Siyuan Li
PSTAT 131
Instructor: Raya Feldman
Final Project
June 4, 2015

Summary:

Glass is one of the greatest discoveries in the human history. Glass is largely used in our daily life, such as Windows, screens, containers, light bulbs, and many others. The goal of my project is to investigate how different types of glasses are categorized based on its compositions.

The task is classification analysis involving different methods of acquiring different classification models. I selected specific attributes from the data set to perform analysis, built different classification models, and compared them with each other. Using classification tree model, k-NN algorithm and K-Fold Cross Validation technique, I was able to acquire a classification model with acceptable misclassification error rate that is able to categorize glass samples with certain compositions with good accuracy.

Introduction:

The data set I was using is Glass Identification Database created by B. German. The data set contains 214 entries and has 11 attributes. Attributes include: the ID of the entry, the Refractive Index of the sample, Sodium content, Magnesium content, Aluminum content Silicon content, Potassium content, Calcium content, Barium content and Iron content of the sample, and the type of glass it is associated with. Since I wasn't given the classification algorithm used by the creator of the data to classify the entries before hand, I decided to reproduce this algorithm using R.

I started out by pre-processing the data, selecting attributes to analyze, visualizing the data for interpretation, then transforming values for classification. I divided the data set into two part, train set and test set; train set to produce the model, and test set to validate. I constructed classification tree with the train set. I then calculated the optimal number of terminal nodes to decide the final classification tree model. I was able to acquire a 10 terminal node decision tree. For a alternative model, I used k-NN algorithm, producing a black box model that classifies data nodes based on its nearest neighbors. Using K-Fold Cross Validation technique, I eliminated a possible variation of k value and finalized a 1-NN algorithm model. Comparing the decision tree model and the Nearest Neighbor algorithm, I found out that the NN model has higher level of accuracy on the prediction of glass types.

Data Itself:

As stated in the introduction, the data set contains 214 entries and has 11 attributes (Figure 1: dimension of the data set). Attributes include: the ID of the entry, the Refractive Index of the sample, Sodium content, Magnesium content, Aluminum content Silicon content, Potassium content, Calcium content, Barium content and Iron content of the sample, and the type of glass it is associated with.

The type variable has the following definitions:

Type	Definition
1	Building Window (float processed)
2	Building Window (non float processed)
3	Vehicle Window (float processed)
4	Vehicle Window (non float processed)
5	Containers
6	Tablewares
7	Headlamps

```
> dim(data.glass) #acquire dimension information for the dataset
[1] 214    11
> summary(data.glass[2:9]) #observe the features of each attribute
Na                  Mg                  Al                  Si
Min.:10.73          Min.:0.000          Min.:0.290          Min.:69.81
1st Qu.:12.91       1st Qu.:2.115       1st Qu.:1.190       1st Qu.:72.28
Median :13.30       Median :3.480       Median :1.360       Median :72.79
Mean:13.41          Mean:2.685          Mean:1.445          Mean:72.65
3rd Qu.:13.82       3rd Qu.:3.600       3rd Qu.:1.630       3rd Qu.:73.09
Max.:17.38          Max.:4.490          Max.:3.500          Max.:75.41
K                   Ca                  Ba                  Fe
Min.:0.0000         Min.:5.430          Min.:0.000          Min.:0.00000
1st Qu.:0.1225      1st Qu.:8.240       1st Qu.:0.000       1st Qu.:0.00000
Median :0.5550      Median :8.600       Median :0.000       Median :0.00000
Mean:0.4971         Mean:8.957          Mean:0.175          Mean:0.05701
3rd Qu.:0.6100      3rd Qu.:9.172       3rd Qu.:0.000       3rd Qu.:0.10000
Max.:6.2100         Max.:16.190         Max.:3.150          Max.:0.51000
```

Figure 1: dimension and summary statistics of attributes

The attributes that are significant enough in the classification process are: Sodium content, Magnesium content, Aluminum content Silicon content, Potassium content, Calcium content, Barium content and Iron content of the sample. Refractive Index indicates how much light is bent when going through the glass. Generally speaking, it measures to what extent the image through the glass is distorted. As you can see from Figure 1, I excluded Refractive Index because all seven types of the sample glasses require similar permeability of light. I also excluded the IDs of the samples from the data set because it is redundant since R automatically index the entries.

Data Visualization, Transformation and Subsetting:

I moved on to visualization of the data set. The best possible graph to display a complex data set with many attribute is a scatterplot. I constructed scatterplot paring each predictor variable I included with the "type" (response) variable. I also added spread to the points on the graph to indicate the amount of occurrence of certain value. From Figure 2, there are distinct content distribution for each type of glass. It is interesting to note that a type 7 glass has significant level of Barium content that almost no other type of glass has.

Certain transformation of the entries has to be made so that minimum errors will be produced during the calculation and classification processes. I noticed in the data set, the attribute, type of glass, is in numeric format, which can cause R to produce regression tree instead of classification tree. Thus, I re-formatted the type attribute into character format and then into factor format so that these values can be identified as unique labels of the entries. I achieved this process by identifying original “Type” attribute, as in 1, 2, 3... etc., create a new variable “type” that is in “one”, “two”, “three” accordingly, remove the original “Type” attribute from the data set and put the “type” attribute into the data set.

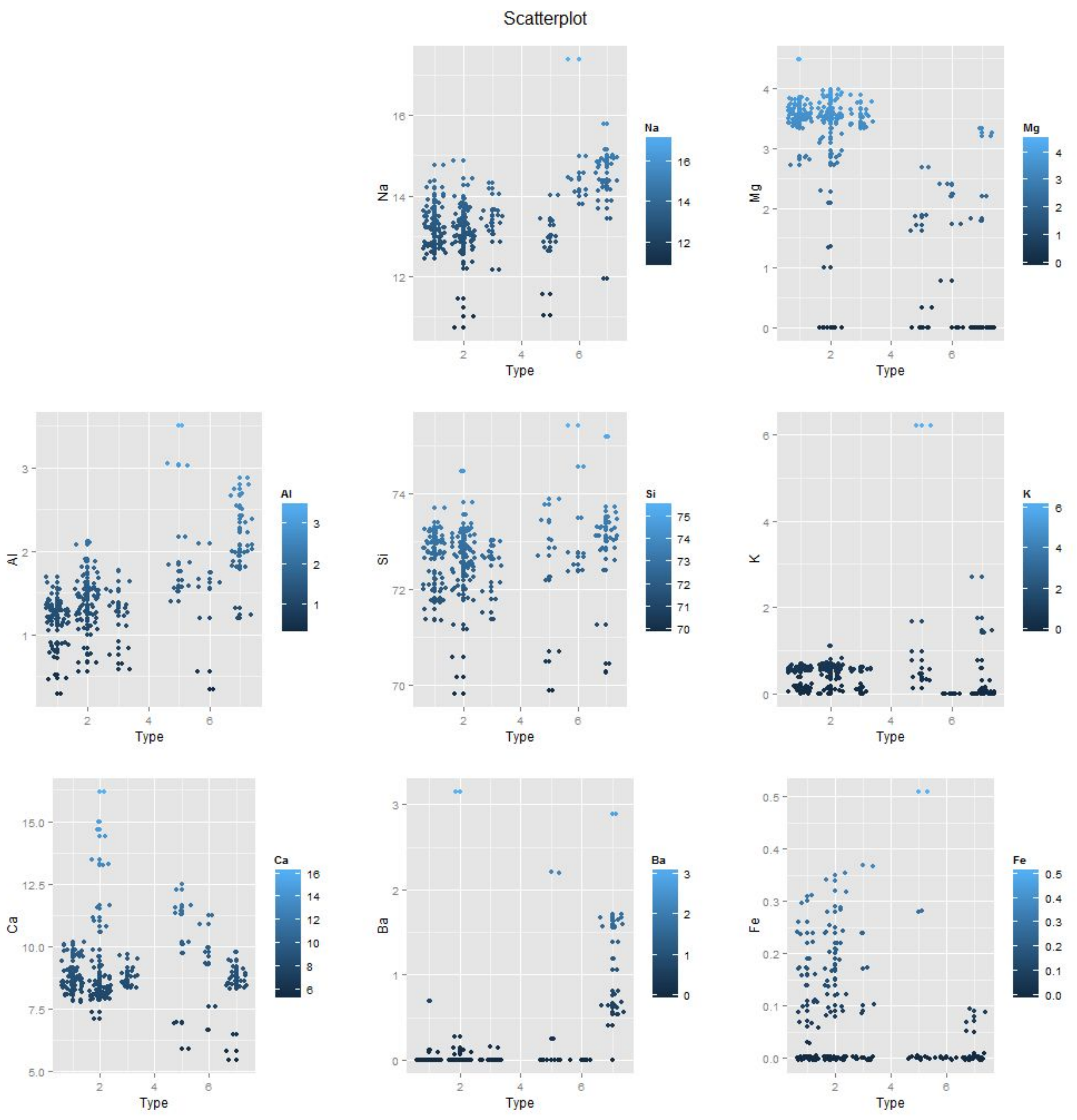

Figure 2: scatter-plot of the data, with each attribute correspond to type

Another preprocessing technique to employ is to split the data set into a train set and a test set. I decided to set train set to include 75% of the original data entries and the test set to have the rest 25%. From Figure 3, we can see that of a total of 214 entries, train set contains 160 entries, test set contains 54 entries. All three sets have the same number of attributes.

```
>#check the dimension of the sets
> dim(data.glass1)
[1] 214   9
> dim(train.glass)
[1] 160   9
> dim(test.glass)
[1] 54  9
```

Figure 3: dimension of the train set and test set

Model: Classification Tree:

With transformations and subsetting done to the data, I proceeded to construct the classification tree model with the training data.

R has functions in tree package that allowed to construct with ease.

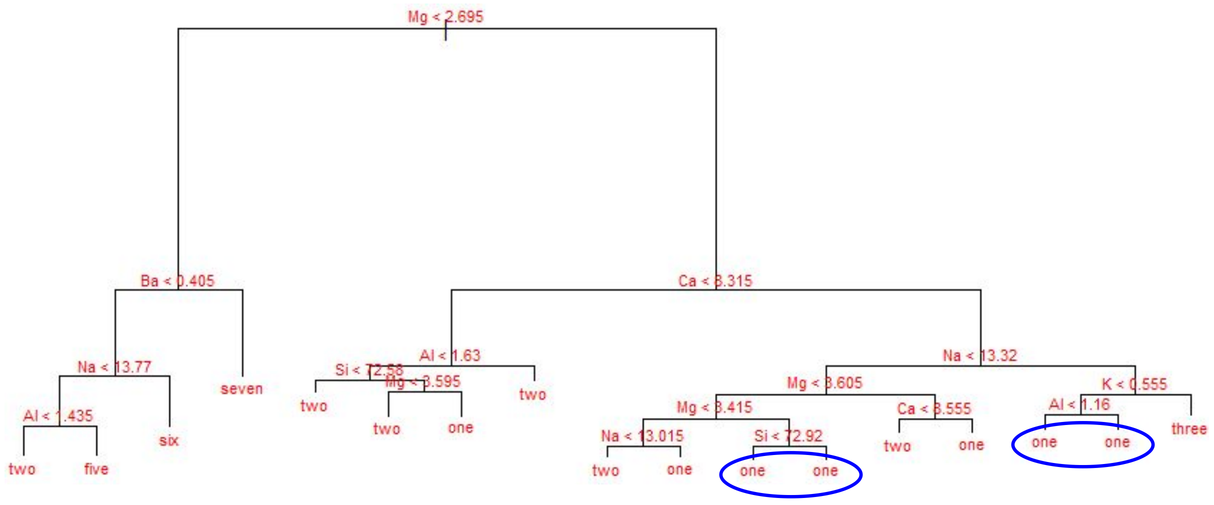

Figure 4: the un-pruned classification tree

Looking at the produced classification tree in Figure 4, we notice that there are two branches that contain the same classification result. This means that the decision process of that branch is a waste of resource, which also means the classification tree needs to be simplified and reduced.

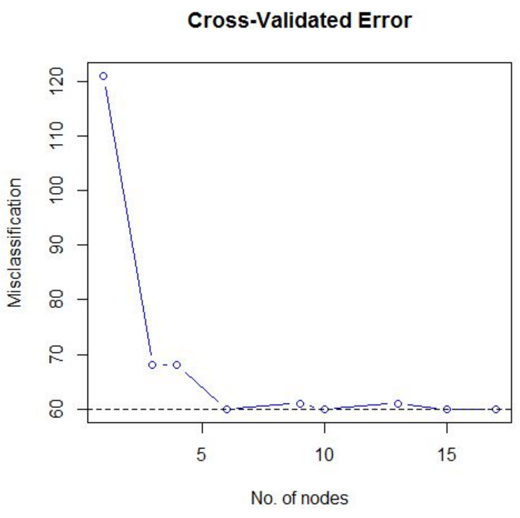

Figure 5: number of misclassifications with respond to number of terminal nodes of the tree

To find the optimal number of terminal nodes for the classification tree. I used the built in cross-validation function to calculate the least number of nodes with minimum misclassification error. As seen in Figure 5, 6 node, 10 node, 15 node and 17 node (which is the original tree) all achieve similar level of misclassification error. Here, I pick 6 node model and check how good it looks.

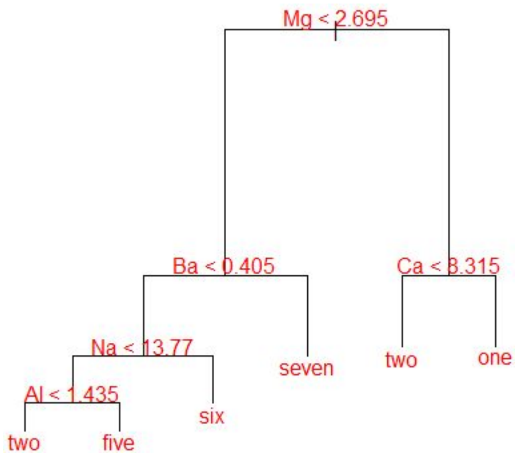

Figure 6: pruned tree with 6 terminal nodes

Figure 6 is the plot of the 6 node model. Notice that, to all six possible classes my model has, the 6 node classification tree only predicts five of the outcomes; no type “three” sample can be classified by the classification tree. Thus I allow more terminal nodes to appear on the classification model, namely the next minimal number of nodes, 10.

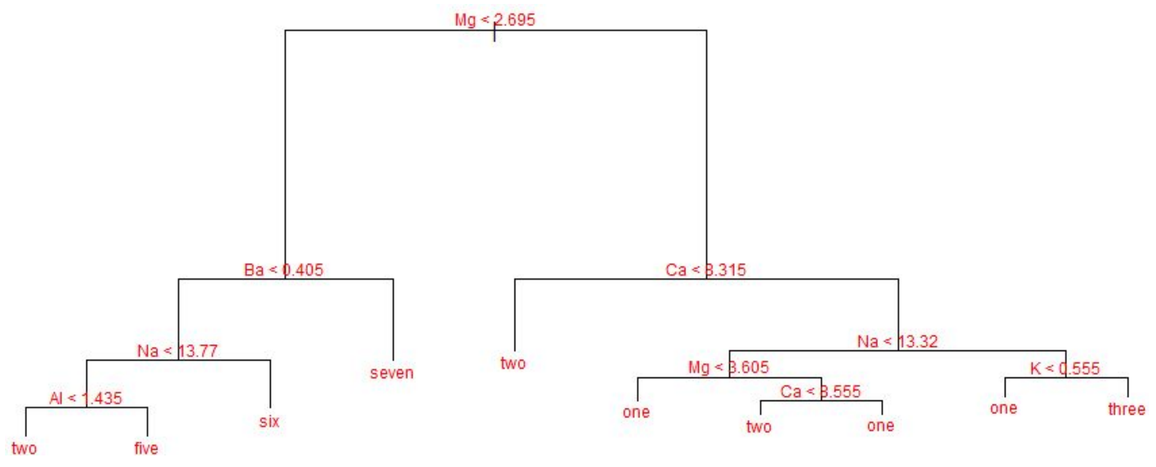

Figure 7: pruned tree with 10 terminal nodes

Figure 7 is the look of the classification tree with 10 terminal nodes. All six “type”s can be classified by the decision tree. I also checked the train set misclassification error by predicting results using pruned(simplified) tree models with 6 and 10 terminal nodes. As can be seen from Figure 8, classification tree with 10 terminal nodes creates less misclassification error than the 6 terminal node tree, meaning 10 node tree is better in classifying train set data.

```
>#prune the tree with optimal terminal nodes
> prune.glass <- prune.misclass(tree.glass,best=6)
# Predict on the training data using pruned tree
> prune.pred <- predict(prune.glass,train.glass,type="class")
# Construct confusion matrix
> train.conf.mat <- table(prune.pred,train.glass$type)
#calculate the misclassification error on training data
> 1-sum(diag(train.conf.mat))/sum(train.conf.mat)
[1] 0.3
> prune.glass <- prune.misclass(tree.glass,best=10)
> prune.pred <- predict(prune.glass,train.glass,type="class")
> train.conf.mat <- table(prune.pred,train.glass$type)
> 1-sum(diag(train.conf.mat))/sum(train.conf.mat)
[1] 0.24375
```

Figure 8: pruned tree misclassification errors

Checking the pruned tree with the original classification tree (17 nodes), it can be seen from Figure 9 (next page) that the original tree has significantly lower misclassification error rate than the pruned tree. However, the original classification tree is over-fitted with redundant terminal nodes. Thus we choose the classification tree with 10 terminal nodes as the first model.

```
>#acquire summary statistics of the tree
> summary(tree.glass)
Classification tree:
tree(formula = type ~ Na + Mg + Al + Si + K + Ca + Ba + Fe, data = train.glass)
Variables actually used in tree construction:
[1] "Mg" "Ba" "Na" "Al" "Ca" "Si" "K"
Number of terminal nodes:  17
Residual mean deviance:  1.01 = 144.5 / 143
Misclassification error rate: 0.1938 = 31 / 160
```

Figure 9: summary statistics of the original classification tree

Model: k-NN

k-NN, k Nearest Neighbor algorithm classifies data nodes by its nearest neighbors, which means if the nearest k number of nodes are of the same class, then the node belongs to that class. I constructed a function that calculates the misclassification error according to the number of nearest neighbor chose, thus producing the “k”.

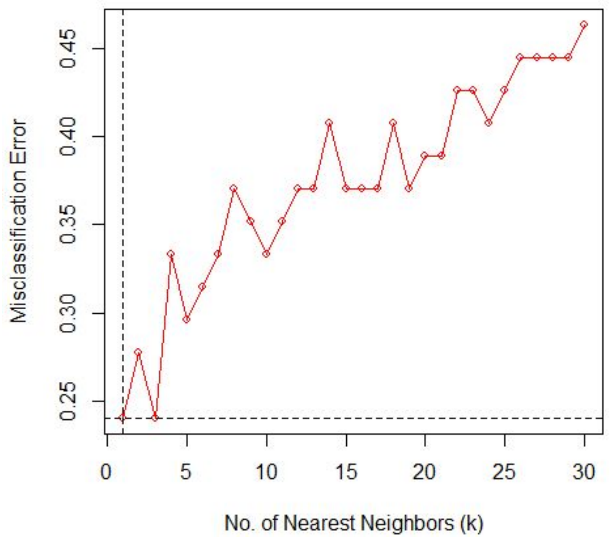

Figure 10: plot of k-NN misclassification error

Figure 10 shows the plot of k-NN misclassification error according to the number of nearest neighbor it chose. As indicated by the lines that marks minimum misclassification error point, a k-NN model with 1 nearest neighbor creates the minimum misclassification error. This simply indicates that the selected node will be assigned to the class of its nearest neighbor. We can also see that 3 nearest neighbor creates similar level of misclassification error. To make certain of the optimal number of nearest neighbor, I used an alternative way of acquiring the “k” value.

The method I use is K-Fold Cross Validation, which randomly partition the data into K equal sized subsets. One of the K subsets are used as test set, and the rest of K-1 sets are used as train set. The k-NN is then calculated K times with every single subset used as test set. I performed 10-Fold Cross Validation, that split the data into 10 subsets and calculate 10 times with each subset as test set.

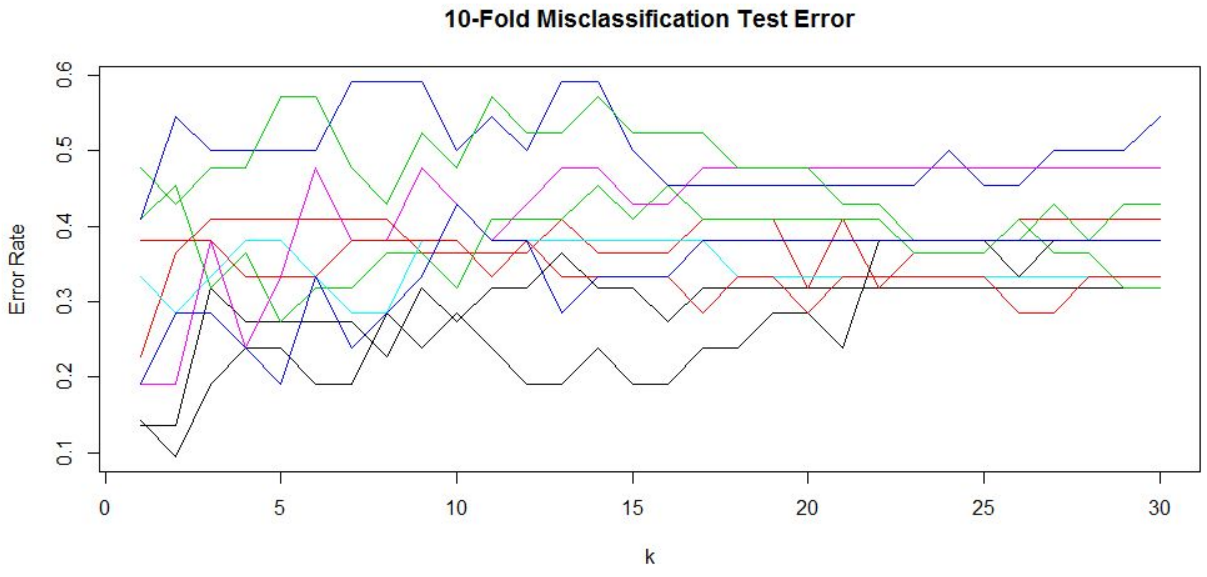

Figure 11: 10-fold misclassification test error

Figure 11 shows the plot of test misclassification error with respond to the number of nearest neighbors. It contains the misclassification error of 10 runs of the k-NN process with each one of ten subsets. Additional modification to the plot is required to interpret the results.

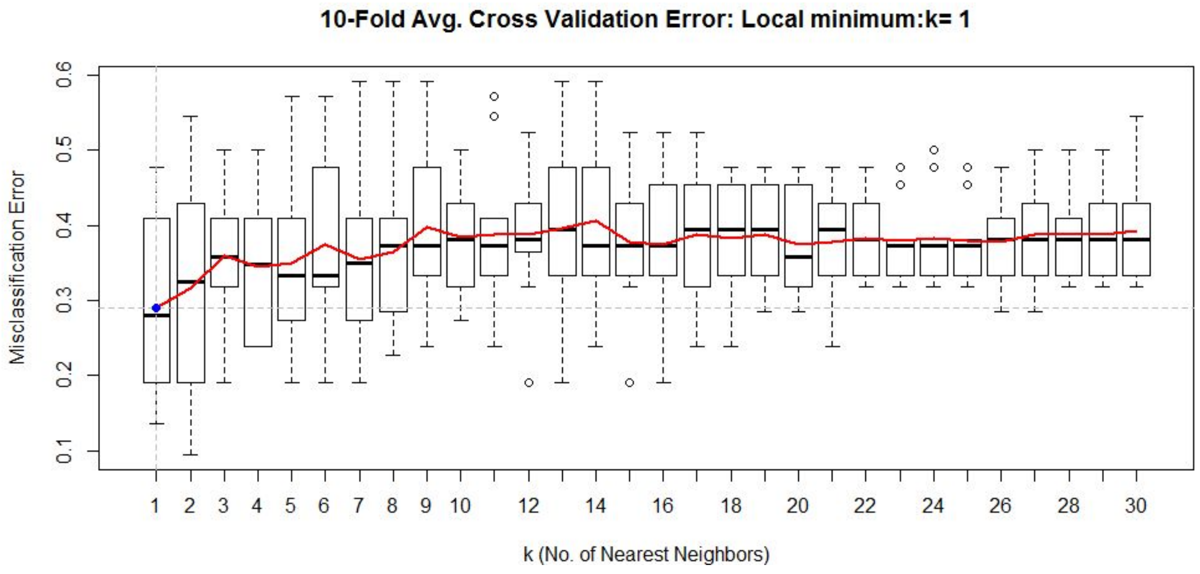

Figure 12: 10-fold avg. cross validation error

As we can see from Figure 12, I calculated the mean misclassification error of 10 runs of the process at each number of nearest neighbors. Plotting the result as boxplot, adding lines indicating minimum, it shows the same result as the original k-NN algorithm. However, it eliminated 3-Nearest Neighbor model as a possible result. Thus I select 1-Nearest Neighbor algorithm as the second model.

Model Comparison:

As analyzing a multiclass data set, I could not use the Receiver Operating Characteristic curve introduced in the lecture to asses the performance of both of my models since it is defined only to work with binary classification problem. Then to evaluate my models, I only look at misclassification errors. It is straightforward, indicating the rate which my models classify entries into a wrong class.

```
> mean.error[which.min(mean.error)]
[1] 0.2896104
> min(knn.error)
[1] 0.2407407
```

Figure 13: 10-Fold CV misclassification error and 1-NN misclassification error

Figure 13 shows that the mean error of 10-Fold Cross Validation of 1-Nearest Neighbor algorithm is 0.2896104 where the minimum of the error is 0.2407407.

```
# Predict on the test data using pruned tree
> prune.test.pred <- predict(prune.glass,test.glass,type="class")
# Construct test data confusion matrix
> test.conf.mat <- table(prune.test.pred,test.glass$type)
#calculate the misclassification error on test data
> 1-sum(diag(test.conf.mat))/sum(test.conf.mat)
[1] 0.2962963
```

Figure 14: 10 node classification tree misclassification error

According to Figure 14, the test data misclassification error of the 10 node classification tree model is 0.2962963, which is higher than the mean error of 1-NN algorithm.

Based on the result of the comparison, it is safe to say that the 1-NN algorithm has higher level of accuracy than the 10 node classification tree model.

Conclusion:

Set out to reproduce the classification algorithm employed by the creator of the data set, using classification tree model, k-NN algorithm and K-Fold Cross Validation technique, I was able to acquire a 1-NN algorithm to categorized glass identification database with about 72% to 76% accuracy and a 10 node classification tree model with about 71% accuracy. However, the nearest neighbor algorithm is very hard for one to read the acquired classification result in a comprehensive way while classification tree models have high level of interpretability. If I will be able to acquire more information on the data set or more classification algorithms I can utilize, I would be able to produce a classification model with even higher accuracy. I would like to give my gratitude to my fellow classmates that provided me with many useful insights on the general direction of the project. I would also like to thank Professor Raya Feldman, for teaching me so many interesting and invaluable real world data mining techniques.

Reference:

Data used:

Glass Identification Database, B. German, Sep 1987, UCI Machine Learning Repository, link: <https://archive.ics.uci.edu/ml/datasets/Glass+Identification>

Appendix:

```
library(ggplot2)
library(gridExtra)
library(class)
library(tree)

#glass identification dataset
#import the data
data.glass <- read.table("D:/Dropbox/UCSB/2015SPRING/PSTAT131/131 Final
Project/glass/glass.txt",sep="," , header = FALSE)
#give column names
colnames(data.glass) <-
c("ID","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type")
#acquire dimension information for the dataset
dim(data.glass)
#exclude ID and RI from the dataset, we do not need it to analyze it
data.glass <- data.glass[3:11]

#create a variable according to the Type of glass
type <- as.factor(ifelse(data.glass$Type == 1,"one",
                 (ifelse(data.glass$Type == 2,"two",
                 (ifelse(data.glass$Type == 3,"three",
                 (ifelse(data.glass$Type == 4,"four",
                 (ifelse(data.glass$Type == 5,"five",
                 (ifelse(data.glass$Type == 6,"six","seven")))))))))))
#select variables into calculation
data.glass <- data.glass[1:8]
#add the type attribute into the data set
data.glass1 <- data.frame(cbind(data.glass,type))

#acquire plots of different attributes in relation to Type
g2 <- qplot(type, Na,  data = data.glass1, colour = Na, geom =
c("point","jitter"))
g3 <- qplot(type, Mg,  data = data.glass1, colour = Mg, geom =
c("point","jitter"))
g4 <- qplot(type, Al,  data = data.glass1, colour = Al, geom =
c("point","jitter"))
g5 <- qplot(type, Si,  data = data.glass1, colour = Si, geom =
c("point","jitter"))
g6 <- qplot(type, K ,   data = data.glass1, colour = K, geom =
c("point","jitter"))
g7 <- qplot(type, Ca,  data = data.glass1, colour = Ca, geom =
c("point","jitter"))
g8 <- qplot(type, Ba,  data = data.glass1, colour = Ba, geom =
c("point","jitter"))
g9 <- qplot(type, Fe,  data = data.glass1, colour = Fe, geom =
c("point","jitter"))
#plot the scatterplot
grid.arrange(g2,g3,g4,g5,g6,g7,g8,g9,ncol = 2, nrow = 4, main =
"Scatterplot")

#set the seed so the answer can be reproducible
set.seed(2)
```

```r
N <- nrow(data.glass1) #set the number of observations of the dataset
#75% for observation
index.train <- sample(1:N,size = floor(N*0.75), replace = FALSE)
#the rest 25% for validation
index.test <- setdiff(1:N,index.train)

#apply the indexing to produce the train set
train.glass <- data.glass1[index.train,]
#apply the indexing to priduce the test set
test.glass <-data.glass1[index.test,]

#check the dimensionality of the sets
dim(data.glass1)
dim(train.glass)
dim(test.glass)

#obtain corresponding values of the response in train and test dataset
class.train <- data.glass1[index.train, "type"]
class.test <-data.glass1[index.test, "type"]

#fit tree with type as response
tree.glass <- tree(type~Na+Mg+Al+Si+K+Ca+Ba+Fe,data=train.glass)
#acquire summary statistics of the tree
summary(tree.glass)
#plot the tree
plot(tree.glass)
#add text to the tree
text(tree.glass,pretty=0,col="red",cex=0.5)

#use the built-in function to calculate the # of optimal nodes
cv.glass <- cv.tree(tree.glass, FUN = prune.misclass, K=2)
#plot the result in response to the misclassification errors they cause
plot(cv.glass$size, cv.glass$dev, type="b", main = "Cross-Validated Error",
    xlab = "No. of nodes", ylab = "Misclassification", col = " blue")

#prune the tree with optimal terminal nodes
prune.glass <- prune.misclass(tree.glass,best=9)
#plot the pruned tree
plot(prune.glass)
#add text to the pruned tree
text(prune.glass,pretty=0,col="red",cex=1)

#predict on the training data using pruned tree
prune.pred <- predict(prune.glass,train.glass,type="class")
#construct confusion matrix
train.conf.mat <- table(prune.pred,train.glass$type)
#calculate the misclassification error on training data
1-sum(diag(train.conf.mat))/sum(train.conf.mat)

#predict on the test data using pruned tree
prune.test.pred <- predict(prune.glass,test.glass,type="class")
#construct test data confusion matrix
test.conf.mat <- table(prune.test.pred,test.glass$type)
#calculate the misclassification error on test data
1-sum(diag(test.conf.mat))/sum(test.conf.mat)

#k-NN
# Select variables into k-NN calculation
vars <-c("Na","Mg","Al","Si","K", "Ca", "Ba", "Fe")
# Maximum number of k's to be considered
m <- 30
```

```r
# Intialize vector for keeping track of knn error
knn.error <- vector()
for (j in 1:m){ # m: Maximum number of values of k
  # Training data with selected variables
  model.knn <- knn(train = train.glass[,vars],
  # Test data with selected variables
  test = test.glass[,vars],
  # labels for training data
      cl = class.train,
  # Number of nearest-neighbors
  k = j,
  # Return posterior probabilities of the class given the obs. data
  prob = T)
  error <- table(model.knn,class.test)
  # Compute Misclassification Error
  knn.error[j] <- 1-sum(diag(error))/sum(error)
}

plot(1:m,knn.error, type="l", col = "red",
     xlab = "No. of Nearest Neighbors (k)",
     ylab = "Misclassification Error")
#add points to previous plot
points(knn.error,col = "red", cex=.8)
#identify the minimum error and add vertical line
abline(v = which.min(knn.error),lty=2)
#horizontal line at the minimum misclassification error
abline(h = min(knn.error),lty=2)

# k-Fold Cross-validation

GetFoldIndex <- function(tt,n.folds){
  #This function creates a vector of indexes that corresponds to n.folds
  #equal size sub-samples taken from tt
  #Arguments:
  #tt: data set with explanatory variables and class variable
  #n.folds: Number of folds (number of sub-samples)
  n <- dim(tt)[1] # Number of observations
  #vector of folds lables
  folds <- rep(1:n.folds,each=floor(n/n.folds))
  remainder <- n-length(folds)
  #number of folds might not be a multiple of total number of obs. so
  #assign remaining obs to a fold systematically: i.e. 1st goes to fold 1,
etc
  if(length(remainder)>0){
    folds <- c(folds,1:remainder)
  }
  #permute indexes
  folds <- sample(folds)
  return(folds)
}

MiscErrorKNN <- function(X,responseY,m,n){
  # Args:
  # X: dataset with explanatory variables
  # responseY : lables
  # m: max value for nearest neighbor
  # n: Number of folds
  error.cv <- list()
  # Add index of sub-samples for n-fold validation
  data.set <- data.frame(X,responseY,Fold=GetFoldIndex(X,n.folds = n))
  # 100% observations plus vector of subsamples
  for(i in 1:n){
    # Training data
    train.set <- subset(data.set, Fold != i)[,colnames(X)]
    # Test data
    test.set <- subset(data.set, Fold == i)[,colnames(X)]
    #Vector of classes
    class.train <- subset(data.set,Fold != i)[,"responseY"]
    class.test <- subset(data.set,Fold == i)[,"responseY"]
    #For these given samples fit k-NN model for several values of k
    knn.error <- vector() # initialize vector
    for (j in 1:m){ # m: Maximum number of values of k
      model.knn <- knn(train = train.set,
                       test = test.set,
                       cl = class.train,
                       k=j,
                       prob=T) # Fit model
      #Compute Error
      error <- table(model.knn, class.test)
      #return(knn.error)
      knn.error[j] <- 1-sum(diag(error))/sum(error)
    }
    error.cv[[i]] <- knn.error
  }
  return(error.cv)
}

CrossValid <- MiscErrorKNN(X = data.glass1[,vars], #Explanatory Variables
                           responseY = data.glass1[,”type”] # labels
                           m=30, #Maximum value of k, for k-NN
                           n=10) #Number of folds (subsamples)
class(CrossValid)
names(CrossValid) <- paste("Sample",1:10) # Assign names

#plot the result of cross validation
matplot(data.frame(CrossValid), type = "l", lty=1,
       ylab = "Error Rate",
       xlab = "k",
       main = "10-Fold Misclassification Test Error")

#get the mean for each k
mean.error <- apply(data.frame(CrossValid),1,mean)
#Box plot of errors
boxplot(t(data.frame(CrossValid)))
#Add mean
lines(1:30,mean.error, type = "l",lwd=2, col = "red")
title(paste("10-Fold Avg. Cross Validation Error: Local minimum:k=",
          which.min(mean.error)),
      xlab = "k (No. of Nearest Neighbors)",
      ylab = "Misclassification Error")
#Place lines to indicate minimum average error and value of k
abline(h=mean.error[which.min(mean.error)], v = which.min(mean.error), col
= "gray",lty = 2)
#Add point to point out the minimum average error and its value of k
points(which.min(mean.error),mean.error[which.min(mean.error)], pch =
19,col = "blue",cex=1)
```