Midterm Project STAT 602

Gavin Gunawardena

3/22/2022

Intro and Objective

For this project, algorithms covered in this course will be tested in their ability to make predictions on beans using features consisting of their measurements. The project relies on a dataset obtained from a study that took the morphometric measurements of several types of beans found in Turkey in order to separate white beans from the others. Furthermore, the factor of cost for each type of bean will be applied to this analysis, as incorrect predictions could financially harm a customer purchasing beans or a business selling the beans. Thus, not only accuracy of the predictive models will be tested in their application to the dataset but also the average cost of error created by the models, in order to determine the best model or models. The objective of this project is to find a model that obtains an accuracy measure of at least 80%, maximizes accuracy, and minimizes the weight and cost differential between predictions and actual values, in order to give a recommendation on which type of algorithm has the most potential for accurately sorting the beans. No major assumptions about the dataset are being made other than that it consists of 3000 observations, 500 observations of each bean type, 7 independent variables, 6 dependent variable classes, and the info shown in tables 1 and 2 about the beans and variables. The type of analysis required by this problem is predictive analytics utilizing multiclass classification as there are 6 classes to fit the bean data in with 7 continuous independent variables to utilize in order to classify them. In this type of analysis, the values of selected independent variables for each observation are used in conjunction with statistical models in order to predict the dependent variable, bean type. The results of the predictions are then compared with the actual results in order to test for stats such as accuracy, weight differential, and price differential which can then be used to check how useful the models are. The dependent variables of the dataset used in this analysis include the bean classes which are as follows: Seker, Bombay, Cali, Horoz, Sira, and Dermason. The independent variables of this dataset include the area, perimeter, major axis length, minor axis length, eccentricity, convex area, and extent measurements of the beans. These variables will be analyzed for each model on their ability to predict the bean classes, and then utilized for the predictions based on the log odds that they attribute towards the classes of beans. Finally, in order to measure the cost of error associated with inaccurate predictions, additional data for the beans was provided for this project. This data includes the price per pound of each bean and the weight in grams per each seed. This information will be used to obtain approximate values of these stats for each bean and therefore the average cost associated with the error rates of the predictive models in order to decide on the optimal model(s).

Methods

Validation Technique

For this project, the dataset will be randomly split 80% for training/validation and 20% for testing of the final chosen model in order to maximize the amount of data used for creating the model and minimize bias as well as potential for overfitting. This should still allow for enough data in the test dataset to test the final model for accuracy and error rates. For the training/validation method, the leave one out cross validation approach was chosen over the validation set approach. This is due to advantages it has over the validation set approach such as being able to maximize the dataset used in the training process and its tendency to not overestimate the validation error rate. Also, its main disadvantage compared to the validation set approach and other cross validation approaches which is that it is computationally expensive is minimized in this project due to the dataset not being immense and only having 3000 observations. The leave one out cross validation approach involves repeatedly running the model on the training/validation dataset while changing what's used for the training/validation split with each iteration. On each iteration, all but one of the training/validation dataset observations are used to train the model, and the unused observation is used to validate the model. This is repeated until each observation is used once to validate the model so that eventually every observation in the training/validation dataset is used in the training as well as validation processes.

source: (James, Witten, Hastie, & Tibshirani, 2021)

Normalization/Standardization Technique

For normalizing the data, a z-score methodology was chosen due to it being common and applicable to this dataset. Also it is known for handling outliers well by giving them slightly more weight than the rest of the data. Regarding the calculation for Z-Score, the standard calculation is used by first calculating the mean and standard deviation of each feature, and then calculating the z score for each individual feature by subtracting the value by the mean and dividing this new value by the standard deviation. The mean and standard deviation of the training dataset are saved for use in future test and production datasets in order to maximize effectiveness of the trained statistical models.

source: (Bobbitt, 2019)

Feature Selection Method

For feature selection, best subset selection via the leaps library in R was chosen for this project. This method and library test all possible combinations of models for the independent variables and the dependent variable via multinomial logistic regression, and chooses the best ones based on residual sum of squares. It then reveals the best variable combinations and statistics for each possible number of variables. In this case, there were seven possible independent variables and thus this method revealed 7 possible combinations of variables, and statistics for them. Next, a variable combination was chosen based on adjusted R squared.

source: (Clark Science Center, 2016)

Statistical Models

Statistical models chosen for this project include multinomial logistic regression, linear discriminant analysis, KNN classifier, quadratic discriminant analysis, and Naive Bayes classifier. Multinomial logistic regression is a model where the log odds of the dependent variable are predicted based on a linear combination of independent variables.

$$ln\left(\frac{P(BeanClass = 1)}{P(BeanClass = 2)}\right) = b_1(Variable_1) + b_2(Variable_2) + b_3(Variable_3)\ldots$$

$$ln\left(\frac{P(BeanClass = 3)}{P(BeanClass = 4)}\right) = b_1(Variable_1) + b_2(Variable_2) + b_3(Variable_3)\ldots$$

source: (UCLA: Statistical Consulting Group, n.d.)

Linear discriminant analysis attempts to map observations of the dependent variable to a data space and utilize a discriminant rule to divide the data space into regions based on the independent variables, utilizing maximum likelihood or Bayesian rules. It assumes that the function of the class is a density function for a multivariate normal random variable with a class specific mean and a shared covariance matrix.

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(-\frac{1}{2\pi\sigma_k^2}(x-\mu_k)^2\right)$$

k=class source: (Xiaozhou, 2020)*

K-nearest neighbor classifier is a model that, given a positive integer, K, and observation, attempts to classify the observation based on the K most similar observations in the training dataset. It improves over time as observations are added to its training dataset.

$$Pr(T = j|X = x_0) = \frac{1}{K}\sum_{i=\epsilon N_0} I\,(y_i = j)$$

j=class K=neighbors x_o = test observations N_o = K points closest to the test observation *source: (James, Witten, Hastie, & Tibshirani, 2021)*

Quadratic discriminant analysis is similar to linear discriminant analysis although it assumes that each class has its own covariance matrix and utilizes a quadratic function. It assumes that the function of the class is a density function for a multivariate normal random variable with class-specific mean and covariance matrix.

$$\delta_k(x) = -\frac{1}{2}(x-\mu_k)^T \sum_k^{-1} (x-\mu_k) - \frac{1}{2} log\left|\sum_k\right| + log\pi_k$$

Epsilon_k = covariance matrix for the kth class k=class *source: (James, Witten, Hastie, & Tibshirani, 2021)*

The Naive bayes model makes class predictions by first assuming that within the kth class, the p predictors are independent, and then utilizes this assumption in a modified Bayes Theorem function.

$$Pr(Y = k|X = x) = \frac{\pi_x \times f_{k1}(x_1) \times f_{k2}(x_2) \times \ldots \times f_{kp}(x_p)}{\sum_{l=1}^{K} \pi_l \times f_{l1}(x_1) \times f_{l2}(x_2) \times \ldots \times f_{lp}(x_p)}$$

for k=1,...,K

Source: (James, Witten, Hastie, & Tibshirani, 2021)

##Initial Data and calculation functions

```
#Suppress warnings
options(warn=-1)
#Install packages necessary for the project
#http://www.salemmarafi.com/code/install-r-package-automatically/
usePackage <- function(p) {
    if (!is.element(p, installed.packages()[,1]))
        install.packages(p, dep = TRUE)
    require(p, character.only = TRUE)
}
# Load packages used in this project
usePackage("readxl")

## Loading required package: readxl

usePackage("ggplot2")

## Loading required package: ggplot2

usePackage("tidyverse")

## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse 1.
3.1 --

## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflict
s() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

usePackage("hrbrthemes")

## Loading required package: hrbrthemes
## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use th
ese themes.

##       Please use hrbrthemes::import_roboto_condensed() to install Roboto C
ondensed and

##       if Arial Narrow is not on your system, please see https://bit.ly/ari
alnarrow

usePackage("viridis")

## Loading required package: viridis

## Loading required package: viridisLite

usePackage("forcats")
usePackage("dplyr")
usePackage("nnet")

## Loading required package: nnet

usePackage("caret")

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

usePackage("leaps")

## Loading required package: leaps

usePackage("MASS")

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

usePackage("class")

## Loading required package: class

usePackage("e1071")
```

```
## Loading required package: e1071
```

```
usePackage("rstatix")
```

```
## Loading required package: rstatix
```

```
## 
## Attaching package: 'rstatix'
```

```
## The following object is masked from 'package:MASS':
## 
##     select
```

```
## The following object is masked from 'package:stats':
## 
##     filter
```

```
usePackage("DT")
```

```
## Loading required package: DT
```

```
#Import data from csv file
SeedDataInitial.df <- as.data.frame(read.csv("labeled.csv"))
#Confirm data wasimported
head(SeedDataInitial.df)



#Create dataframes with necessary key information for calculating bean value
and cost of error

#Create dataframe to store bean data
BeanKey.df <- data.frame (ClassID = factor(seq(1,6,1)),
  Class  = c("Bombay", "Cali", "Dermason", "Horoz", "Seker", "Sira"),
                  dollars_per_lb = c(5.56,6.02,1.98,2.43,2.72,5.4),
                  grams_per_seed = c(1.92,.61,.28,.52,.49,.38)
                  )
BeanKey.df$Class <- toupper(BeanKey.df$Class)
attach(BeanKey.df)
  BeanKey.df$approx_lbs_per_seed <- round(grams_per_seed/453.592,4)
attach(BeanKey.df)
```

```
## The following objects are masked from BeanKey.df (pos = 3):
## 
##     Class, ClassID, dollars_per_lb, grams_per_seed
```

```
  BeanKey.df$approx_dollars_per_seed <- round(approx_lbs_per_seed * dollars_p
er_lb,4)
attach(BeanKey.df)
```

```
## The following objects are masked from BeanKey.df (pos = 3):
## 
##     approx_lbs_per_seed, Class, ClassID, dollars_per_lb, grams_per_seed
## The following objects are masked from BeanKey.df (pos = 4):
##
##     Class, ClassID, dollars_per_lb, grams_per_seed
```

```
  BeanKey.df$approx_dollars_per_gram <- round(dollars_per_lb / 453.592,4)


#Create dataframe to store descriptions of the independent variables of the b
ean data
bean_column_descriptions.df <- data.frame(ID = seq(1,7,1),
  name = c("Area","Perimeter", "MajorAxisLength", "MinorAxisLength", "Eccentr
icity", "ConvexArea", "Extent"),
                                            # abbreviation = c("A","P","L","I
","Ec","C", "Ex"),
                                            description = c("The area of a be
an zone and the number of pixels within its boundaries.","Bean circumference 
is defined as the length of its border.","The distance between the ends of th
e longest line that can be drawn from a bean.","The longest line that can be 
drawn from the bean while standing perpendicular to the main axis.","Eccentri
city of the ellipse having the same moments as the region.","Number of pixels
in the smallest convex polygon that can contain the area of a bean seed.","Th
e ratio of the pixels in the bounding box to the bean area.")
                                            )
#Create functions to calculate bean data

#Calculate weight of a sample of beans in grams
get.Bean.Weight <- function(BeanList, BeanKey){
  temp.df <- left_join(BeanList, BeanKey, by='ClassID')
  return(sum(temp.df$grams_per_seed, na.rm = TRUE))
}

#Calculate cost of a sample of beans
get.Bean.Cost <- function(BeanList, BeanKey){
  temp.df <- left_join(BeanList, BeanKey, by='ClassID')
  return(sum(temp.df$approx_dollars_per_seed, na.rm = TRUE))
}
#Compare weights of 2 samples of beans
compare.Bean.Weight <- function(BeanListActuals, BeanListPredicted, BeanKey){
  beansActualWeight <- get.Bean.Weight(BeanListActuals, BeanKey)
  beansPredictedWeight <- get.Bean.Weight(BeanListPredicted, BeanKey)
  return(abs(beansActualWeight - beansPredictedWeight))
}
#Compare costs of 2 samples of beans
compare.Bean.Cost <- function(BeanListActuals, BeanListPredicted, BeanKey){
    beansActualCost <- get.Bean.Cost(BeanListActuals, BeanKey)
  beansPredictedCost <- get.Bean.Cost(BeanListPredicted, BeanKey)
  return(abs(beansActualCost - beansPredictedCost))
}
```

```
# Create functions to calculate Z Scores, Skewness, and Kurtosis
# Z score function
ZScore.Solve <- function(x.dat, y_hat = mean(x.dat), s_dev = sd(x.dat)) {
 z_score <- (x.dat - y_hat)/sqrt(s_dev)
 return(z_score)
}
#Calculate Skewness function
Skewness.fun <- function(arrayOfValues){
  mean.x <- 0
  stDev.x <- 0
  sum.x <- 0
  skewness.x <- 0
  kurtosis.x <- 0
  runningCalcTotal1.x <- 0 #standard deviation
  runningCalcTotal2.x <- 0 #skewness
  runningCalcTotal3.x <- 0 #kurtosis
  n <- 0
  #Mean calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      sum.x <- sum.x + arrayOfValues[i]
      n <- n+1
    }
  }
  mean.x <- sum.x/n

  #Standard Deviation Calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal1.x <- runningCalcTotal1.x + ((arrayOfValues[i]-mean.x)
* (arrayOfValues[i]-mean.x))
    }
  }
  stDev.x <- sqrt(runningCalcTotal1.x/(n-1))

  #Fisher-Pearson  skewness Calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal2.x <- (runningCalcTotal2.x + ((arrayOfValues[i]-mean.x
) * (arrayOfValues[i]-mean.x)^2))
     }
  }
  skewness.x <- ((runningCalcTotal2.x/n) / stDev.x^3)

  return(skewness.x)
}
#Calculate Kurtosis function
Kurtosis.fun <- function(arrayOfValues){
  mean.x <- 0
  stDev.x <- 0
  sum.x <- 0
  skewness.x <- 0
  kurtosis.x <- 0
  runningCalcTotal1.x <- 0 #standard deviation
  runningCalcTotal2.x <- 0 #skewness
  runningCalcTotal3.x <- 0 #kurtosis
  n <- 0
  #Mean calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      sum.x <- sum.x + arrayOfValues[i]
      n <- n+1
    }
  }
  mean.x <- sum.x/n

  #Standard Deviation Calculation
  for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal1.x <- runningCalcTotal1.x + ((arrayOfValues[i]-mean.x)
* (arrayOfValues[i]-mean.x))
    }
  }
  stDev.x <- sqrt(runningCalcTotal1.x/(n-1))


  #kurtosis Calculation
    for (i in 1:length(arrayOfValues)){
    if(!is.na(arrayOfValues[i])){
      runningCalcTotal3.x <- (runningCalcTotal3.x + ((arrayOfValues[i]-mean.x
) * (arrayOfValues[i]-mean.x)^3))
     }
  }
  kurtosis.x <- ((runningCalcTotal3.x/n) / stDev.x^4)
  return(kurtosis.x)
}

#Functions to convert Class Variable to and from a numeric ID
#Convert class names to class IDs and return all dependent and independent va
riables
convert.class.to.classID <- function(BeanData.df, BeanKey=BeanKey.df){
  temp.1.df <- left_join(BeanData.df, BeanKey.df, by='Class')
  return(subset(temp.1.df, select = c("Area","Perimeter","MajorAxisLength", "
MinorAxisLength", "Eccentricity", "ConvexArea","Extent", "ClassID") ))
}

#Convert class ID to class names and return all dependent and independent var
```

```
iables
convert.classID.to.class <- function(BeanData.df, BeanKey=BeanKey.df){
  temp.1.df <- left_join(BeanData.df, BeanKey.df, by='ClassID')
 return(subset(temp.1.df, select = c("Area","Perimeter","MajorAxisLength", "
MinorAxisLength", "Eccentricity", "ConvexArea","Extent", "Class") ))
}
#Convert class IDs to Class names and return the class names
convert.classID.to.class.2 <- function(BeanData.df, BeanKey=BeanKey.df){
  temp.1.df <- left_join(BeanData.df, BeanKey.df, by='ClassID')
 return(subset(temp.1.df, select = c("Class") ))
}
#Convert a factor of class IDs to Class Names and return the class names as a
factor
convert.classID.to.class.2.factor <- function(BeanData, BeanKey=BeanKey.df){
 tempBeanDf <- data.frame(ClassID = BeanData)
  temp.1.df <- left_join(tempBeanDf, BeanKey.df, by='ClassID')
  result <- factor(subset(temp.1.df, select = Class )$Class, levels = c("BO
MBAY", "CALI", "DERMASON", "HOROZ", "SEKER", "SIRA"))
 return(result)
}

#Create function to remove outliers
remove_outliers_beans_dataset <- function(x.df){

   #Create temporary dataset with row numbers as a column
 Data.All.df.No.Outliers.Temp <- x.df
 Data.All.df.No.Outliers.Temp$row_names <- row.names(Data.All.df.No.Outliers
.Temp)
 Dependent.Var.Columns <-c("Area","Perimeter","MajorAxisLength", "MinorAxisL
ength", "Eccentricity", "ConvexArea","Extent")

 for (i in Dependent.Var.Columns) {
 #Identify the values in each aliquot that are outliers
 Data.All.df.No.Outliers.Temp2 <- Data.All.df.No.Outliers.Temp %>%
  group_by(Class) %>%
  identify_outliers(all_of(i))

 #Use recursion to rerun the function for identifying and removing outliers
in case there are any left after each run
 while (length(Data.All.df.No.Outliers.Temp2[,all_of(i)]) > 0)
     {
     # create a list of all row numbers that are outliers
     list.outliers <- as.list(Data.All.df.No.Outliers.Temp2$row_names)

     # Remove the outliers from the main dataset based on the row number
s of outliers list
      Data.All.df.No.Outliers.Temp <- Data.All.df.No.Outliers.Temp[!Data
.All.df.No.Outliers.Temp$row_names %in% list.outliers,]
```

```
           #Identify the values in each aliquot that are outliers
          Data.All.df.No.Outliers.Temp2 <- Data.All.df.No.Outliers.Temp %>%
            group_by(Class) %>%
            identify_outliers(all_of(i))

  }
  }
 return(Data.All.df.No.Outliers.Temp)
}

#Display some of the created dataframes
print(BeanKey.df)
print(bean_column_descriptions.df)
```

#Exploratory Analysis

```
#Suppress warnings
options(warn=-1)
#Check distribution between classes and look for class imbalance
#https://www.statology.org/r-frequency-table-by-group/

beanCounts <- SeedDataInitial.df %>%
  group_by(Class) %>%
  summarize(Freq=n())

print(beanCounts)
#Check distribution of data within classes and look for outliers that could b
e the result of an error
#https://r-graph-gallery.com/histogram_several_group.html
#https://cmdlinetips.com/2019/02/how-to-make-grouped-boxplots-with-ggplot2/




#Convert data to long format
SeedDataLong.df <- SeedDataInitial.df %>%                                   #
Apply pivot_longer function
  pivot_longer(c("Area","Perimeter", "MajorAxisLength", "MinorAxisLength", "E
ccentricity", "ConvexArea", "Extent"), names_to = "variable")


#Plots




#Box plots of the dataset
for (var_ in unique(SeedDataLong.df$variable)) {
  plotValue <- SeedDataLong.df %>%
    filter(variable %in% var_) %>%
    ggplot(aes(x=Class, y=value, fill=Class)) +
    geom_boxplot() +
    stat_boxplot(geom ='errorbar', width = 0.6) +
    geom_jitter(width=0.1,alpha=0.2) +
    labs(y=var_)

  print(plotValue)
}

#Histograms of the dataset
  for (var_ in unique(SeedDataLong.df$variable)) {
  plotValue2 <- SeedDataLong.df %>%
    filter(variable %in% var_) %>%
  ggplot( aes(x=value, color=Class, fill=Class)) +
    geom_histogram(alpha=0.6) +
    scale_fill_viridis(discrete=TRUE) +
    scale_color_viridis(discrete=TRUE) +
    theme_ipsum() +
    theme(
      legend.position="none",
      panel.spacing = unit(0.1, "lines"),
      axis.text.x = element_text(angle=45)
    ) +
    facet_wrap(~Class)+
    labs(subtitle=var_)
  print(plotValue2)
  }
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
#View Kurtosis and Skewness of the dataset

Class_val <- list()
Variable_val <- list()
Skewness_val <- list()
Kurtosis_val <- list()
i <- 1
for (bean_ in unique(SeedDataLong.df$Class)){
  for (var_ in unique(SeedDataLong.df$variable)){
     Class_val[[i]] <- bean_
     Variable_val[[i]] <- var_
     Skewness_val[[i]] <- Skewness.fun(subset(SeedDataInitial.df, Class == be
an_)[,var_])
    Kurtosis_val[[i]] <- Kurtosis.fun(subset(SeedDataInitial.df, Class == bea
n_)[,var_])
    i <- i + 1
  }
}
#
# beanSkewnessKurtosis.df <- data.frame(Class = unlist(Class_val),
#                                       Variable = unlist(Variable_val),
#                                       Skewness = unlist(Skewness_val),
#                                       Kurtosis = unlist(Kurtosis_val)
#                                       )
# print(beanSkewnessKurtosis.df)
```

For the exploratory analysis, the data was grouped by bean type and then box plots and histograms of each variable were created in order to check the distributions and outliers of the data. Most of the data seems to be normally distributed with varying levels of kurtosis while the rest have varying levels of skewness, especially within the extent variable. No assumptions have been made that the data has been previously cleaned of any error or abnormalities, and thus the outliers will be removed based on anything that is above quartile 3 plus 1.5 times the interquartile range or below quartile 1 minus 1.5 times the interquartile range.

Remove Outliers

```
#Suppress warnings
options(warn=-1)


# Run function to remove outliers
SeedDataNoOutliers.df <- remove_outliers_beans_dataset(SeedDataInitial.df)
#Run the function to remove outliers once more as 2 outliers were found after
the first run
SeedDataNoOutliers.df <- remove_outliers_beans_dataset(SeedDataNoOutliers.df)
```

View Exploratory Analysis after outlier removal

```
#Suppress warnings
options(warn=-1)

# View bean counts per category
print(SeedCountsNoOutliers <- SeedDataNoOutliers.df %>%
  group_by(Class) %>%
  summarize(Freq=n()))

#Convert data to long format
SeedDataLong.df <- SeedDataNoOutliers.df %>%
# Apply pivot_longer function
```

```
  pivot_longer(c("Area","Perimeter", "MajorAxisLength", "MinorAxisLength", "E
ccentricity", "ConvexArea", "Extent"), names_to = "variable")


#Plots
#Box plots of the dataset
for (var_ in unique(SeedDataLong.df$variable)) {
  plotValue <- SeedDataLong.df %>%
    filter(variable %in% var_) %>%
    ggplot(aes(x=Class, y=value, fill=Class)) +
    geom_boxplot() +
    stat_boxplot(geom ='errorbar', width = 0.6) +
    geom_jitter(width=0.1,alpha=0.2) +
    labs(y=var_)

  print(plotValue)
}

#Histograms of the dataset
  for (var_ in unique(SeedDataLong.df$variable)) {
  plotValue2 <- SeedDataLong.df %>%
    filter(variable %in% var_) %>%
  ggplot( aes(x=value, color=Class, fill=Class)) +
    geom_histogram(alpha=0.6) +
    scale_fill_viridis(discrete=TRUE) +
    scale_color_viridis(discrete=TRUE) +
    # theme_ipsum() +
    theme(
      legend.position="none",
      panel.spacing = unit(0.1, "lines"),
      axis.text.x = element_text(angle=45)
    ) +
    facet_wrap(~Class)+
    labs(subtitle=var_)
  print(plotValue2)
  }
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

After removing the outliers, the dataset has been reduced from 3000 observations to 2731 observations with 422 being the lowest amount of observations in a class. These outliers are visibly gone from the upper and lower limits of the box plots as well as the tails of the histograms. # Predictive Analysis

Numeric dependent variable conversion

```
#Suppress warnings
options(warn=-1)
#Update Dataset to have a numeric version of response variable via the  beanK
ey.df dataframe
SeedDataNumeric.df <- convert.class.to.classID(SeedDataNoOutliers.df, BeanKey
.df)
```

Data Split

```
#Suppress warnings
options(warn=-1)

# Split dataset 80/20
set.seed (10)
rows.trn=sort(sample(1:length(SeedDataNumeric.df[,1]), size = length(SeedData
Numeric.df[,1])*.8, replace = F))

trn.dat=SeedDataNumeric.df[rows.trn,]
tst.dat=SeedDataNumeric.df[-rows.trn,]
```

Normalization/Standardization

```
#Suppress warnings
options(warn=-1)
# Normalize the data

#Store normalization parameters of mean and standard deviation for use on the
test dataset later
NormalizationParams <- data.frame(mean = apply(trn.dat[,1:7],2,mean), sd = ap
ply(trn.dat[,1:7],2,sd))

#Make copy of unnormalized training dataset in case needed later
trn.norm.dat<-trn.dat

#Normalize via the newly created Z-score function
trn.norm.dat$Area <- ZScore.Solve(trn.dat$Area)
trn.norm.dat$Perimeter <- ZScore.Solve(trn.dat$Perimeter)
trn.norm.dat$MajorAxisLength <- ZScore.Solve(trn.dat$MajorAxisLength)
trn.norm.dat$MinorAxisLength <- ZScore.Solve(trn.dat$MinorAxisLength)
trn.norm.dat$Eccentricity <- ZScore.Solve(trn.dat$Eccentricity)
trn.norm.dat$ConvexArea <- ZScore.Solve(trn.dat$ConvexArea)
trn.norm.dat$Extent <- ZScore.Solve(trn.dat$Extent)
```

```
#Make copy of unnormalized test dataset in case needed later
tst.norm.dat<-tst.dat

#Normalize the test dataset, utilizing the parameters obtained from the train
ing dataset
tst.norm.dat$Area <- ZScore.Solve(tst.dat$Area, NormalizationParams["Area","m
ean"], NormalizationParams["Area","sd"])
tst.norm.dat$Perimeter <- ZScore.Solve(tst.dat$Perimeter, NormalizationParams
["Perimeter","mean"], NormalizationParams["Perimeter","sd"])
tst.norm.dat$MajorAxisLength <- ZScore.Solve(tst.dat$MajorAxisLength, Normali
zationParams["MajorAxisLength","mean"], NormalizationParams["MajorAxisLength"
,"sd"])
tst.norm.dat$MinorAxisLength <- ZScore.Solve(tst.dat$MinorAxisLength, Normali
zationParams["MinorAxisLength","mean"], NormalizationParams["MinorAxisLength"
,"sd"])
tst.norm.dat$Eccentricity <- ZScore.Solve(tst.dat$Eccentricity, Normalization
Params["Eccentricity","mean"], NormalizationParams["Eccentricity","sd"])
tst.norm.dat$ConvexArea <- ZScore.Solve(tst.dat$ConvexArea, NormalizationPara
ms["ConvexArea","mean"], NormalizationParams["ConvexArea","sd"])
tst.norm.dat$Extent <- ZScore.Solve(tst.dat$Extent, NormalizationParams["Exte
nt","mean"], NormalizationParams["Extent","sd"])

#Check for any missing values in the datasets
 print(paste("Missing values in training dataset: ",sum (is.na( trn.norm.dat
))))
 print(paste("Missing values in test dataset: ",sum (is.na( tst.norm.dat ))))
```

Feature Selection

```
#Suppress warnings
options(warn=-1)

# Run best subset selection
regfit.full<-regsubsets(ClassID ~.,data=trn.norm.dat,nvmax = 7)


reg.summary <- summary(regfit.full)




#Display the results and highlight the row with the highest adjusted R^2 valu
e
FeatureSelectionResults <- cbind(Num_Features = row.names(reg.summary$which),
reg.summary$which,'AdjRSquared' = reg.summary$adjr2)

datatable(FeatureSelectionResults[,-2]) %>% formatStyle(
  'AdjRSquared',
  target = 'row',
    backgroundColor = styleEqual(
      max(FeatureSelectionResults[,'AdjRSquared']), c('lightgreen')
    )
)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs().
If it is installed, please make sure the phantomjs executable can be found vi
a the PATH variable.
```

```
formulaVar <- as.formula("ClassID ~ Area + Eccentricity + Perimeter + ConvexA
rea + Extent")
formulaVarOrig <- as.formula("ClassID ~ Area + Eccentricity  + ConvexArea + E
xtent")
```

Statistical Models

Multinomial Logistic Regression

```
#Suppress warnings
options(warn=-1)
defaultW <- getOption("warn")
options(warn = -1)
#Found compatible loocv code here; https://www.r-bloggers.com/2015/09/predict
ing-creditability-using-logistic-regression-in-r-cross-validating-the-classif
ier-part-2-2/

#Run multinomial logistic regression model with loocv
acc <- NULL
resultsLogModel <- factor(rep(1,length(trn.norm.dat)),levels=seq(1,6))
for(i in 1:nrow(trn.norm.dat))
{
    # Train-test splitting
    train <- trn.norm.dat[-i,]
    validation <- trn.norm.dat[i,]

    # Fitting
    LogModel <- multinom(formulaVar, data=train, trace=FALSE)

    # Predict results
    resultsLogModel[i] <- predict(LogModel,subset(validation,select=c(1:7)),
type="class")

}

# Capture and print results
correctRes <- factor(trn.norm.dat$ClassID,levels = seq(1,6))

correctResForCompare <- data.frame(ClassID = correctRes)

resultsLogModelForCompare <- data.frame(ClassID = resultsLogModel)

print("weight")
print(LogBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, resultsL
ogModelForCompare, BeanKey.df))

print("cost")
print(LogBeanCostDiff <- compare.Bean.Cost(correctResForCompare, resultsLogMo
delForCompare, BeanKey.df))

print(LogConfusionMatrix <- confusionMatrix(data = convert.classID.to.class.2
.factor(resultsLogModel), reference = convert.classID.to.class.2.factor(corre
ctRes)))
```

Linear Discriminant Analysis

```
#Suppress warnings
options(warn=-1)

#Run Linear Discriminant Analysis with loocv
acc <- NULL
resultsLdaModel <- factor(rep(1,length(trn.norm.dat)),levels=seq(1,6))
for(i in 1:nrow(trn.norm.dat))
{
    # Train-test splitting
    train <- trn.norm.dat[-i,]
    validation <- trn.norm.dat[i,]

    # Fitting
    LdaModel <- lda(formulaVar, data=train)

    # Predict results
    resultsLdaModel[i] <- predict(LdaModel,subset(validation,select=c(1:7)),
type="class")$class

}

#Capture and print results
correctRes <- factor(trn.norm.dat$ClassID,levels = seq(1,6))


correctResForCompare <- data.frame(ClassID = correctRes)

resultsLdaModelForCompare <- data.frame(ClassID = resultsLdaModel)

print("weight")
print(LdaBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, resultsL
daModelForCompare, BeanKey.df))

print("cost")
print(LdaBeanCostDiff <- compare.Bean.Cost(correctResForCompare, resultsLdaMo
delForCompare, BeanKey.df))
```

```r
print(LdaConfusionMatrix <- confusionMatrix(data = convert.classID.to.class.2
.factor(resultsLdaModel), reference = convert.classID.to.class.2.factor(corre
ctRes)))
```

KNN Classifier

```r
#Suppress warnings
options(warn=-1)
#Run K Nearest Neighbors with loocv
acc <- NULL
resultsKnnModel <- factor(rep(1,length(trn.norm.dat)),levels=seq(1,6))
columnsKnn <- c("Area", "Perimeter","Eccentricity","ConvexArea","Extent", "Cl
assID")

for(i in 1:nrow(trn.norm.dat))
{
    # Train-test splitting
    train <- trn.norm.dat[-i,columnsKnn]
    validation <- trn.norm.dat[i,columnsKnn]

    # Fitting
    resultsKnnModel[i] <- knn(train[,1:5],validation[,1:5],train[,6], k=30)


}

#Capture and print results
correctRes <- factor(trn.norm.dat$ClassID,levels = seq(1,6))

correctResForCompare <- data.frame(ClassID = correctRes)

resultsKnnModelForCompare <- data.frame(ClassID = resultsKnnModel)

print("weight")
print(KnnBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, resultsK
nnModelForCompare, BeanKey.df))

print("cost")
print(KnnBeanCostDiff <- compare.Bean.Cost(correctResForCompare, resultsKnnMo
delForCompare, BeanKey.df))

print(KnnConfusionMatrix <- confusionMatrix(data = convert.classID.to.class.2
.factor(resultsKnnModel), reference = convert.classID.to.class.2.factor(corre
ctRes)))
```

Quadratic Discriminant Analysis

```r
#Suppress warnings
options(warn=-1)
```

```r
#Run Quadratic Discriminant Analysis with loocv
acc <- NULL
resultsQdaModel <- factor(rep(1,length(trn.norm.dat)),levels=seq(1,6))
for(i in 1:nrow(trn.norm.dat))
{
    # Train-test splitting
    train <- trn.norm.dat[-i,]
    validation <- trn.norm.dat[i,]

    # Fitting
    QdaModel <- qda(formulaVar, data=train)

    # Predict results
    resultsQdaModel[i] <- predict(QdaModel,subset(validation,select=c(1:7)), type="class")$class

}

#Capture and print results
correctRes <- factor(trn.norm.dat$ClassID,levels = seq(1,6))


correctResForCompare <- data.frame(ClassID = correctRes)

resultsQdaModelForCompare <- data.frame(ClassID = resultsQdaModel)

print("weight")
print(QdaBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, resultsQdaModelForCompare, BeanKey.df))

print("cost")
print(QdaBeanCostDiff <- compare.Bean.Cost(correctResForCompare, resultsQdaModelForCompare, BeanKey.df))

print(QdaConfusionMatrix <- confusionMatrix(data = convert.classID.to.class.2.factor(resultsQdaModel), reference = convert.classID.to.class.2.factor(correctRes)))
```

Naive Bayes

```r
#Suppress warnings
options(warn=-1)

#Run Naive Bayes model with loocv
acc <- NULL
resultsNbModel <- factor(rep(1,length(trn.norm.dat)),levels=seq(1,6))
for(i in 1:nrow(trn.norm.dat))
{
    # Train-test splitting
    train <- trn.norm.dat[-i,]
    validation <- trn.norm.dat[i,]

    # Fitting
    NbModel <- naiveBayes(formulaVar, data=train)

    # Predict results
    resultsNbModel[i] <- predict(NbModel,subset(validation,select=c(1:7)), ty
pe="class")

}

#Capture and print results
correctRes <- factor(trn.norm.dat$ClassID,levels = seq(1,6))

correctResForCompare <- data.frame(ClassID = correctRes)

resultsNbModelForCompare <- data.frame(ClassID = resultsNbModel)

print("weight")
print(NbBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, resultsNb
ModelForCompare, BeanKey.df))

print("cost")
print(NbBeanCostDiff <- compare.Bean.Cost(correctResForCompare, resultsNbMode
lForCompare, BeanKey.df))

print(NbConfusionMatrix <- confusionMatrix(data = convert.classID.to.class.2.
factor(resultsNbModel), reference = convert.classID.to.class.2.factor(correct
Res)))


options(warn = defaultW)
```

Running the best models on a test dataset to get the final accuracy rate

```
#Suppress warnings
options(warn=-1)
# Multinomial Logistic Regression and Linear Discriminant Analysis

# #Setup for KNN Classifier
# IndependentVars <- c('Area','Eccentricity','ConvexArea','Extent')

    # Predict results
       resultsFinalModelLog <- predict(LogModel,subset(tst.norm.dat,select=c(
1:7)), type="class")
       resultsFinalModelQDA <- predict(LdaModel,subset(tst.norm.dat,select=c(
1:7)), type="class")$class
```

```
correctRes <- factor(tst.norm.dat$ClassID,levels = seq(1,6))




#Capture and print results
correctResForCompare <- data.frame(ClassID = correctRes)

resultsLogModelForCompare <- data.frame(ClassID = resultsFinalModelLog)
resultsLdaModelForCompare <- data.frame(ClassID = resultsFinalModelQDA)

print("Multinomial Logistic Regression weight")
print(FinalLogBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, res
ultsLogModelForCompare, BeanKey.df))

print("Multinomial Logistic Regression cost")
print(FinalLogBeanCostDiff <- compare.Bean.Cost(correctResForCompare, results
LogModelForCompare, BeanKey.df))

print("Multinomial Logistic Regression")
print(FinalLogConfusion <- confusionMatrix(data = convert.classID.to.class.2.
factor(resultsLogModelForCompare), reference = convert.classID.to.class.2.fac
tor(correctRes)))

print("Linear Discriminant Analysis weight")
print(FinalLdaBeanWeightDiff <- compare.Bean.Weight(correctResForCompare, res
ultsLdaModelForCompare, BeanKey.df))

print("Linear Discriminant Analysis cost")
print(FinalLdaBeanCostDiff <- compare.Bean.Cost(correctResForCompare, results
LdaModelForCompare, BeanKey.df))

print("Linear Discriminant Analysis")
print(FinalLdaConfusion <- confusionMatrix(data = convert.classID.to.class.2.
factor(resultsLdaModelForCompare), reference = convert.classID.to.class.2.fac
tor(correctRes)))
```

The predictive analysis included a number of steps to first prepare the data, create models, and then run the models. It involved first converting the dependent variable into a numeric format between 1 and 6 based on the BeanKey.df dataframe. Next it involved normalizing the data via converting them to z-scores and variable selection utilizing the Best Subset Selection method in conjunction with the adjusted r^2 measure. The variable selection process determined that the best combination of variables to use for this analysis were “Area”, “Perimeter”, “Eccentricity”, ConvexArea, and “Extent”. Afterwards the dataset was randomly split, 80% for training/validation and 20% for testing. The models were created with the formula: ClassID = Area + Perimeter + Eccentricity + ConvexArea + Extent, set up in Leave-One-Out-Cross-Validation, and modified via some minor hyperparameter tuning such as optimization of K Nearest Neighbors K value. Finally, the models were run and the best model was chosen based on accuracy, price comparison between predictions and actual results, and weight comparison between predictions and actual results.

#Results Analysis

Results of all of the models when run on the validation dataset: (models with the maximum accuracy, minimum weight differential, and minimum cost differential are highlighted)

```
#Suppress warnings
options(warn=-1)
#Results Table of final model results with a highlight on the chosen model

Model_Name <- c('Multinomial Logistic Regression', 'Linear Discriminant Analy
sis', 'K-Nearest Neighbors', 'Quadratic Discriminant Analysis', 'Naive Bayes 
Analysis')
Accuracy_Rates <- c(LogConfusionMatrix$overall['Accuracy'],LdaConfusionMatrix
$overall['Accuracy'],KnnConfusionMatrix$overall['Accuracy'],QdaConfusionMatri
x$overall['Accuracy'],NbConfusionMatrix$overall['Accuracy'])
Bean_Weights <- c(LogBeanWeightDiff,LdaBeanWeightDiff,KnnBeanWeightDiff,QdaBe
anWeightDiff,NbBeanWeightDiff)
Bean_Costs <- c(LogBeanCostDiff,LdaBeanCostDiff,KnnBeanCostDiff,QdaBeanCostDi
ff,NbBeanCostDiff)
# Join the variables to create a data frame
ValidationResultsDisplay.df <- data.frame(ModelID = seq(1,5),Model_Name, Accu
racy = round(Accuracy_Rates,4), Weight_Differential = round(Bean_Weights,4), 
Cost_Differential =round(Bean_Costs,4))


#Utilize datatable to show results and highlight rows of models with the high
est accuracy and lowest weight and cost differentials
datatable(ValidationResultsDisplay.df) %>% formatStyle(
  c('Accuracy','Weight_Differential','Cost_Differential'),
  target = 'row',
    backgroundColor = styleEqual(
      c(max(ValidationResultsDisplay.df[,'Accuracy']),min(ValidationResultsDi
splay.df[,'Weight_Differential']),min(ValidationResultsDisplay.df[,'Cost_Diff
erential'])), c('lightgreen','lightgreen','lightgreen')
    )
)
```

The results of testing the models on the validation dataset via leave one out cross validation were that the most accurate models were multinomial logistic regression and Quadratic Discriminant Analysis as both had accuracy rates over 91%. The model with the lowest weight differential between predictions and actual values was K Nearest Neighbors. The model with the lowest cost differential between predictions and actual values was Linear Discriminant Analysis. Thus, the multinomial logistic regression model as well as the Linear Discriminant Analysis model were both chosen to be run on the test dataset to get final prediction values as they are both superior in different measures and to simplify the results, accuracy and cost differential were prioritized.

Results of the 2 top chosen models, multinomial logistic regression and linear discriminant analysis, from the validation dataset when run on the normalized test dataset.:

```r
#Suppress warnings
options(warn=-1)

Model_Name <- c('Multinomial Logistic Regression', 'Linear Discriminant Analysis')
Accuracy_Rates <- c(FinalLogConfusion$overall['Accuracy'],FinalLdaConfusion$overall['Accuracy'])
Bean_Weights <- c(FinalLogBeanWeightDiff,FinalLdaBeanWeightDiff)
Bean_Costs <- c(FinalLogBeanCostDiff,FinalLdaBeanCostDiff)
# Join the variables to create a data frame
TestResultsDisplay.df <- data.frame(ModelID = seq(1,2),Model_Name, Accuracy = round(Accuracy_Rates,4), Weight_Differential = round(Bean_Weights,4), Cost_Differential =round(Bean_Costs,4))
#Utilize datatable to show results and highlight rows of models with the highest accuracy and lowest weight and cost differentials
datatable(TestResultsDisplay.df)


options(warn=0)
```

The results on the test dataset indicated strong results on all three measures by both models with each maintaining accuracy rates of at least 87%. #Conclusions The project resulted in two models being chosen for accurately predicting bean data, multinomial logistic regression and linear discriminant analysis. This is due to them either having the highest accuracy rate or the lowest cost differential when tested on the validation dataset compared to the other tested models.