Pred: CL3     27.3%     75.7%

library(scales)
cl1pred1 <- percent(conf.mat.svm.test[1,1]/sum(conf.mat.svm.test[,1]))
cl1pred3 <- percent(conf.mat.svm.test[3,1]/sum(conf.mat.svm.test[,1]))

cl3pred3 <- percent(conf.mat.svm.test[3,3]/sum(conf.mat.svm.test[,3]))
cl3pred1 <- percent(conf.mat.svm.test[1,3]/sum(conf.mat.svm.test[,3]))

row1 <- c(cl1pred1, cl3pred1)
row3 <- c(cl1pred3, cl3pred3)


conf.matrix.percent.svm.test <- rbind(row1, row3)
rownames(conf.matrix.percent.svm.test) <- c("Pred: CL1", "Pred: CL3")
colnames(conf.matrix.percent.svm.test) <- c("True: CL1", "True: CL3")
as.data.frame(conf.matrix.percent.svm.test)

##           True: CL1 True: CL3
## Pred: CL1     64.0%     27.2%
## Pred: CL3     36.0%     72.8%

write.csv(data,"C:\\Users\\Melinda B\\Documents\\College\\Graduate\\6350 - Az
encott\\Final\\csv's\\data.csv")
write.csv(TRAIN,"C:\\Users\\Melinda B\\Documents\\College\\Graduate\\6350 - A
zencott\\Final\\csv's\\TRAIN.csv")
write.csv(TEST, "C:\\Users\\Melinda B\\Documents\\College\\Graduate\\6350 - A
zencott\\Final\\csv's\\TEST.csv")
write.csv(newTRAsvm, "C:\\Users\\Melinda B\\Documents\\College\\Graduate\\635
0 - Azencott\\Final\\csv's\\newTRAsvm.csv")
```

```r
write.csv(newTESTsvm, "C:\\Users\\Melinda B\\Documents\\College\\Graduate\\63
50 - Azencott\\Final\\csv's\\newTESTsvm.csv")
```