THE iSCHOOL
Syracuse University

IST707 – Data Analytics
Prof.: Dr. Bolton

Homework 6&7 Report:

kNN, Random Forest, SVM, Decision Tree and Naïve Bayes

Name: Yehuda Perry
Course: IST707 – Data Analytics
Professor: Dr. Bolton
Homework: 6 & 7 (Combined)
Date: 05/18/2020

Table of Contents

1. Introduction

MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, the classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.[1]

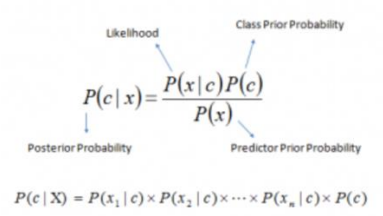
In the Kaggle competition, the goal is to correctly identify digits from a dataset of tens of thousands of handwritten images using machine learning algorithms. This paper will discuss digit recognizer using Naïve Bayes, Decision Tree, kNN, Random Forest and SVM algorithms.

1.1 Naïve Bayes

Naive Bayes has three flaws when applied to document classification. First, a word’s non-appearance counts just as much its appearance, whereas surely a document’s class is determined by the words that are in it rather than those that aren’t? Second, Naive Bayes doesn’t take account of the number of appearances of a word, whereas surely frequently occurring words should have a greater influence on the class than ones that only appear once? Third, it treats all words the same, whereas surely unusual words like “weka” and “breakfast” should count more than common ones like “and” and “the”? Multinomial Naive Bayes is a classification method that solves these problems and is generally better and faster than plain Naive Bayes.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naïve Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c) as the following equation below:

[1] https://www.kaggle.com/c/digit-recognizer



Above,

- $P(c|x)$ is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Figure 1: Bayes Theorem

1.2 Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

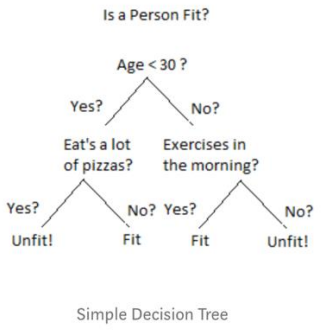


Figure 2: Sample Decision Tree

How Do Decision Trees Work? There are several steps involved in the building of a decision tree. **Splitting:** The process of partitioning the data set into subsets. Splits are formed on a particular variable.

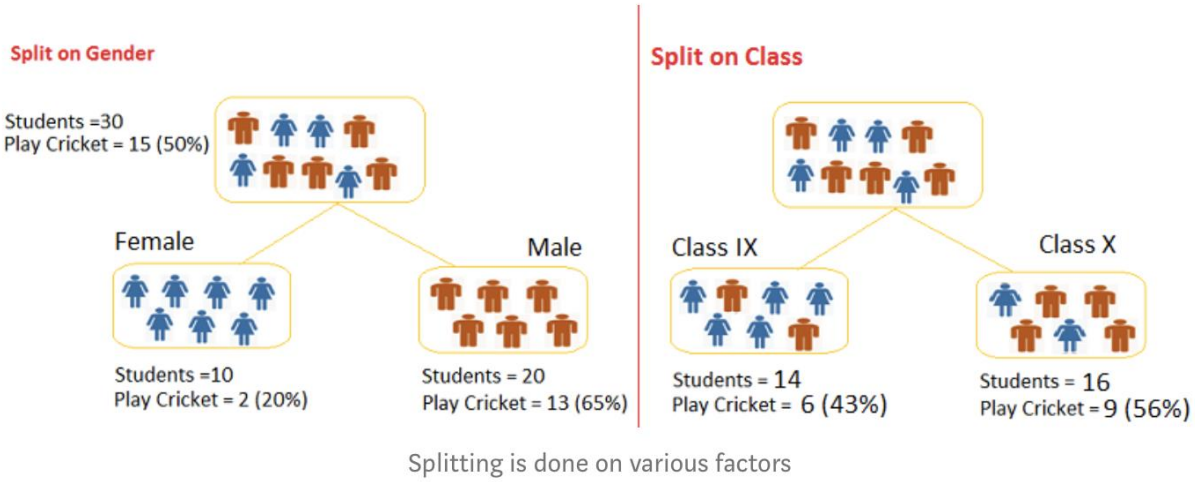


Figure 3: Decision Tree Splitting

Pruning: he shortening of branches of the tree. Pruning is the process of reducing the size of the tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch. Pruning is useful because classification trees may fit the training data well, but may do a poor job of classifying new values. A simpler tree often avoids over-fitting.

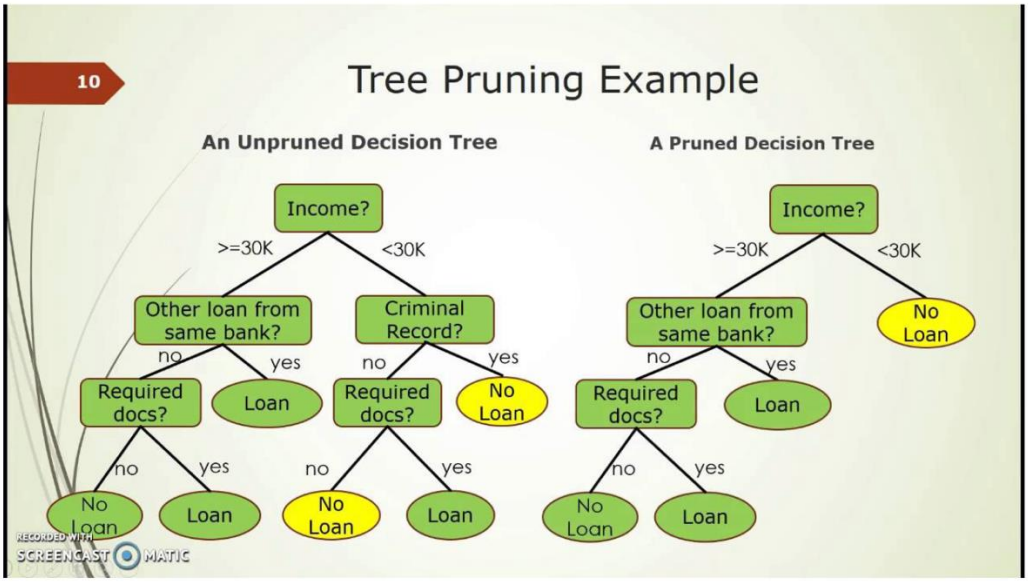


Figure 4: Tree Pruning Example

As we can see above, a pruned tree has less nodes and has less sparsity than a unpruned decision tree.

Tree Selection: The process of finding the smallest tree that fits the data. Usually this is the tree that yields the lowest cross-validated error.

1.3 K-nearest neighbors (kNN)

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

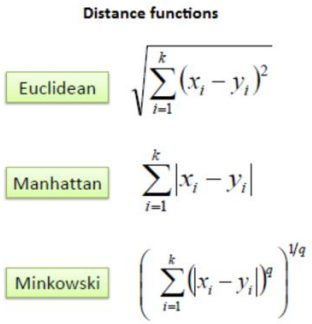
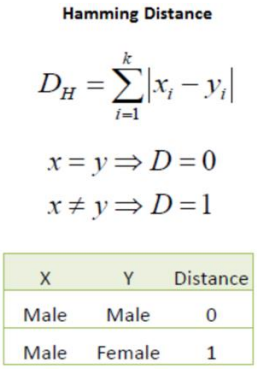


Figure 5: Distance Function

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables, the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.



X	Y	Distance
Male	Male	0
Male	Female	1

Figure 6: Hamming Distance

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

1.4 Random Forest

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks).

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below is how a random forest would look like with two trees:

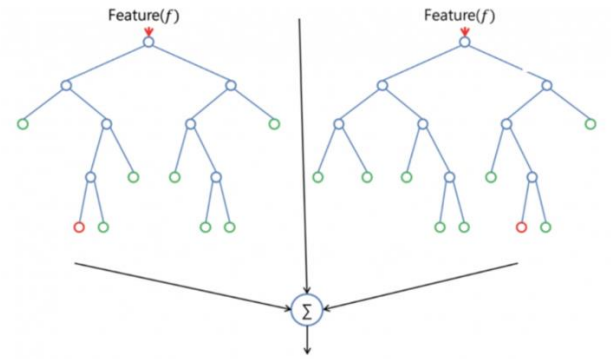


Figure 7: Random Forest two trees

Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because it can easily use the classifier-class of random forest. With random forest, it can also deal with regression tasks by using the algorithm's regressor.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. It can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).[2]

1.5 Support Vector Machine (SVM)

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

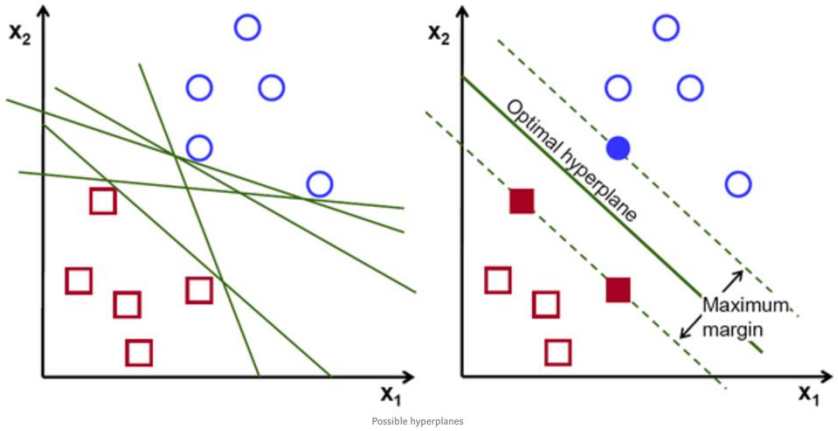


Figure 8: Possible Hyperplane

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

[2] https://builtin.com/data-science/random-forest-algorithm

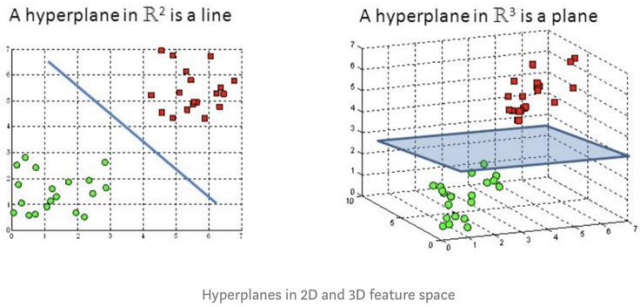


Figure 9: Hyperplane in 2D and 3D

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

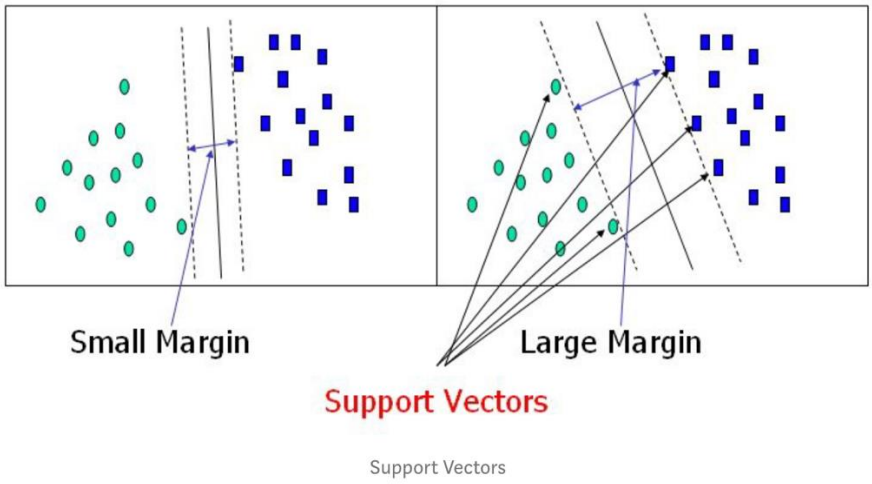


Figure 10: Support Vectors

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, it maximizes the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help to build SVM.

1.6 Research Instructions (HW6)

Now that we have learned two classification algorithms, decision tree and Naïve Bayes, let's think further on the question of choosing algorithms for a specific task. Note that there is no silver bullet in terms of algorithm comparison – no algorithm would outperform all other algorithms on all data sets. Therefore, choosing appropriate algorithms is an important decision, and it requires knowledge of both the data set and the candidate algorithms. In this homework, you will compare Naïve Bayes and decision tree for handwriting recognition.

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 10, 20 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

Tip: check out the Kaggle forum to see if there are some patterns other people have found that you can use to build better models.

1.7 Research Instructions (HW7)

In this homework, you will use SVMs, kNN, and Random Forest algorithms for handwriting recognition, and compare their performance with the Naïve Bayes and decision tree models you built in previous week.

Steps:

1. Describe data pre-processing steps and the chosen evaluation method and measure(s)

2. Use the train set to build kNN, SVM, and Random Forest models. Submit these models' prediction results to Kaggle. Report test performance, compare them, and use the theoretic knowledge to explain whether the algorithm performance difference makes sense or not.
3. Write a report to describe what you did, including the data preparation, transformation, algorithm tuning, the generated models and their performance. In the end, summarize which model works the best and why.
4. If you use Weka to do the experiment, write your report in Microsoft Word. Up to 8 pages. NO PDF PLEASE.
5. If you use R, use R Markdown tool to prepare your document and output to Word. Don't print out excessively large amount of output, such as prediction results, or the entire dataset.

2. Analysis and Models

2.1 About the data

Dataset name: 'train' and 'test' (From Kaggle)
Dataset format: csv files
Description: train: 42000 obs with 784 variables
test: 28000 obs with 784 variables
Data Dictionary: Data Dictionary was evaluated to support the dataset.
Reference R Code: '

2.2 Libraries, Data Loading and Exploring the Data

First, Loading require packages to R to restructure and visualize the dataset.

```
library(datasets)
library(class)
library(ggplot2)
library(plyr)
library(dplyr)
library(lattice)
library(caret)
library(e1071)
library(gmodels)
library(randomForest)
library(tree)
library(MASS)
library(rpart)
library(rpart.plot)
library(proto)
library(readr)
library(e1071)
```

```
library(naivebayes)
library(statsr)
library(statsExpressions)
library(statsguRu)
library(stats19)
library(tidyverse)
library(gbm)
library(rpart.utils)
library(rpart.LAD)
library(Rcpp)
library(Rtsne)
library(RColorBrewer)
library(rattle)
library(lattice)
library(caret)
library(cluster)
```

```
library(class)
library(foreign)
library(tree)
library(maptree)
```

Figure 11: Loading require R packages

The next step is to load and read both train and test datasets.

```
train <- read.csv("~/Downloads/trainkg.csv", header = TRUE)
test <- read.csv("~/Downloads/testkg.csv", header = TRUE)
```

Figure 12: Load and read the datasets

Now, exploring the data structure:

```
> nrow(train)
[1] 42000
> nrow(test)
[1] 28000
```

Figure 13: nrow

```
#explorer the data
#ETD1
train
test
```

```
> train
  label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9 pixel10 pixel11 pixel12 pixel13 pixel14
1     1      0      0      0      0      0      0      0      0      0      0       0       0       0       0       0
  pixel15 pixel16 pixel17 pixel18 pixel19 pixel20 pixel21 pixel22 pixel23 pixel24 pixel25 pixel26 pixel27 pixel28
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel29 pixel30 pixel31 pixel32 pixel33 pixel34 pixel35 pixel36 pixel37 pixel38 pixel39 pixel40 pixel41 pixel42
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel43 pixel44 pixel45 pixel46 pixel47 pixel48 pixel49 pixel50 pixel51 pixel52 pixel53 pixel54 pixel55 pixel56
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel57 pixel58 pixel59 pixel60 pixel61 pixel62 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68 pixel69 pixel70
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel71 pixel72 pixel73 pixel74 pixel75 pixel76 pixel77 pixel78 pixel79 pixel80 pixel81 pixel82 pixel83 pixel84
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel85 pixel86 pixel87 pixel88 pixel89 pixel90 pixel91 pixel92 pixel93 pixel94 pixel95 pixel96 pixel97 pixel98
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel99 pixel100 pixel101 pixel102 pixel103 pixel104 pixel105 pixel106 pixel107 pixel108 pixel109 pixel110 pixel111
1       0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel112 pixel113 pixel114 pixel115 pixel116 pixel117 pixel118 pixel119 pixel120 pixel121 pixel122 pixel123 pixel124
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel125 pixel126 pixel127 pixel128 pixel129 pixel130 pixel131 pixel132 pixel133 pixel134 pixel135 pixel136 pixel137
1        0        0        0        0        0        0        0      188      255       94        0        0        0
  pixel138 pixel139 pixel140 pixel141 pixel142 pixel143 pixel144 pixel145 pixel146 pixel147 pixel148 pixel149 pixel150
1        0        0        0        0        0        0        0        0        0        0        0        0        0
```

Figure 14: train dataset

```
> test
  pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9 pixel10 pixel11 pixel12 pixel13 pixel14 pixel15
1      0      0      0      0      0      0      0      0      0      0       0       0       0       0       0       0
  pixel16 pixel17 pixel18 pixel19 pixel20 pixel21 pixel22 pixel23 pixel24 pixel25 pixel26 pixel27 pixel28 pixel29
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel30 pixel31 pixel32 pixel33 pixel34 pixel35 pixel36 pixel37 pixel38 pixel39 pixel40 pixel41 pixel42 pixel43
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel44 pixel45 pixel46 pixel47 pixel48 pixel49 pixel50 pixel51 pixel52 pixel53 pixel54 pixel55 pixel56 pixel57
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel58 pixel59 pixel60 pixel61 pixel62 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68 pixel69 pixel70 pixel71
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel72 pixel73 pixel74 pixel75 pixel76 pixel77 pixel78 pixel79 pixel80 pixel81 pixel82 pixel83 pixel84 pixel85
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel86 pixel87 pixel88 pixel89 pixel90 pixel91 pixel92 pixel93 pixel94 pixel95 pixel96 pixel97 pixel98 pixel99
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel100 pixel101 pixel102 pixel103 pixel104 pixel105 pixel106 pixel107 pixel108 pixel109 pixel110 pixel111 pixel112
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel113 pixel114 pixel115 pixel116 pixel117 pixel118 pixel119 pixel120 pixel121 pixel122 pixel123 pixel124 pixel125
1        0        0        0        0        0        0        0        0        0       10       17       17       17
  pixel126 pixel127 pixel128 pixel129 pixel130 pixel131 pixel132 pixel133 pixel134 pixel135 pixel136 pixel137 pixel138
1       17       81      180      180       35        0        0        0        0        0        0        0        0
  pixel139 pixel140 pixel141 pixel142 pixel143 pixel144 pixel145 pixel146 pixel147 pixel148 pixel149 pixel150 pixel151
1        0        0        0        0        0        0        0        0        0        0        0      139      253
  pixel152 pixel153 pixel154 pixel155 pixel156 pixel157 pixel158 pixel159 pixel160 pixel161 pixel162 pixel163 pixel164
1      253      253      253      253      253      253       48        0        0        0        0        0        0
  pixel165 pixel166 pixel167 pixel168 pixel169 pixel170 pixel171 pixel172 pixel173 pixel174 pixel175 pixel176 pixel177
1        0        0        0        0        0        0        0        0        0        0        0        0       60
```

Figure 15: test dataset

Head(train)

```
> #ETD2
> head(train)
  label pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9 pixel10 pixel11 pixel12 pixel13 pixel14
1     1      0      0      0      0      0      0      0      0      0      0       0       0       0       0       0
  pixel15 pixel16 pixel17 pixel18 pixel19 pixel20 pixel21 pixel22 pixel23 pixel24 pixel25 pixel26 pixel27 pixel28
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel29 pixel30 pixel31 pixel32 pixel33 pixel34 pixel35 pixel36 pixel37 pixel38 pixel39 pixel40 pixel41 pixel42
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel43 pixel44 pixel45 pixel46 pixel47 pixel48 pixel49 pixel50 pixel51 pixel52 pixel53 pixel54 pixel55 pixel56
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel57 pixel58 pixel59 pixel60 pixel61 pixel62 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68 pixel69 pixel70
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel71 pixel72 pixel73 pixel74 pixel75 pixel76 pixel77 pixel78 pixel79 pixel80 pixel81 pixel82 pixel83 pixel84
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel85 pixel86 pixel87 pixel88 pixel89 pixel90 pixel91 pixel92 pixel93 pixel94 pixel95 pixel96 pixel97 pixel98
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel99 pixel100 pixel101 pixel102 pixel103 pixel104 pixel105 pixel106 pixel107 pixel108 pixel109 pixel110 pixel111
1       0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel112 pixel113 pixel114 pixel115 pixel116 pixel117 pixel118 pixel119 pixel120 pixel121 pixel122 pixel123 pixel124
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel125 pixel126 pixel127 pixel128 pixel129 pixel130 pixel131 pixel132 pixel133 pixel134 pixel135 pixel136 pixel137
1        0        0        0        0        0        0        0      188      255       94        0        0        0
  pixel138 pixel139 pixel140 pixel141 pixel142 pixel143 pixel144 pixel145 pixel146 pixel147 pixel148 pixel149 pixel150
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel151 pixel152 pixel153 pixel154 pixel155 pixel156 pixel157 pixel158 pixel159 pixel160 pixel161 pixel162 pixel163
1        0        0        0        0        0        0        0        0      191      250      253       93        0
  pixel164 pixel165 pixel166 pixel167 pixel168 pixel169 pixel170 pixel171 pixel172 pixel173 pixel174 pixel175 pixel176
1        0        0        0        0        0        0        0        0        0        0        0        0        0
```

Figure 16: head(train)

Str(train):

```
> str(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 ...
 $ pixel11 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel12 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel13 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel14 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel15 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel16 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel17 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel18 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel19 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel20 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel21 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel22 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel23 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel24 : int  0 0 0 0 0 0 0 0 0 0 ...
```

Figure 17: str(train)

Head(test)

```
> head(test)
  pixel0 pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9 pixel10 pixel11 pixel12 pixel13 pixel14 pixel15
1      0      0      0      0      0      0      0      0      0      0       0       0       0       0       0       0
  pixel16 pixel17 pixel18 pixel19 pixel20 pixel21 pixel22 pixel23 pixel24 pixel25 pixel26 pixel27 pixel28 pixel29
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel30 pixel31 pixel32 pixel33 pixel34 pixel35 pixel36 pixel37 pixel38 pixel39 pixel40 pixel41 pixel42 pixel43
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel44 pixel45 pixel46 pixel47 pixel48 pixel49 pixel50 pixel51 pixel52 pixel53 pixel54 pixel55 pixel56 pixel57
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel58 pixel59 pixel60 pixel61 pixel62 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68 pixel69 pixel70 pixel71
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel72 pixel73 pixel74 pixel75 pixel76 pixel77 pixel78 pixel79 pixel80 pixel81 pixel82 pixel83 pixel84 pixel85
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel86 pixel87 pixel88 pixel89 pixel90 pixel91 pixel92 pixel93 pixel94 pixel95 pixel96 pixel97 pixel98 pixel99
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel100 pixel101 pixel102 pixel103 pixel104 pixel105 pixel106 pixel107 pixel108 pixel109 pixel110 pixel111 pixel112
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel113 pixel114 pixel115 pixel116 pixel117 pixel118 pixel119 pixel120 pixel121 pixel122 pixel123 pixel124 pixel125
1        0        0        0        0        0        0        0        0        0       10       17       17       17
  pixel126 pixel127 pixel128 pixel129 pixel130 pixel131 pixel132 pixel133 pixel134 pixel135 pixel136 pixel137 pixel138
1       17       81      180      180       35        0        0        0        0        0        0        0        0
  pixel139 pixel140 pixel141 pixel142 pixel143 pixel144 pixel145 pixel146 pixel147 pixel148 pixel149 pixel150 pixel151
1        0        0        0        0        0        0        0        0        0        0        0      139      253
  pixel152 pixel153 pixel154 pixel155 pixel156 pixel157 pixel158 pixel159 pixel160 pixel161 pixel162 pixel163 pixel164
1      253      253      253      253      253      253       48        0        0        0        0        0        0
  pixel165 pixel166 pixel167 pixel168 pixel169 pixel170 pixel171 pixel172 pixel173 pixel174 pixel175 pixel176 pixel177
1        0        0        0        0        0        0        0        0        0        0        0        0       60
  pixel178 pixel179 pixel180 pixel181 pixel182 pixel183 pixel184 pixel185 pixel186 pixel187 pixel188 pixel189 pixel190
```

Figure 18: head(test)

```
> str(test)
'data.frame':   28000 obs. of  784 variables:
 $ 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 ...
 $ pixel11 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel12 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel13 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel14 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel15 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel16 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel17 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel18 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel19 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel20 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel21 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel22 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel23 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel24 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pixel25 : int  0 0 0 0 0 0 0 0 0 0 ...
```

Figure 19: str(test)

```
> #ETD3
> dim(train)
[1] 42000   785
> dim(test)
[1] 28000   784
```

Figure 20: dim(train) and din(test)

```
#Disterbution of values - train
hist(train$label, col='orange', breaks = seq(from=-0.5, to=9.5, by=1),
     main="Distribution of Values", xlab='Value')
```

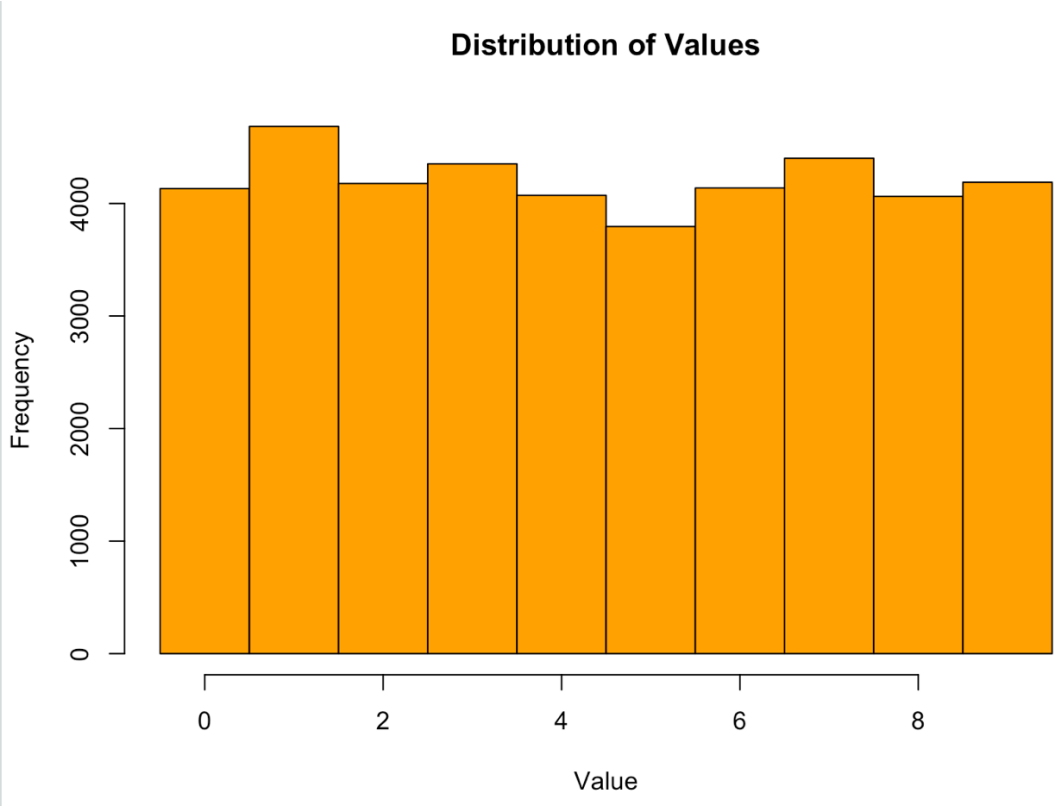


Figure 21: Plotting the matrix (train): distribution of values

```r
#Plotting the matrix
flip <- function(matrix){
  apply(matrix, 2, rev)
}
#random sampling
plotdigit<- function(datarow, rm1=F){
  # function will produce an image of the data given
  # the function takes away the first value because it is the target
  title<- datarow[1] # get actual value from training data
  if(rm1){
    datarow<- datarow[-1] # remove the actual value
  }
  datarow<- as.numeric(datarow) # convert to numeric
  x<- rep(0:27)/27
  y<- rep(0:27)/27
  z<- matrix(datarow, ncol=28, byrow=T)
  rotate <- function(x) t(apply(x, 2, rev))
  z<- rotate(z)
  image(x,y,z, main=paste("Actual Value:", title), col=gray.colors(255, start=1, end=0), asp=1,
        xlim=c(0,1), ylim=c(-0.1,1.1), useRaster = T, axes=F, xlab='', ylab='')
}

par(mfrow=c(3,4))
set.seed(1)
rows<- sample( 1:42000, size=12)
for(i in rows){
  plotdigit(train[i,],rm1=T)
}
```

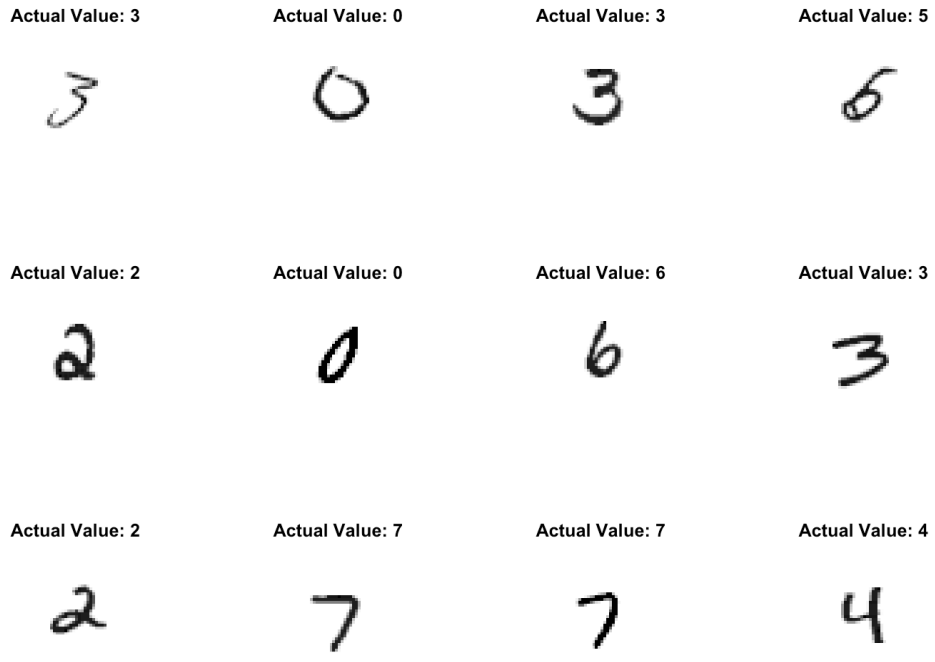


Figure 22: Plotting the matrix (train): random sampling

2.3 Data Transformation and Cleansing

2.3.1. Creating Samples

Creating samples for training and testing datasets as follows:

```
> #Creating samples
> train_sam <- train[seq(1, nrow(train), 10), ]
> test_sam <- test[seq(1, nrow(test), 10), ]
```

Figure 23: Creating Samples

2.3.2. Removing pixels that have 0 in all images

In this step, we are about to remove all pixels that have 0 in all images.

```
> #Removing pixels that have 0 in all images
> train_clean <- train_sam[, colSums(train_sam != 0) > 0]
```

Figure 24: Removing pixels that have 0 in all images

2.3.3. Removing pixels with low variance

Here we are removing pixels with low variance as follows:

```
> #Removing pixels with low variances
> all_var <- data.frame(apply(train_clean[-1], 2, var))
> colnames(all_var) <- "Variances"
```

Figure 25: Removing pixels with low variance

2.3.4. Sorting variance and creating number labels for variance Incl. summary of all variance

Sorting variance and creating number labels for variance:

```
> #Sorting variances
> all_var <- all_var[order(all_var$Variances), , drop = FALSE]
> #Creating number labels for variances
> num_labels <- c(1:661)
> numbered_var <- cbind(all_var,num_labels)
> summary(all_var)
   Variances
 Min.   :    0.001
 1st Qu.:  191.852
 Median : 3157.256
 Mean   : 5181.810
 3rd Qu.:11005.109
 Max.   :13102.379
```

Figure 26: Sorting variance and creating number labels for variance Incl. summary of all variance

2.3.5. Plotting all variances

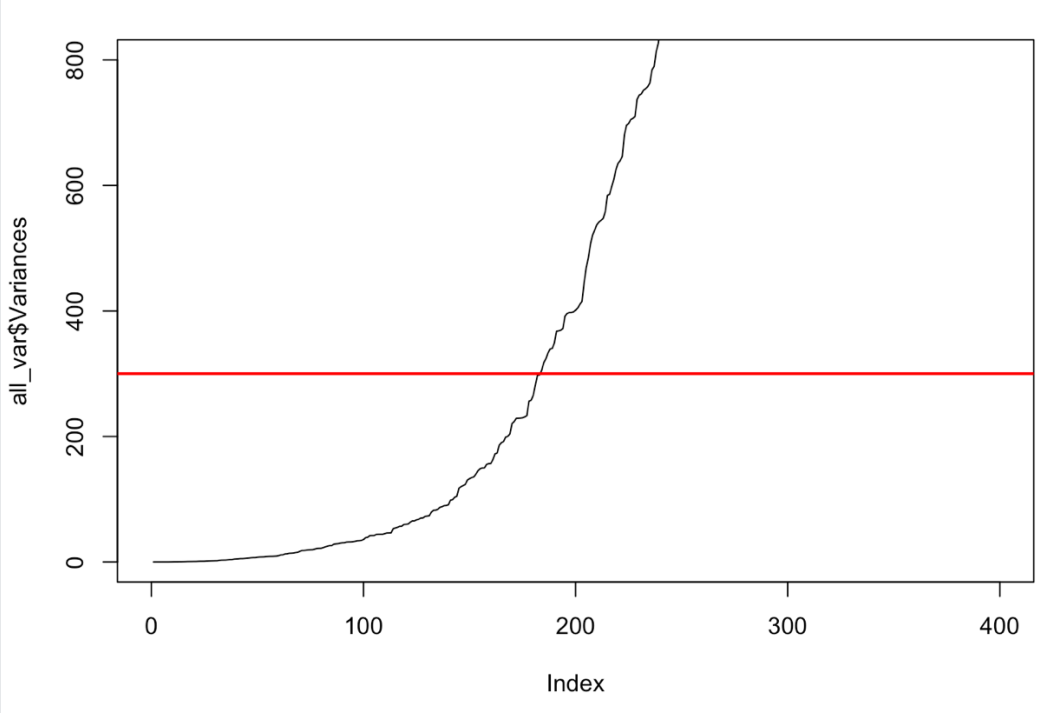


Figure 27: Plotting all variances

2.3.6. Creating clean train set with good variance

In this step, we are creating clean trainset with good variance:

```
> plot(all_var$Variances, type = "l",xlab="Pixel", ylab="Pixel variance", lwd=2)
> abline(h=5181, col="red", lwd=2)
> plot(all_var$Variances, type = "l", xlim = c(0,400), ylim=c(0,800))
> abline(h=300, col="red", lwd=2)
> good_var <- subset(all_var, all_var$Variances >= 300, "Variances")
> good_var_pixels <- row.names(good_var)
> train_clean <- train_clean[, c("label", good_var_pixels)]
```

Figure 28: Creating clean train set with good variance

2.3.7. Normalizing the data

Normalizing the data:

```
> #Normalizing the Data
> min_max_func <- function (x) {
+   a <- (x - min(x))
+   b <- (max(x) - min(x))
+   return(a / b)
+ }
>
> clean_train_nolabel <- train_clean[, -1]
> clean_train_nolabel_normalize <- as.data.frame(lapply(clean_train_nolabel, min_max_func))
> train_clean <- cbind(label = train_clean$label, clean_train_nolabel_normalize)
>
> head(clean.train)
  label pixel34 pixel35 pixel36 pixel37 pixel38 pixel39 pixel40 pixel41 pixel42 pixel43 pixel44 pixel45 pixel46 pixel47
1     1       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel48 pixel49 pixel61 pixel62 pixel63 pixel64 pixel65 pixel66 pixel67 pixel68 pixel69 pixel70 pixel71 pixel72
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel73 pixel74 pixel75 pixel76 pixel77 pixel78 pixel79 pixel89 pixel90 pixel91 pixel92 pixel93 pixel94 pixel95
1       0       0       0       0       0       0       0       0       0       0       0       0       0       0
  pixel96 pixel97 pixel98 pixel99 pixel100 pixel101 pixel102 pixel103 pixel104 pixel105 pixel106 pixel107 pixel108
1       0       0       0       0        0        0        0        0        0        0        0        0        0
  pixel109 pixel115 pixel116 pixel117 pixel118 pixel119 pixel120 pixel121 pixel122 pixel123 pixel124 pixel125 pixel126
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel127 pixel128 pixel129 pixel130 pixel131 pixel132 pixel133 pixel134 pixel135 pixel136 pixel137 pixel138 pixel142
1        0        0        0        0        0      188      255       94        0        0        0        0        0
  pixel143 pixel144 pixel145 pixel146 pixel147 pixel148 pixel149 pixel150 pixel151 pixel152 pixel153 pixel154 pixel155
1        0        0        0        0        0        0        0        0        0        0        0        0        0
  pixel156 pixel157 pixel158 pixel159 pixel160 pixel161 pixel162 pixel163 pixel164 pixel165 pixel166 pixel167 pixel170
1        0        0        0      191      250      253       93        0        0        0        0        0        0
```

Figure 29: Normalizing the data

2.3.8. Creating Test and Train sets for the Training Data Set

```
> #Creating Test and Train sets for the Training Data Set
> clean.train$label <- as.factor(train_clean$label)
> set.seed(123)
> idx <- sample(1:nrow(train_clean), size = 0.8 * nrow(train_clean))
> train_set <- train_clean[idx, ]
> test_set <- train_clean[ -idx, ]
```

Figure 30: Creating test and train sets for the training data set

2.4 kNN, Random Forest and SVM (Part II)

2.4.1 Building the kNN Model – Train and testing sets

Now, we are stating the build the different models and we will start with kNN as follows:

```
> #Building the KNN Model
> #Dividing test and train sets into attributes only and labels only
> train_set_numonly <- train_set[,-1]
> train_set_labels <- train_set[,1]
> test_set_numonly <- test_set[,-1]
> test_set_labels <- test_set[,1]
```

Figure 31: Building the kNN model

2.4.2 kNN - Choosing the K and Building the model

```
> #Choosing the K and building the model
> k <- round(sqrt(nrow(train_clean)))
>
> KNN_Model <- knn(train = train_set_numonly, test = test_set_numonly,
+                  cl= train_set_labels ,k = k, prob=TRUE)
>
```

Figure 32: Choosing the K and Building the Model

2.4.3 kNN - Confusion Matrix and Accuracy

The kNN model received 82.86%

```
> confusionMatrix(test_set_labels, KNN_Model)
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2  3  4  5  6  7  8  9
         0 69  0  0  0  0  3  3  0  0  0
         1  0 84  0  1  0  0  0  0  0  0
         2  3 12 62  3  1  0  0  5  1  0
         3  0  9  1 78  0  6  1  0  0  1
         4  0  6  0  0 80  0  1  3  1 13
         5  1  4  0  6  0 55  2  0  1  1
         6  0  4  0  0  1  4 81  0  0  0
         7  0  5  0  1  0  0  0 68  0  4
         8  0 12  0  4  1  3  0  1 55  1
         9  3  3  0  1  2  0  0  5  0 64

Overall Statistics

               Accuracy : 0.8286
                 95% CI : (0.8013, 0.8535)
    No Information Rate : 0.1655
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8093

 Mcnemar's Test P-Value : NA
```

Figure 32: kNN confusion matrix

```
Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity           0.90789   0.6043  0.98413  0.82979  0.94118  0.77465  0.92045  0.82927  0.94828  0.76190
Specificity           0.99215   0.9986  0.96782  0.97587  0.96821  0.98049  0.98803  0.98681  0.97187  0.98148
Pos Pred Value        0.92000   0.9882  0.71264  0.81250  0.76923  0.78571  0.90000  0.87179  0.71429  0.82051
Neg Pred Value        0.99085   0.9272  0.99867  0.97849  0.99321  0.97922  0.99067  0.98163  0.99607  0.97375
Prevalence            0.09048   0.1655  0.07500  0.11190  0.10119  0.08452  0.10476  0.09762  0.06905  0.10000
Detection Rate        0.08214   0.1000  0.07381  0.09286  0.09524  0.06548  0.09643  0.08095  0.06548  0.07619
Detection Prevalence  0.08929   0.1012  0.10357  0.11429  0.12381  0.08333  0.10714  0.09286  0.09167  0.09286
Balanced Accuracy     0.95002   0.8014  0.97598  0.90283  0.95469  0.87757  0.95424  0.90804  0.96007  0.87169
```

Figure 33: kNN statistics by Class

2.4.4 Random Forest – Building the Random Forest Model

Building the Random Forest Model:

```
> #Build the Random Fores Model
> RF_Model <- randomForest(label~., test_set)
> RF_Pred <- predict(RF_Model, test_set)
```

Figure 34: Building the Random Forest Model

2.4.5 Random Forest – Confusion Matrix and Accuracy

Random Forest received 99.56% accuracy.

```
> confusionMatrix(test_set$label, RF_Pred)
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0  75   0   0   0   0   0   0   0   0   0
         1   0  85   0   0   0   0   0   0   0   0
         2   0   0  87   0   0   0   0   0   0   0
         3   0   0   0  96   0   0   0   0   0   0
         4   0   0   0   0 104   0   0   0   0   0
         5   0   0   0   0   0  70   0   0   0   0
         6   0   0   0   0   0   0  90   0   0   0
         7   0   0   0   0   0   0   0  78   0   0
         8   0   0   0   0   0   0   0   0  77   0
         9   0   0   0   0   0   0   0   0   0  78

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9956, 1)
    No Information Rate : 0.1238
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```

Figure 34: Random Forest Confusion Matrix

Statistics by Class:

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Sensitivity	1.00000	1.0000	1.0000	1.0000	1.0000	1.00000	1.0000	1.00000	1.00000	1.00000
Specificity	1.00000	1.0000	1.0000	1.0000	1.0000	1.00000	1.0000	1.00000	1.00000	1.00000
Pos Pred Value	1.00000	1.0000	1.0000	1.0000	1.0000	1.00000	1.0000	1.00000	1.00000	1.00000
Neg Pred Value	1.00000	1.0000	1.0000	1.0000	1.0000	1.00000	1.0000	1.00000	1.00000	1.00000
Prevalence	0.08929	0.1012	0.1036	0.1143	0.1238	0.08333	0.1071	0.09286	0.09167	0.09286
Detection Rate	0.08929	0.1012	0.1036	0.1143	0.1238	0.08333	0.1071	0.09286	0.09167	0.09286
Detection Prevalence	0.08929	0.1012	0.1036	0.1143	0.1238	0.08333	0.1071	0.09286	0.09167	0.09286
Balanced Accuracy	1.00000	1.0000	1.0000	1.0000	1.0000	1.00000	1.0000	1.00000	1.00000	1.00000

Figure 35: RF statistics by Class

2.4.6 Plotting RF Model

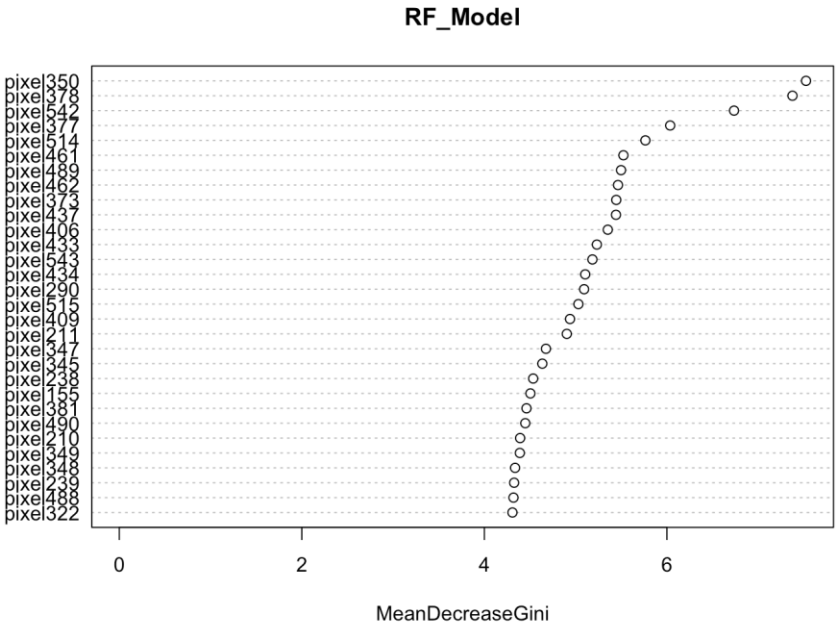


Figure 36: RF Model

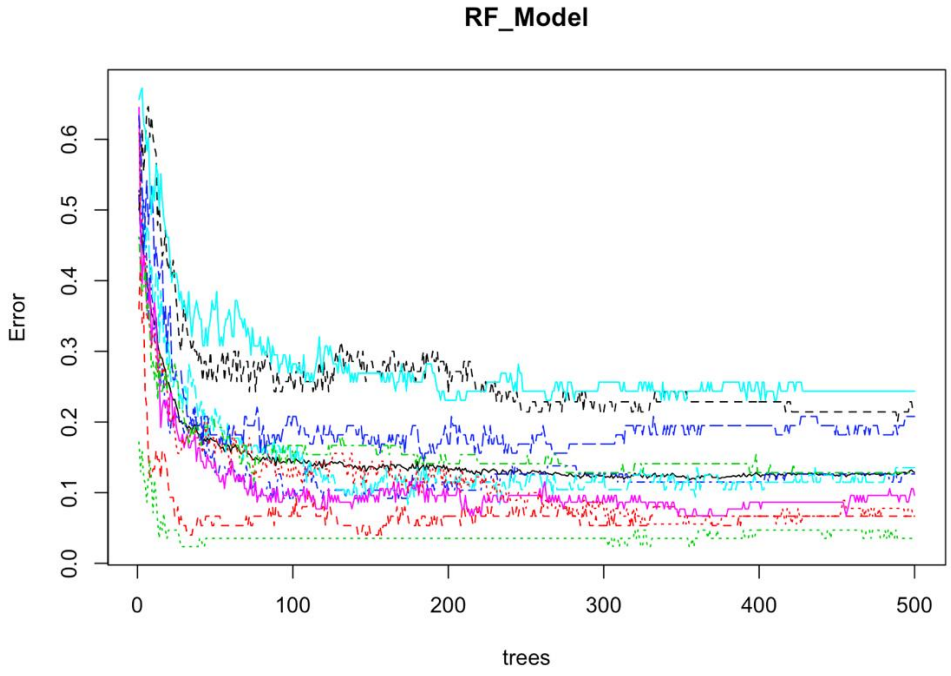


Figure 37: RF Model by trees and Errors

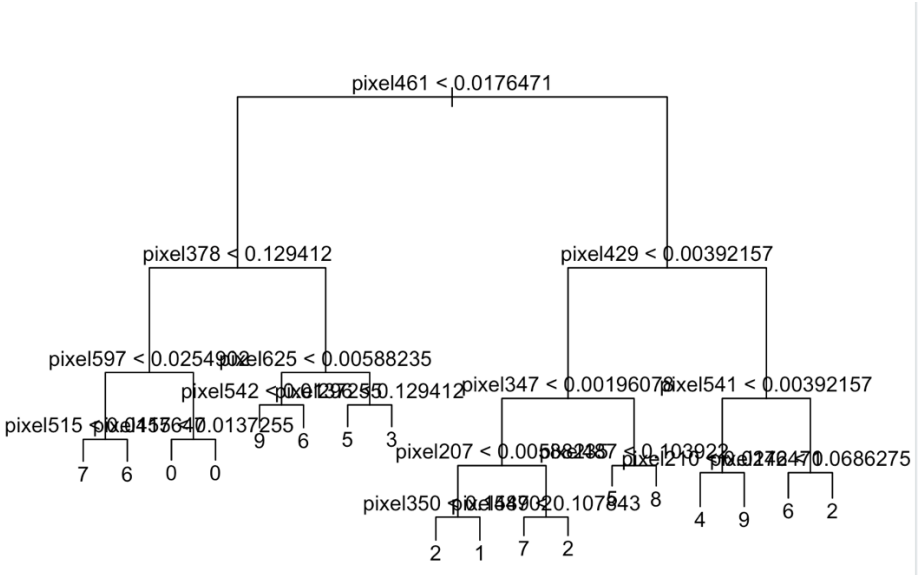


Figure 38: RF Model trees

2.4.7 SVM Model – Building the Model

Building the SVM Model:

```
> SVM <- svm(label~., data=train_set, kernel=KT_P, cost=100, scale=FALSE)
> print(SVM)

Call:
svm(formula = label ~ ., data = train_set, kernel = KT_P, cost = 100, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  100
     degree:  3
     coef.0:  0

Number of Support Vectors:  1757
```

Figure 39: Building the SVM Model

2.4.8 SVM Model – Predication

```
> #SVM Predication
> SVM_Pred <- predict(SVM, test_set_numonly , type="class")
```

Figure 40: SVM Predication

2.4.9 SVM Model – Confusion Matrix and Accuracy

Accuracy received: 88.45%

```
> confusionMatrix(test_set_labels, SVM_Pred)
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2  3  4  5  6  7  8  9
         0 69  0  0  0  0  5  1  0  0  0
         1  0 84  0  1  0  0  0  0  0  0
         2  2  6 74  0  0  1  0  3  0  1
         3  0  6  1 81  1  5  0  0  1  1
         4  0  7  0  0 91  0  1  1  0  4
         5  0  1  0  3  0 63  2  0  1  0
         6  0  3  0  0  2  3 81  0  1  0
         7  0  4  2  0  1  0  0 68  0  3
         8  0  5  0  0  1  3  0  0 67  1
         9  1  2  1  1  4  1  0  3  0 65

Overall Statistics

               Accuracy : 0.8845
                 95% CI : (0.861, 0.9054)
    No Information Rate : 0.1405
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8715

 Mcnemar's Test P-Value : NA
```

Figure 41: SVM Confusion Matrix

```
Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity           0.95833   0.7119  0.94872  0.94186   0.9100  0.77778  0.95294  0.90667  0.95714  0.86667
Specificity           0.99219   0.9986  0.98294  0.98011   0.9824  0.99078  0.98808  0.98693  0.98701  0.98301
Pos Pred Value        0.92000   0.9882  0.85057  0.84375   0.8750  0.90000  0.90000  0.87179  0.87013  0.83333
Neg Pred Value        0.99608   0.9550  0.99469  0.99328   0.9878  0.97662  0.99467  0.99081  0.99607  0.98688
Prevalence            0.08571   0.1405  0.09286  0.10238   0.1190  0.09643  0.10119  0.08929  0.08333  0.08929
Detection Rate        0.08214   0.1000  0.08810  0.09643   0.1083  0.07500  0.09643  0.08095  0.07976  0.07738
Detection Prevalence  0.08929   0.1012  0.10357  0.11429   0.1238  0.08333  0.10714  0.09286  0.09167  0.09286
Balanced Accuracy     0.97526   0.8552  0.96583  0.96098   0.9462  0.88428  0.97051  0.94680  0.97208  0.92484
```

Figure 42: SVM Statistics by Class

2.4.10 Naïve Bayes – Building the Model

Building the Naïve Bayes model:

```
#Naive Bayes
NB <- naiveBayes(label~., data=train_set, kernel=KT_P, cost=100, scale=FALSE)
print(NB)
```

```
Y       [,1]     [,2]
 0 0.11892044 0.2817960
 1 0.64252701 0.4000161
 2 0.66419118 0.4039180
 3 0.08823529 0.2491862
 4 0.46038312 0.4208039
 5 0.21024668 0.3603020
 6 0.49451754 0.4330077
 7 0.60633134 0.4152582
 8 0.37776212 0.4162964
 9 0.37196805 0.4119040

  pixel352
Y       [,1]     [,2]
 0 0.04384719 0.1616616
 1 0.52769108 0.4070160
 2 0.45518382 0.4376923
 3 0.70401494 0.3846749
 4 0.47412556 0.4211176
 5 0.27997470 0.3998559
 6 0.29815531 0.3997760
 7 0.42763285 0.4199880
 8 0.65270635 0.3903785
 9 0.58194142 0.4121841

  pixel210
Y       [,1]     [,2]
 0 0.7275073 0.3694106
 1 0.3905462 0.4383860
 2 0.4683333 0.4380928
 3 0.6055322 0.4147567
 4 0.1564220 0.3186634
 5 0.5986211 0.4280781
 6 0.4657379 0.4330869
 7 0.4535038 0.4148618
 8 0.6039216 0.3997403
 9 0.7441298 0.3237961

  pixel521
```

```
Y       [,1]     [,2]
 0 0.45165652 0.4404825
 1 0.03355342 0.1550239
 2 0.55224265 0.4323362
 3 0.44283380 0.4336549
 4 0.43068317 0.4249618
 5 0.43638204 0.4364089
 6 0.60900413 0.4083187
 7 0.36289855 0.4125371
 8 0.40328754 0.4378859
 9 0.49450496 0.4314044

  pixel408
Y       [,1]      [,2]
 0 0.01836827 0.09960667
 1 0.26699680 0.34012204
 2 0.58264706 0.42551753
 3 0.59595005 0.41908367
 4 0.76167712 0.34854918
 5 0.44412397 0.43370559
 6 0.61333849 0.41385313
 7 0.63893151 0.40980911
 8 0.69311964 0.37872049
 9 0.79011135 0.32219749

  pixel380
Y       [,1]     [,2]
 0 0.02174893 0.1189441
 1 0.41546619 0.3834960
 2 0.52295343 0.4328291
 3 0.69942810 0.3939619
 4 0.61930271 0.4135442
 5 0.38885515 0.4251007
 6 0.49218266 0.4399053
 7 0.52393294 0.4315859
 8 0.72485617 0.3571277
 9 0.72409828 0.3747768

  pixel657
Y       [,1]     [,2]
 0 0.55670498 0.4093116
 1 0.30038015 0.3911367
 2 0.10120098 0.2635586
 3 0.67843137 0.3929199
 4 0.25350059 0.3706419
 5 0.56455408 0.4407969
 6 0.01324819 0.1020448
 7 0.46878090 0.4410443
 8 0.72877774 0.3515610
 9 0.32833454 0.4056471
```

pixel240

Y	[,1]	[,2]
0	0.5771242	0.4228047
1	0.5745698	0.4272598
2	0.4384926	0.4316941
3	0.5697362	0.4131880
4	0.2163870	0.3614388
5	0.4383428	0.4258330
6	0.2024123	0.3557413
7	0.7097926	0.3751072
8	0.3762710	0.4095520
9	0.6801017	0.3708195

pixel598

Y	[,1]	[,2]
0	0.77740590	0.3327586
1	0.34742897	0.4337444
2	0.64814951	0.4176493
3	0.44953315	0.4302004
4	0.07802452	0.2399204
5	0.50943707	0.4264198
6	0.42567079	0.4077993
7	0.16858198	0.3367288
8	0.55000587	0.4057598
9	0.06872428	0.2252449

pixel545

Y	[,1]	[,2]
0	0.21548343	0.3567772
1	0.67551020	0.3891298
2	0.67495098	0.3948082
3	0.09597339	0.2537450
4	0.29744024	0.4002326
5	0.20631246	0.3587605
6	0.69484004	0.3926409
7	0.46741688	0.4348248
8	0.30942820	0.3965972
9	0.24065602	0.3809296

pixel213

Y	[,1]	[,2]
0	0.7180077	0.3724686
1	0.4491697	0.4416003
2	0.5598775	0.4203353
3	0.6160714	0.4161937
4	0.3065265	0.4057293
5	0.5344972	0.4316871
6	0.1653380	0.3281919
7	0.4202444	0.4230280

```
  8 0.5955266 0.4054251
  9 0.5985355 0.4112060

   pixel520
Y       [,1]     [,2]
  0 0.3108519 0.4118483
  1 0.1104242 0.2653328
  2 0.6085294 0.4152034
  3 0.3012722 0.3973312
  4 0.5860315 0.4258282
  5 0.3817963 0.4203928
  6 0.5231037 0.4236679
  7 0.5914635 0.4128268
  8 0.4410825 0.4220692
  9 0.6237110 0.4020479

   pixel351
Y       [,1]     [,2]
  0 0.04894073 0.1844142
  1 0.86352541 0.2590397
  2 0.32840686 0.4148788
  3 0.75246265 0.3486674
  4 0.23950952 0.3567238
  5 0.34161923 0.4136230
  6 0.23264964 0.3588468
  7 0.19616937 0.3356522
  8 0.57443936 0.4155485
  9 0.37636166 0.4128662

   pixel235
Y       [,1]     [,2]
  0 0.59637142 0.4217146
  1 0.03103241 0.1420803
  2 0.31609069 0.4172427
  3 0.34126984 0.4039118
  4 0.33259875 0.4004994
  5 0.45340923 0.4293379
  6 0.41429309 0.4353753
  7 0.67006536 0.3987671
  8 0.58088529 0.4177791
  9 0.67006778 0.3793795

   pixel548
Y       [,1]     [,2]
  0 0.4880550 0.4421342
  1 0.1214886 0.2820120
  2 0.5773652 0.4259893
  3 0.4209734 0.4317038
  4 0.4761485 0.4329153
  5 0.4258318 0.4249483
```

6 0.7046053 0.3784591
7 0.4572890 0.4265720
8 0.4463544 0.4211924
9 0.5424837 0.4204101

pixel577

Y	[,1]	[,2]
0	0.67134325	0.3921455
1	0.04712885	0.1859942
2	0.47622549	0.4352034
3	0.62226891	0.3988556
4	0.29239984	0.3991912
5	0.49373814	0.4399908
6	0.65158669	0.3921229
7	0.16736573	0.3256596
8	0.44570858	0.4330508
9	0.35048414	0.4124193

pixel549

Y	[,1]	[,2]
0	0.60215236	0.4217234
1	0.03778511	0.1562665
2	0.54561275	0.4201442
3	0.52783613	0.4199748
4	0.34697406	0.4223525
5	0.46040481	0.4401412
6	0.72107843	0.3632188
7	0.24180733	0.3622213
8	0.42507925	0.4365448
9	0.41909949	0.4262781

pixel291

Y	[,1]	[,2]
0	0.65105927	0.4090329
1	0.02185874	0.1216729
2	0.11976716	0.2865726
3	0.14124650	0.2955524
4	0.44989546	0.4290367
5	0.55528147	0.4219112
6	0.52699948	0.4320706
7	0.52564933	0.4340002
8	0.56498767	0.4191112
9	0.58270395	0.4145422

pixel604

Y	[,1]	[,2]
0	0.6822853	0.3786055
1	0.1500200	0.3113679
2	0.3378554	0.4237009
3	0.6808007	0.3777618

4 0.3439905 0.4208890
5 0.5632764 0.4303668
6 0.5392157 0.4016889
7 0.2595851 0.3863432
8 0.5168369 0.4278655
9 0.4001573 0.4280979

pixel599

Y	[,1]	[,2]
0	0.7260536	0.3688003
1	0.4666867	0.4393243
2	0.6100613	0.4207528
3	0.4520075	0.4257325
4	0.1461378	0.3134431
5	0.5205566	0.4265433
6	0.5863132	0.3981621
7	0.2845240	0.3991354
8	0.5274862	0.4188696
9	0.1377875	0.3095805

pixel576

Y	[,1]	[,2]
0	0.6441740	0.4040629
1	0.1344038	0.2954014
2	0.4867525	0.4363631
3	0.5768207	0.4170223
4	0.3919308	0.4180672
5	0.4918659	0.4349062
6	0.7674923	0.3237915
7	0.3389031	0.4075167
8	0.4760127	0.4276318
9	0.4612443	0.4338905

pixel574

Y	[,1]	[,2]
0	0.4900270	0.4392895
1	0.4696979	0.4404518
2	0.5344730	0.4310737
3	0.3503852	0.4139130
4	0.4087359	0.4220349
5	0.3936496	0.4174786
6	0.8403767	0.2870555
7	0.6296903	0.4142169
8	0.3616414	0.4080600
9	0.4345195	0.4344534

pixel489

Y	[,1]	[,2]
0	0.02423935	0.1156079
1	0.84108643	0.2737123

2 0.69712010 0.3882885
3 0.07658730 0.2357523
4 0.42562016 0.4194603
5 0.14031626 0.3007461
6 0.52444530 0.4223542
7 0.28519466 0.3944461
8 0.56894446 0.4295545
9 0.31637618 0.3981589

pixel465

Y	[,1]	[,2]
0	0.17971602	0.33252215
1	0.01592637	0.08304341
2	0.47763480	0.44436504
3	0.47450980	0.42632910
4	0.73372888	0.35879795
5	0.41934219	0.42584039
6	0.42403251	0.42354535
7	0.62250639	0.40904562
8	0.38971469	0.41582759
9	0.70525297	0.37141553

pixel354

Y	[,1]	[,2]
0	0.18181204	0.3346291
1	0.05211084	0.1958246
2	0.48268382	0.4322539
3	0.36539449	0.4149537
4	0.64694581	0.3972606
5	0.15311828	0.3126744
6	0.33426213	0.4147653
7	0.68475135	0.3788130
8	0.54271457	0.4239719
9	0.69885016	0.3875648

pixel241

Y	[,1]	[,2]
0	0.58777327	0.4251416
1	0.44341737	0.4394661
2	0.51096814	0.4299318
3	0.58552754	0.4280611
4	0.35825281	0.4215903
5	0.41757116	0.4320697
6	0.09183437	0.2427976
7	0.68613811	0.3883681
8	0.45436186	0.4202763
9	0.63298233	0.3947220

pixel214

Y	[,1]	[,2]

0 0.72928781 0.3759077
1 0.32024810 0.4229124
2 0.52074755 0.4365497
3 0.49543651 0.4346628
4 0.37193875 0.4295948
5 0.47547122 0.4310502
6 0.08560372 0.2371624
7 0.35760159 0.4099512
8 0.62837854 0.3964049
9 0.43248608 0.4206628

pixel353
Y [,1] [,2]
0 0.07195177 0.2129966
1 0.16953782 0.3018265
2 0.53910539 0.4226642
3 0.56245331 0.4113599
4 0.65959202 0.3890470
5 0.22308665 0.3723910
6 0.33526832 0.4203950
7 0.63627167 0.4020508
8 0.64636609 0.3900742
9 0.73671024 0.3588891

pixel290
Y [,1] [,2]
0 0.647960334 0.40229301
1 0.007833133 0.08284056
2 0.136164216 0.30358546
3 0.104586835 0.26120967
4 0.483144036 0.42293773
5 0.521366224 0.42488620
6 0.455392157 0.43237250
7 0.541847116 0.43228776
8 0.543313373 0.42440731
9 0.678915517 0.37423906

pixel264
Y [,1] [,2]
0 0.6604012 0.4135513
1 0.1094038 0.2768623
2 0.2044240 0.3552141
3 0.1818394 0.3283483
4 0.3568740 0.4222409
5 0.5258697 0.4344890
6 0.5094298 0.4335451
7 0.7045524 0.3744396
8 0.5037807 0.4239564
9 0.6348100 0.3885224

pixel572

Y	[,1]	[,2]
0	0.4228307	0.4176163
1	0.5736695	0.4234653
2	0.6587623	0.3951862
3	0.1982376	0.3349886
4	0.2029610	0.3594468
5	0.3047944	0.3997454
6	0.8438080	0.2912909
7	0.3677067	0.4264788
8	0.3774921	0.4188011
9	0.1721859	0.3422132

pixel626

Y	[,1]	[,2]
0	0.77053189	0.3341296
1	0.35169068	0.4329239
2	0.45269608	0.4360547
3	0.63879552	0.4051916
4	0.09895463	0.2637450
5	0.56075901	0.4161626
6	0.10819143	0.2675506
7	0.23336175	0.3829288
8	0.58852882	0.4163461
9	0.11778020	0.2878692

pixel239

Y	[,1]	[,2]
0	0.5809331	0.4274428
1	0.6015906	0.4209028
2	0.3775245	0.4179784
3	0.4794935	0.4163996
4	0.1430525	0.3056457
5	0.4593548	0.4314347
6	0.3153380	0.4139308
7	0.7268542	0.3729720
8	0.3787719	0.4198304
9	0.6864803	0.3797023

pixel603

Y	[,1]	[,2]
0	0.7181542	0.3702359
1	0.2786615	0.4119792
2	0.3520466	0.4221484
3	0.6606209	0.3898614
4	0.3799966	0.4282088
5	0.5586464	0.4263935
6	0.6721104	0.3731844
7	0.4343620	0.4319548
8	0.5217330	0.4255577

9 0.4486807 0.4409716

pixel573

Y	[,1]	[,2]
0	0.4353505	0.4349335
1	0.5914866	0.4178415
2	0.6021936	0.4184444
3	0.2488912	0.3641740
4	0.3206871	0.4072765
5	0.3330803	0.4047520
6	0.8377580	0.2953651
7	0.5279113	0.4349359
8	0.3210872	0.3933801
9	0.2788429	0.4046373

pixel656

Y	[,1]	[,2]
0	0.59861393	0.4056733
1	0.35670268	0.4138773
2	0.13099265	0.2929504
3	0.69216853	0.3753573
4	0.21392326	0.3667675
5	0.57620493	0.4288355
6	0.01348039	0.1080395
7	0.47065644	0.4390745
8	0.72902430	0.3445163
9	0.27961753	0.4091561

pixel547

Y	[,1]	[,2]
0	0.3646383	0.4269047
1	0.3030212	0.4131211
2	0.6026961	0.4183638
3	0.2728058	0.3856663
4	0.5075210	0.4373287
5	0.3437571	0.3980416
6	0.6708075	0.3949051
7	0.6452174	0.4146176
8	0.3934367	0.4288257
9	0.5449770	0.4269111

pixel546

Y	[,1]	[,2]
0	0.2686049	0.3910686
1	0.5440676	0.4260860
2	0.6386887	0.4146480
3	0.1547502	0.3107776
4	0.4448890	0.4319171
5	0.2594687	0.3925925
6	0.6615712	0.3963387

```
  7 0.6409321 0.4075362
  8 0.3221087 0.4085676
  9 0.4065359 0.4298948

   pixel270
Y       [,1]     [,2]
  0 0.49043272 0.4320667
  1 0.23661465 0.3776119
  2 0.50980392 0.4247456
  3 0.48125584 0.4311544
  4 0.50157654 0.4298861
  5 0.24651486 0.3830161
  6 0.03583591 0.1607071
  7 0.74641660 0.3707887
  8 0.53457790 0.4152490
  9 0.55313483 0.4134075

   pixel381
Y       [,1]     [,2]
  0 0.05353843 0.1935037
  1 0.09497799 0.2341751
  2 0.55263480 0.4205599
  3 0.59815593 0.4083612
  4 0.73277957 0.3482985
  5 0.32475648 0.4086179
  6 0.52573529 0.4277660
  7 0.68707019 0.3876696
  8 0.58849360 0.4062141
  9 0.80359477 0.3199293

   pixel631
Y       [,1]     [,2]
  0 0.6319135 0.3890084
  1 0.2606543 0.4008943
  2 0.1936887 0.3487374
  3 0.6940243 0.3799641
  4 0.3391422 0.4209874
  5 0.5956736 0.4239436
  6 0.2108359 0.3492185
  7 0.3613413 0.4137631
  8 0.5936245 0.4168433
  9 0.4025781 0.4430821

   pixel263
Y       [,1]     [,2]
  0 0.66915709 0.3987229
  1 0.02810124 0.1410171
  2 0.21671569 0.3733765
  3 0.16642157 0.3184569
  4 0.40683732 0.4085814
```

5 0.51614168 0.4342895
6 0.47494840 0.4304935
7 0.70567775 0.3696661
8 0.57891276 0.4184796
9 0.71526265 0.3495803

pixel382

Y	[,1]	[,2]
0	0.1628240	0.3245262
1	0.0207583	0.1111265
2	0.4662255	0.4389090
3	0.4310341	0.4257303
4	0.6592756	0.4011239
5	0.2377483	0.3712757
6	0.5233101	0.4331355
7	0.6684513	0.3895512
8	0.4055066	0.4125426
9	0.6937182	0.3949751

pixel236

Y	[,1]	[,2]
0	0.6800992	0.3959198
1	0.1089536	0.2792813
2	0.3131373	0.4164193
3	0.3273693	0.3997144
4	0.2968639	0.3927035
5	0.5138899	0.4310883
6	0.4856553	0.4347223
7	0.6991645	0.3815343
8	0.5841376	0.4140828
9	0.7626604	0.3222479

pixel463

Y	[,1]	[,2]
0	0.03477575	0.1513037
1	0.51926771	0.3888435
2	0.64340686	0.4167415
3	0.24169001	0.3724342
4	0.79401028	0.3166405
5	0.32417457	0.4033772
6	0.51096491	0.4292391
7	0.64553566	0.4121011
8	0.61280967	0.4139600
9	0.62952070	0.4059175

pixel519

Y	[,1]	[,2]
0	0.1846405	0.3410472
1	0.3363846	0.4134859
2	0.6501716	0.4133989

3 0.1636788 0.3249215
4 0.6003051 0.4157899
5 0.3004048 0.4108068
6 0.4768189 0.4248877
7 0.7084854 0.3919810
8 0.4221322 0.4235146
9 0.5681554 0.4205545

pixel575

Y	[,1]	[,2]
0	0.5772143	0.4303806
1	0.2872949	0.4183411
2	0.5012623	0.4355866
3	0.4658263	0.4451923
4	0.4386958	0.4399147
5	0.4633650	0.4323179
6	0.8356424	0.2789395
7	0.5438250	0.4234693
8	0.4506047	0.4326909
9	0.5005810	0.4304774

pixel629

Y	[,1]	[,2]
0	0.7690331	0.3449407
1	0.4154162	0.4274387
2	0.2865809	0.3999310
3	0.7290850	0.3645984
4	0.2977793	0.3962586
5	0.6573308	0.3997621
6	0.2726909	0.3890080
7	0.5232396	0.4288137
8	0.6863215	0.3728091
9	0.3381748	0.4155650

pixel181

Y	[,1]	[,2]
0	0.6074037	0.4083677
1	0.2176771	0.3710131
2	0.5919240	0.4234807
3	0.7408847	0.3656070
4	0.1272193	0.2850643
5	0.4713725	0.4328166
6	0.4450464	0.4407826
7	0.1263200	0.2888853
8	0.6538218	0.4089072
9	0.2863229	0.3732097

pixel492

Y	[,1]	[,2]
0	0.1603448	0.3201406

1 0.1084634 0.2538034
2 0.5828309 0.4432236
3 0.2542017 0.3720104
4 0.7373227 0.3588288
5 0.3566856 0.4179339
6 0.4127709 0.4254329
7 0.7055527 0.3732081
8 0.4561465 0.4292955
9 0.7048656 0.3723897

pixel238

Y	[,1]	[,2]
0	0.6382466	0.4139296
1	0.4441677	0.4378873
2	0.3208701	0.4072276
3	0.3901027	0.4167842
4	0.1701531	0.3338896
5	0.5152435	0.4296424
6	0.4307147	0.4328271
7	0.7289798	0.3731411
8	0.4127627	0.4236859
9	0.7191237	0.3750278

pixel242

Y	[,1]	[,2]
0	0.63505747	0.4048933
1	0.28866547	0.4104528
2	0.51797794	0.4245751
3	0.50435341	0.4337948
4	0.44213143	0.4294940
5	0.37781151	0.4271598
6	0.03973168	0.1754769
7	0.63205456	0.4211641
8	0.55636961	0.4213677
9	0.54163641	0.4148043

pixel410

Y	[,1]	[,2]
0	0.184099617	0.34830327
1	0.003841537	0.03277532
2	0.421789216	0.44269589
3	0.531594304	0.43385938
4	0.681968695	0.39170997
5	0.344313725	0.40968952
6	0.607636739	0.42930889
7	0.614992896	0.40424707
8	0.279523306	0.38352745
9	0.634507383	0.40316794

pixel405

Y [,1] [,2]
0 0.03887762 0.1624713
1 0.75570228 0.2961414
2 0.43712010 0.4372352
3 0.61663165 0.4184568
4 0.43761089 0.4265884
5 0.54622391 0.4247115
6 0.41124871 0.4303518
7 0.05150327 0.1923068
8 0.82207350 0.2907942
9 0.50273542 0.4210004

pixel436

Y [,1] [,2]
0 0.03619563 0.1561145
1 0.17310924 0.2954536
2 0.57675245 0.4270806
3 0.45383987 0.4216397
4 0.85506018 0.2736605
5 0.42575585 0.4285088
6 0.58190144 0.4192179
7 0.71594203 0.3813745
8 0.59958906 0.4094149
9 0.81072380 0.3059351

pixel262

Y [,1] [,2]
0 0.578183457 0.43148861
1 0.007452981 0.07561024
2 0.218468137 0.37537758
3 0.174556489 0.33016817
4 0.406995536 0.42533704
5 0.420860215 0.43168945
6 0.383501032 0.43318211
7 0.691628303 0.38464406
8 0.547164495 0.43226830
9 0.674642944 0.38364381

pixel600

Y [,1] [,2]
0 0.6932049 0.3884136
1 0.5263806 0.4418898
2 0.5547304 0.4363579
3 0.4688492 0.4288470
4 0.2389105 0.3859758
5 0.5212524 0.4221980
6 0.7099458 0.3613336
7 0.4377380 0.4403748
8 0.4805448 0.4223848
9 0.2175986 0.3788202

pixel491

Y	[,1]	[,2]
0	0.08545188	0.2512403
1	0.39619848	0.4039495
2	0.63578431	0.4307536
3	0.15093371	0.3195316
4	0.70472962	0.3758161
5	0.29566097	0.4065352
6	0.44317595	0.4372914
7	0.71554419	0.3837673
8	0.49935423	0.4274057
9	0.59870491	0.4019393

pixel185

Y	[,1]	[,2]
0	0.7307190	0.3790708
1	0.4265506	0.4387336
2	0.5831985	0.4230654
3	0.5786998	0.4224020
4	0.2314517	0.3607693
5	0.5505123	0.4366326
6	0.2579463	0.3954999
7	0.1056323	0.2656544
8	0.6950100	0.3808237
9	0.2960421	0.3871848

pixel184

Y	[,1]	[,2]
0	0.7412779	0.3702903
1	0.4484694	0.4194516
2	0.6282721	0.4138742
3	0.6935691	0.3830092
4	0.1465672	0.3049873
5	0.5473498	0.4282771
6	0.3563983	0.4203427
7	0.1135436	0.2783884
8	0.7027709	0.3704955
9	0.3855483	0.4150891

pixel434

Y	[,1]	[,2]
0	0.01132522	0.0903629
1	0.94181673	0.1511231
2	0.60549020	0.4072323
3	0.39181839	0.4271198
4	0.68498616	0.3673251
5	0.38963947	0.4161870
6	0.54887771	0.4164002
7	0.24953680	0.3657680

```
  8 0.80852413 0.3272383
  9 0.56664246 0.4122804

   pixel237
Y       [,1]     [,2]
  0 0.6836939 0.4173562
  1 0.2545218 0.3923330
  2 0.3030515 0.4064369
  3 0.3316643 0.3996561
  4 0.2503136 0.3859131
  5 0.5511828 0.4328247
  6 0.4847781 0.4383825
  7 0.7242626 0.3747599
  8 0.4983680 0.4282820
  9 0.7556161 0.3472707

   pixel379
Y       [,1]     [,2]
  0 0.02287582 0.1153121
  1 0.83817527 0.2826281
  2 0.42866422 0.4365736
  3 0.74347572 0.3606456
  4 0.42434311 0.4235746
  5 0.44475648 0.4356108
  6 0.42720588 0.4409531
  7 0.26331344 0.3703568
  8 0.75480803 0.3435249
  9 0.55513193 0.4218778

   pixel490
Y       [,1]     [,2]
  0 0.04730674 0.1836920
  1 0.77102841 0.3294863
  2 0.67658088 0.4049727
  3 0.10331466 0.2730029
  4 0.54506414 0.4147833
  5 0.21407970 0.3635511
  6 0.50379257 0.4396484
  7 0.53066212 0.4376529
  8 0.51010919 0.4335666
  9 0.40106512 0.4169287

   pixel464
Y       [,1]     [,2]
  0 0.07332657 0.2267566
  1 0.12847139 0.2667538
  2 0.57506127 0.4365300
  3 0.32467320 0.4035898
  4 0.85033622 0.2720756
  5 0.36638836 0.4164202
```

6 0.44681373 0.4248790
7 0.74675760 0.3574184
8 0.51020312 0.4270318
9 0.77828613 0.3291133

pixel628

Y	[,1]	[,2]
0	0.7934753	0.3310545
1	0.4681172	0.4381247
2	0.3548162	0.4167384
3	0.7114963	0.3765582
4	0.2412725	0.3819065
5	0.6578115	0.3933380
6	0.2405315	0.3744614
7	0.4679170	0.4456324
8	0.6746859	0.3821400
9	0.2608085	0.3958503

pixel182

Y	[,1]	[,2]
0	0.6982984	0.3877911
1	0.3562325	0.4324736
2	0.5940931	0.4167910
3	0.7550070	0.3590110
4	0.1069447	0.2681972
5	0.5236306	0.4306545
6	0.4811920	0.4379552
7	0.1240011	0.2818570
8	0.7001292	0.3757563
9	0.3743040	0.4050055

pixel462

Y	[,1]	[,2]
0	0.01126888	0.08201629
1	0.88210284	0.22988803
2	0.66285539	0.39822975
3	0.20151727	0.35046206
4	0.69626490	0.37482770
5	0.26519924	0.38706399
6	0.54340815	0.42159901
7	0.39905655	0.41661546
8	0.68884584	0.39244583
9	0.48220770	0.41534123

pixel630

Y	[,1]	[,2]
0	0.7324544	0.3580115
1	0.3543117	0.4293975
2	0.2323284	0.3724063
3	0.7280345	0.3689790

4 0.3355032 0.4240802
5 0.6340923 0.4147405
6 0.2590041 0.3832723
7 0.4943905 0.4458634
8 0.6651168 0.3773580
9 0.4066933 0.4356269

pixel409

Y	[,1]	[,2]
0	0.0654947	0.2133368
1	0.0444978	0.1522589
2	0.5286642	0.4256122
3	0.5966737	0.4123960
4	0.7957846	0.3161245
5	0.4100822	0.4304948
6	0.6336558	0.4130207
7	0.7231373	0.3687525
8	0.4910649	0.4188457
9	0.8275720	0.2909963

pixel183

Y	[,1]	[,2]
0	0.7472053	0.3622981
1	0.4455782	0.4375083
2	0.6103309	0.4168870
3	0.7401961	0.3623875
4	0.1021981	0.2672981
5	0.5428210	0.4275316
6	0.4387126	0.4355349
7	0.1200227	0.2862005
8	0.7072913	0.3598166
9	0.4138949	0.4173733

pixel627

Y	[,1]	[,2]
0	0.8045526	0.3125679
1	0.4286114	0.4393696
2	0.4173897	0.4329643
3	0.6814309	0.3950009
4	0.1801548	0.3484989
5	0.6208476	0.4050268
6	0.1671956	0.3227020
7	0.3602501	0.4305988
8	0.6484913	0.3827332
9	0.1866739	0.3567731

pixel437

Y	[,1]	[,2]
0	0.10352716	0.2653850
1	0.01997799	0.0927278

2 0.49003676 0.4417366
3 0.53790850 0.4167777
4 0.81729107 0.3030684
5 0.43396584 0.4365158
6 0.56247420 0.4217240
7 0.71363456 0.3734449
8 0.41124809 0.4261904
9 0.79316146 0.3173964

pixel461

Y	[,1]	[,2]
0	0.008260086	0.06052797
1	0.853961585	0.26822375
2	0.671409314	0.39336313
3	0.179306723	0.32809786
4	0.647465672	0.39646238
5	0.228298545	0.36790825
6	0.516937564	0.42608080
7	0.177220801	0.33578971
8	0.715979805	0.37443731
9	0.443064633	0.41665357

pixel433

Y	[,1]	[,2]
0	0.01831192	0.1051047
1	0.83209284	0.2700512
2	0.57772059	0.4267242
3	0.39644024	0.4365570
4	0.65778381	0.3866478
5	0.37762176	0.4186423
6	0.47318111	0.4414534
7	0.10672350	0.2720074
8	0.83058589	0.2996083
9	0.53828371	0.4208548

pixel378

Y	[,1]	[,2]
0	0.03940726	0.1665277
1	0.94275710	0.1404872
2	0.34698529	0.4259148
3	0.76322362	0.3489446
4	0.25780641	0.3743999
5	0.49623023	0.4271960
6	0.34686533	0.4090192
7	0.08493322	0.2268660
8	0.70346366	0.3896035
9	0.42832244	0.4182290

pixel406

Y	[,1]	[,2]

0 0.02196304 0.1316857
1 0.96281513 0.1173645
2 0.48845588 0.4436545
3 0.60910364 0.4236871
4 0.49261457 0.4178517
5 0.51592663 0.4321306
6 0.46631837 0.4482933
7 0.13982381 0.2989515
8 0.83100857 0.2870056
9 0.56719923 0.4206561

Figure 43: Naïve Bayes model

2.4.11 Naïve Bayes - Predication

Process Naïve Bayes predication:

```
> NB_Pred <- predict(NB, test_set_numonly , type="class")
```

Figure 44: Naïve Bayes predication model

2.4.12 Naïve Bayes - Confusion Matrix and Accuracy

Accuracy received: 70.12%

```
> confusionMatrix(test_set_labels, NB_Pred)
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2  3  4  5  6  7  8  9
         0 62  1  1  0  0  1  6  0  4  0
         1  0 83  0  1  0  0  0  0  1  0
         2  1  1 57  3  1  1 13  2  8  0
         3  1  8  8 58  1  2  4  2  8  4
         4  1  4  2  0 56  0  4  4  2 31
         5  4  5  0  9  5 26  6  2  9  4
         6  0  1  0  0  0  1 83  1  4  0
         7  1  2  0  4  2  0  0 63  0  6
         8  1 15  3  0  1  4  2  0 39 12
         9  1  5  0  0  3  0  0  4  3 62

Overall Statistics

               Accuracy : 0.7012
                 95% CI : (0.669, 0.732)
    No Information Rate : 0.1488
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6677

 Mcnemar's Test P-Value : NA
```

Figure 45: Naïve Bayes Confusion Matrix and Accuracy

```
Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8 Class: 9
Sensitivity           0.86111  0.66400  0.80282  0.77333  0.81159  0.74286  0.70339  0.80769  0.50000  0.52101
Specificity           0.98307  0.99720  0.96099  0.95033  0.93774  0.94534  0.99030  0.98031  0.95013  0.97781
Pos Pred Value        0.82667  0.97647  0.65517  0.60417  0.53846  0.37143  0.92222  0.80769  0.50649  0.79487
Neg Pred Value        0.98693  0.94437  0.98141  0.97715  0.98234  0.98831  0.95333  0.98031  0.94889  0.92520
Prevalence            0.08571  0.14881  0.08452  0.08929  0.08214  0.04167  0.14048  0.09286  0.09286  0.14167
Detection Rate        0.07381  0.09881  0.06786  0.06905  0.06667  0.03095  0.09881  0.07500  0.04643  0.07381
Detection Prevalence  0.08929  0.10119  0.10357  0.11429  0.12381  0.08333  0.10714  0.09286  0.09167  0.09286
Balanced Accuracy     0.92209  0.83060  0.88190  0.86183  0.87467  0.84410  0.84685  0.89400  0.72507  0.74941
```

Figure 45: Naïve Bayes Statistics by Class

2.4.13 Decision Tree – Building the Model

Building the Decision Tree model:

```
> DT <- rpart(train_set$label ~ .,method = "class", data = train_set)
> printcp(DT)

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

Variables actually used in tree construction:
 [1] pixel154 pixel211 pixel270 pixel290 pixel352 pixel375 pixel377 pixel409 pixel431 pixel434 pixel542 pixel657

Root node error: 2968/3360 = 0.88333

n= 3360

         CP nsplit rel error  xerror      xstd
1  0.096530      0   1.00000 1.00000 0.0062696
2  0.070586      2   0.80694 0.80761 0.0088311
3  0.056941      4   0.66577 0.68632 0.0095421
4  0.053571      5   0.60883 0.63039 0.0097018
5  0.041442      6   0.55526 0.59468 0.0097526
6  0.026954      7   0.51381 0.50674 0.0097113
7  0.022574      8   0.48686 0.49528 0.0096885
8  0.021226      9   0.46429 0.47776 0.0096456
9  0.011792     11   0.42183 0.44306 0.0095318
10 0.010000     12   0.41004 0.42453 0.0094550
```

Figure 46: Decision Tree Building the model

2.4.14 Decision Tree – Predication

Predication for Decision Tree:

```
> DT_Pred <- predict(DT, test_set_numonly , type="class")
```

Figure 47: Decision Tree Predication

2.4.15 Decision Tree – Confusion Matrix and Accuracy

Accuracy received: 58.93%

```
> #Confusion Matrix
> confusionMatrix(test_set_labels, DT_Pred)
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2  3  4  5  6  7  8  9
         0 60  0  0  0  1  1  1  3  9  0
         1  0 73  7  1  0  0  0  2  2  0
         2  5 10 28  2  4  2 23  4  9  0
         3  5  3  7 41  6  5  3  4 19  3
         4  2  2  3  7 65  3  3  5  5  9
         5  7  4  2  8  2 26  4  4  8  5
         6  8  7  4  1  1  5 57  0  7  0
         7  8  1  4  2  3  3  1 46  4  6
         8  1  4  3  5  1  5  0  1 55  2
         9  0  1  6  3  5 10  1  5  3 44

Overall Statistics

               Accuracy : 0.5893
                 95% CI : (0.5552, 0.6228)
    No Information Rate : 0.144
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5436

 Mcnemar's Test P-Value : NA
```

Figure 48: Decision Tree Decision Matrix and Accuracy

Statistics by Class:

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4	Class: 5	Class: 6	Class: 7	Class: 8	Class: 9
Sensitivity	0.62500	0.6952	0.43750	0.58571	0.73864	0.43333	0.61290	0.62162	0.45455	0.63768
Specificity	0.97984	0.9837	0.92397	0.92857	0.94814	0.94359	0.95582	0.95822	0.96940	0.95590
Pos Pred Value	0.80000	0.8588	0.32184	0.42708	0.62500	0.37143	0.63333	0.58974	0.71429	0.56410
Neg Pred Value	0.95294	0.9576	0.95219	0.96102	0.96875	0.95584	0.95200	0.96325	0.91350	0.96719
Prevalence	0.11429	0.1250	0.07619	0.08333	0.10476	0.07143	0.11071	0.08810	0.14405	0.08214
Detection Rate	0.07143	0.0869	0.03333	0.04881	0.07738	0.03095	0.06786	0.05476	0.06548	0.05238
Detection Prevalence	0.08929	0.1012	0.10357	0.11429	0.12381	0.08333	0.10714	0.09286	0.09167	0.09286
Balanced Accuracy	0.80242	0.8395	0.68073	0.75714	0.84339	0.68846	0.78436	0.78992	0.71197	0.79679

Figure 48: Decision Tree Decision Statistics by Class

2.4.16 Decision Tree – Visualizations

Here are the Decision Tree Visualizations:

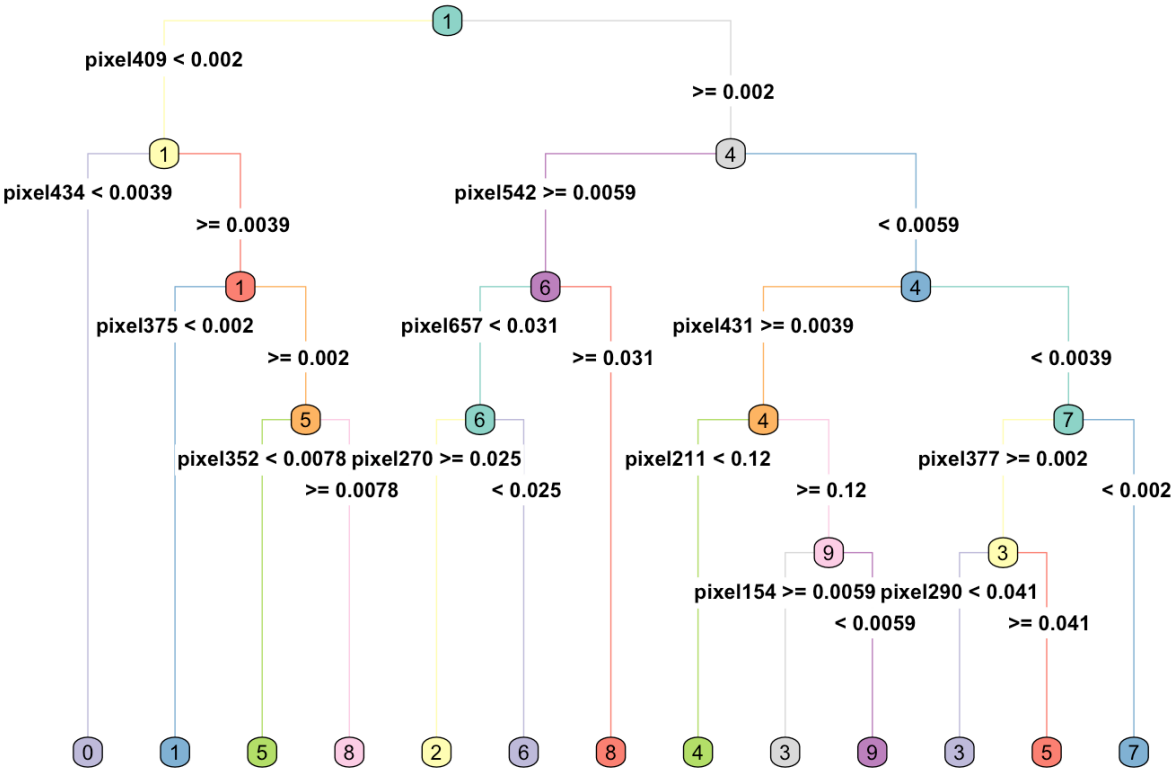


Figure 49: Decision Tree Visualization – Heat Tree

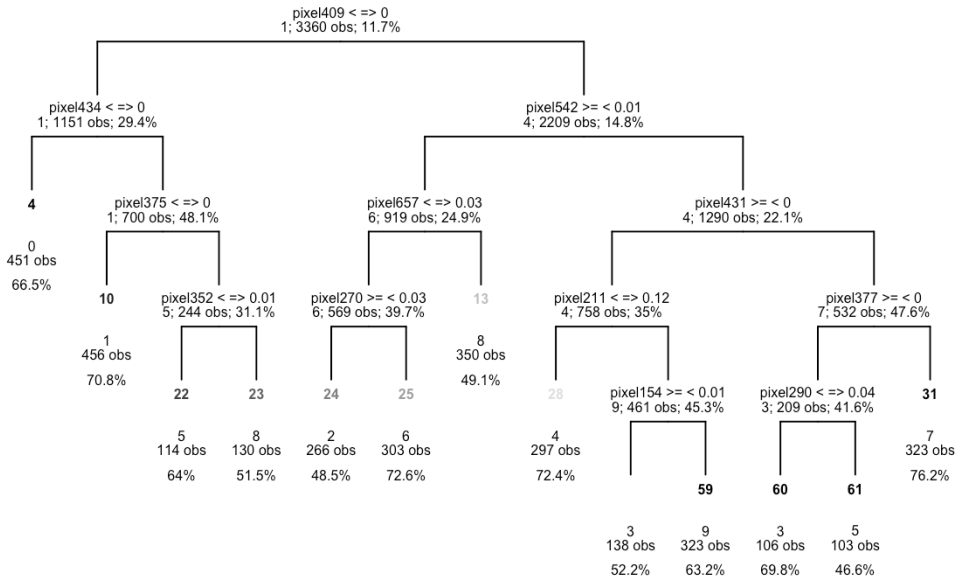


Figure 50: Decision Tree Visualization – Draw Tree Nodes

3.0 Results

The accuracy results received are as follows:

kNN: 82.86%

Random Forest: 100%

Support Vector Machine (SVM): 88.45%

Naïve Bayes: 70.12%

Decision Tree: 58.93%

4.0 Conclusions

This research work contributed in so many ways for me as being a Data Scientist. If the researcher had to draw the main reason for the goal of this work, the answer will be pre-processing. Improving the pre-processing mechanism allowed the researcher, to achieve the most accuracy of each algorithms. Building all the algorithms was not the majority of the work but rather, improving the datasets in a way that each algorithm will provide the best results.

The researcher still wondering what was the main reason for low accuracy received in Decision Tree model but if the researcher had to guess what will be the main reason for the low accuracy received, it will probably will indicate that the pre-processing can be improved.

In addition, the researcher realized that the majority of the pre-processing steps did assist to improve the accuracy. Nevertheless, with similar research which requires to combined multiple algorithms, I would consider to separate the low accuracy models from the rest of the algorithms that received high accuracy and build a separate preprocessing model.

Lastly, I would like to Thank Dr. Bolton. His knowledge and guidance allowed me to improve my research goals with all the model. Thank you so much.

5.0 References

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

https://builtin.com/data-science/random-forest-algorithm