Decision Tree and Naïve Bayes Algorithm for MNIST

Table of Contents

Instructions……………………………………..……………3

Introduction and Overview of Dataset……………...4

Data Cleaning ………..………………….………………...5

Exploratory Data Analysis……………………………...9

Model Data Pre-Processing…………………………13

Decision Tree Algorithm…………………………..…16

Naïve Bayes Algorithm………………………..…...44

Model Comparison and Conclusion………………..49

Appendix (PCA Analysis…………………………..…51

Instructions

Task description:

The data set comes from the Kaggle Digit Recognizer competition. The goal is to recognize digits 0 to 9 in handwriting images. Because the original data set is too large to be loaded in Weka GUI, I have systematically sampled 10% of the data by selecting the 10th, 20th examples and so on. You are going to use the sampled data to construct prediction models using naïve Bayes and decision tree algorithms. Tune their parameters to get the best model (measured by cross validation) and compare which algorithms provide better model for this task. Due to the large size of the test data, submission to Kaggle is not required for this task. However, 1 extra point will be given to successful submissions. One solution for the large test set is to separate it to several smaller test set, run prediction on each subset, and merge all prediction results to one file for submission. You can also try use the entire training data set, or re-sample a larger sample.

https://www.kaggle.com/c/digit-recognizer/data

Section 1: Introduction Briefly describe the classification problem and general data preprocessing. Note that some data preprocessing steps maybe specific to a particular algorithm. Report those steps under each algorithm section.

Section 2: Decision tree Build a decision tree model. Tune the parameters, such as the pruning options, and report the 3-fold CV accuracy.

Section 3: Naïve Bayes Build a naïve Bayes model. Tune the parameters, such as the discretization options, to compare results.

Section 4: Algorithm performance comparison Compare the results from the two algorithms. Which one reached higher accuracy? Which one runs faster? Can you explain why?

Section 5: Kaggle test result - Report the test accuracy for the naïve Bayes and decision tree models. Discuss whether overfitting occurs in these models.

Introduction

MNIST “Modified National Institute of Standards and Technology” is a database of large collections of handwritten digits ranging from numbers 0-9 in 28X28 pixel boxes. This database is largely used for training machines to “read” the images based on gradient of each pixel. Through various machine learning techniques, including K-Nearest Neighbors, Random Forest, SVMs, and Neural Networks, data scientists have achieved a near-human accuracy of processing the correct number in these images, with an error rate of between 0.4% and 0.2%.[1]

In this report, we will use two prediction models to attempt to correctly classify the handwritten digits based on the population of various pixels: Decision Tree Algorithms and the Naive Bayes Algorithm.

Overview of the Dataset

The digit_train dataset has 785 attributes and 42,000 observations. The first column is the label, or the integer drawn by the user. The next 784 columns each represent a pixel (28 pixels high and 28 pixels wide). Each row is one drawing. The name of each pixel (starting from 0, Pixel0-Pixel783) Indicates where the pixel is located on the image. Pixel0-Pixel27 are the top 1/28th of the image, the next row is Pixel28-Pixel55, and so on.

The digit_test dataset includes 28,000 observations that are labeled as “unknown”. The purpose of the analysis is to assign accurate labels of the digits to each image.

Challenges of the Dataset

This dataset is unconventional compared to many datasets used for machine learning due to its high dimensionality and extremely large sample size. This may lend to two issues in our analysis.

Firstly, the high dimensionality (783 dimensions) may impair the algorithm with excess noise. This is known as the “curse of dimensionality”. Certain models such as SVM models automatically reduce dimensionality, but decision trees and Naive Bayes algorithms do not. Therefore, we may have better results by reducing the dimensionality of the dataset through Principal Component Analysis or other methods.

Secondly, the extremely large sample size (42,000 observations) may cause slow processing speed as we train our models. Therefore, it may be useful to decrease the size of the dataset.

[1] https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

Loading Dataset and Required Packages

Load Required Packages:

```
library(rpart)
library(rpart.plot)
library(e1071)
library(tree)
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.3     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.4.4     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.0
## ✓ purrr     1.0.2
## ── Conflicts ──────────────────────────────────────────── tidyverse_conflicts()
──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
errors

library(caret)

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

Load Data:

```
digit_test<- read.csv("~/Downloads/Kaggle-digit-test-1 (2).csv")
digit_train<- read.csv("~/Downloads/Kaggle-digit-train (2).csv")
head(digit_train)
```

```
##   label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
## 1     1      0      0      0      0      0      0      0      0      0      0
## 2     0      0      0      0      0      0      0      0      0      0      0
## 3     1      0      0      0      0      0      0      0      0      0      0
## 4     4      0      0      0      0      0      0      0      0      0      0
## 5     0      0      0      0      0      0      0      0      0      0      0
## 6     0      0      0      0      0      0      0      0      0      0      0
##   pixel10 pixel11 pixel12 pixel13 pixel14 pixel15 pixel16 pixel17 pixel18
## 1       0       0       0       0       0       0       0       0       0
## 2       0       0       0       0       0       0       0       0       0
## 3       0       0       0       0       0       0       0       0       0
## 4       0       0       0       0       0       0       0       0       0
## 5       0       0       0       0       0       0       0       0       0
## 6       0       0       0       0       0       0       0       0       0
[TRUNCATED]
```

```
#Looking at Structure of Datasets
str(digit_train)
```

```
## 'data.frame':    42000 obs. of  785 variables:
##  $ label   : int  1 0 1 4 0 0 7 3 5 3 ...
##  $ pixel0  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel1  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel2  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel3  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel4  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel5  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel6  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel7  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel8  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel9  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel10 : int  0 0 0 0 0 0 0 0 0 0 ...

##   [list output truncated]

str(digit_test)

## 'data.frame':    28000 obs. of  785 variables:
##  $ label   : chr  "?" "?" "?" "?" ...
##  $ pixel0  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel1  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel2  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel3  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel4  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel5  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel6  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel7  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel8  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel9  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pixel10 : int  0 0 0 0 0 0 0 0 0 0 ...

##   [list output truncated]

#Analyzing dimensionality of datasets
dim(digit_train)

## [1] 42000   785

#any dataset with over 100 dimensions is regarded as high dimensionality
```

Initial Data Cleaning

Data Cleaning is an extremely important step of the data analysis process. If we do not feed manageable and easily interpretable data to our models, we will likely not get a good accuracy result in our predictions.

Firstly, we will transform the “label” column to factors rather than characters. This will improve the interpretability of this column for our models.

```
digit_train$label<-as.factor(digit_train$label)
```

The train data currently includes 42,000 observations, while the test data includes 28,000 observations. It would be beneficial to to decrease the size of both datasets to improve the speed of the analysis. Therefore, I will take a sample of 10,000 from the train dataset and 5000 from the test dataset.

```
set.seed(22)
sample_digit_train<-digit_train[sample(nrow(digit_train),10000), ]
sample_digit_test<-digit_test[sample(nrow(digit_test),5000), ]
```

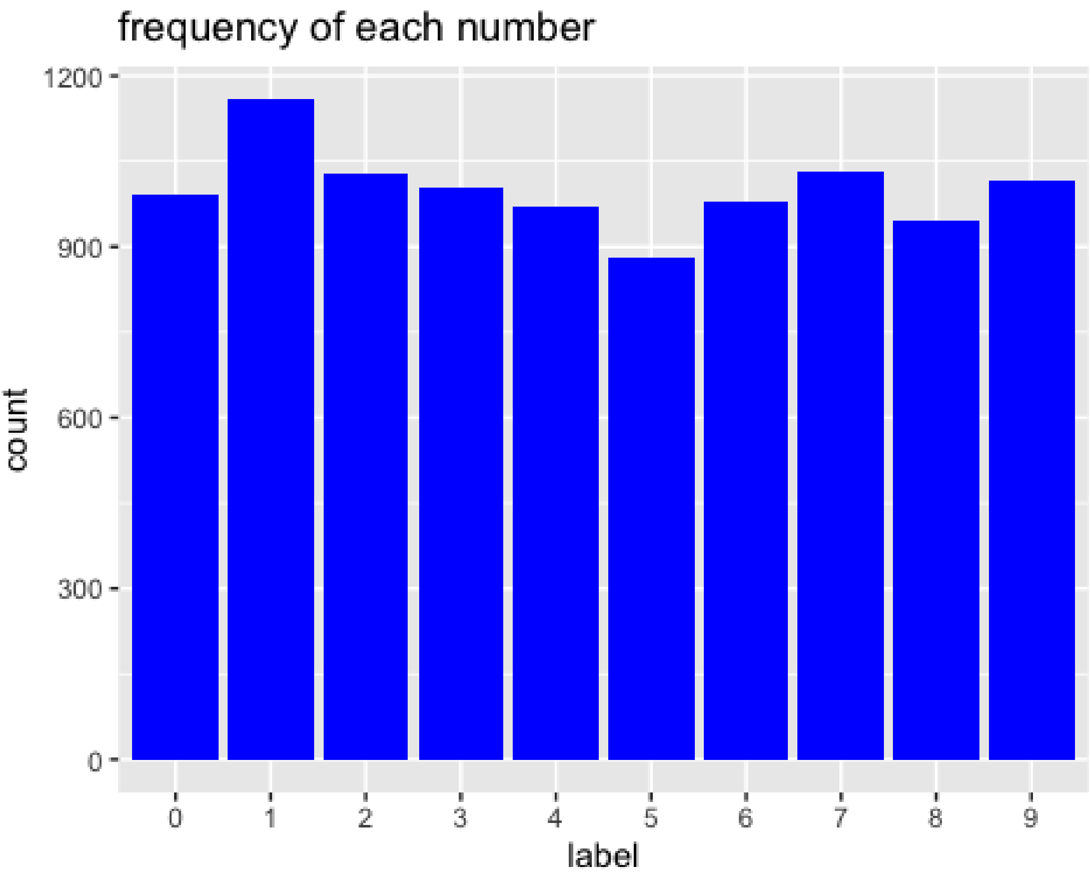
Finally, we will binarize the data in the pixel columns so that all values over 0 will be interpreted as 1s. Therefore, there is no gradient of how dark the pixel is, but rather the pixel will either be populated (1) or not populated (0). This simplifies our data and improves the results of our analysis.

```
sample_digit_train_nolab<-sample_digit_train[,-1]
sample_digit_train_nolab[sample_digit_train_nolab > 0] <- 1
sample_digit_train_nolab$label<-sample_digit_train$label
sample_digit_train<-sample_digit_train_nolab
head(sample_digit_train)

##       pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
## 31113      0      0      0      0      0      0      0      0      0      0
## 34122      0      0      0      0      0      0      0      0      0      0
## 24421      0      0      0      0      0      0      0      0      0      0
## 39808      0      0      0      0      0      0      0      0      0      0
## 5392       0      0      0      0      0      0      0      0      0      0
## 11464      0      0      0      0      0      0      0      0      0      0
##       pixel10 pixel11 pixel12 pixel13 pixel14 pixel15 pixel16 pixel17 pixel18
## 31113       0       0       0       0       0       0       0       0       0
## 34122       0       0       0       0       0       0       0       0       0
## 24421       0       0       0       0       0       0       0       0       0
## 39808       0       0       0       0       0       0       0       0       0
## 5392        0       0       0       0       0       0       0       0       0
## 11464       0       0       0       0       0       0       0       0       0
[TRUNCATED
```

Exploratory Data Analysis

To get a better understanding of the dataset, first we will take a look graphically at how many labels we have by count in our dataset.

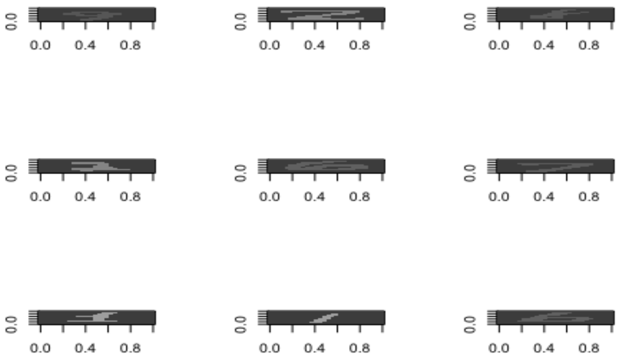
```
label_freq<-sample_digit_train%>%count(label)
ggplot(data=sample_digit_train, aes(x=label)) + geom_bar(stat="count", fill = "blue") +
labs(title = "frequency of each number")
```

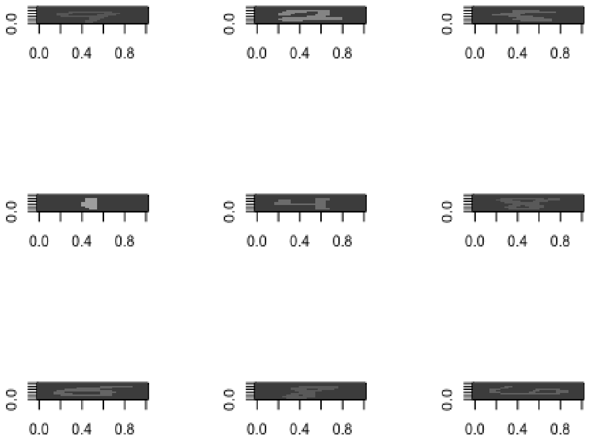


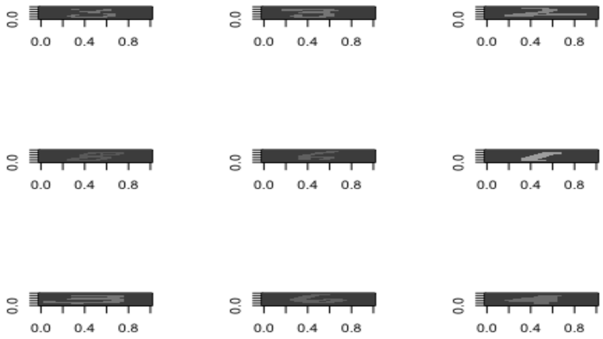
Each number appears relatively equally in our dataset, which shows us that our sample has sufficient examples of each label to continue in our analysis.

Now, let’s visualize the first 9 numbers from our dataset in their image format:

```
flip <- function(matrix){
apply(matrix, 2, rev)
}
par(mfrow=c(3,3))
for (i in 1:27){
dit <- flip(matrix(rev(as.numeric(sample_digit_train[i,-c(1, 786)])), nrow = 28)) #look at DigitTotalDF
image(dit, col = grey.colors(255))
}
```







```
flip(sample_digit_train[1:9,])

##       pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
## 14075 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 23617 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 24351 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 11464 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 5392  "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 39808 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 24421 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 34122 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
## 31113 "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"    "0"
##       pixel10 pixel11 pixel12 pixel13 pixel14 pixel15 pixel16 pixel17 pixel18
## 14075 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 23617 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 24351 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 11464 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 5392  "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 39808 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 24421 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 34122 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
## 31113 "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"     "0"
##
##       pixel780 pixel781 pixel782 pixel783 label
## 14075 "0"      "0"      "0"      "0"      "6"
## 23617 "0"      "0"      "0"      "0"      "1"
## 24351 "0"      "0"      "0"      "0"      "1"
## 11464 "0"      "0"      "0"      "0"      "7"
## 5392  "0"      "0"      "0"      "0"      "6"
## 39808 "0"      "0"      "0"      "0"      "2"
## 24421 "0"      "0"      "0"      "0"      "8"
## 34122 "0"      "0"      "0"      "0"      "2"
## 31113 "0"      "0"      "0"      "0"      "9"

[TRUNCATED]
```

It will also be useful for us to take a look at individual digits to understand the variance in pixel population for a single digit. “8” is a digit that covers a lot of surface area and has a relatively unique shape. Therefore, I will subset the data to only include digits labeled “8” for our analysis.

```
eight<-sample_digit_train[sample_digit_train$label == "8",]
eight_pixel_sum<-eight%>%
  select(pixel1:pixel783) %>%
  pivot_longer(cols = everything(), names_to = "pixel", values_to = "frequency") %>%
  group_by(pixel) %>%
  summarise(Total = sum(frequency))
eight_pixel_sum<-eight_pixel_sum%>%arrange(desc(Total))
eight_populated<-eight_pixel_sum[eight_pixel_sum$Total>0,]
nrow(eight_populated)/nrow(eight_pixel_sum)

## [1] 0.6487867

nrow(eight_pixel_sum)

## [1] 783
```

Theoretically, if you placed all 783 of the "8" images on top of each other, they would cover 63.6% of the white space in the image. This shows large variability in the locations and types of handwriting.

```
mean(rowSums(eight == 1))

## [1] 175.746

#average number of pixels for one image
(mean(rowSums(eight == 1)))/783

## [1] 0.2244521
```

The average "8" image has covers only 22.05% of the white space in the image. This confirms that there is a large variability between images of the same digit, and therefore there may be difficulty in determining the correct digit based on the population of each specific pixel.

Preprocessing of Data to feed to Decision Tree and Naive Bayes Models

In order to begin training our model, we will first divide the training set into two datasets: train and test. After training our models with the “train” set, we will apply it to the test dataset to see the accuracy rate in which is predicts the correct label.

```
set.seed(22)
#use 70% as training set and 30% as test set
sample <- sample(c(TRUE, FALSE), nrow(sample_digit_train), replace=TRUE,
prob=c(0.7,0.3))
train<-sample_digit_train[sample,]
test<-sample_digit_train[!sample,]
```

Check new train and test samples:

```
dim(train)

## [1] 7028  785

dim(test)
```

```
## [1] 2972  785
```

The train and test datasets are very high dimensional datasets (over 100 attributes) and therefore are prone to overfitting and the “curse of dimensionality”. One way to reduce the dimensionality is to remove pixels that are rarely populated, such as pixels in the margin and corners of the image.

```
head(train)

##       pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9
## 31113      0      0      0      0      0      0      0      0      0      0
## 34122      0      0      0      0      0      0      0      0      0      0
## 39808      0      0      0      0      0      0      0      0      0      0
## 24351      0      0      0      0      0      0      0      0      0      0
## 14075      0      0      0      0      0      0      0      0      0      0
## 16928      0      0      0      0      0      0      0      0      0      0
##       pixel10 pixel11 pixel12 pixel13 pixel14 pixel15 pixel16 pixel17 pixel18
## 31113       0       0       0       0       0       0       0       0       0
## 34122       0       0       0       0       0       0       0       0       0
## 39808       0       0       0       0       0       0       0       0       0
## 24351       0       0       0       0       0       0       0       0       0
## 14075       0       0       0       0       0       0       0       0       0
## 16928       0       0       0       0       0       0       0       0       0
0
##       pixel780 pixel781 pixel782 pixel783 label
## 31113        0        0        0        0     9
## 34122        0        0        0        0     2
## 39808        0        0        0        0     2
## 24351        0        0        0        0     1
## 14075        0        0        0        0     6
## 16928        0        0        0        0     9

[TRUNCATED]

pixel_sum<-train%>%
  select(pixel1:pixel783) %>%
  pivot_longer(cols = everything(), names_to = "pixel", values_to = "frequency") %>%
  group_by(pixel) %>%
  summarise(Total = sum(frequency))
pixel_sum

## # A tibble: 783 × 2
##    pixel    Total
##    <chr>    <dbl>
##  1 pixel1       0
##  2 pixel10      0
##  3 pixel100   570
##  4 pixel101   515
##  5 pixel102   428
##  6 pixel103   306
##  7 pixel104   205
##  8 pixel105   120
##  9 pixel106    51
## 10 pixel107    19
## # ℹ 773 more rows

#add new column showing frequency percent
numtrainimages<-nrow(train)
pixel_sum<-pixel_sum%>%mutate(percent = (Total/numtrainimages)*100)

#order by most frequently occurring
pixel_sum<-pixel_sum%>%arrange(desc(Total))

pixel_sum

## # A tibble: 783 × 3
##    pixel    Total percent
##    <chr>    <dbl>   <dbl>
##  1 pixel435  5143    73.2
##  2 pixel407  5127    73.0
##  3 pixel408  5082    72.3
##  4 pixel211  5024    71.5
##  5 pixel210  4959    70.6
##  6 pixel380  4944    70.3
##  7 pixel212  4943    70.3
##  8 pixel434  4919    70.0
##  9 pixel436  4918    70.0
## 10 pixel602  4913    69.9
## # ℹ 773 more rows
```

If the pixel occurs in more than 5% of cases, then it is likely useful for us to include in our analysis. Therefore, we can remove all pixels that occur in less than 5% of cases.

```
pixel_sum_above5<-pixel_sum[pixel_sum$percent>0.05,]
pixels_above5<-pixel_sum_above5$pixel
```

Now decrease dataset to include only pixels that occur in at least 5% of the dataset.

```
train1<-train
train1<-train1[,pixels_above5]
train1$label<-train$label #ensure labels are still attached

test1<-test
test1<-test1[,pixels_above5]
test1$label<-test$label #ensure labels are still attached


#also repeat for test data

sample_digit_test<-sample_digit_test[,pixels_above5]

dim(train1)

## [1] 7028  621
```

Now that the dimensionality is reduced from 783 to 625 attributes, this may reduce overfitting in our model. We will use this new, reduced dimensionality dataset to feed to our model and test it against using the full dimensionality dataset to see which performs better for both our decision tree and naive bayes model.

Decision Tree

Overview of Decision Tree Model

A decision tree model is a hierarchal, non-parametric supervised learning algorithm for classification and regression tasks. In our analysis, we will be using its classification capabilities to see if it can correctly classify each digit based on binary decision nodes, which create homogenous subsets.[2] The tree consists of a root node, internal nodes, and several leaf nodes which represent the outcomes of the analysis.

Decision trees are notoriously prone to overfitting, or overcomplexity that hinders the accuracy of the model. As a decision tree grows in size and complexity, it generally becomes less accurate. Occam's Razor states that simplicity generally offers a better classification result. Therefore, we will prune the tree based on an optimal cp (complexity parameter) value.

Decision Tree Building

```
#rpart summary
digit_train_rpart = rpart(label ~ ., data = train1,
                    method = "class")
summary(digit_train_rpart)

## Call:
## rpart(formula = label ~ ., data = train1, method = "class")
##   n= 7028
##
##            CP nsplit rel error    xerror        xstd
## 1  0.09572759      0 1.0000000 1.0000000 0.004281212
## 2  0.09235464      1 0.9042724 0.9092515 0.005329738
## 3  0.06553164      2 0.8119178 0.8119178 0.006050610
## 4  0.06360424      4 0.6808545 0.6855124 0.006575696
## 5  0.05011243      5 0.6172502 0.6236749 0.006695284
## 6  0.04690010      6 0.5671378 0.5907485 0.006725170
## 7  0.04031481      7 0.5202377 0.5266624 0.006717434
```

[2] https://www.ibm.com/topics/decision-trees)

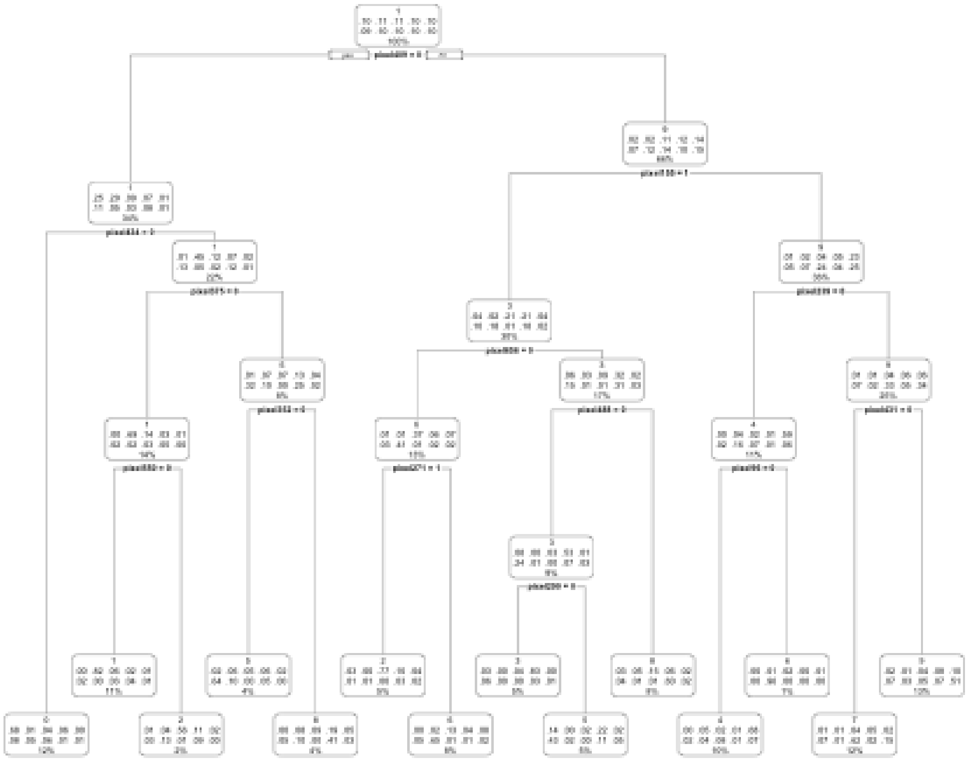
```
## 8  0.02280758      8 0.4799229 0.4853839 0.006666190
## 9  0.01831031      9 0.4571153 0.4656280 0.006628596
## 10 0.01590106     10 0.4388050 0.4473177 0.006586036
## 11 0.01349181     11 0.4229040 0.4317379 0.006543868
## 12 0.01060071     12 0.4094121 0.4187279 0.006504378
## 13 0.01000000     13 0.3988114 0.4106971 0.006478026
## 
## Variable importance
## pixel433 pixel461 pixel434 pixel406 pixel462 pixel409 pixel435 pixel155 
##        4        4        3        3        3        3        2        2 
## pixel403 pixel239 pixel156 pixel154 pixel656 pixel431 pixel375 pixel657 
##        2        2        2        2        2        2        2        2 
## pixel238 pixel655 pixel153 pixel157 pixel381 pixel437 pixel240 pixel271 
##        2        2        2        2        2        2        2        1 
## pixel432 pixel430 pixel488 pixel658 pixel211 pixel410 pixel654 pixel127 
##        1        1        1        1        1        1        1        1 
## pixel408 pixel243 pixel382 pixel347 pixel299 pixel489 pixel487 pixel429 
##        1        1        1        1        1        1        1        1 
## pixel267 pixel374 pixel516 pixel515 pixel237 pixel659 pixel550  pixel96 
##        1        1        1        1        1        1        1        1 
## pixel272 pixel346 pixel215 pixel402 pixel244 pixel551 pixel124  pixel97 
##        1        1        1        1        1        1        1        1 
## pixel290  pixel95 pixel123 pixel522  pixel68 pixel352 pixel552 pixel578 
##        1        1        1        1        1        1        1        1 
## pixel523 pixel291 pixel289 pixel262 pixel263 pixel318 
##        1        1        1        1        1        1 
## 
## Node number 1: 7028 observations,    complexity param=0.09572759
##   predicted class=1  expected loss=0.885885  P(node) =1
##     class counts:   696   802   739   710   690   613   669   714   676   719
##    probabilities: 0.099 0.114 0.105 0.101 0.098 0.087 0.095 0.102 0.096 0.102 
##   left son=2 (2379 obs) right son=3 (4649 obs)
##   Primary splits:
```

```
##       pixel409 < 0.5 to the left,  improve=285.8426, (0 missing)
##       pixel461 < 0.5 to the left,  improve=267.9799, (0 missing)
##       pixel433 < 0.5 to the left,  improve=267.4597, (0 missing)
##       pixel568 < 0.5 to the right, improve=267.2841, (0 missing)
##       pixel437 < 0.5 to the left,  improve=267.1093, (0 missing)
##   Surrogate splits:
##       pixel381 < 0.5 to the left,  agree=0.875, adj=0.632, (0 split)
##       pixel437 < 0.5 to the left,  agree=0.874, adj=0.628, (0 split)
##       pixel410 < 0.5 to the left,  agree=0.841, adj=0.532, (0 split)
##       pixel408 < 0.5 to the left,  agree=0.835, adj=0.513, (0 split)
##       pixel382 < 0.5 to the left,  agree=0.834, adj=0.510, (0 split)
## 
## Node number 2: 2379 observations,    complexity param=0.09235464
##   predicted class=1  expected loss=0.7053384  P(node) =0.3385031
##     class counts:   590   701   212   155    35   268   123    77   196    22
##    probabilities: 0.248 0.295 0.089 0.065 0.015 0.113 0.052 0.032 0.082 0.009 
##   left son=4 (851 obs) right son=5 (1528 obs)
##   Primary splits:
##       pixel434 < 0.5 to the left,  improve=372.6681, (0 missing)
##       pixel462 < 0.5 to the left,  improve=363.4885, (0 missing)
##       pixel406 < 0.5 to the left,  improve=360.9173, (0 missing)
##       pixel433 < 0.5 to the left,  improve=355.2999, (0 missing)
##       pixel378 < 0.5 to the left,  improve=343.3097, (0 missing)
##   Surrogate splits:
##       pixel433 < 0.5 to the left,  agree=0.954, adj=0.871, (0 split)
##       pixel406 < 0.5 to the left,  agree=0.944, adj=0.843, (0 split)
##       pixel462 < 0.5 to the left,  agree=0.939, adj=0.830, (0 split)
##       pixel435 < 0.5 to the left,  agree=0.914, adj=0.760, (0 split)
##       pixel461 < 0.5 to the left,  agree=0.910, adj=0.747, (0 split)
## 
## Node number 3: 4649 observations,    complexity param=0.06553164
##   predicted class=9  expected loss=0.8500753  P(node) =0.6614969
##     class counts:   106   101   527   555   655   345   546   637   480   697
##    probabilities: 0.023 0.022 0.113 0.119 0.141 0.074 0.117 0.137 0.103 0.150 
##   left son=6 (2084 obs) right son=7 (2565 obs)
##   Primary splits:
##       pixel155 < 0.5 to the right, improve=274.2560, (0 missing)
##       pixel154 < 0.5 to the right, improve=272.2530, (0 missing)
##       pixel542 < 0.5 to the right, improve=253.1650, (0 missing)
##       pixel569 < 0.5 to the right, improve=253.0963, (0 missing)
##       pixel156 < 0.5 to the right, improve=241.2201, (0 missing)
##   Surrogate splits:
##       pixel156 < 0.5 to the right, agree=0.934, adj=0.852, (0 split)
##       pixel154 < 0.5 to the right, agree=0.925, adj=0.833, (0 split)
##       pixel157 < 0.5 to the right, agree=0.851, adj=0.668, (0 split)
##       pixel153 < 0.5 to the right, agree=0.851, adj=0.668, (0 split)
##       pixel127 < 0.5 to the right, agree=0.793, adj=0.538, (0 split)
## 
## Node number 4: 851 observations
##   predicted class=0  expected loss=0.3172738  P(node) =0.1210871
##     class counts:   581     6    33    50     3    68    44    49     9     8
##    probabilities: 0.683 0.007 0.039 0.059 0.004 0.080 0.052 0.058 0.011 0.009 
## 
## Node number 5: 1528 observations,    complexity param=0.02280758
##   predicted class=1  expected loss=0.5451571  P(node) =0.2174161
##     class counts:     9   695   179   105    32   200    79    28   187    14
##    probabilities: 0.006 0.455 0.117 0.069 0.021 0.131 0.052 0.018 0.122 0.009 
##   left son=10 (954 obs) right son=11 (574 obs)
##   Primary splits:
##       pixel375 < 0.5 to the left,  improve=189.3687, (0 missing)
##       pixel521 < 0.5 to the left,  improve=183.4745, (0 missing)
##       pixel347 < 0.5 to the left,  improve=175.2548, (0 missing)
##       pixel346 < 0.5 to the left,  improve=171.4535, (0 missing)
##       pixel403 < 0.5 to the left,  improve=170.6849, (0 missing)
##   Surrogate splits:
##       pixel347 < 0.5 to the left,  agree=0.913, adj=0.768, (0 split)
##       pixel374 < 0.5 to the left,  agree=0.882, adj=0.685, (0 split)
##       pixel403 < 0.5 to the left,  agree=0.874, adj=0.664, (0 split)
##       pixel346 < 0.5 to the left,  agree=0.855, adj=0.615, (0 split)
##       pixel402 < 0.5 to the left,  agree=0.850, adj=0.601, (0 split)
##
## Node number 6: 2084 observations,    complexity param=0.05011243
##   predicted class=3  expected loss=0.7912668  P(node) =0.2965282
##     class counts:    79    41   430   435    76   205   379    11   382    46
##    probabilities: 0.038 0.020 0.206 0.209 0.036 0.098 0.182 0.005 0.183 0.022
##   left son=12 (890 obs) right son=13 (1194 obs)
##   Primary splits:
##       pixel656 < 0.5 to the left,  improve=208.4248, (0 missing)
##       pixel657 < 0.5 to the left,  improve=207.1139, (0 missing)
##       pixel655 < 0.5 to the left,  improve=189.4093, (0 missing)
##       pixel658 < 0.5 to the left,  improve=187.4302, (0 missing)
##       pixel515 < 0.5 to the left,  improve=173.9747, (0 missing)
##   Surrogate splits:
##       pixel657 < 0.5 to the left,  agree=0.959, adj=0.904, (0 split)
##       pixel655 < 0.5 to the left,  agree=0.953, adj=0.891, (0 split)
##       pixel658 < 0.5 to the left,  agree=0.897, adj=0.758, (0 split)
##       pixel654 < 0.5 to the left,  agree=0.881, adj=0.721, (0 split)
##       pixel659 < 0.5 to the left,  agree=0.823, adj=0.585, (0 split)
##
## Node number 7: 2565 observations,    complexity param=0.06553164
##   predicted class=9  expected loss=0.7461988  P(node) =0.3649687
##     class counts:    27    60    97   120   579   140   167   626    98   651
##    probabilities: 0.011 0.023 0.038 0.047 0.226 0.055 0.065 0.244 0.038 0.254
##   left son=14 (802 obs) right son=15 (1763 obs)
##   Primary splits:
##       pixel239 < 0.5 to the left,  improve=253.7769, (0 missing)
##       pixel238 < 0.5 to the left,  improve=237.3069, (0 missing)
##       pixel431 < 0.5 to the right, improve=226.6129, (0 missing)
##       pixel430 < 0.5 to the right, improve=223.8479, (0 missing)
##       pixel211 < 0.5 to the left,  improve=215.5119, (0 missing)
##   Surrogate splits:
##       pixel238 < 0.5 to the left,  agree=0.919, adj=0.742, (0 split)
##       pixel240 < 0.5 to the left,  agree=0.901, adj=0.683, (0 split)
##       pixel211 < 0.5 to the left,  agree=0.881, adj=0.618, (0 split)
##       pixel267 < 0.5 to the left,  agree=0.847, adj=0.511, (0 split)
##       pixel237 < 0.5 to the left,  agree=0.838, adj=0.483, (0 split)
##
## Node number 10: 954 observations,    complexity param=0.01349181
##   predicted class=1  expected loss=0.3144654  P(node) =0.1357427
##     class counts:     2   654   138    32    11    17    22    28    46     4
##    probabilities: 0.002 0.686 0.145 0.034 0.012 0.018 0.023 0.029 0.048 0.004
##   left son=20 (792 obs) right son=21 (162 obs)
##   Primary splits:
##       pixel550 < 0.5 to the left,  improve=118.9158, (0 missing)
##       pixel551 < 0.5 to the left,  improve=110.3422, (0 missing)
##       pixel206 < 0.5 to the left,  improve=105.3515, (0 missing)
##       pixel579 < 0.5 to the left,  improve=104.9301, (0 missing)
##       pixel485 < 0.5 to the left,  improve=104.6566, (0 missing)
##   Surrogate splits:
##       pixel551 < 0.5 to the left,  agree=0.963, adj=0.784, (0 split)
##       pixel522 < 0.5 to the left,  agree=0.938, adj=0.636, (0 split)
##       pixel552 < 0.5 to the left,  agree=0.928, adj=0.574, (0 split)
##       pixel578 < 0.5 to the left,  agree=0.927, adj=0.568, (0 split)
##       pixel523 < 0.5 to the left,  agree=0.925, adj=0.556, (0 split)
##
## Node number 11: 574 observations,    complexity param=0.01831031
##   predicted class=5  expected loss=0.6811847  P(node) =0.08167331
##     class counts:     7    41    41    73    21   183    57     0   141    10
##    probabilities: 0.012 0.071 0.071 0.127 0.037 0.319 0.099 0.000 0.246 0.017
##   left son=22 (263 obs) right son=23 (311 obs)
##   Primary splits:
##       pixel352 < 0.5 to the left,  improve=72.09103, (0 missing)
##       pixel351 < 0.5 to the left,  improve=68.32429, (0 missing)
##       pixel380 < 0.5 to the left,  improve=65.21370, (0 missing)
##       pixel379 < 0.5 to the left,  improve=63.53266, (0 missing)
##       pixel353 < 0.5 to the left,  improve=58.63524, (0 missing)
##   Surrogate splits:
##       pixel351 < 0.5 to the left,  agree=0.902, adj=0.787, (0 split)
##       pixel324 < 0.5 to the left,  agree=0.897, adj=0.776, (0 split)
##       pixel380 < 0.5 to the left,  agree=0.889, adj=0.757, (0 split)
##       pixel353 < 0.5 to the left,  agree=0.885, adj=0.749, (0 split)
##       pixel325 < 0.5 to the left,  agree=0.885, adj=0.749, (0 split)
##
## Node number 12: 890 observations,    complexity param=0.04031481
##   predicted class=6  expected loss=0.5898876  P(node) =0.1266363
##     class counts:    10    10   328    53    58    30   365     5    15    16
##    probabilities: 0.011 0.011 0.369 0.060 0.065 0.034 0.410 0.006 0.017 0.018
##   left son=24 (329 obs) right son=25 (561 obs)
##   Primary splits:
##       pixel271 < 0.5 to the right, improve=170.8303, (0 missing)
##       pixel270 < 0.5 to the right, improve=167.9733, (0 missing)
##       pixel319 < 0.5 to the left,  improve=167.4015, (0 missing)
##       pixel298 < 0.5 to the right, improve=164.0751, (0 missing)
##       pixel347 < 0.5 to the left,  improve=153.7797, (0 missing)
##   Surrogate splits:
##       pixel243 < 0.5 to the right, agree=0.946, adj=0.854, (0 split)
##       pixel299 < 0.5 to the right, agree=0.943, adj=0.845, (0 split)
##       pixel272 < 0.5 to the right, agree=0.883, adj=0.684, (0 split)
##       pixel215 < 0.5 to the right, agree=0.882, adj=0.681, (0 split)
##       pixel244 < 0.5 to the right, agree=0.873, adj=0.657, (0 split)
##
## Node number 13: 1194 observations,    complexity param=0.0469001
##   predicted class=3  expected loss=0.680067  P(node) =0.1698919
##     class counts:    69    31   102   382    18   175    14     6   367    30
##    probabilities: 0.058 0.026 0.085 0.320 0.015 0.147 0.012 0.005 0.307 0.025
##   left son=26 (658 obs) right son=27 (536 obs)
##   Primary splits:
##       pixel488 < 0.5 to the left,  improve=169.3419, (0 missing)
##       pixel487 < 0.5 to the left,  improve=161.8602, (0 missing)
##       pixel515 < 0.5 to the left,  improve=151.1248, (0 missing)
##       pixel489 < 0.5 to the left,  improve=146.0450, (0 missing)
##       pixel516 < 0.5 to the left,  improve=131.4159, (0 missing)
##   Surrogate splits:
##       pixel489 < 0.5 to the left,  agree=0.924, adj=0.830, (0 split)
##       pixel487 < 0.5 to the left,  agree=0.911, adj=0.802, (0 split)
##       pixel516 < 0.5 to the left,  agree=0.886, adj=0.746, (0 split)
##       pixel515 < 0.5 to the left,  agree=0.877, adj=0.726, (0 split)
##       pixel461 < 0.5 to the left,  agree=0.876, adj=0.724, (0 split)
##
## Node number 14: 802 observations,    complexity param=0.01590106
##   predicted class=4  expected loss=0.4064838  P(node) =0.114115
##     class counts:     2    36    19     9   476    17   131    53    10    49
##    probabilities: 0.002 0.045 0.024 0.011 0.594 0.021 0.163 0.066 0.012 0.061
##   left son=28 (698 obs) right son=29 (104 obs)
##   Primary splits:
##       pixel96  < 0.5 to the left,  improve=118.08080, (0 missing)
##       pixel97  < 0.5 to the left,  improve=105.30730, (0 missing)
##       pixel124 < 0.5 to the left,  improve=100.53830, (0 missing)
##       pixel95  < 0.5 to the left,  improve= 95.95058, (0 missing)
##       pixel123 < 0.5 to the left,  improve= 92.59187, (0 missing)
##   Surrogate splits:
##       pixel124 < 0.5 to the left,  agree=0.968, adj=0.750, (0 split)
##       pixel97  < 0.5 to the left,  agree=0.968, adj=0.750, (0 split)
##       pixel95  < 0.5 to the left,  agree=0.958, adj=0.673, (0 split)
##       pixel123 < 0.5 to the left,  agree=0.955, adj=0.654, (0 split)
##       pixel68  < 0.5 to the left,  agree=0.953, adj=0.635, (0 split)
##
## Node number 15: 1763 observations,    complexity param=0.06360424
##   predicted class=9  expected loss=0.6585366  P(node) =0.2508537
##     class counts:    25    24    78   111   103   123    36   573    88   602
##    probabilities: 0.014 0.014 0.044 0.063 0.058 0.070 0.020 0.325 0.050 0.341
##   left son=30 (844 obs) right son=31 (919 obs)
##   Primary splits:
##       pixel431 < 0.5 to the left,  improve=205.8435, (0 missing)
##       pixel405 < 0.5 to the left,  improve=202.1331, (0 missing)
##       pixel432 < 0.5 to the left,  improve=202.0210, (0 missing)
##       pixel403 < 0.5 to the left,  improve=188.3853, (0 missing)
##       pixel404 < 0.5 to the right, improve=187.3827, (0 missing)
##   Surrogate splits:
##       pixel432 < 0.5 to the left,  agree=0.917, adj=0.826, (0 split)
##       pixel430 < 0.5 to the left,  agree=0.915, adj=0.823, (0 split)
##       pixel433 < 0.5 to the left,  agree=0.833, adj=0.650, (0 split)
##       pixel403 < 0.5 to the left,  agree=0.833, adj=0.650, (0 split)
##       pixel429 < 0.5 to the left,  agree=0.827, adj=0.639, (0 split)
##
## Node number 20: 792 observations
##   predicted class=1  expected loss=0.1818182  P(node) =0.1126921
##     class counts:     1   648    48    14     7    12     1    26    31     4
##    probabilities: 0.001 0.818 0.061 0.018 0.009 0.015 0.001 0.033 0.039 0.005
##
## Node number 21: 162 observations
##   predicted class=2  expected loss=0.4444444  P(node) =0.02305065
##     class counts:     1     6    90    18     4     5    21     2    15     0
##    probabilities: 0.006 0.037 0.556 0.111 0.025 0.031 0.130 0.012 0.093 0.000
##
## Node number 22: 263 observations
##   predicted class=5  expected loss=0.3612167  P(node) =0.03742174
##     class counts:     6    16    14    15     4   168    27     0    12     1
##    probabilities: 0.023 0.061 0.053 0.057 0.015 0.639 0.103 0.000 0.046 0.004
##
## Node number 23: 311 observations
##   predicted class=8  expected loss=0.585209  P(node) =0.04425157
##     class counts:     1    25    27    58    17    15    30     0   129     9
##    probabilities: 0.003 0.080 0.087 0.186 0.055 0.048 0.096 0.000 0.415 0.029
##
## Node number 24: 329 observations
##   predicted class=2  expected loss=0.231003  P(node) =0.04681275
##     class counts:     9     0   253    32    12     2     2     1    11     7
##    probabilities: 0.027 0.000 0.769 0.097 0.036 0.006 0.006 0.003 0.033 0.021
##
## Node number 25: 561 observations
##   predicted class=6  expected loss=0.3529412  P(node) =0.07982356
##     class counts:     1    10    75    21    46    28   363     4     4     9
##    probabilities: 0.002 0.018 0.134 0.037 0.082 0.050 0.647 0.007 0.007 0.016
##
## Node number 26: 658 observations,    complexity param=0.01060071
##   predicted class=3  expected loss=0.4665653  P(node) =0.0936255
##     class counts:    54     2    20   351     7   155     6     1    44    18
##    probabilities: 0.082 0.003 0.030 0.533 0.011 0.236 0.009 0.002 0.067 0.027
##   left son=52 (341 obs) right son=53 (317 obs)
##   Primary splits:
##       pixel290 < 0.5 to the left,  improve=87.02635, (0 missing)
##       pixel269 < 0.5 to the right, improve=74.60786, (0 missing)
##       pixel291 < 0.5 to the left,  improve=74.01110, (0 missing)
##       pixel296 < 0.5 to the right, improve=73.45300, (0 missing)
##       pixel263 < 0.5 to the left,  improve=69.20867, (0 missing)
##   Surrogate splits:
##       pixel291 < 0.5 to the left,  agree=0.881, adj=0.754, (0 split)
##       pixel289 < 0.5 to the left,  agree=0.845, adj=0.678, (0 split)
##       pixel262 < 0.5 to the left,  agree=0.843, adj=0.675, (0 split)
##       pixel263 < 0.5 to the left,  agree=0.840, adj=0.669, (0 split)
##       pixel318 < 0.5 to the left,  agree=0.834, adj=0.656, (0 split)
##
## Node number 27: 536 observations
##   predicted class=8  expected loss=0.3973881  P(node) =0.07626636
##     class counts:    15    29    82    31    11    20     8     5   323    12
##    probabilities: 0.028 0.054 0.153 0.058 0.021 0.037 0.015 0.009 0.603 0.022
## 
## Node number 28: 698 observations
##   predicted class=4  expected loss=0.3194842  P(node) =0.09931702
##     class counts:     2    35    17     9   475    17    31    53    10    49
##    probabilities: 0.003 0.050 0.024 0.013 0.681 0.024 0.044 0.076 0.014 0.070
## 
## Node number 29: 104 observations
##   predicted class=6  expected loss=0.03846154  P(node) =0.01479795
##     class counts:     0     1     2     0     1     0   100     0     0     0
##    probabilities: 0.000 0.010 0.019 0.000 0.010 0.000 0.962 0.000 0.000 0.000
## 
## Node number 30: 844 observations
##   predicted class=7  expected loss=0.3767773  P(node) =0.1200911
##     class counts:     8    11    37    39    13    55     5   526    20   130
##    probabilities: 0.009 0.013 0.044 0.046 0.015 0.065 0.006 0.623 0.024 0.154
## 
## Node number 31: 919 observations
##   predicted class=9  expected loss=0.4863983  P(node) =0.1307627
##     class counts:    17    13    41    72    90    68    31    47    68   472
##    probabilities: 0.018 0.014 0.045 0.078 0.098 0.074 0.034 0.051 0.074 0.514
## 
## Node number 52: 341 observations
##   predicted class=3  expected loss=0.1730205  P(node) =0.0485202
##     class counts:     9     1    15   282     1    20     1     1     9     2
##    probabilities: 0.026 0.003 0.044 0.827 0.003 0.059 0.003 0.003 0.026 0.006
## 
## Node number 53: 317 observations
##   predicted class=5  expected loss=0.5741325  P(node) =0.04510529
##     class counts:    45     1     5    69     6   135     5     0    35    16
##    probabilities: 0.142 0.003 0.016 0.218 0.019 0.426 0.016 0.000 0.110 0.050
```

Rpart Plot

Now we can plot the tree using the rpart.plot() function.

```
rpart.plot(digit_train_rpart)

## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```



Analysis of importance of variables

It is useful to analyze the importance of variables when constructing a decision tree. Based on the below analysis, the pixels in the 400s (center of the image) are the most useful for determining the differences between digits.

```
digit_train_rpart$variable.importance
```

```
##  pixel433  pixel461  pixel434  pixel406  pixel462  pixel409  pixel435  pixel155
## 458.39295 401.09904 372.66806 313.98707 309.16997 285.84264 283.33282 274.25601
##  pixel403  pixel239  pixel156  pixel154  pixel656  pixel431  pixel375  pixel657
## 259.59180 253.77692 233.59137 228.59054 208.42480 205.84353 189.36870 188.51906
##  pixel238  pixel655  pixel153  pixel157  pixel381  pixel437  pixel240  pixel271
## 188.27590 185.70884 183.18828 183.18828 180.58911 179.50774 173.40368 170.83029
##  pixel432  pixel430  pixel488  pixel658  pixel211  pixel410  pixel654  pixel127
## 169.99164 169.50385 169.34189 158.07499 156.94932 151.99283 150.34688 147.65607
##  pixel408  pixel243  pixel382  pixel347  pixel299  pixel489  pixel487  pixel429
## 146.58597 145.90672 145.74490 145.49059 144.34900 140.59168 135.85264 131.45695
##  pixel267  pixel374  pixel516  pixel515  pixel237  pixel659  pixel550   pixel96
## 129.73633 129.65488 126.37455 122.89925 122.45844 122.01048 118.91584 118.08080
##  pixel272  pixel346  pixel215  pixel402  pixel244  pixel551  pixel124   pixel97
## 116.82922 116.45845 116.30998 113.81917 112.15605  93.22414  88.56060  88.56060
##  pixel290   pixel95  pixel123  pixel522   pixel68  pixel352  pixel552  pixel578
##  87.02635  79.47746  77.20668  75.60698  74.93589  72.09103  68.26650  67.53245
##  pixel523  pixel291  pixel289  pixel262  pixel263  pixel318  pixel351  pixel324
##  66.06435  65.61293  59.02418  58.74965  58.20059  57.10246  56.74085  55.91852
##  pixel380  pixel325  pixel353
##  54.54797  53.99974  53.99974
```

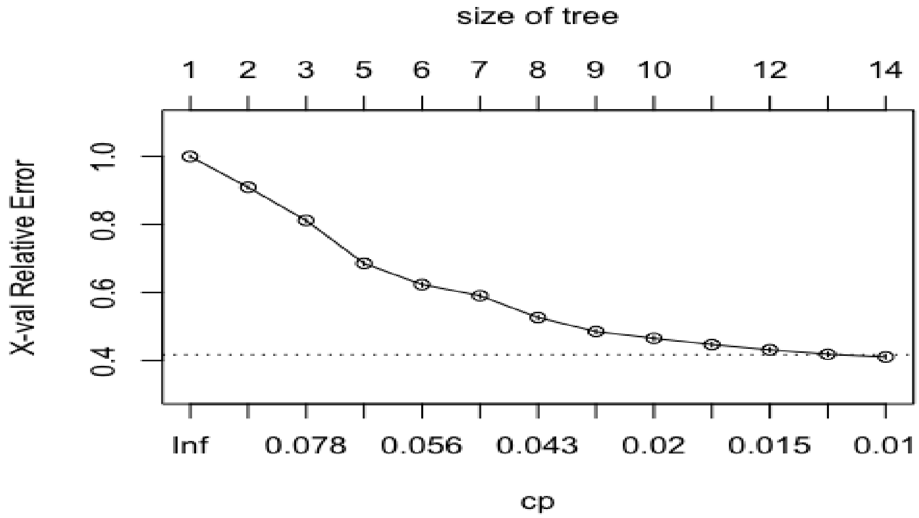
In order to prune the tree, we must first call the complexity parameter (cp) and tune this to its optimal value. The complexity parameter is the minimum improvement in the model that is required for each node, defined by the following equation:

$$\sum_{Terminal\ Nodes} Misclass_i + \lambda * (Splits)$$

The cp value is a "stopping parameter" that prunes splits that do not meet this criteria.[3]

[3] https://www.learnbymarketing.com/tutorials/rpart-decision-trees-in-r/

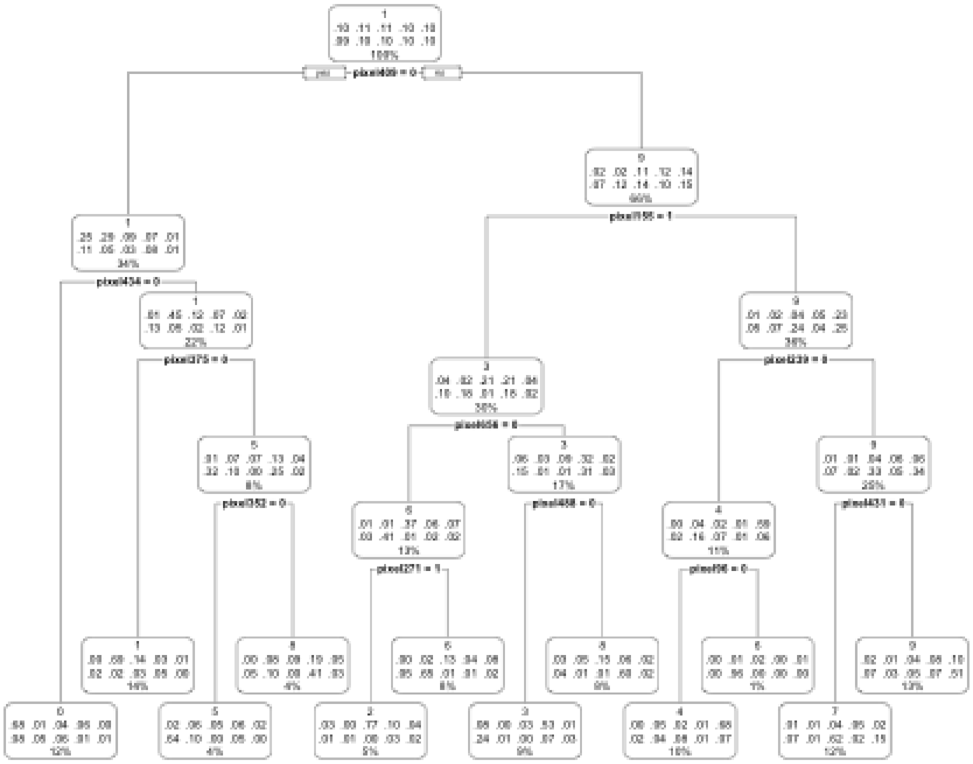
```
printcp(digit_train_rpart)

##
## Classification tree:
## rpart(formula = label ~ ., data = train1, method = "class")
##
## Variables actually used in tree construction:
##  [1] pixel155 pixel239 pixel271 pixel290 pixel352 pixel375 pixel409 pixel431
##  [9] pixel434 pixel488 pixel550 pixel656 pixel96
##
## Root node error: 6226/7028 = 0.88589
##
## n= 7028
##
##          CP nsplit rel error  xerror      xstd
## 1  0.095728      0   1.00000 1.00000 0.0042812
## 2  0.092355      1   0.90427 0.90925 0.0053297
## 3  0.065532      2   0.81192 0.81192 0.0060506
## 4  0.063604      4   0.68085 0.68551 0.0065757
## 5  0.050112      5   0.61725 0.62367 0.0066953
## 6  0.046900      6   0.56714 0.59075 0.0067252
## 7  0.040315      7   0.52024 0.52666 0.0067174
## 8  0.022808      8   0.47992 0.48538 0.0066662
## 9  0.018310      9   0.45712 0.46563 0.0066286
## 10 0.015901     10   0.43881 0.44732 0.0065860
## 11 0.013492     11   0.42290 0.43174 0.0065439
## 12 0.010601     12   0.40941 0.41873 0.0065044
## 13 0.010000     13   0.39881 0.41070 0.0064780

plotcp(digit_train_rpart)
```



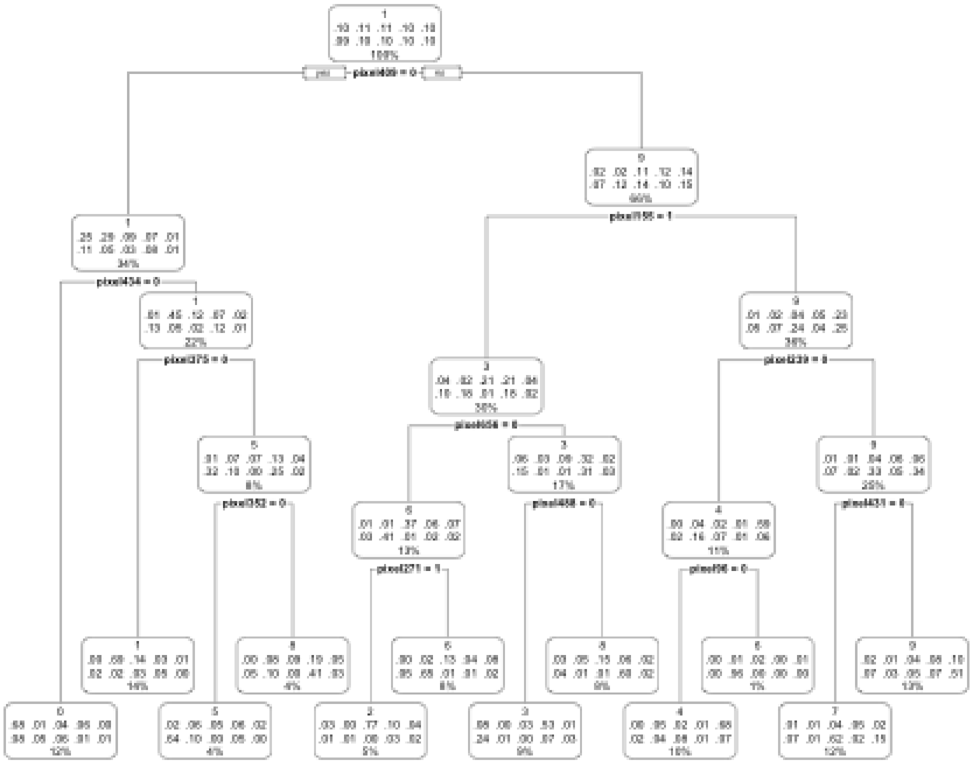
```
#set cp to 0.015
```

There are two ways that we can go about pruning the tree. Firstly, we can make a new tree with a cp of 0.015, which is shown through our graph above. Secondly, we can prune the original tree with the prune function and our new cp.

```
#building new tree
digit_train_rpart2<-rpart(label~., method="class", data=train1, control=rpart.control(minsplit=2,
cp=0.015))
rpart.plot(digit_train_rpart2)

## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```



```
#pruning original tree
pruned<-prune(digit_train_rpart, cp=0.015)
rpart.plot(pruned)

## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```



Now that our tree is finalized, we will use this tree to predict the values in our test set (derived from our training data so we can cross validate the results).

```
test_dt<-predict(pruned, test1, type='class')
#confusion matrix
test1_table<-table(label=test_dt, true=test1$label)
#length(test_dt)
#length(test1$label) #ensuring that these are the same length
confusionMatrix(test1_table)

## Confusion Matrix and Statistics
##
##      true
## label   0   1   2   3   4   5   6   7   8   9
##     0 232   1  16  27   5  35  31  21   4   5
##     1   2 297  41  10   5  14   8  13  11   6
##     2   7   0 106  12  12   0   4   2   6   7
##          3  23   0   7 149   1  78   3   0  22   7
##          4   0  13   3   1 181   2  17  24   4  15
##          5   4   5   6  13   2  64  13   0   7   0
##          6   0   3  36  14  16   8 186   1   3   4
##          7   4  10  10  14   8  28   3 237   5  52
##          8  12  22  44  29  11   7  16   5 169   9
##          9  10   6  19  24  38  31  29  16  38 191
##
## Overall Statistics
##
##                Accuracy : 0.6097
##                  95% CI : (0.5919, 0.6273)
##     No Information Rate : 0.1201
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5655
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.78912  0.83193  0.36806  0.50853  0.64875  0.23970
## Specificity           0.94586  0.95793  0.98137  0.94737  0.97066  0.98152
## Pos Pred Value        0.61538  0.72973  0.67949  0.51379  0.69615  0.56140
## Neg Pred Value        0.97611  0.97661  0.93537  0.94631  0.96386  0.92897
## Prevalence            0.09892  0.12012  0.09690  0.09859  0.09388  0.08984
## Detection Rate        0.07806  0.09993  0.03567  0.05013  0.06090  0.02153
## Detection Prevalence  0.12685  0.13694  0.05249  0.09758  0.08748  0.03836
## Balanced Accuracy     0.86749  0.89493  0.67471  0.72795  0.80971  0.61061
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.60000  0.74295  0.62825  0.64527
## Specificity           0.96807  0.94949  0.94266  0.92115
## Pos Pred Value       0.68635  0.63881  0.52160  0.47512
## Neg Pred Value       0.95409  0.96847  0.96224  0.95914
## Prevalence           0.10431  0.10734  0.09051  0.09960
## Detection Rate       0.06258  0.07974  0.05686  0.06427
## Detection Prevalence 0.09118  0.12483  0.10902  0.13526
## Balanced Accuracy    0.78403  0.84622  0.78545  0.78321
```

The accuracy rate is 60.97%, which is not as accurate as we would like it to be. From our results, we can see that the model positively predicted digit 1 with the most accuracy at 72.97%. It struggled the most with positively predicting digit 9, with an accuracy rate of 47.51%. It scored relatively high with negative predictions, which ranged from 92.11% to 97.66%. With balanced accuracy, it performed the best for digit 1 with an accuracy rate of 89.49%.

Ultimately, these results are not as good as one may hope for. One possibility is that our preprocessing step of reducing dimensionality actually impaired the algorithm's ability to classify correctly. Therefore, I will take resample from the original data without reducing dimensionality to attempt to improve our results.

Resampling data:

```
set.seed(22)
new_sample<-sample(1:nrow(sample_digit_train), size = 0.8 * nrow(sample_digit_train))
train_all<-sample_digit_train[new_sample,]
test_all<-sample_digit_train[-new_sample,]
```

Retry decision tree:

```
digit_train_rpart2 = rpart(label ~ ., data = train_all,
                     method = "class")
summary(digit_train_rpart2)

## Call:
## rpart(formula = label ~ ., data = train_all, method = "class")
##   n= 8000
##
##           CP nsplit rel error    xerror        xstd
## 1  0.09349766      0 1.0000000 1.0000000 0.004082049
## 2  0.09137272      1 0.9065023 0.9169854 0.004979482
## 3  0.06608585      2 0.8151296 0.8151296 0.005693792
## 4  0.06530670      4 0.6829579 0.7145488 0.006115776
## 5  0.04844879      5 0.6176512 0.6186429 0.006308645
## 6  0.04476555      6 0.5692024 0.5713274 0.006335152
## 7  0.04136563      7 0.5244369 0.5361949 0.006326209
## 8  0.02096614      8 0.4830713 0.4866128 0.006271849
## 9  0.01954951      9 0.4621051 0.4710299 0.006244511
## 10 0.01544128     10 0.4425556 0.4460972 0.006190330
## 11 0.01289134     11 0.4271143 0.4310809 0.006151376
## 12 0.01147471     12 0.4142230 0.4197478 0.006118763
## 13 0.01000000     13 0.4027483 0.4136563 0.006100074
## 
## Variable importance
## pixel433 pixel461 pixel434 pixel406 pixel462 pixel409 pixel435 pixel155 
##        4        4        3        3        3        3        3        2 
## pixel403 pixel239 pixel156 pixel154 pixel431 pixel657 pixel375 pixel656 
##        2        2        2        2        2        2        2        2 
## pixel238 pixel153 pixel437 pixel157 pixel381 pixel658 pixel432 pixel430 
##        2        2        2        2        2        2        2        2 
## pixel240 pixel488 pixel271 pixel655 pixel410 pixel211 pixel347 pixel382 
##        2        1        1        1        1        1        1        1 
## pixel127 pixel243 pixel408 pixel299 pixel659 pixel489 pixel404 pixel487 
##        1        1        1        1        1        1        1        1 
## pixel654 pixel374 pixel267 pixel237 pixel516 pixel346 pixel215 pixel550 
##        1        1        1        1        1        1        1        1 
## pixel515  pixel96 pixel402 pixel272 pixel270 pixel551 pixel290 pixel124 
##        1        1        1        1        1        1        1        1 
##  pixel97  pixel95 pixel522 pixel123  pixel68 pixel352 pixel578 pixel291 
##        1        1        1        1        1        1        1        1 
## pixel552 pixel523 pixel289 pixel263 pixel318 
##        1        1        1        1        1 
## 
## Node number 1: 8000 observations,    complexity param=0.09349766
##   predicted class=1  expected loss=0.882375  P(node) =1
##     class counts:   788   941   824   801   781   707   770   818   767   803
##    probabilities: 0.099 0.118 0.103 0.100 0.098 0.088 0.096 0.102 0.096 0.100
##   left son=2 (2747 obs) right son=3 (5253 obs)
##   Primary splits:
##       pixel409 < 0.5 to the left,  improve=325.3122, (0 missing)
##       pixel461 < 0.5 to the left,  improve=311.5538, (0 missing)
##       pixel433 < 0.5 to the left,  improve=305.7788, (0 missing)
##       pixel437 < 0.5 to the left,  improve=305.0178, (0 missing)
##       pixel290 < 0.5 to the left,  improve=301.3592, (0 missing)
##   Surrogate splits:
##       pixel437 < 0.5 to the left,  agree=0.874, adj=0.634, (0 split)
##       pixel381 < 0.5 to the left,  agree=0.872, adj=0.626, (0 split)
##       pixel410 < 0.5 to the left,  agree=0.841, adj=0.536, (0 split)
##       pixel382 < 0.5 to the left,  agree=0.833, adj=0.513, (0 split)
##       pixel408 < 0.5 to the left,  agree=0.830, adj=0.505, (0 split)
##
## Node number 2: 2747 observations,    complexity param=0.09137272
##   predicted class=1  expected loss=0.6985803  P(node) =0.343375
##     class counts:   663   828   231   176    45   308   148    91   227    30
##    probabilities: 0.241 0.301 0.084 0.064 0.016 0.112 0.054 0.033 0.083 0.011
##   left son=4 (971 obs) right son=5 (1776 obs)
##   Primary splits:
##       pixel434 < 0.5 to the left,  improve=420.0695, (0 missing)
##       pixel462 < 0.5 to the left,  improve=408.8474, (0 missing)
##       pixel406 < 0.5 to the left,  improve=401.2964, (0 missing)
##       pixel433 < 0.5 to the left,  improve=397.4082, (0 missing)
##       pixel461 < 0.5 to the left,  improve=384.9772, (0 missing)
##   Surrogate splits:
##       pixel433 < 0.5 to the left,  agree=0.950, adj=0.858, (0 split)
##       pixel406 < 0.5 to the left,  agree=0.941, adj=0.833, (0 split)
##       pixel462 < 0.5 to the left,  agree=0.936, adj=0.819, (0 split)
##       pixel435 < 0.5 to the left,  agree=0.916, adj=0.763, (0 split)
##       pixel461 < 0.5 to the left,  agree=0.913, adj=0.755, (0 split)
##
## Node number 3: 5253 observations,    complexity param=0.06608585
##   predicted class=9  expected loss=0.852846  P(node) =0.656625
##     class counts:   125   113   593   625   736   399   622   727   540   773
##    probabilities: 0.024 0.022 0.113 0.119 0.140 0.076 0.118 0.138 0.103 0.147
##   left son=6 (2349 obs) right son=7 (2904 obs)
##   Primary splits:
##       pixel155 < 0.5 to the right, improve=309.5633, (0 missing)
##       pixel154 < 0.5 to the right, improve=308.7080, (0 missing)
##       pixel542 < 0.5 to the right, improve=292.4972, (0 missing)
##       pixel569 < 0.5 to the right, improve=285.0350, (0 missing)
##       pixel156 < 0.5 to the right, improve=277.2995, (0 missing)
##   Surrogate splits:
##       pixel156 < 0.5 to the right, agree=0.934, adj=0.853, (0 split)
##       pixel154 < 0.5 to the right, agree=0.924, adj=0.829, (0 split)
##       pixel153 < 0.5 to the right, agree=0.852, adj=0.668, (0 split)
##       pixel157 < 0.5 to the right, agree=0.850, adj=0.665, (0 split)
##       pixel127 < 0.5 to the right, agree=0.791, adj=0.532, (0 split)
##
## Node number 4: 971 observations
##   predicted class=0  expected loss=0.3316169  P(node) =0.121375
##     class counts:   649     4    35    60     8    82    57    55    10    11
##    probabilities: 0.668 0.004 0.036 0.062 0.008 0.084 0.059 0.057 0.010 0.011
##
## Node number 5: 1776 observations,    complexity param=0.02096614
##   predicted class=1  expected loss=0.536036  P(node) =0.222
##     class counts:    14   824   196   116    37   226    91    36   217    19
##    probabilities: 0.008 0.464 0.110 0.065 0.021 0.127 0.051 0.020 0.122 0.011
##   left son=10 (1107 obs) right son=11 (669 obs)
##   Primary splits:
##       pixel375 < 0.5 to the left,  improve=218.1079, (0 missing)
##       pixel347 < 0.5 to the left,  improve=204.4210, (0 missing)
##       pixel521 < 0.5 to the left,  improve=200.4910, (0 missing)
##       pixel374 < 0.5 to the left,  improve=199.2714, (0 missing)
##       pixel346 < 0.5 to the left,  improve=199.1335, (0 missing)
##   Surrogate splits:
##       pixel347 < 0.5 to the left,  agree=0.914, adj=0.771, (0 split)
##       pixel374 < 0.5 to the left,  agree=0.875, adj=0.668, (0 split)
##       pixel403 < 0.5 to the left,  agree=0.873, adj=0.662, (0 split)
##       pixel346 < 0.5 to the left,  agree=0.854, adj=0.613, (0 split)
##       pixel402 < 0.5 to the left,  agree=0.844, adj=0.586, (0 split)
##
## Node number 6: 2349 observations,    complexity param=0.04844879
##   predicted class=3  expected loss=0.7879949  P(node) =0.293625
##     class counts:    94    42   482   498    84   238   428    11   423    49
##    probabilities: 0.040 0.018 0.205 0.212 0.036 0.101 0.182 0.005 0.180 0.021
##   left son=12 (1027 obs) right son=13 (1322 obs)
##   Primary splits:
##       pixel657 < 0.5 to the left,  improve=232.8541, (0 missing)
##       pixel656 < 0.5 to the left,  improve=230.4477, (0 missing)
##       pixel658 < 0.5 to the left,  improve=206.6091, (0 missing)
##       pixel655 < 0.5 to the left,  improve=204.3235, (0 missing)
##       pixel489 < 0.5 to the left,  improve=197.6531, (0 missing)
##   Surrogate splits:
##       pixel656 < 0.5 to the left,  agree=0.960, adj=0.907, (0 split)
##       pixel658 < 0.5 to the left,  agree=0.940, adj=0.863, (0 split)
##       pixel655 < 0.5 to the left,  agree=0.908, adj=0.789, (0 split)
##       pixel659 < 0.5 to the left,  agree=0.854, adj=0.666, (0 split)
##       pixel654 < 0.5 to the left,  agree=0.839, adj=0.631, (0 split)
##
## Node number 7: 2904 observations,    complexity param=0.06608585
##   predicted class=9  expected loss=0.7506887  P(node) =0.363
##     class counts:    31    71   111   127   652   161   194   716   117   724
##    probabilities: 0.011 0.024 0.038 0.044 0.225 0.055 0.067 0.247 0.040 0.249
##   left son=14 (900 obs) right son=15 (2004 obs)
##   Primary splits:
##       pixel239 < 0.5 to the left,  improve=288.8554, (0 missing)
##       pixel238 < 0.5 to the left,  improve=262.5413, (0 missing)
##       pixel431 < 0.5 to the right, improve=260.5896, (0 missing)
##       pixel430 < 0.5 to the right, improve=251.5240, (0 missing)
##       pixel432 < 0.5 to the right, improve=248.5253, (0 missing)
##   Surrogate splits:
##       pixel238 < 0.5 to the left,  agree=0.917, adj=0.731, (0 split)
##       pixel240 < 0.5 to the left,  agree=0.896, adj=0.666, (0 split)
##       pixel211 < 0.5 to the left,  agree=0.873, adj=0.591, (0 split)
##       pixel267 < 0.5 to the left,  agree=0.843, adj=0.494, (0 split)
##       pixel237 < 0.5 to the left,  agree=0.843, adj=0.493, (0 split)
##
## Node number 10: 1107 observations,    complexity param=0.01289134
##   predicted class=1  expected loss=0.3026197  P(node) =0.138375
##     class counts:     4   772   148    35    12    26    20    34    50     6
##    probabilities: 0.004 0.697 0.134 0.032 0.011 0.023 0.018 0.031 0.045 0.005
##   left son=20 (933 obs) right son=21 (174 obs)
##   Primary splits:
##       pixel550 < 0.5 to the left,  improve=130.5992, (0 missing)
##       pixel206 < 0.5 to the left,  improve=129.6435, (0 missing)
##       pixel551 < 0.5 to the left,  improve=126.5073, (0 missing)
##       pixel178 < 0.5 to the left,  improve=125.7001, (0 missing)
##       pixel179 < 0.5 to the left,  improve=120.7799, (0 missing)
##   Surrogate splits:
##       pixel551 < 0.5 to the left,  agree=0.965, adj=0.776, (0 split)
##       pixel522 < 0.5 to the left,  agree=0.941, adj=0.626, (0 split)
##       pixel578 < 0.5 to the left,  agree=0.936, adj=0.592, (0 split)
##       pixel552 < 0.5 to the left,  agree=0.932, adj=0.569, (0 split)
##       pixel523 < 0.5 to the left,  agree=0.930, adj=0.552, (0 split)
##
## Node number 11: 669 observations,    complexity param=0.01954951
##   predicted class=5  expected loss=0.7010463  P(node) =0.083625
##     class counts:    10    52    48    81    25   200    71     2   167    13
##    probabilities: 0.015 0.078 0.072 0.121 0.037 0.299 0.106 0.003 0.250 0.019
##   left son=22 (308 obs) right son=23 (361 obs)
##   Primary splits:
##       pixel352 < 0.5 to the left,  improve=78.25714, (0 missing)
##       pixel351 < 0.5 to the left,  improve=68.95125, (0 missing)
##       pixel380 < 0.5 to the left,  improve=64.63170, (0 missing)
##       pixel379 < 0.5 to the left,  improve=64.28421, (0 missing)
##       pixel353 < 0.5 to the left,  improve=63.73417, (0 missing)
##   Surrogate splits:
##       pixel380 < 0.5 to the left,  agree=0.889, adj=0.760, (0 split)
##       pixel324 < 0.5 to the left,  agree=0.883, adj=0.747, (0 split)
##       pixel351 < 0.5 to the left,  agree=0.883, adj=0.747, (0 split)
##       pixel325 < 0.5 to the left,  agree=0.874, adj=0.727, (0 split)
##       pixel353 < 0.5 to the left,  agree=0.867, adj=0.711, (0 split)
##
## Node number 12: 1027 observations,    complexity param=0.04136563
##   predicted class=6  expected loss=0.600779  P(node) =0.128375
##     class counts:    21    11   381    68    56    28   410     6    21    25
##    probabilities: 0.020 0.011 0.371 0.066 0.055 0.027 0.399 0.006 0.020 0.024
##   left son=24 (415 obs) right son=25 (612 obs)
##   Primary splits:
##       pixel271 < 0.5 to the right, improve=189.0644, (0 missing)
##       pixel319 < 0.5 to the left,  improve=184.3829, (0 missing)
##       pixel298 < 0.5 to the right, improve=181.9553, (0 missing)
##       pixel270 < 0.5 to the right, improve=181.2120, (0 missing)
##       pixel243 < 0.5 to the right, improve=174.6929, (0 missing)
##   Surrogate splits:
##       pixel243 < 0.5 to the right, agree=0.947, adj=0.870, (0 split)
##       pixel299 < 0.5 to the right, agree=0.936, adj=0.841, (0 split)
##       pixel215 < 0.5 to the right, agree=0.875, adj=0.692, (0 split)
##       pixel272 < 0.5 to the right, agree=0.857, adj=0.646, (0 split)
##       pixel270 < 0.5 to the right, agree=0.853, adj=0.636, (0 split)
```

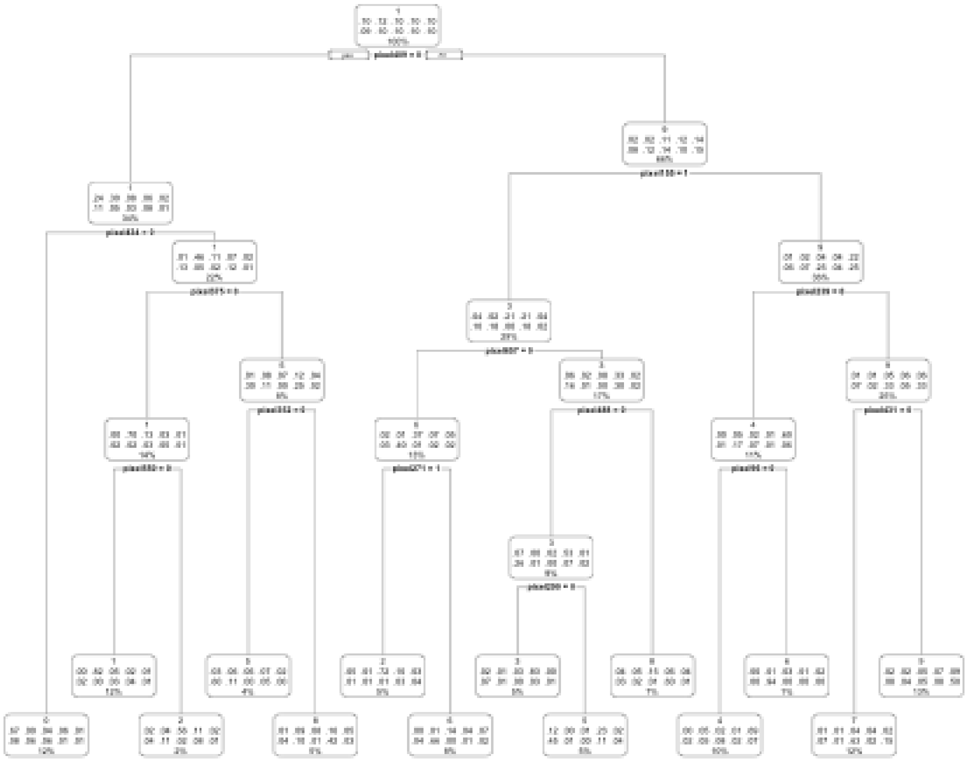
```
## 
## Node number 13: 1322 observations,    complexity param=0.04476555
##   predicted class=3  expected loss=0.6747352  P(node) =0.16525
##     class counts:    73    31   101   430    28   210    18     5   402    24
##    probabilities: 0.055 0.023 0.076 0.325 0.021 0.159 0.014 0.004 0.304 0.018 
##   left son=26 (737 obs) right son=27 (585 obs)
##   Primary splits:
##       pixel488 < 0.5 to the left,  improve=189.6047, (0 missing)
##       pixel487 < 0.5 to the left,  improve=183.1813, (0 missing)
##       pixel515 < 0.5 to the left,  improve=172.9308, (0 missing)
##       pixel489 < 0.5 to the left,  improve=170.6557, (0 missing)
##       pixel514 < 0.5 to the left,  improve=146.7924, (0 missing)
##   Surrogate splits:
##       pixel489 < 0.5 to the left,  agree=0.916, adj=0.810, (0 split)
##       pixel487 < 0.5 to the left,  agree=0.902, adj=0.778, (0 split)
##       pixel516 < 0.5 to the left,  agree=0.874, adj=0.716, (0 split)
##       pixel461 < 0.5 to the left,  agree=0.872, adj=0.711, (0 split)
##       pixel515 < 0.5 to the left,  agree=0.859, adj=0.682, (0 split)
## 
## Node number 14: 900 observations,    complexity param=0.01544128
##   predicted class=4  expected loss=0.4022222  P(node) =0.1125
##     class counts:     1    42    20     9   538    13   150    61    12    54
##    probabilities: 0.001 0.047 0.022 0.010 0.598 0.014 0.167 0.068 0.013 0.060 
##   left son=28 (782 obs) right son=29 (118 obs)
##   Primary splits:
##       pixel96  < 0.5 to the left,  improve=128.5411, (0 missing)
##       pixel97  < 0.5 to the left,  improve=116.5920, (0 missing)
##       pixel124 < 0.5 to the left,  improve=111.6414, (0 missing)
##       pixel95  < 0.5 to the left,  improve=105.5772, (0 missing)
##       pixel123 < 0.5 to the left,  improve=102.6003, (0 missing)
##   Surrogate splits:
##       pixel124 < 0.5 to the left,  agree=0.963, adj=0.720, (0 split)
##       pixel97  < 0.5 to the left,  agree=0.961, adj=0.703, (0 split)
##       pixel95  < 0.5 to the left,  agree=0.958, adj=0.678, (0 split)
##       pixel68  < 0.5 to the left,  agree=0.951, adj=0.627, (0 split)
##       pixel123 < 0.5 to the left,  agree=0.951, adj=0.627, (0 split)
##
## Node number 15: 2004 observations,    complexity param=0.0653067
##   predicted class=9  expected loss=0.6656687  P(node) =0.2505
##     class counts:    30    29    91   118   114   148    44   655   105   670
##    probabilities: 0.015 0.014 0.045 0.059 0.057 0.074 0.022 0.327 0.052 0.334
##   left son=30 (959 obs) right son=31 (1045 obs)
##   Primary splits:
##       pixel431 < 0.5 to the left,  improve=239.2058, (0 missing)
##       pixel432 < 0.5 to the left,  improve=235.9353, (0 missing)
##       pixel405 < 0.5 to the left,  improve=232.9446, (0 missing)
##       pixel404 < 0.5 to the left,  improve=213.8825, (0 missing)
##       pixel403 < 0.5 to the left,  improve=210.5316, (0 missing)
##   Surrogate splits:
##       pixel432 < 0.5 to the left,  agree=0.914, adj=0.820, (0 split)
##       pixel430 < 0.5 to the left,  agree=0.910, adj=0.811, (0 split)
##       pixel403 < 0.5 to the left,  agree=0.834, adj=0.653, (0 split)
##       pixel433 < 0.5 to the left,  agree=0.833, adj=0.651, (0 split)
##       pixel404 < 0.5 to the left,  agree=0.823, adj=0.630, (0 split)
##
## Node number 20: 933 observations
##   predicted class=1  expected loss=0.1800643  P(node) =0.116625
##     class counts:     1   765    50    16     9    19     1    31    36     5
##    probabilities: 0.001 0.820 0.054 0.017 0.010 0.020 0.001 0.033 0.039 0.005
##
## Node number 21: 174 observations
##   predicted class=2  expected loss=0.4367816  P(node) =0.02175
##     class counts:     3     7    98    19     3     7    19     3    14     1
##    probabilities: 0.017 0.040 0.563 0.109 0.017 0.040 0.109 0.017 0.080 0.006
##
## Node number 22: 308 observations
##   predicted class=5  expected loss=0.3961039  P(node) =0.0385
##     class counts:     8    18    18    22     6   186    34     0    15     1
##    probabilities: 0.026 0.058 0.058 0.071 0.019 0.604 0.110 0.000 0.049 0.003 
## 
## Node number 23: 361 observations
##   predicted class=8  expected loss=0.5789474  P(node) =0.045125
##     class counts:     2    34    30    59    19    14    37     2   152    12
##    probabilities: 0.006 0.094 0.083 0.163 0.053 0.039 0.102 0.006 0.421 0.033 
## 
## Node number 24: 415 observations
##   predicted class=2  expected loss=0.2843373  P(node) =0.051875
##     class counts:    19     3   297    41    14     4     5     3    14    15
##    probabilities: 0.046 0.007 0.716 0.099 0.034 0.010 0.012 0.007 0.034 0.036 
## 
## Node number 25: 612 observations
##   predicted class=6  expected loss=0.3382353  P(node) =0.0765
##     class counts:     2     8    84    27    42    24   405     3     7    10
##    probabilities: 0.003 0.013 0.137 0.044 0.069 0.039 0.662 0.005 0.011 0.016 
## 
## Node number 26: 737 observations,    complexity param=0.01147471
##   predicted class=3  expected loss=0.4654003  P(node) =0.092125
##     class counts:    52     2    14   394     7   191     8     1    50    18
##    probabilities: 0.071 0.003 0.019 0.535 0.009 0.259 0.011 0.001 0.068 0.024 
##   left son=52 (373 obs) right son=53 (364 obs)
##   Primary splits:
##       pixel290 < 0.5 to the left,  improve=97.20250, (0 missing)
##       pixel296 < 0.5 to the right, improve=97.18806, (0 missing)
##       pixel269 < 0.5 to the right, improve=95.08527, (0 missing)
##       pixel297 < 0.5 to the right, improve=90.30301, (0 missing)
##       pixel268 < 0.5 to the right, improve=83.24156, (0 missing)
##   Surrogate splits:
##       pixel291 < 0.5 to the left,  agree=0.887, adj=0.772, (0 split)
##       pixel289 < 0.5 to the left,  agree=0.844, adj=0.684, (0 split)
##       pixel263 < 0.5 to the left,  agree=0.843, adj=0.681, (0 split)
##       pixel318 < 0.5 to the left,  agree=0.839, adj=0.673, (0 split)
##       pixel317 < 0.5 to the left,  agree=0.826, adj=0.648, (0 split)
## 
## Node number 27: 585 observations
##   predicted class=8  expected loss=0.3982906  P(node) =0.073125
##     class counts:    21    29    87    36    21    19    10     4   352     6
##    probabilities: 0.036 0.050 0.149 0.062 0.036 0.032 0.017 0.007 0.602 0.010 
## 
## Node number 28: 782 observations
##   predicted class=4  expected loss=0.314578  P(node) =0.09775
##     class counts:     1    41    17     8   536    13    39    61    12    54
##    probabilities: 0.001 0.052 0.022 0.010 0.685 0.017 0.050 0.078 0.015 0.069 
## 
## Node number 29: 118 observations
##   predicted class=6  expected loss=0.05932203  P(node) =0.01475
##     class counts:     0     1     3     1     2     0   111     0     0     0
##    probabilities: 0.000 0.008 0.025 0.008 0.017 0.000 0.941 0.000 0.000 0.000 
## 
## Node number 30: 959 observations
##   predicted class=7  expected loss=0.3680918  P(node) =0.119875
##     class counts:     7    13    42    43    15    66     5   606    17   145
##    probabilities: 0.007 0.014 0.044 0.045 0.016 0.069 0.005 0.632 0.018 0.151 
## 
## Node number 31: 1045 observations
##   predicted class=9  expected loss=0.4976077  P(node) =0.130625
##     class counts:    23    16    49    75    99    82    39    49    88   525
##    probabilities: 0.022 0.015 0.047 0.072 0.095 0.078 0.037 0.047 0.084 0.502 
## 
## Node number 52: 373 observations
##   predicted class=3  expected loss=0.1689008  P(node) =0.046625
##     class counts:     7     2    11   310     1    26     3     1    10     2
##    probabilities: 0.019 0.005 0.029 0.831 0.003 0.070 0.008 0.003 0.027 0.005 
```

```
##
## Node number 53: 364 observations
##   predicted class=5  expected loss=0.5467033  P(node) =0.0455
##     class counts:    45     0     3    84     6   165     5     0    40    16
##    probabilities: 0.124 0.000 0.008 0.231 0.016 0.453 0.014 0.000 0.110 0.044
```

Plotting new tree with all pixels:

```
rpart.plot(digit_train_rpart2)
```

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```

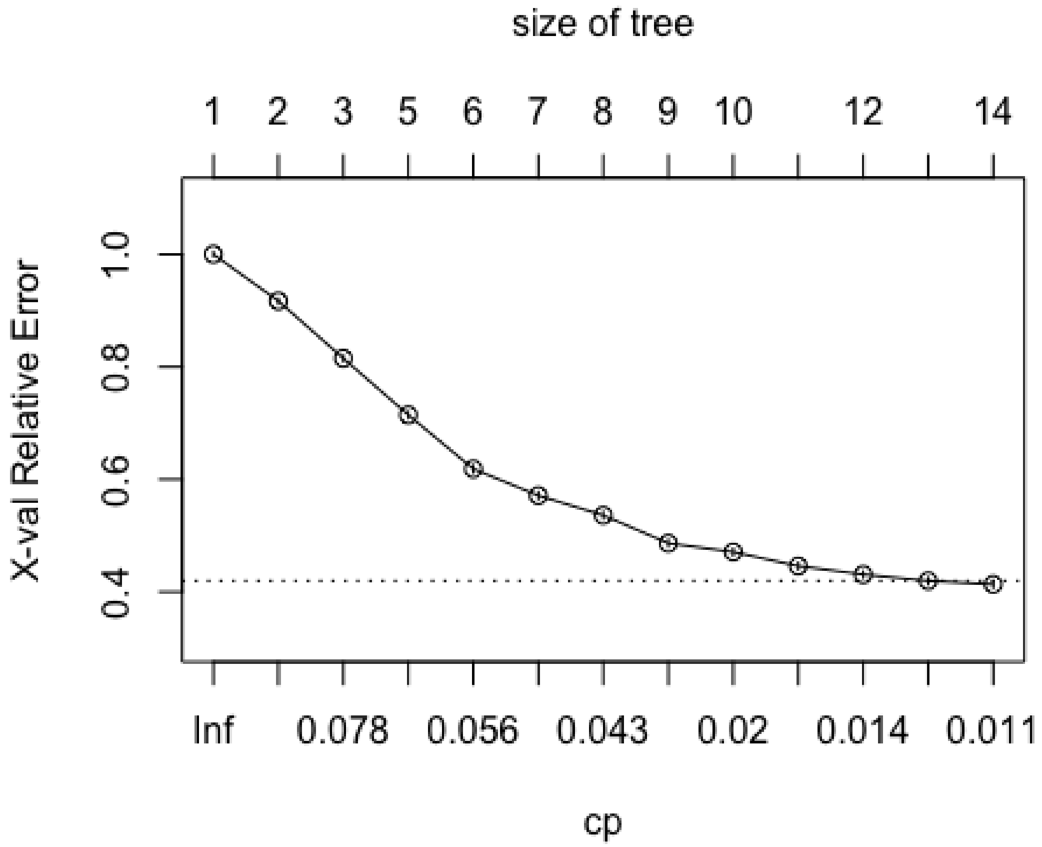


Finding New CP:

```
printcp(digit_train_rpart2)
```

```
##
## Classification tree:
## rpart(formula = label ~ ., data = train_all, method = "class")
##
## Variables actually used in tree construction:
##  [1] pixel155 pixel239 pixel271 pixel290 pixel352 pixel375 pixel409 pixel431
##  [9] pixel434 pixel488 pixel550 pixel657 pixel96
##
## Root node error: 7059/8000 = 0.88238
##
## n= 8000
##
##          CP nsplit rel error  xerror      xstd
## 1  0.093498      0   1.00000 1.00000 0.0040820
## 2  0.091373      1   0.90650 0.91699 0.0049795
## 3  0.066086      2   0.81513 0.81513 0.0056938
## 4  0.065307      4   0.68296 0.71455 0.0061158
## 5  0.048449      5   0.61765 0.61864 0.0063086
## 6  0.044766      6   0.56920 0.57133 0.0063352
## 7  0.041366      7   0.52444 0.53619 0.0063262
## 8  0.020966      8   0.48307 0.48661 0.0062718
## 9  0.019550      9   0.46211 0.47103 0.0062445
## 10 0.015441     10   0.44256 0.44610 0.0061903
## 11 0.012891     11   0.42711 0.43108 0.0061514
## 12 0.011475     12   0.41422 0.41975 0.0061188
## 13 0.010000     13   0.40275 0.41366 0.0061001
```

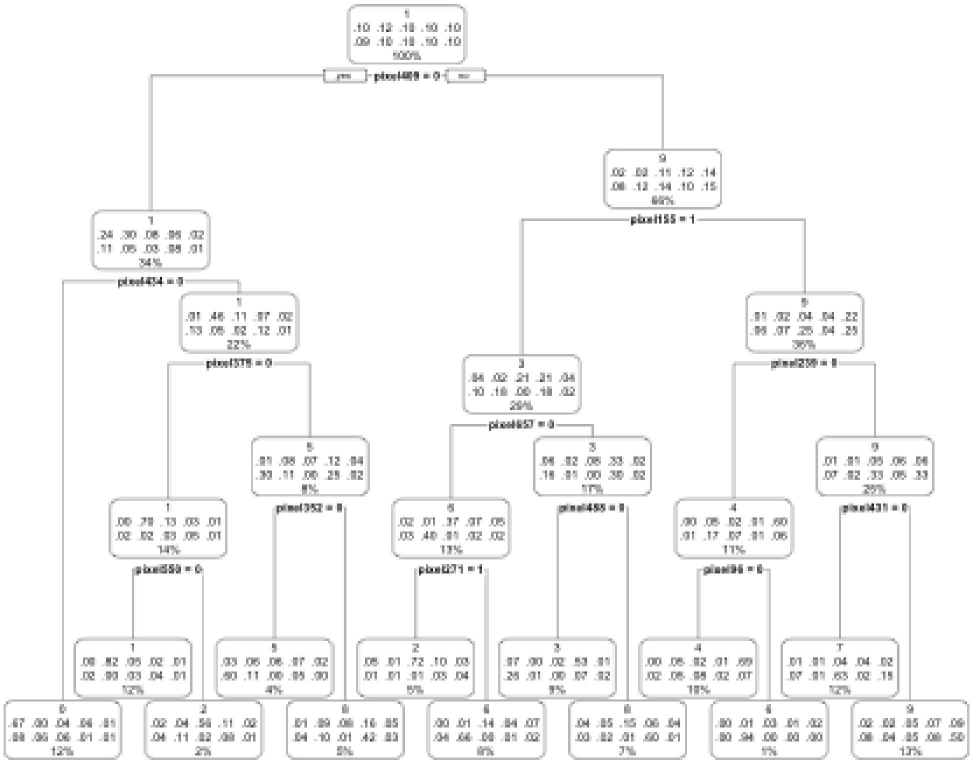
```
plotcp(digit_train_rpart2)
```



Prune tree with new CP of 0.012:

```
pruned2<-prune(digit_train_rpart2, cp=0.012)
rpart.plot(pruned2)
```

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 10 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```



New confusion matrix:

```
test_dt2<-predict(pruned2, test_all, type='class')
#confusion matrix
test1_table2<-table(label=test_dt2, true=test_all$label)
#length(test_dt)
#length(test1$label) #ensuring that these are the same length
confusionMatrix(test1_table2)

## Confusion Matrix and Statistics
##
##      true
## label   0   1   2   3   4   5   6   7   8   9
##     0 164   3  14  17   0  21  18  15   3   2
##     1   0 176   9   1   3   2   2   7   3   3
##     2   2   3 109  17   8   4   9   0   7   2
##     3  20   1   5  99   2  41   2   0  19   5
##          4   1   7   3   2 120   6   9  16   2  10
##          5   2   3   2   6   0  46   6   0   4   0
##          6   0   3  25   5  15  11 132   2   1   1
##          7   5   8   5  10   6  17   3 157   8  37
##          8   4  11  20  24   5   8   7   4 113  14
##          9   4   3  11  21  29  17  21  14  18 138
## 
## Overall Statistics
##                                           
##                Accuracy : 0.627           
##                  95% CI : (0.6054, 0.6482)
##     No Information Rate : 0.109           
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.5849          
##                                           
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8119   0.8073   0.5369   0.4901   0.6383   0.2659
## Specificity            0.9483   0.9832   0.9711   0.9472   0.9691   0.9874
## Pos Pred Value         0.6381   0.8544   0.6770   0.5103   0.6818   0.6667
## Neg Pred Value         0.9782   0.9766   0.9489   0.9430   0.9627   0.9342
## Prevalence             0.1010   0.1090   0.1015   0.1010   0.0940   0.0865
## Detection Rate         0.0820   0.0880   0.0545   0.0495   0.0600   0.0230
## Detection Prevalence   0.1285   0.1030   0.0805   0.0970   0.0880   0.0345
## Balanced Accuracy      0.8801   0.8953   0.7540   0.7186   0.8037   0.6267
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.6316   0.7302   0.6348   0.6509
## Specificity            0.9648   0.9445   0.9468   0.9228
## Pos Pred Value         0.6769   0.6133   0.5381   0.5000
## Neg Pred Value        0.9573   0.9667   0.9637   0.9571
## Prevalence            0.1045   0.1075   0.0890   0.1060
## Detection Rate        0.0660   0.0785   0.0565   0.0690
## Detection Prevalence  0.0975   0.1280   0.1050   0.1380
## Balanced Accuracy     0.7982   0.8374   0.7908   0.7869
```

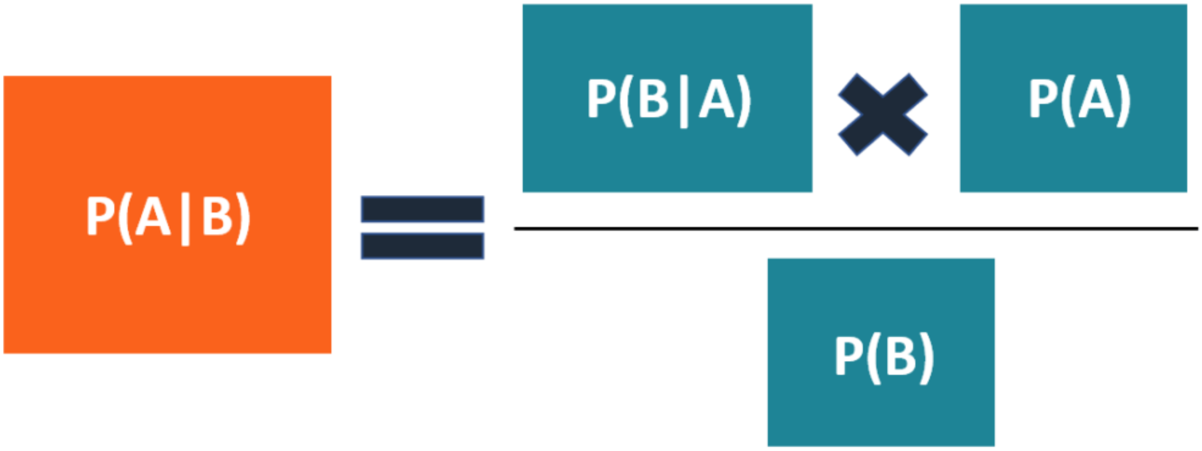
When we add back in the pixels that we had originally removed, we get a slightly better result for our model (62.1% overall accuracy rate). This implies that the pixels we removed, even though they were the sparsest pixels in the images, were somewhat significant for identifying the numbers in the images.

Again, the model was most adept at predicting digit 1, with a balanced accuracy of 89.53%, and performed with least accuracy for predicting digit 5 with an accuracy rate of 62.67%.

Naive Bayes

Overview of Naïve Bayes Model

The Naïve Bayes algorithm is a collection of classification algorithms that is based on Bayes' theorem, which is a mathematical formula used to determine the conditional probability of events and is defined by the following equation:



P(A|B) is the probability of event A occurring given that event B has already occurred. It is calculated by multiplying P(B|A) – its inverse, which states the probability of event B occurring given A has already occurred – by P(A), or the probability of A independently – and then dividing this by the P(B) – the probability of event B occurring independently.[4]

Naïve Bayes takes its name from its primary assumption, which is that all inputs occur independently from one another and are not correlated with one another. The Naïve Bayes algorithm, similar to the decision tree, prioritizes simplicity and practicality over complexity. It is also very useful for its speed, which is useful in modelling datasets with a high number of attributes and observations, such as the MNIST.[5]

Building Naïve Bayes Model

```
nb<- naiveBayes(label~., data=train1, kernel=KT_P, cost=100, scale=FALSE)
print(nb)

## 
## Naive Bayes Classifier for Discrete Predictors
## 
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace, kernel = ..1, 
##     cost = 100, scale = FALSE)
## 
## A-priori probabilities:
## Y
##          0          1          2          3          4          5          6 
## 0.09903244 0.11411497 0.10515083 0.10102447 0.09817871 0.08722254 0.09519067 
##          7          8          9 
## 0.10159363 0.09618668 0.10230507 
## 
## Conditional probabilities:
##    pixel435
## Y         [,1]      [,2]
##   0 0.04022989 0.1966393
```

[4] https://corporatefinanceinstitute.com/resources/data-science/bayes-theorem/

[5] https://www.geeksforgeeks.org/naive-bayes-classifiers/

```
##   1 0.91770574 0.2749840
##   2 0.83761840 0.3690503
##   3 0.61549296 0.4868215
##   4 0.92318841 0.2664854
##   5 0.60358891 0.4895511
##   6 0.80119581 0.3993990
##   7 0.71288515 0.4527328
##   8 0.92011834 0.2713106
##   9 0.90264256 0.2966502
##
##    pixel407
## Y        [,1]      [,2]
##   0 0.0316092 0.1750831
##   1 0.9538653 0.2099076
##   2 0.7604871 0.4270752
##   3 0.8126761 0.3904464
##   4 0.8217391 0.3830098
##   5 0.6313214 0.4828406
##   6 0.8146487 0.3888730
##   7 0.6008403 0.4900690
##   8 0.9393491 0.2388656
##   9 0.8873435 0.3163925
##
##    pixel408
## Y        [,1]      [,2]
##   0 0.0545977 0.2273567
##   1 0.5723192 0.4950510
##   2 0.7889039 0.4083628
##   3 0.8169014 0.3870200
##   4 0.9202899 0.2710404
##   5 0.5840131 0.4932938
##   6 0.8340807 0.3722865
##   7 0.8263305 0.3790906
##   8 0.8801775 0.3249943
##   9 0.9541029 0.2094076
##
```

Now that the model is constructed, we can use it to predict the digits and then put our results in a confusion matrix for analysis.

```
nb_pred<- predict(nb, test1, type="class")
#confusion matrix
test1nb_table<-table(label=nb_pred, true=test1$label)
#length(test_dt)
#length(test1$label)
confusionMatrix(test1nb_table)
```

```
## Confusion Matrix and Statistics
##
##      true
## label   0   1   2   3   4   5   6   7   8   9
##     0 259   0  30   8   4  31   4   3  13   3
##     1   0 343   6  13   2  12  13   5  45  11
##     2   3   0  93   5   3   2   5   3   1   1
##     3   5   3  43 184   3  29   1   6   7   0
##     4   0   0   0   1  66   1   0   3   1   3
##     5   1   3   0   1   5  61   0   2   5   1
##     6   4   2  74  14  17   9 278   1   0   1
##     7   0   0   0   4   1   0   0 112   0   5
##     8  10   3  38  32  24  95   6   3 156   1
##     9  12   3   4  31 154  27   3 181  41 270
##
## Overall Statistics
##
##                Accuracy : 0.6131
##                  95% CI : (0.5953, 0.6306)
##     No Information Rate : 0.1201
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                  Kappa : 0.5692
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.88095   0.9608  0.32292  0.62799  0.23656  0.22846
## Specificity           0.96415   0.9591  0.99143  0.96379  0.99666  0.99335
## Pos Pred Value        0.72958   0.7622  0.80172  0.65480  0.88000  0.77215
## Neg Pred Value        0.98663   0.9944  0.93172  0.95949  0.92648  0.92879
## Prevalence            0.09892   0.1201  0.09690  0.09859  0.09388  0.08984
## Detection Rate        0.08715   0.1154  0.03129  0.06191  0.02221  0.02052
## Detection Prevalence  0.11945   0.1514  0.03903  0.09455  0.02524  0.02658
## Balanced Accuracy     0.92255   0.9599  0.65717  0.79589  0.61661  0.61091
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.89677  0.35110  0.57993  0.91216
## Specificity           0.95417  0.99623  0.92157  0.82960
## Pos Pred Value        0.69500  0.91803  0.42391  0.37190
## Neg Pred Value        0.98756  0.92737  0.95661  0.98842
## Prevalence            0.10431  0.10734  0.09051  0.09960
## Detection Rate        0.09354  0.03769  0.05249  0.09085
## Detection Prevalence  0.13459  0.04105  0.12382  0.24428
## Balanced Accuracy     0.92547  0.67366  0.75075  0.87088
```

Based on these results, the Naïve Bayes Model performed at par with the decision tree model, with a overall accuracy rate of 61.31%. Similar to the previous model, it classified digit 1 with the most accuracy at 95.99% (though this is significantly better than the decision tree model for this digit. This model also classified digits 0 and 6 with accuracy rates of 92%, which is much higher than the previous model. However, there is more variability in the accuracy rates, which range from as low as 61% for digits 4 and 5, to upwards of 92% for digits 0, 1, and 6.

The decision tree model performed slightly better when we included all of the dimensions. Therefore, I will retry the Naïve Bayes model with the all of the pixels to see if this improves the model's accuaracy.

```
#building model
nb2<- naiveBayes(label~., data=train_all, kernel=KT_P, cost=100, scale=FALSE)
#prediction
nb_pred2<- predict(nb2, test_all, type="class")
#confusion matrix
test1nb_table2<-table(label=nb_pred2, true=test_all$label)
#length(test_dt)
#length(test1$label)
confusionMatrix(test1nb_table2)
```

```
## Confusion Matrix and Statistics
##
##      true
## label   0   1   2   3   4   5   6   7   8   9
##     0 185   0  23  20  10  39   3   1  15   3
##     1   0 205   7  14   0   4  11   2  25   3
##     2   1   3  94   9   3   2   4   1   0   0
##     3   1   1  11 105   1   5   0   4   3   1
##     4   0   0   0   0  44   3   1   4   1   1
##     5   2   1   0   0   3  45   0   2   4   1
##     6   5   2  46   6  10  10 183   2   2   2
##     7   0   0   0   4   0   0   0  66   0   2
##     8   6   1  21  26  15  46   3   4  94   2
##     9   2   5   1  18 102  19   4 129  34 197
##
## Overall Statistics
##
##                Accuracy : 0.609
##                  95% CI : (0.5872, 0.6305)
##     No Information Rate : 0.109
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5645
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9158   0.9404   0.4631   0.5198   0.2340   0.2601
## Specificity            0.9366   0.9630   0.9872   0.9850   0.9945   0.9929
## Pos Pred Value         0.6187   0.7565   0.8034   0.7955   0.8148   0.7759
## Neg Pred Value         0.9900   0.9925   0.9421   0.9481   0.9260   0.9341
## Prevalence             0.1010   0.1090   0.1015   0.1010   0.0940   0.0865
## Detection Rate         0.0925   0.1025   0.0470   0.0525   0.0220   0.0225
## Detection Prevalence   0.1495   0.1355   0.0585   0.0660   0.0270   0.0290
## Balanced Accuracy      0.9262   0.9517   0.7251   0.7524   0.6143   0.6265
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.8756   0.3070   0.5281   0.9292
## Specificity            0.9525   0.9966   0.9319   0.8244
## Pos Pred Value         0.6828   0.9167   0.4312   0.3855
## Neg Pred Value         0.9850   0.9227   0.9529   0.9899
## Prevalence             0.1045   0.1075   0.0890   0.1060
## Detection Rate         0.0915   0.0330   0.0470   0.0985
## Detection Prevalence   0.1340   0.0360   0.1090   0.2555
## Balanced Accuracy      0.9141   0.6518   0.7300   0.8768
```

Surprisingly, the model performed slightly worse when all of the dimensions in the dataset were included, coming in with an overall accuracy rate of 60.9%. This is surprising, considering that the decision tree model performed better when it was fed higher dimensionality data.

Model Comparison and Conclusion

The overall accuracy rates for the models are as follows:

	Decision Tree	Naïve Bayes
Lower Dimensionality Data	60.97%	61.31%
Higher Dimensionality Data	62.7%	60.9%

The model with the greatest overall accuracy was the decision tree after it was pruned and fed the higher dimensionality data (original data). Surprisingly, the model with the lowest accuracy was the Naïve Bayes algorithm when it was also provided with the higher dimensionality data.

This implies a key distinction between the models: the Decision Tree may function better with more dimensions and attributes while the Naïve Bayes model may function better with a more simplified dataset. However, all the models performed relatively at par with one another, with accuracy rates from between 60.9% to 62.7%. This is a very small difference between the four different models, so further analysis is needed to determine how each model truly performs.

Both the Decision Tree Model and the Naïve Bayes model performed better in identifying certain digits over other ones, as demonstrated below:

Digit	Decision Tree Accuracy Rate (Higher Dimension Data)	Naïve Bayes Accuracy Rate (Lower Dimension Data)
0	88.01%	92.25%
1	89.53%	95.99%
2	75.40%	65.72%
3	71.86%	79.59%
4	80.37%	61.66%
5	62.67%	61.09%

6	79.82%	92.55%
7	83.74%	67.37%
8	79.08%	75.08%
9	78.69%	87.09%

Digits with high accuracy at highlighted in green (above 85%), digits with low accuracy are highlighted in red (below 65%), and digits with moderate accuracy are highlighted in orange. In the above figure, you can see that the Naïve Bayes model had much more variability in its predictions compared to the Decision Tree model. Both models performed best with digits 0 and 1, while the performed most poorly with digit 5.

Ultimately, I believe that better pre-processing of the data would lend to more accuracy in the predictions. One solution that could be used in the future is the Principal Components Analysis. While I attempted this (see appendix), I could not figure out how to construct a dataset with the new variables that could be inputted into a confusion matrix with the test set.

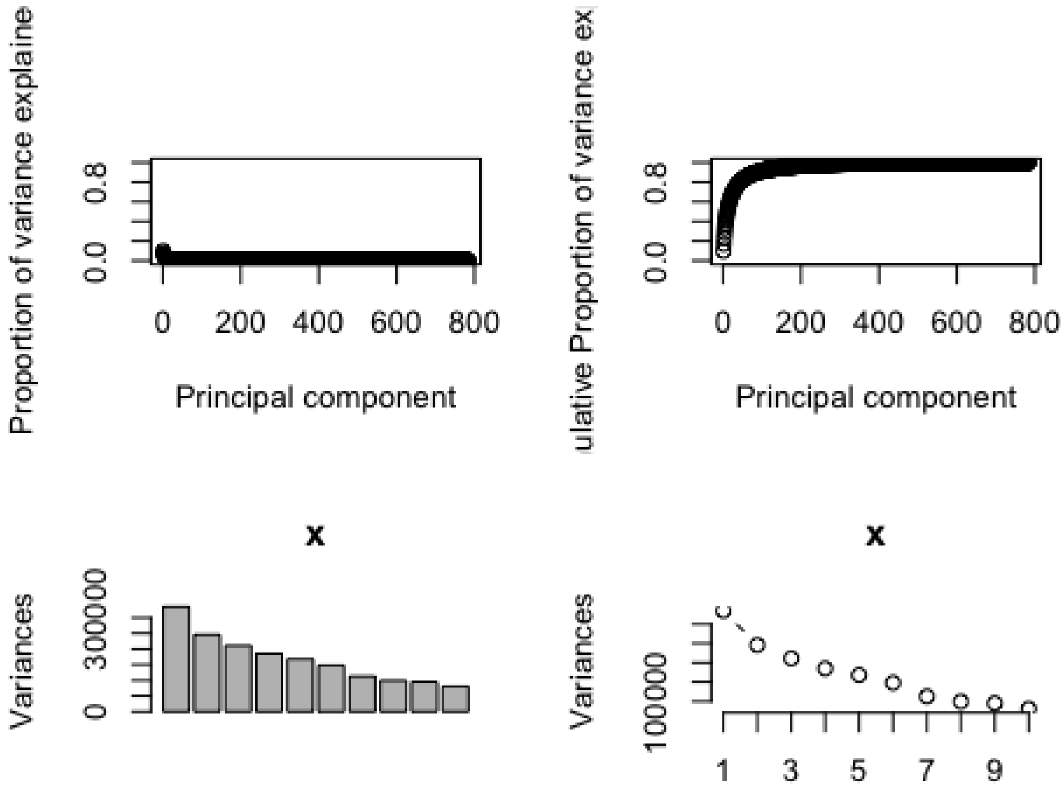
Appendix (PCA Analysis)

```
library(rpart)
library(rpart.plot)
library(e1071)
library(tree)
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.3     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ──────────────────────────────────────────── tidyverse_conflicts()
──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
errors

library(caret)

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift

digit_test<- read.csv("~/Downloads/Kaggle-digit-test-1 (2).csv")
digit_train<- read.csv("~/Downloads/Kaggle-digit-train (2).csv")

digit_train_nolab<-digit_train[,-1]
```

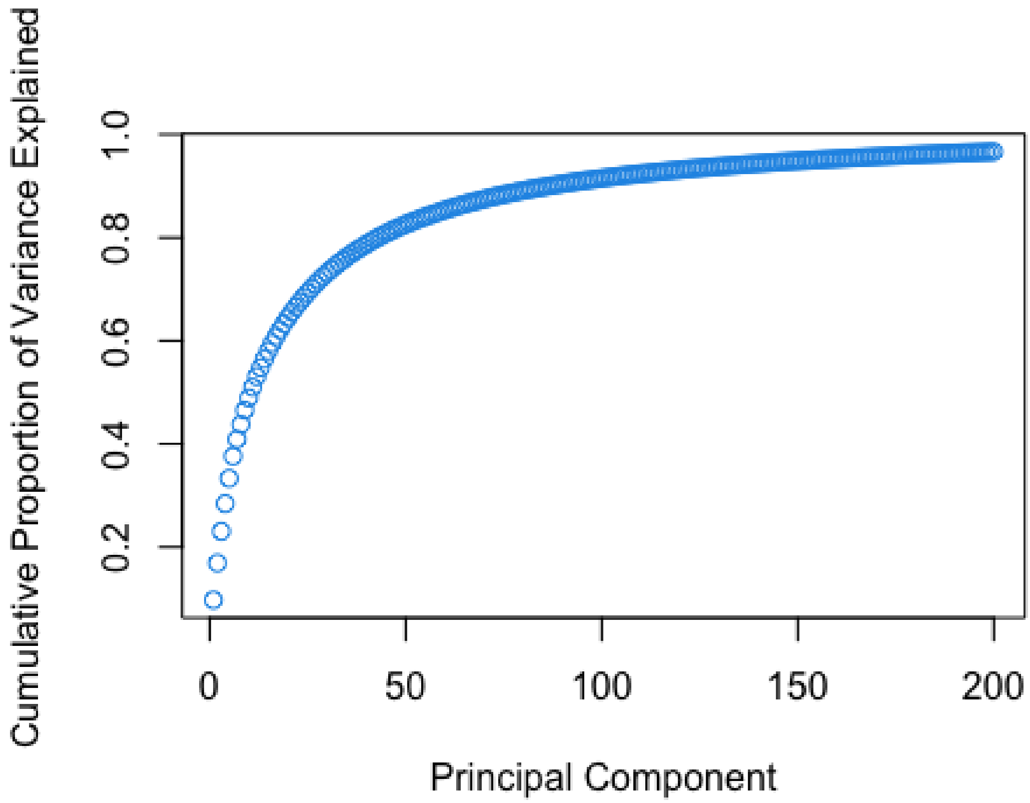
R Markdown

```
require(factoextra)

## Loading required package: factoextra

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

require(plotly)

## Loading required package: plotly

##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

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

## The following object is masked from 'package:graphics':
##
##     layout

digit_train.pca <- prcomp(digit_train_nolab,center = TRUE)
#print(sample_digit_train.pca)
#find dimensions with highest variance
#summary(sample_digit_train)
pcaCharts <- function(x) {
  x.var <- x$sdev ^ 2
  x.pvar <- x.var/sum(x.var)
  print("proportions of variance:")
  print(x.pvar)

  par(mfrow=c(2,2))
  plot(x.pvar,xlab="Principal component", ylab="Proportion of variance explained", ylim=c(0,1),
type='b')
   plot(cumsum(x.pvar),xlab="Principal component", ylab="Cumulative Proportion of variance
explained", ylim=c(0,1), type='b')
   screeplot(x)
   screeplot(x,type="l")
   par(mfrow=c(1,1))
}
pcaCharts(digit_train.pca)
```

```
## [1] "proportions of variance:"
##   [1] 9.748938e-02 7.160266e-02 6.145903e-02 5.379302e-02 4.894262e-02
##   [6] 4.303214e-02 3.277051e-02 2.892103e-02 2.766902e-02 2.348871e-02
##  [11] 2.099325e-02 2.059001e-02 1.702553e-02 1.692787e-02 1.581126e-02
##  [16] 1.483240e-02 1.319688e-02 1.282727e-02 1.187976e-02 1.152755e-02
##  [21] 1.072191e-02 1.015199e-02 9.649023e-03 9.128461e-03 8.876409e-03
##  [26] 8.387663e-03 8.118559e-03 7.774057e-03 7.406351e-03 6.866615e-03
##  [31] 6.579822e-03 6.387986e-03 5.993670e-03 5.889134e-03 5.643352e-03
##  [36] 5.409670e-03 5.092219e-03 4.875049e-03 4.755694e-03 4.665447e-03
##  [41] 4.529525e-03 4.449892e-03 4.182553e-03 3.975058e-03 3.845420e-03
##  [46] 3.749195e-03 3.610132e-03 3.485222e-03 3.364878e-03 3.207381e-03
##  [51] 3.154671e-03 3.091455e-03 2.937092e-03 2.865413e-03 2.807594e-03
##  [56] 2.696184e-03 2.658314e-03 2.562986e-03 2.538211e-03 2.461783e-03
##  [61] 2.397162e-03 2.387396e-03 2.275914e-03 2.215184e-03 2.139336e-03
##  [66] 2.061334e-03 2.028511e-03 1.959767e-03 1.936386e-03 1.884853e-03
##  [71] 1.867510e-03 1.816700e-03 1.768913e-03 1.725924e-03 1.661208e-03
##  [76] 1.633095e-03 1.606013e-03 1.544724e-03 1.468497e-03 1.423759e-03
##  [81] 1.410985e-03 1.402284e-03 1.388348e-03 1.354173e-03 1.323072e-03
##  [86] 1.307799e-03 1.296738e-03 1.242404e-03 1.222491e-03 1.196245e-03
##  [91] 1.158403e-03 1.138590e-03 1.122635e-03 1.104752e-03 1.081331e-03
##  [96] 1.074126e-03 1.038656e-03 1.033220e-03 1.014946e-03 9.999650e-04
## [101] 9.748190e-04 9.450577e-04 9.386418e-04 9.122210e-04 9.073135e-04
## [106] 8.888716e-04 8.636997e-04 8.442350e-04 8.355416e-04 8.166521e-04
## [111] 7.876812e-04 7.815596e-04 7.774646e-04 7.719337e-04 7.578423e-04
## [116] 7.502156e-04 7.344758e-04 7.257721e-04 7.153227e-04 7.003250e-04
## [121] 6.930496e-04 6.857411e-04 6.799331e-04 6.657187e-04 6.561385e-04
## [126] 6.448018e-04 6.353918e-04 6.261249e-04 6.185142e-04 6.057413e-04
## [131] 6.038548e-04 5.914527e-04 5.858988e-04 5.846332e-04 5.754819e-04
## [136] 5.697168e-04 5.644971e-04 5.531741e-04 5.343447e-04 5.257771e-04
## [141] 5.219656e-04 5.111938e-04 5.051442e-04 4.999250e-04 4.953237e-04
## [146] 4.923486e-04 4.843951e-04 4.766862e-04 4.746660e-04 4.678909e-04
## [151] 4.652967e-04 4.613623e-04 4.563365e-04 4.517562e-04 4.494886e-04
## [156] 4.413574e-04 4.387901e-04 4.243893e-04 4.203146e-04 4.163554e-04
## [161] 4.138303e-04 4.070993e-04 3.985484e-04 3.943591e-04 3.940671e-04
## [166] 3.914013e-04 3.816656e-04 3.792309e-04 3.755363e-04 3.738317e-04
## [171] 3.663377e-04 3.630741e-04 3.593076e-04 3.570320e-04 3.530286e-04
## [176] 3.527546e-04 3.453024e-04 3.432769e-04 3.418910e-04 3.391659e-04
## [181] 3.348615e-04 3.298425e-04 3.255064e-04 3.247036e-04 3.210138e-04
## [186] 3.204329e-04 3.171947e-04 3.168191e-04 3.109165e-04 3.101730e-04
## [191] 3.068330e-04 3.038232e-04 2.997089e-04 2.983271e-04 2.940932e-04
## [196] 2.933920e-04 2.929772e-04 2.892764e-04 2.849890e-04 2.836973e-04
## [201] 2.810822e-04 2.762584e-04 2.741035e-04 2.716020e-04 2.675224e-04
## [206] 2.667939e-04 2.628922e-04 2.620466e-04 2.614872e-04 2.582536e-04
## [211] 2.566039e-04 2.553872e-04 2.541116e-04 2.528139e-04 2.509125e-04
## [216] 2.482861e-04 2.476159e-04 2.444614e-04 2.432867e-04 2.409924e-04
## [221] 2.402759e-04 2.393645e-04 2.385944e-04 2.364660e-04 2.320577e-04
## [226] 2.309875e-04 2.289827e-04 2.272866e-04 2.259669e-04 2.247819e-04
## [231] 2.209783e-04 2.194530e-04 2.173839e-04 2.159275e-04 2.149510e-04
## [236] 2.136149e-04 2.114882e-04 2.108641e-04 2.081134e-04 2.051707e-04
## [241] 2.037462e-04 2.032718e-04 2.017844e-04 1.996661e-04 1.979844e-04
## [246] 1.968069e-04 1.948171e-04 1.937177e-04 1.926669e-04 1.921244e-04
## [251] 1.902499e-04 1.883889e-04 1.868300e-04 1.849897e-04 1.846963e-04
## [256] 1.841008e-04 1.827881e-04 1.823236e-04 1.813922e-04 1.796128e-04
## [261] 1.767108e-04 1.756164e-04 1.748565e-04 1.734837e-04 1.725241e-04
## [266] 1.714765e-04 1.711332e-04 1.687730e-04 1.681334e-04 1.676928e-04
## [271] 1.662683e-04 1.642935e-04 1.639027e-04 1.630261e-04 1.622473e-04
## [276] 1.604343e-04 1.602755e-04 1.588478e-04 1.583913e-04 1.572491e-04
## [281] 1.554212e-04 1.544494e-04 1.536261e-04 1.518331e-04 1.508824e-04
## [286] 1.504021e-04 1.490375e-04 1.482894e-04 1.466982e-04 1.459181e-04
## [291] 1.434703e-04 1.434138e-04 1.429473e-04 1.418640e-04 1.414345e-04
## [296] 1.403223e-04 1.383426e-04 1.374465e-04 1.363785e-04 1.354505e-04
## [301] 1.351599e-04 1.344041e-04 1.333983e-04 1.322870e-04 1.307529e-04
## [306] 1.298907e-04 1.285353e-04 1.278489e-04 1.269745e-04 1.266083e-04
## [311] 1.258839e-04 1.251527e-04 1.240269e-04 1.225415e-04 1.223236e-04
## [316] 1.211846e-04 1.208661e-04 1.200818e-04 1.190165e-04 1.182379e-04
## [321] 1.172344e-04 1.157411e-04 1.153602e-04 1.145606e-04 1.136267e-04
## [326] 1.131858e-04 1.117558e-04 1.102935e-04 1.099606e-04 1.093495e-04
## [331] 1.091768e-04 1.084774e-04 1.077975e-04 1.069175e-04 1.067235e-04
## [336] 1.060787e-04 1.044644e-04 1.043962e-04 1.030692e-04 1.022149e-04
## [341] 1.013709e-04 1.005946e-04 9.976689e-05 9.944067e-05 9.858291e-05
## [346] 9.701599e-05 9.697845e-05 9.629361e-05 9.461096e-05 9.412853e-05
## [351] 9.304408e-05 9.257216e-05 9.158093e-05 9.107929e-05 9.050181e-05
## [356] 9.003006e-05 8.961549e-05 8.898158e-05 8.832001e-05 8.752939e-05
## [361] 8.712553e-05 8.589760e-05 8.504838e-05 8.504020e-05 8.463656e-05
## [366] 8.383057e-05 8.273177e-05 8.250521e-05 8.216028e-05 8.085855e-05
## [371] 7.970905e-05 7.900246e-05 7.838146e-05 7.803764e-05 7.745423e-05
## [376] 7.660364e-05 7.575152e-05 7.493093e-05 7.423327e-05 7.317602e-05
## [381] 7.300815e-05 7.254482e-05 7.224468e-05 7.175438e-05 7.127845e-05
## [386] 6.954879e-05 6.919512e-05 6.855459e-05 6.786175e-05 6.702182e-05
## [391] 6.649535e-05 6.584339e-05 6.500547e-05 6.388378e-05 6.290813e-05
## [396] 6.266935e-05 6.131352e-05 6.076087e-05 6.027283e-05 5.948505e-05
## [401] 5.879971e-05 5.826196e-05 5.775367e-05 5.750266e-05 5.627282e-05
## [406] 5.609293e-05 5.499435e-05 5.446265e-05 5.395817e-05 5.382608e-05
## [411] 5.313103e-05 5.275921e-05 5.219731e-05 5.137443e-05 5.125531e-05
## [416] 5.064784e-05 5.008442e-05 4.935020e-05 4.926479e-05 4.826127e-05
## [421] 4.721410e-05 4.684736e-05 4.611518e-05 4.594422e-05 4.558186e-05
## [426] 4.437119e-05 4.361689e-05 4.291449e-05 4.260114e-05 4.242961e-05
## [431] 4.182061e-05 4.111446e-05 4.049884e-05 4.040296e-05 4.010153e-05
## [436] 3.837037e-05 3.791703e-05 3.767531e-05 3.748136e-05 3.720804e-05
## [441] 3.648971e-05 3.608664e-05 3.530905e-05 3.513495e-05 3.476555e-05
## [446] 3.443352e-05 3.402032e-05 3.372430e-05 3.320936e-05 3.312085e-05
## [451] 3.214127e-05 3.155662e-05 3.129649e-05 3.082178e-05 3.015722e-05
## [456] 2.982009e-05 2.944766e-05 2.908948e-05 2.888922e-05 2.844263e-05
## [461] 2.825941e-05 2.789678e-05 2.689222e-05 2.668279e-05 2.562381e-05
## [466] 2.556813e-05 2.540919e-05 2.482816e-05 2.473559e-05 2.438074e-05
## [471] 2.423827e-05 2.403310e-05 2.361807e-05 2.305838e-05 2.267943e-05
## [476] 2.236841e-05 2.177244e-05 2.137985e-05 2.121537e-05 2.111529e-05
## [481] 2.078512e-05 2.064776e-05 2.031171e-05 1.996141e-05 1.978354e-05
## [486] 1.947611e-05 1.890664e-05 1.884581e-05 1.855807e-05 1.817872e-05
## [491] 1.788657e-05 1.771081e-05 1.743040e-05 1.725148e-05 1.611400e-05
## [496] 1.580814e-05 1.569456e-05 1.565934e-05 1.529315e-05 1.518174e-05
## [501] 1.500891e-05 1.474412e-05 1.445963e-05 1.430036e-05 1.420527e-05
## [506] 1.389297e-05 1.385690e-05 1.367917e-05 1.341038e-05 1.324788e-05
## [511] 1.316656e-05 1.252166e-05 1.233069e-05 1.231431e-05 1.223922e-05
## [516] 1.184710e-05 1.165821e-05 1.163007e-05 1.159093e-05 1.146784e-05
## [521] 1.107291e-05 1.092944e-05 1.065816e-05 1.044859e-05 1.037343e-05
## [526] 1.030392e-05 9.858283e-06 9.499780e-06 9.298707e-06 9.185678e-06
## [531] 9.065298e-06 8.974464e-06 8.915254e-06 8.572538e-06 8.510337e-06
## [536] 8.189111e-06 8.002535e-06 7.904847e-06 7.718658e-06 7.670241e-06
## [541] 7.449644e-06 7.398019e-06 7.277331e-06 7.053970e-06 6.939784e-06
## [546] 6.685944e-06 6.627613e-06 6.476094e-06 6.440528e-06 6.081937e-06
## [551] 6.023899e-06 5.835582e-06 5.662937e-06 5.538908e-06 5.439308e-06
## [556] 5.336225e-06 5.203551e-06 5.179646e-06 5.104440e-06 5.008242e-06
## [561] 4.927635e-06 4.801141e-06 4.755103e-06 4.539302e-06 4.359327e-06
## [566] 4.283655e-06 4.211796e-06 4.048709e-06 3.997179e-06 3.973068e-06
## [571] 3.832559e-06 3.813016e-06 3.779690e-06 3.556923e-06 3.409421e-06
## [576] 3.372608e-06 3.266698e-06 3.122047e-06 3.049520e-06 3.038813e-06
## [581] 2.987089e-06 2.830493e-06 2.709116e-06 2.664806e-06 2.618400e-06
## [586] 2.606192e-06 2.544426e-06 2.516392e-06 2.449461e-06 2.368870e-06
## [591] 2.311004e-06 2.265028e-06 2.234702e-06 2.197117e-06 2.137343e-06
## [596] 2.058384e-06 1.949946e-06 1.876784e-06 1.782735e-06 1.762363e-06
## [601] 1.757505e-06 1.653312e-06 1.607786e-06 1.591173e-06 1.574526e-06
## [606] 1.480167e-06 1.434589e-06 1.392824e-06 1.382899e-06 1.282288e-06
## [611] 1.278578e-06 1.239309e-06 1.206412e-06 1.181376e-06 1.135853e-06
## [616] 1.082742e-06 1.065598e-06 1.004103e-06 9.461886e-07 9.218848e-07
## [621] 8.723958e-07 8.616638e-07 8.362935e-07 8.275123e-07 7.995455e-07
## [626] 7.828223e-07 7.658973e-07 7.181268e-07 6.985372e-07 6.954597e-07
## [631] 6.916828e-07 6.533242e-07 6.380940e-07 6.208021e-07 6.001076e-07
## [636] 5.545331e-07 5.441608e-07 5.194321e-07 5.110798e-07 4.957448e-07
## [641] 4.900111e-07 4.819120e-07 3.797032e-07 3.721904e-07 3.627632e-07
## [646] 3.529544e-07 3.238378e-07 3.235120e-07 3.174470e-07 3.068858e-07
## [651] 3.002305e-07 2.815474e-07 2.485342e-07 2.457845e-07 2.339119e-07
## [656] 2.302431e-07 2.190187e-07 2.056246e-07 1.976633e-07 1.898105e-07
## [661] 1.894057e-07 1.860778e-07 1.753207e-07 1.709149e-07 1.651990e-07
## [666] 1.123102e-07 1.096545e-07 9.065878e-08 8.727857e-08 7.502624e-08
## [671] 7.480487e-08 6.891332e-08 6.814494e-08 5.148387e-08 4.803269e-08
## [676] 4.609692e-08 4.271066e-08 3.940602e-08 3.801991e-08 3.237713e-08
## [681] 2.351720e-08 2.214034e-08 2.068346e-08 1.971342e-08 1.933952e-08
## [686] 1.762724e-08 1.709607e-08 1.519626e-08 1.317004e-08 1.181564e-08
## [691] 9.806628e-09 7.425530e-09 7.075454e-09 5.755694e-09 3.812521e-09
## [696] 3.309532e-09 2.012697e-09 1.857054e-09 8.673978e-10 1.623083e-10
## [701] 1.064140e-10 8.869195e-11 1.757791e-11 1.601492e-11 2.772125e-31
## [706] 1.137526e-31 1.006629e-31 6.962796e-32 6.147171e-32 3.258383e-32
## [711] 2.610033e-32 2.120875e-32 1.797840e-32 1.738180e-32 1.577728e-32
## [716] 1.317350e-32 1.115899e-32 9.855838e-33 9.262806e-33 8.518789e-33
## [721] 6.960777e-33 5.855957e-33 5.158154e-33 4.561457e-33 4.129502e-33
## [726] 3.308579e-33 3.057365e-33 2.473505e-33 2.096832e-33 1.779104e-33
## [731] 1.395141e-33 1.236174e-33 7.837795e-34 5.341635e-34 4.932921e-34
## [736] 4.355100e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [741] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [746] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [751] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [756] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [761] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [766] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [771] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34
## [776] 4.013092e-34 4.013092e-34 4.013092e-34 4.013092e-34 3.984085e-34
## [781] 2.553531e-34 9.792730e-35 4.741410e-35 6.388275e-36
```



```
pca<-princomp(digit_train_nolab)
std_dev <- pca[1:260]$sdev
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)
#plot cumulative variance
plot(cumsum(prop_varex[1:200]), xlab = "Principal Component",
ylab = "Cumulative Proportion of Variance Explained",
type = "b",
col = 4)
```



Based on the above charts, we can see that the largest proportion of variance is explained by only 200 attributes, meaning we should be able to reduce the dimensionality to at least this size.