Parin Patel

Naïve Bayes and Decision Tree for Handing Writing Recognition

Introduction:

The ability to recognize handwriting is an invaluable tool for various organizations and people wanting to convert text into digitized documents. This is especially common in our current digital world, where even chemical laboratories are now working to convert older documents into digital formats on their databases. Not only does it allow for ease-of-access, but this conversion allows documents to be better archived, distributed, and analyzed further.

The level of where organizations are using text analysis varies. For example, Microsoft has developed a service called Project Ink Analysis that converts handwritings and drawings into text. This is especially useful for those who work in fields where diagramming and drawing are often used alongside standard texts, like chemist working in a laboratory or database systems architects. Some organizations, however, are just beginning to shuffle through old journals and logs and convert them to digital records. In the past, most paperwork and notes were written on paper and then filed manually.

Today, while most organizations now scan documents for digital access, often the key topics and important information found in the past documents gets overlooked or misinterpreted due to simple misunderstanding of the written text. To combat this, and provide a more systematic record keeping methodology, organizations have begun using handwriting recognition to convert written text into digital ledgers.

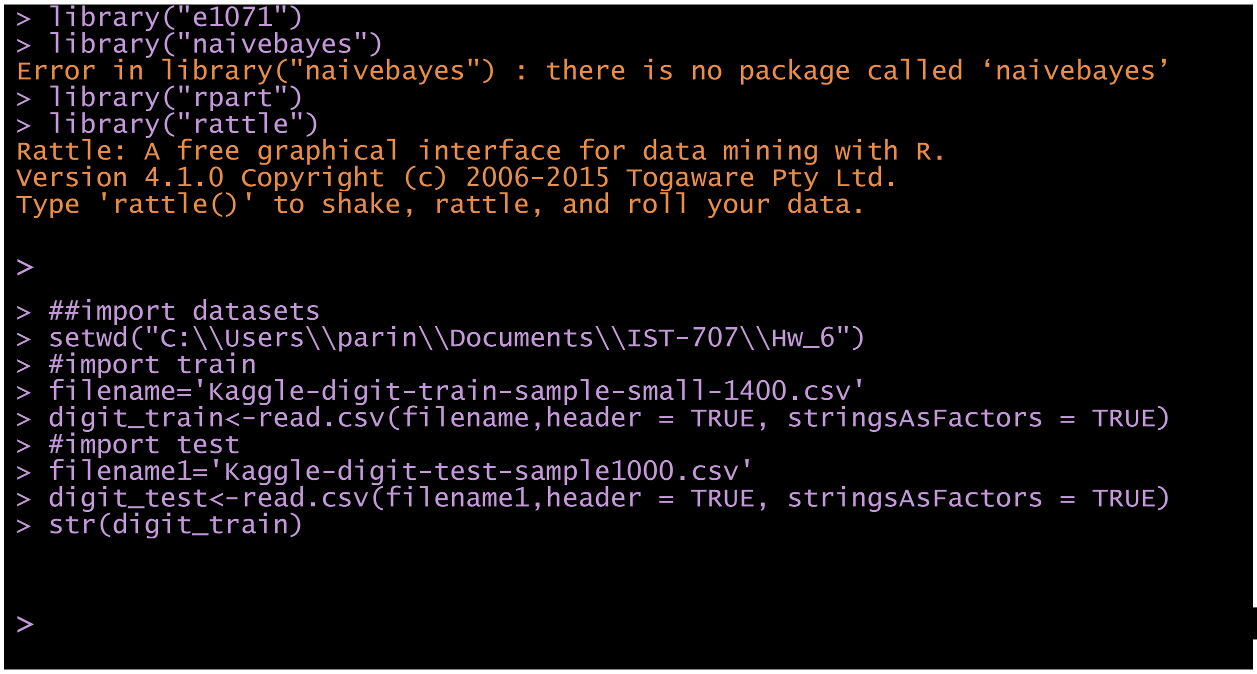
There are numerous prediction methods that have been developed, and knowing when to use each type, based on how it fits the data is key to making a successful prediction. For our purpose, we will use the Naïve Bayes algorithm and decision tree to make the best prediction model in predicting the shape of a hand-written number, zero through nine.

Analysis:

Data Preparation and Processing:

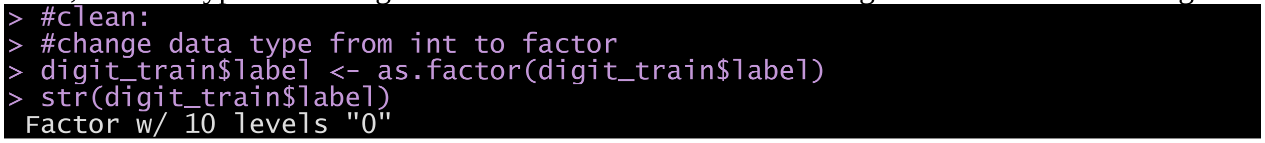
The dataset used was a small sample of the large data set containing information about pixel gray-scale values for 1400 handwritten digits. Each column represented a pixel and its value, which ranged from 0 to 255. This value indicated how dark the pixel is, so the larger the value the darker the pixel. The original training dataset contained 42,000 records over 785 columns, we chose to use the provided smaller dataset, which had 1400 observations over 785 variables. Same went for the test dataset; which originally contained 28,000 observations, where we chose to use the smaller set with 1000observations over 784 variables. The first column in the train set is called “label”. This was removed from the test set in the provided dataset. We mainly chose these sets because the master file is simply too large and would have been split anyway.

The first step in preparing the data required reading the testing and training data.



```
> library("e1071")
> library("naivebayes")
Error in library("naivebayes") : there is no package called ‘naivebayes’
> library("rpart")
> library("rattle")
Rattle: A free graphical interface for data mining with R.
Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
Type 'rattle()' to shake, rattle, and roll your data.

>

> ##import datasets
> setwd("C:\\Users\\parin\\Documents\\IST-707\\Hw_6")
> #import train
> filename='Kaggle-digit-train-sample-small-1400.csv'
> digit_train<-read.csv(filename,header = TRUE, stringsAsFactors = TRUE)
> #import test
> filename1='Kaggle-digit-test-sample1000.csv'
> digit_test<-read.csv(filename1,header = TRUE, stringsAsFactors = TRUE)
> str(digit_train)



>
```

Next, the data type was changed for the label column in the training data to factor from integer.



```
> #clean:
> #change data type from int to factor
> digit_train$label <- as.factor(digit_train$label)
> str(digit_train$label)
 Factor w/ 10 levels "0"
```

Overall, we had a pretty minimum level of cleaning since we used the pre-split sets.

Decision Tree:

For the decision tree, we had to first do some prep-work to properly build multiple trees and ideally get a more accurate tuned model from the test data. First, we excluded pixel values that corresponded with the lighter, outer lines of any written document. We did this, so we could focus on pixels that were darker, so more centered to the handwritten numbers, rather than on the edge. This helps us get a much more accurate understanding of the shape of the symbol. Also, this way, the tree will have fewer values to consider, and therefore, it can generate a potentially more accurate result.

```
> ##Decision Tree Prep
> digit_cleaned <- digit_train
> digit_cleaned2 <- digit_train
> digit_cleaned <- replace(digit_cleaned[,-1],digit_cleaned[,-1] < 100, 0)
> digit_cleaned$label <- digit_train$label
> digit_cleaned2 <- replace(digit_cleaned2[,-1],digit_cleaned2[,-1] < 150, 0)
> digit_cleaned2$label <- digit_train$label

>
```

Finally, I created three models from the pre-processed tables. After this we went ahead and ran our model, which will be discussed in the results section. However, below are visualizations of the three trees. Figure 1 is fit, figure 2 is fit2, and figure 3 is fit3.

```
> fit <- rpart(digit_cleaned$label ~ ., data = digit_cleaned, method="class")
> fit2 <- rpart(digit_train$label ~ ., data = digit_train, method="class")
> fit3 <- rpart(digit_cleaned2$label ~ ., data = digit_cleaned2, method="class")

> fancyRpartPlot(fit)
> fancyRpartPlot(fit2)
> fancyRpartPlot(fit3)
```

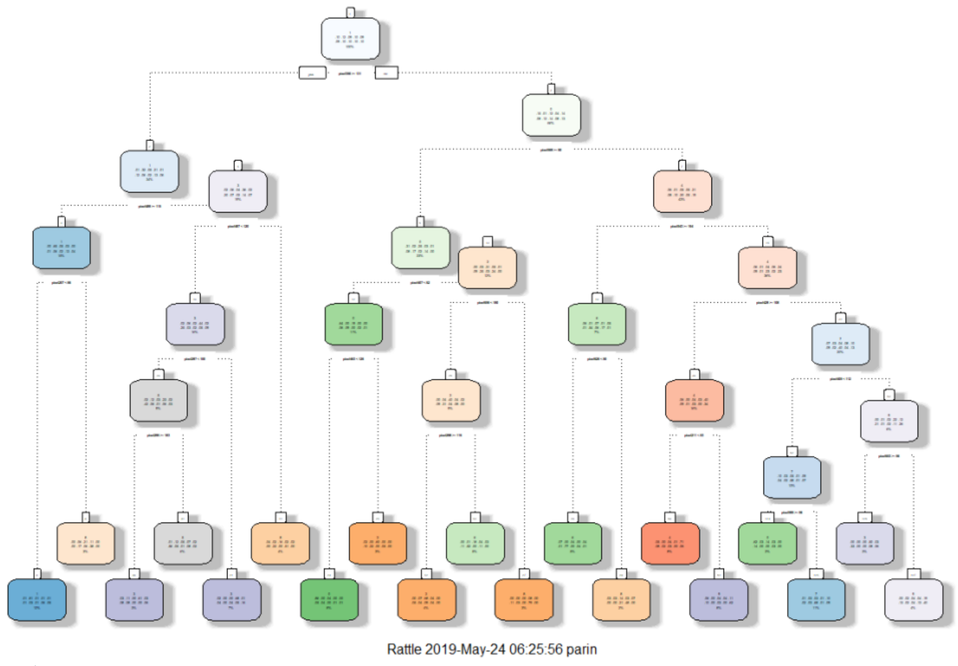


Figure 1

Parin Patel

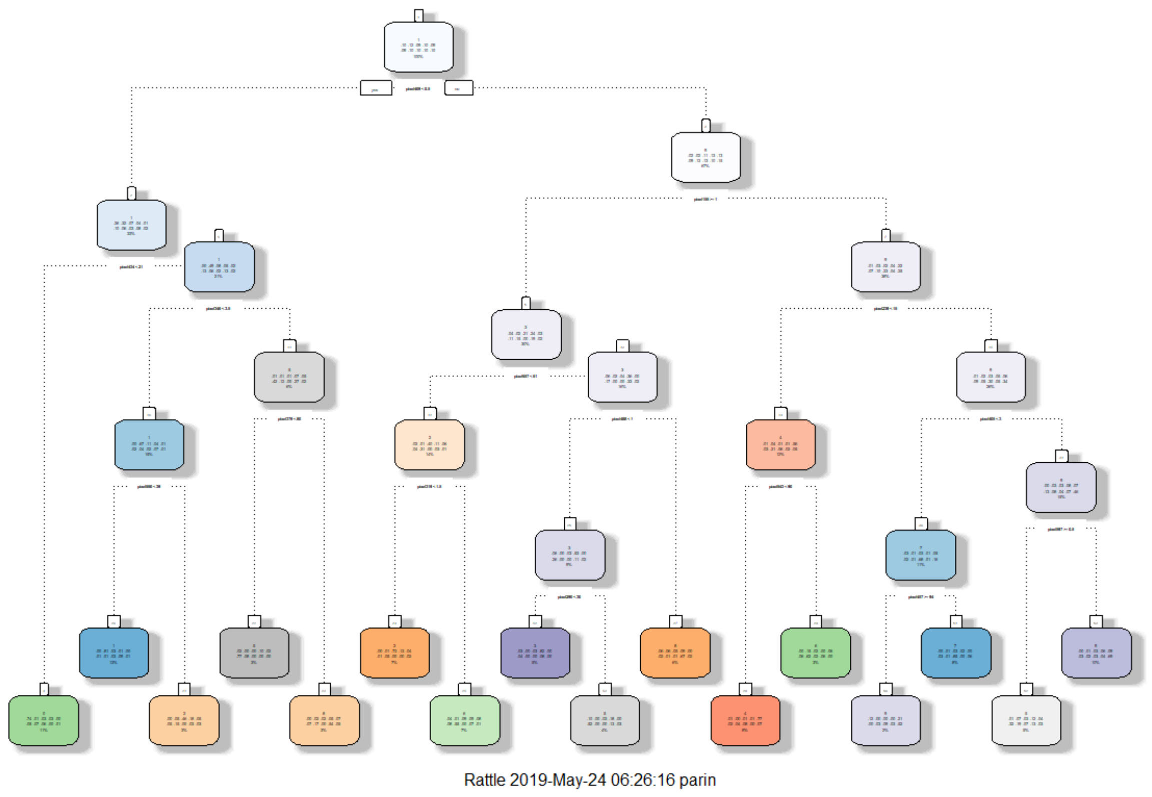


Figure 2

Parin Patel

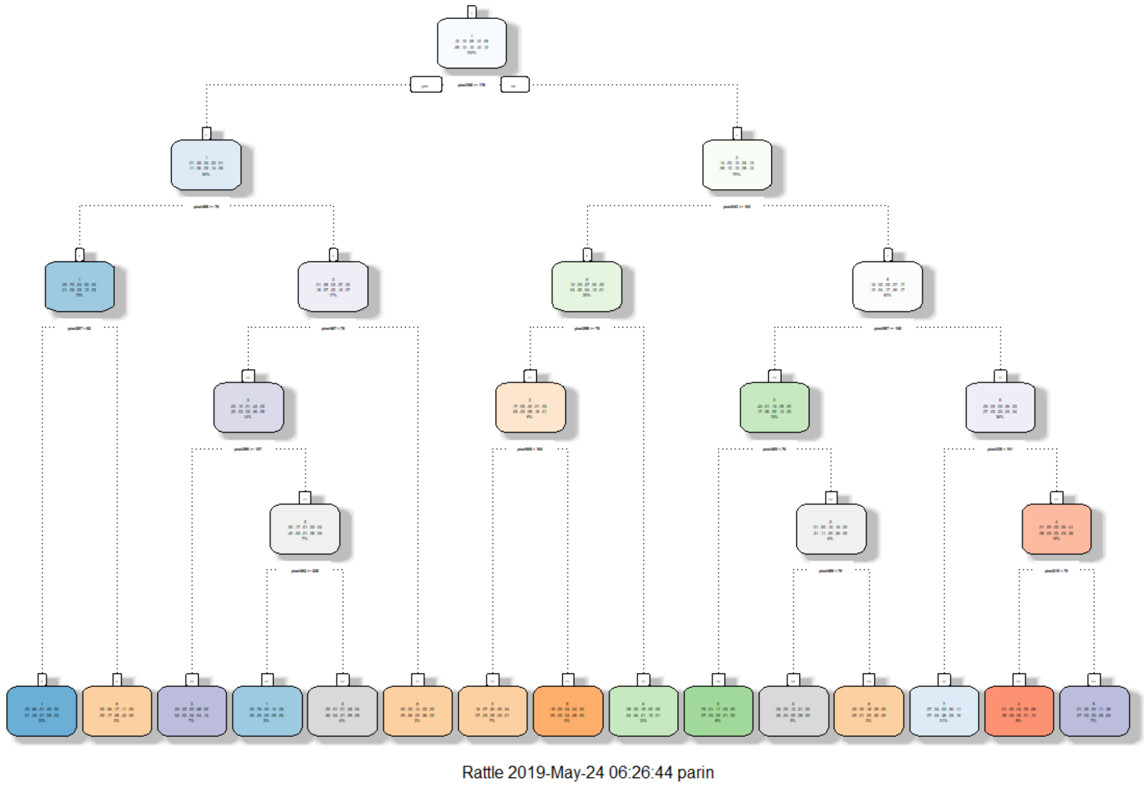
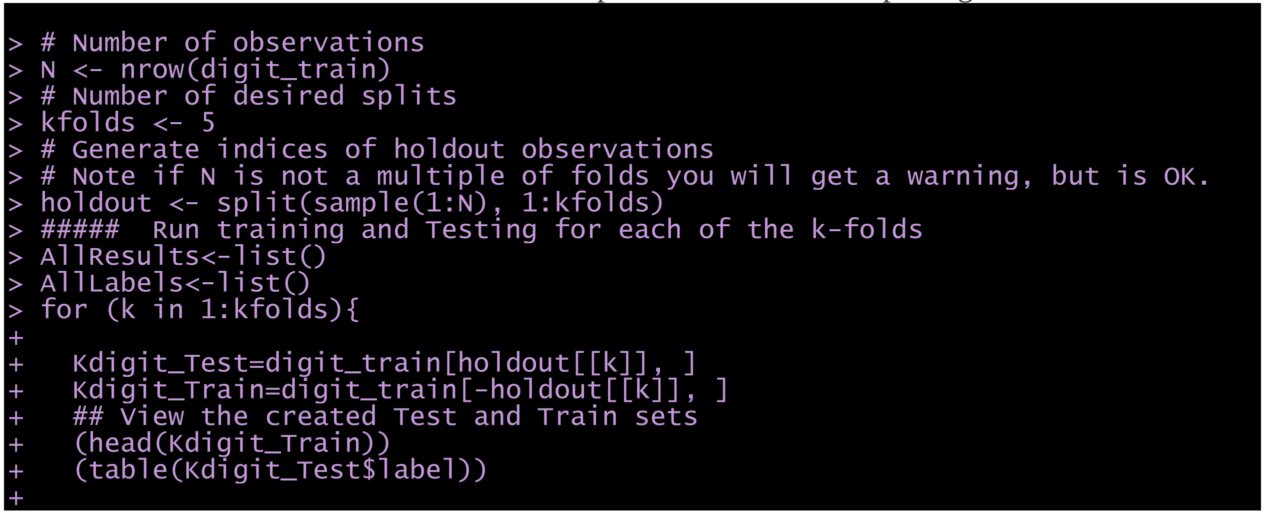


Figure 3

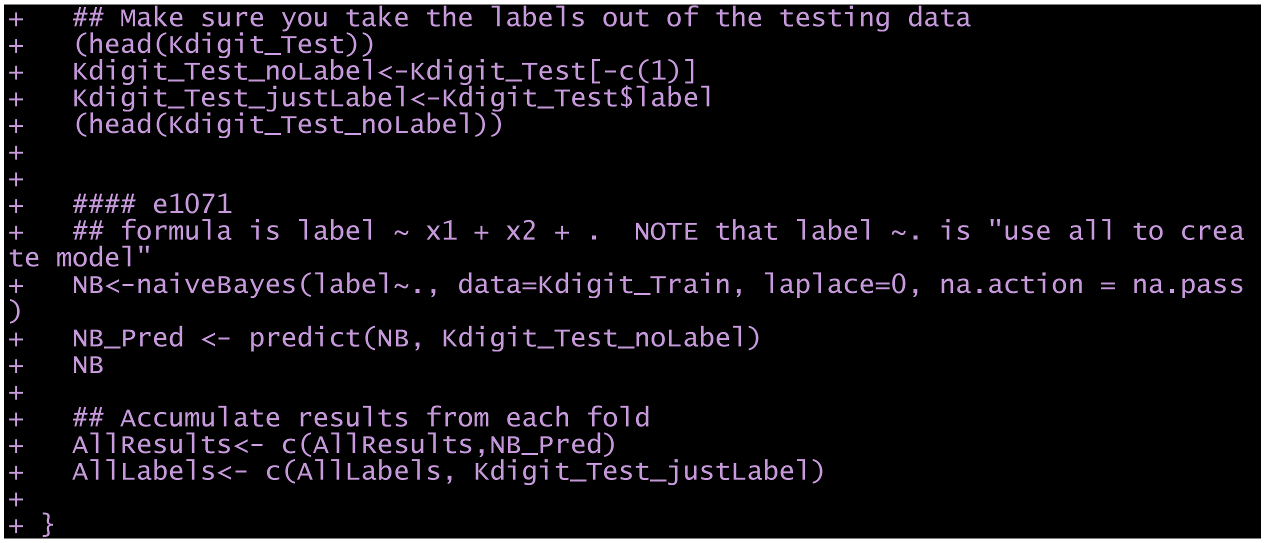
Naïve Bayes

For the Naive Bayes model, two different algorithms were run to compare accuracy of the results. For the purposes of validation, the algorithm was run through a for loop to be able to use cross validation across 5 folds. The first attempt was from the e1071package.



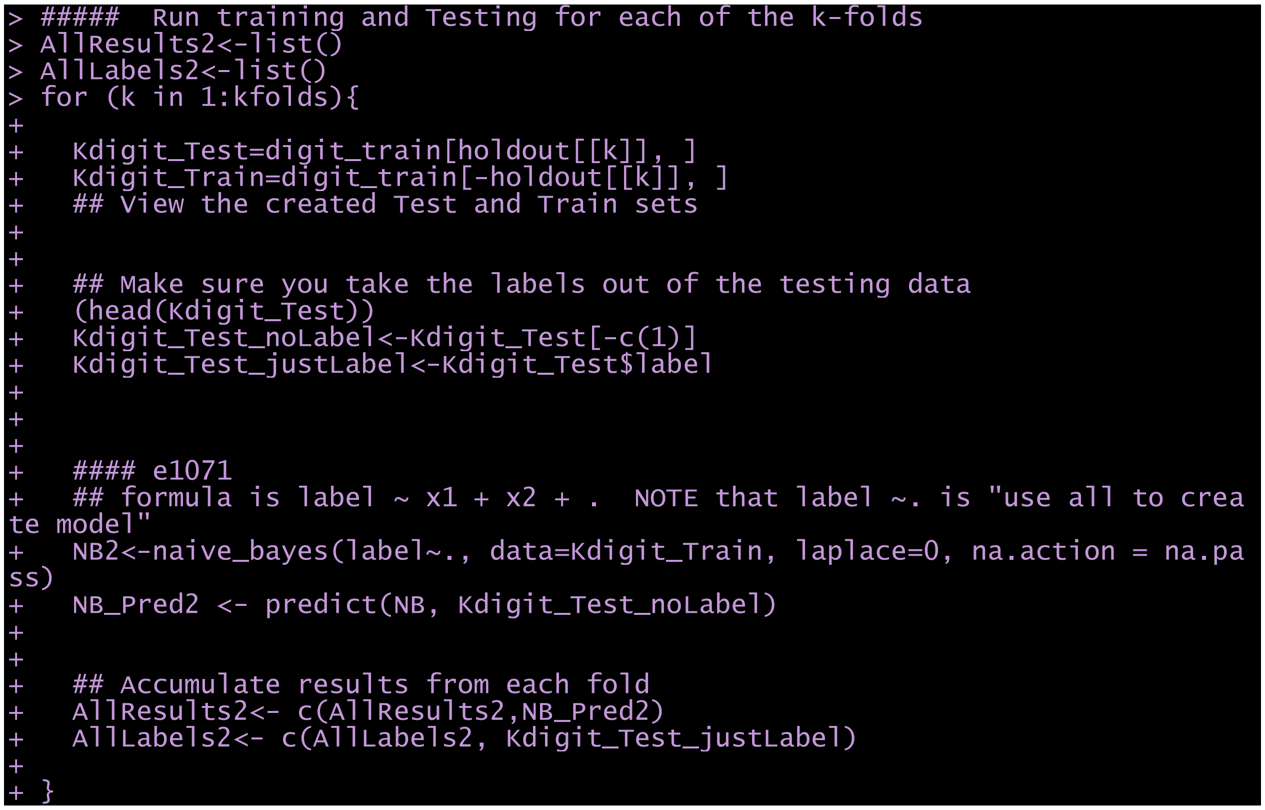
```
> # Number of observations
> N <- nrow(digit_train)
> # Number of desired splits
> kfolds <- 5
> # Generate indices of holdout observations
> # Note if N is not a multiple of folds you will get a warning, but is OK.
> holdout <- split(sample(1:N), 1:kfolds)
> #####  Run training and Testing for each of the k-folds
> AllResults<-list()
> AllLabels<-list()
> for (k in 1:kfolds){
+ 
+   Kdigit_Test=digit_train[holdout[[k]], ]
+   Kdigit_Train=digit_train[-holdout[[k]], ]
+   ## View the created Test and Train sets
+   (head(Kdigit_Train))
+   (table(Kdigit_Test$label))
+ 
```

Parin Patel



```
+   ## Make sure you take the labels out of the testing data
+   (head(Kdigit_Test))
+   Kdigit_Test_noLabel<-Kdigit_Test[-c(1)]
+   Kdigit_Test_justLabel<-Kdigit_Test$label
+   (head(Kdigit_Test_noLabel))
+
+
+   #### e1071
+   ## formula is label ~ x1 + x2 + .  NOTE that label ~. is "use all to crea
te model"
+   NB<-naiveBayes(label~., data=Kdigit_Train, laplace=0, na.action = na.pass
)
+   NB_Pred <- predict(NB, Kdigit_Test_noLabel)
+   NB
+
+   ## Accumulate results from each fold
+   AllResults<- c(AllResults,NB_Pred)
+   AllLabels<- c(AllLabels, Kdigit_Test_justLabel)
+
+ }
```

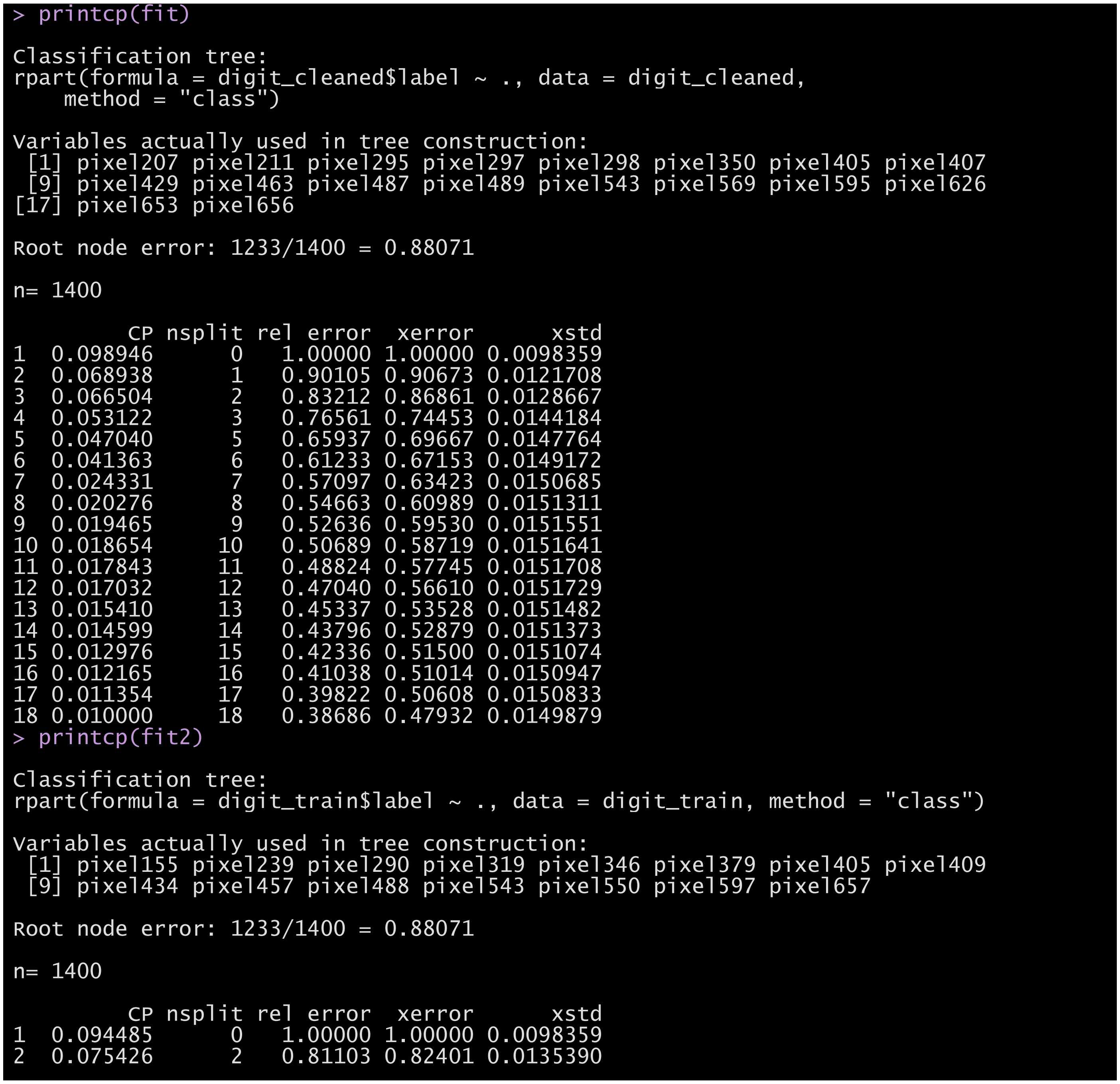
The second model using Naive Bayes was created from the naivebayes packages. The same structure was used to create the cross validated model as above.



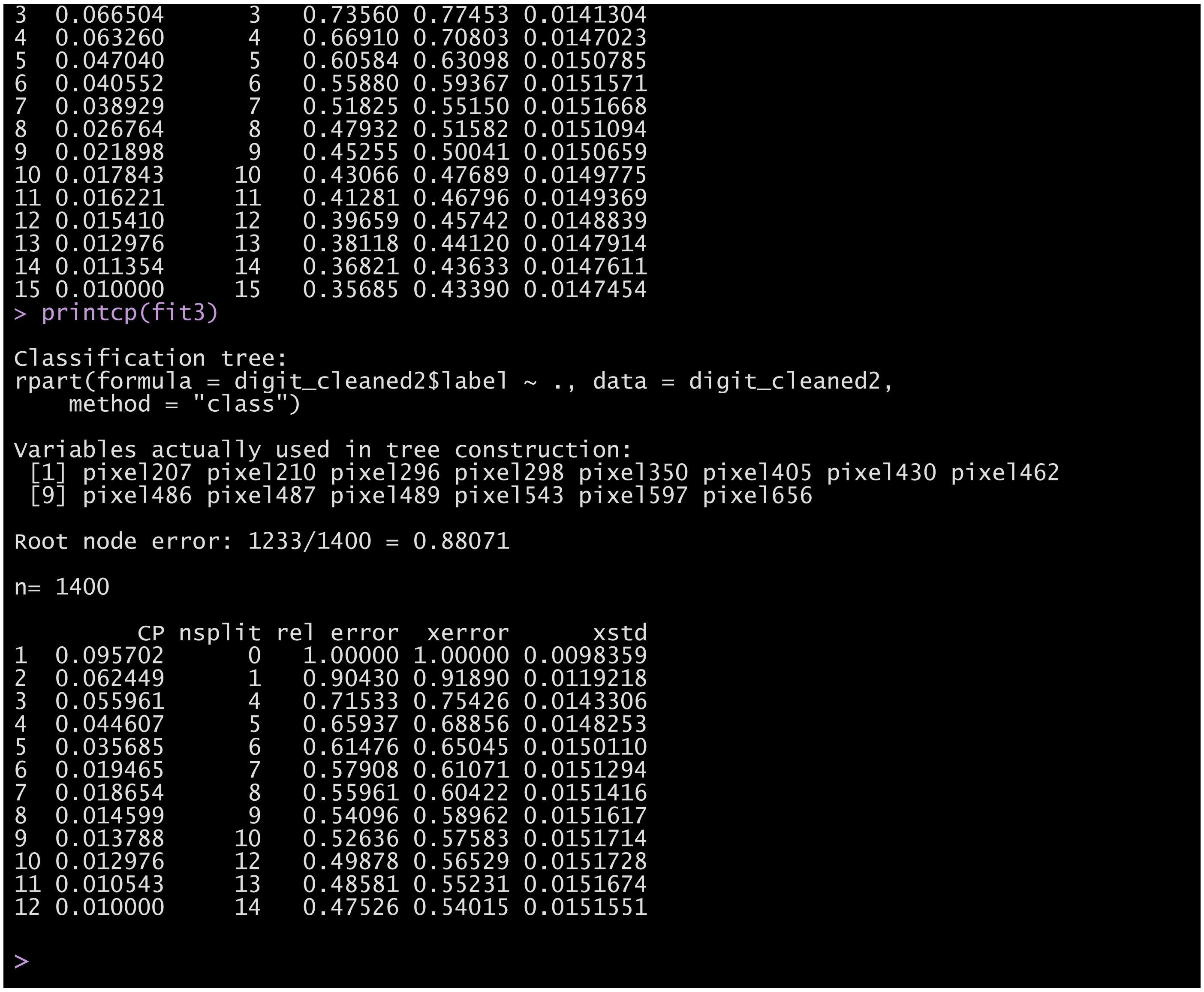
```
> #####  Run training and Testing for each of the k-folds
> AllResults2<-list()
> AllLabels2<-list()
> for (k in 1:kfolds){
+
+   Kdigit_Test=digit_train[holdout[[k]], ]
+   Kdigit_Train=digit_train[-holdout[[k]], ]
+   ## View the created Test and Train sets
+
+
+   ## Make sure you take the labels out of the testing data
+   (head(Kdigit_Test))
+   Kdigit_Test_noLabel<-Kdigit_Test[-c(1)]
+   Kdigit_Test_justLabel<-Kdigit_Test$label
+
+
+
+   #### e1071
+   ## formula is label ~ x1 + x2 + .  NOTE that label ~. is "use all to crea
te model"
+   NB2<-naive_bayes(label~., data=Kdigit_Train, laplace=0, na.action = na.pa
ss)
+   NB_Pred2 <- predict(NB, Kdigit_Test_noLabel)
+
+
+   ## Accumulate results from each fold
+   AllResults2<- c(AllResults2,NB_Pred2)
+   AllLabels2<- c(AllLabels2, Kdigit_Test_justLabel)
+
+ }
```

Results:

Decision Tree:
To measure the accuracy of the decision trees, an output of important values can be generated based on each model. The numbers generated can help calculate the cross-validation error for each model.



```
> printcp(fit)

Classification tree:
rpart(formula = digit_cleaned$label ~ ., data = digit_cleaned,
    method = "class")

Variables actually used in tree construction:
 [1] pixel207 pixel211 pixel295 pixel297 pixel298 pixel350 pixel405 pixel407
 [9] pixel429 pixel463 pixel487 pixel489 pixel543 pixel569 pixel595 pixel626
[17] pixel653 pixel656

Root node error: 1233/1400 = 0.88071

n= 1400

         CP nsplit rel error  xerror      xstd
1  0.098946      0   1.00000 1.00000 0.0098359
2  0.068938      1   0.90105 0.90673 0.0121708
3  0.066504      2   0.83212 0.86861 0.0128667
4  0.053122      3   0.76561 0.74453 0.0144184
5  0.047040      5   0.65937 0.69667 0.0147764
6  0.041363      6   0.61233 0.67153 0.0149172
7  0.024331      7   0.57097 0.63423 0.0150685
8  0.020276      8   0.54663 0.60989 0.0151311
9  0.019465      9   0.52636 0.59530 0.0151551
10 0.018654     10   0.50689 0.58719 0.0151641
11 0.017843     11   0.48824 0.57745 0.0151708
12 0.017032     12   0.47040 0.56610 0.0151729
13 0.015410     13   0.45337 0.53528 0.0151482
14 0.014599     14   0.43796 0.52879 0.0151373
15 0.012976     15   0.42336 0.51500 0.0151074
16 0.012165     16   0.41038 0.51014 0.0150947
17 0.011354     17   0.39822 0.50608 0.0150833
18 0.010000     18   0.38686 0.47932 0.0149879
> printcp(fit2)

Classification tree:
rpart(formula = digit_train$label ~ ., data = digit_train, method = "class")

Variables actually used in tree construction:
 [1] pixel155 pixel239 pixel290 pixel319 pixel346 pixel379 pixel405 pixel409
 [9] pixel434 pixel457 pixel488 pixel543 pixel550 pixel597 pixel657

Root node error: 1233/1400 = 0.88071

n= 1400

         CP nsplit rel error  xerror      xstd
1  0.094485      0   1.00000 1.00000 0.0098359
2  0.075426      2   0.81103 0.82401 0.0135390
```

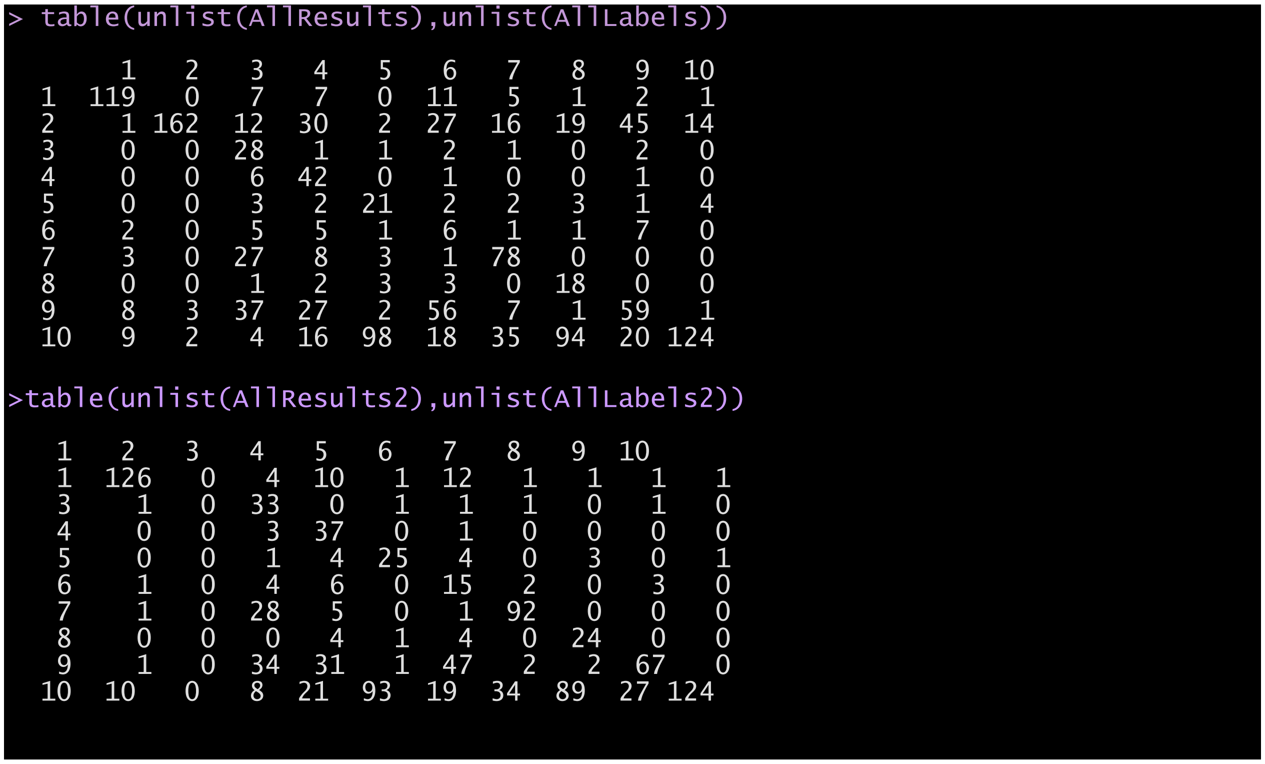


```
3  0.066504      3   0.73560 0.77453 0.0141304
4  0.063260      4   0.66910 0.70803 0.0147023
5  0.047040      5   0.60584 0.63098 0.0150785
6  0.040552      6   0.55880 0.59367 0.0151571
7  0.038929      7   0.51825 0.55150 0.0151668
8  0.026764      8   0.47932 0.51582 0.0151094
9  0.021898      9   0.45255 0.50041 0.0150659
10 0.017843     10   0.43066 0.47689 0.0149775
11 0.016221     11   0.41281 0.46796 0.0149369
12 0.015410     12   0.39659 0.45742 0.0148839
13 0.012976     13   0.38118 0.44120 0.0147914
14 0.011354     14   0.36821 0.43633 0.0147611
15 0.010000     15   0.35685 0.43390 0.0147454
> printcp(fit3)

Classification tree:
rpart(formula = digit_cleaned2$label ~ ., data = digit_cleaned2,
    method = "class")

Variables actually used in tree construction:
 [1] pixel207 pixel210 pixel296 pixel298 pixel350 pixel405 pixel430 pixel462
 [9] pixel486 pixel487 pixel489 pixel543 pixel597 pixel656

Root node error: 1233/1400 = 0.88071

n= 1400

         CP nsplit rel error  xerror      xstd
1  0.095702      0   1.00000 1.00000 0.0098359
2  0.062449      1   0.90430 0.91890 0.0119218
3  0.055961      4   0.71533 0.75426 0.0143306
4  0.044607      5   0.65937 0.68856 0.0148253
5  0.035685      6   0.61476 0.65045 0.0150110
6  0.019465      7   0.57908 0.61071 0.0151294
7  0.018654      8   0.55961 0.60422 0.0151416
8  0.014599      9   0.54096 0.58962 0.0151617
9  0.013788     10   0.52636 0.57583 0.0151714
10 0.012976     12   0.49878 0.56529 0.0151728
11 0.010543     13   0.48581 0.55231 0.0151674
12 0.010000     14   0.47526 0.54015 0.0151551

>
```

In each case, the root node errors did not change from model to model. However, the xerror value changed with each model. By multiplying the rood node errors value with the final xerror value, a percentage of misclassification error based on cross validation can be generated. For each fit they would be as such: fit = 0.880710.49392 = 43.5% fit = 0.880710.43877 = 38.6% fit = 0.88071*0.53447 = 47.1%

Naïve Bayes

For the Naive Bayes models, matrices can be made to show what predictions were made during the cross validation.



```
> table(unlist(AllResults),unlist(AllLabels))

       1   2   3   4   5   6   7   8   9  10
  1  119   0   7   7   0  11   5   1   2   1
  2    1 162  12  30   2  27  16  19  45  14
  3    0   0  28   1   1   2   1   0   2   0
  4    0   0   6  42   0   1   0   0   1   0
  5    0   0   3   2  21   2   2   3   1   4
  6    2   0   5   5   1   6   1   1   7   0
  7    3   0  27   8   3   1  78   0   0   0
  8    0   0   1   2   3   3   0  18   0   0
  9    8   3  37  27   2  56   7   1  59   1
  10   9   2   4  16  98  18  35  94  20 124

>table(unlist(AllResults2),unlist(AllLabels2))

   1   2   3   4   5   6   7   8   9  10
   1  126   0   4  10   1  12   1   1   1   1
   3    1   0  33   0   1   1   1   0   1   0
   4    0   0   3  37   0   1   0   0   0   0
   5    0   0   1   4  25   4   0   3   0   1
   6    1   0   4   6   0  15   2   0   3   0
   7    1   0  28   5   0   1  92   0   0   0
   8    0   0   0   4   1   4   0  24   0   0
   9    1   0  34  31   1  47   2   2  67   0
  10   10   0   8  21  93  19  34  89  27 124
```

Following the diagonal from top left to bottom right shows the total counts of correctly predicted shapes. The first tables show that 683/1400 were correctly predicted, or 48.8% correct. The second table shows 689/1400 were correctly predicted, or 49.2% correct. When these two algorithms were rerun with Laplace smoothing, the results were less accurate or equally accurate as before.

Conclusion:

Based on the results of the accuracy measurements, the decision tree with the mid-tier threshold performed the best. It had the lowest percentage error and was also faster to build. By comparison, the decision trees were all faster to build because it needs to check against few points; the Naive Bayes needs to construct probabilities for many more combinations compared to the node splitting that needs to occur for the decision trees. To make even better models, a better tuning of the threshold selecting in the decision tree could be done. As was demonstrated, by excluding values below 100, the model became more accurate, but when set to 150, the model was less accurate. There could be a threshold value that would make an even better prediction. Outside of these two models, a neural network would most likely be able to produce more accurate results. An algorithm could even be built that could include a pixel “cut off” margin. That is to say, if a certain shape of pixels is dark enough, the algorithm could cut off early and know whatever number would be drawn. For example, if it detects the angle of a 7, something unique among the shape of numbers, it could cut off its analysis and fill in the rest by assuming it's a 7 and move on to the next shape to analyze. This method could potentially speed up the predictions but would run into issues with numbers that have similar structures, like 3 and 8. Finally, there are options in both the Naive Bayes and decision trees that were not accounted for here that could build a better prediction model. For the naive Bayes, kernel density could be used to estimate the probabilities as opposed to a normal distribution. In the decision trees, a matrix could be included to weight the cost of incorrect predictions to try and refine its model.

Appendix A: R Code

```
install.packages("e1071")
install.packages("naivebayes")
install.packages("rpart")
install.packages("rattle")

library("e1071")
library("naivebayes")
library("rpart")
library("rattle")


##import datasets
setwd("C:\\Users\\parin\\Documents\\IST-707\\Hw_6")
#import train
filename='Kaggle-digit-train-sample-small-1400.csv'
digit_train<-read.csv(filename,header = TRUE, stringsAsFactors = TRUE)
#import test
filename1='Kaggle-digit-test-sample1000.csv'
digit_test<-read.csv(filename1,header = TRUE, stringsAsFactors = TRUE)


str(digit_train)
str(digit_train)

#clean:
#change data type from int to factor
digit_train$label <- as.factor(digit_train$label)
str(digit_train$label)

##Decision Tree Prep
digit_cleaned <- digit_train
digit_cleaned2 <- digit_train
digit_cleaned <- replace(digit_cleaned[,-1],digit_cleaned[,-1] < 100, 0)
digit_cleaned$label <- digit_train$label
digit_cleaned2 <- replace(digit_cleaned2[,-1],digit_cleaned2[,-1] < 150, 0)
```

Parin Patel

```
digit_cleaned2$label <- digit_train$label

fit <- rpart(digit_cleaned$label ~ ., data = digit_cleaned, method="class")
fit2 <- rpart(digit_train$label ~ ., data = digit_train, method="class")
fit3 <- rpart(digit_cleaned2$label ~ ., data = digit_cleaned2, method="class")
fancyRpartPlot(fit)
fancyRpartPlot(fit2)
fancyRpartPlot(fit3)

predict_tree= predict(fit,digit_test, type="class")


##################################################################
#############  Create k-folds for k-fold validation ###########
##################################################################


# Number of observations
N <- nrow(digit_train)
# Number of desired splits
kfolds <- 5
# Generate indices of holdout observations
# Note if N is not a multiple of folds you will get a warning, but is OK.
holdout <- split(sample(1:N), 1:kfolds)


#####  Run training and Testing for each of the k-folds
AllResults<-list()
AllLabels<-list()
for (k in 1:kfolds){

  Kdigit_Test=digit_train[holdout[[k]], ]
  Kdigit_Train=digit_train[-holdout[[k]], ]
  ## View the created Test and Train sets
  (head(Kdigit_Train))
  (table(Kdigit_Test$label))

  ## Make sure you take the labels out of the testing data
  (head(Kdigit_Test))
  Kdigit_Test_noLabel<-Kdigit_Test[-c(1)]
  Kdigit_Test_justLabel<-Kdigit_Test$label
  (head(Kdigit_Test_noLabel))


  #### e1071
  ## formula is label ~ x1 + x2 + .  NOTE that label ~. is "use all to create model"
  NB<-naiveBayes(label~., data=Kdigit_Train, laplace=0, na.action = na.pass)
  NB_Pred <- predict(NB, Kdigit_Test_noLabel)
  NB

  ## Accumulate results from each fold
  AllResults<- c(AllResults,NB_Pred)
  AllLabels<- c(AllLabels, Kdigit_Test_justLabel)

}
### end crossvalidation -- present results for all folds
table(unlist(AllResults),unlist(AllLabels))
#####  Run training and Testing for each of the k-folds
AllResults2<-list()
AllLabels2<-list()
for (k in 1:kfolds){

  Kdigit_Test=digit_train[holdout[[k]], ]
  Kdigit_Train=digit_train[-holdout[[k]], ]
  ## View the created Test and Train sets


  ## Make sure you take the labels out of the testing data
  (head(Kdigit_Test))
  Kdigit_Test_noLabel<-Kdigit_Test[-c(1)]
  Kdigit_Test_justLabel<-Kdigit_Test$label



  #### e1071
  ## formula is label ~ x1 + x2 + .  NOTE that label ~. is "use all to create model"
  NB2<-naive_bayes(label~., data=Kdigit_Train, laplace=0, na.action = na.pass)
  NB_Pred2 <- predict(NB, Kdigit_Test_noLabel)


  ## Accumulate results from each fold
  AllResults2<- c(AllResults2,NB_Pred2)
  AllLabels2<- c(AllLabels2, Kdigit_Test_justLabel)

}

table(unlist(AllResults2),unlist(AllLabels2))

printcp(fit)
printcp(fit2)
printcp(fit3)
```

Parin Patel