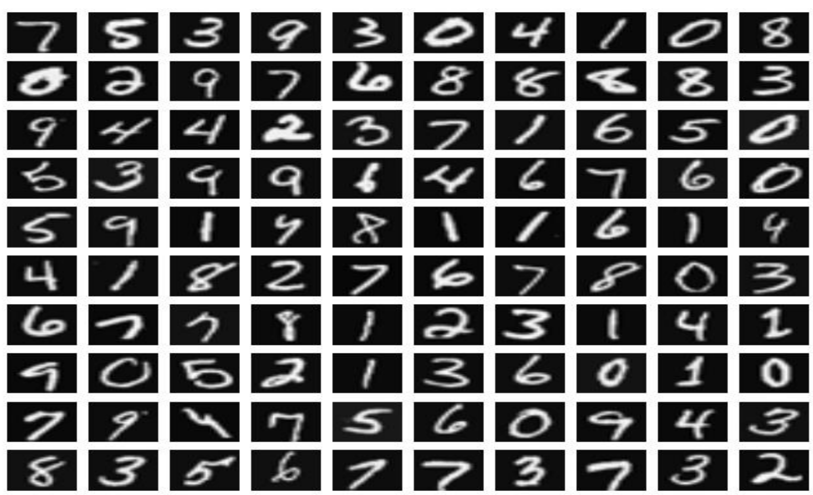
Optical Recognition of Handwritten Digits



Presented By: Varsha Tekale

Business Analytics

5th August 2016

Optical Recognition of Handwritten Digits

Data Gathering

Download the dataset available on my GitHub Account:

https://github.com/VarshaVT/Handwritten-Digits-Recognition/blob/master/uci_files-selected.zip

I got the processed training data set and validation data sets.

I have used R programming language for this project.

First of all, set your working directory.

```
> ##Set the directory
> getwd()
[1] "C:/Users/Varsha/Music/AnalyticsPracticum/uci_files-selected"
> setwd("C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_files-selected")
```

Load the datasets and convert it into the data frames.

```
> ##Load the data
> tra <- read.table(file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_files-selected\\optdigits-orig_tra_linear.dat")
> cv <- read.table(file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_files-selected\\optdigits-orig_cv_linear.dat")
> tra <- as.data.frame(tra)
> cv <- as.data.frame(cv)
```

Exploratory Analysis

Next step is to know about your data and understand it properly for that perform the exploratory data analysis.

```
> ##Data Exploration of Training dataset
> head(tra)
  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33
1  0  0  0  0  0  0  0  0  0   0   0   0   0   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2  0  0  0  0  0  0  0  0  0   0   0   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3  0  0  0  0  0  0  0  0  0   0   0   0   0   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0
4  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   1   1   1   1   1   0   0   0   0   0   0   0   0   0
5  0  0  0  0  0  0  0  0  0   0   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6  0  0  0  0  0  0  0  0  0   0   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

> str(tra)
'data.frame':   1934 obs. of  1026 variables:
 $ V1   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V2   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V3   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V4   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V5   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V6   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V7   : int  0 0 0 0 0 0 0 0 0 0 ...

> dim(tra)
[1] 1934 1026
> attributes(tra)
$names
   [1] "V1"   "V2"   "V3"   "V4"   "V5"   "V6"   "V7"   "V8"   "V9"   "V10"  "V11"  "V12"  "V13"  "V14"
  [15] "V15"  "V16"  "V17"  "V18"  "V19"  "V20"  "V21"  "V22"  "V23"  "V24"  "V25"  "V26"  "V27"  "V28"

> nrow(tra)
[1] 1934
> ncol(tra)
[1] 1026
```

```
row.names(tra)
colnames(tra)
summary(tra)
sapply(tra[1,], class)
```

Similarly, do it for validation dataset

```
##Data Exploration of validation dataset
head(cv)
str(cv)
dim(cv)
attributes(cv)
nrow(cv)
ncol(cv)
row.names(cv)
colnames(cv)
summary(cv)
sapply(cv[1,], class)
```

Check if any missing values are there.

```
> ## Check for missing values
> sum(is.na(tra))
[1] 0
> sum(is.na(cv))
[1] 0
```

After carefully observing, I found that column 1025th and 1026th are exactly same. So, remove the extra column.

```
> #remove the last 1026th column
> tra <- tra[, -1026]
> cv <- cv[, -1026]
```

From the summary output it has been noted that 1025th column is consisting of labels 0 – 9 digits. Lets rename the columns and change the data type of it as factor.

```
> ##Change the label of last column of training set and make it as factor
> tra[,1025] <- as.factor(tra[,1025])
> colnames(tra) <- c(paste("X.", 1:1024, sep = ""), "Y")
> class(tra[,1025])
[1] "factor"
> ##Change the label of last column of validation set and make it as factor
> cv[,1025] <- as.factor(cv[,1025])
> colnames(cv) <- c(paste("X.", 1:1024, sep = ""), "Y")
> class(cv[,1025])
[1] "factor"
```

Let's see the levels of 1025th column and check the class for training and validation datasets.

```
> ###See the digits 0-9
> levels(tra[,1025])
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
> sapply(tra[1,], class)

> ###See the digits 0-9
> levels(cv[,1025])
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
> sapply(cv[1,], class)
```

Move the label column to first for the ease of model application.

```
> ## Move the label to first column
> tra <- tra[c(1025,1:1024)]
> cv <- cv[c(1025,1:1024)]
```

We will install some packages that will help us in model building.

library(RColorBrewer):

RColorBrewer is an R packages that uses the work from http://colorbrewer2.org/ to help you choose sensible colour schemes for figures in R.

library(ElemStatLearn):

Data Sets, Functions and Examples from the Book: "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" by Trevor Hastie, Robert Tibshirani and Jerome Friedman

library(foreign):

Functions for reading and writing data stored by some versions of Epi Info, Minitab, S, SAS, SPSS, Stata, Systat and Weka and for reading and writing some dBase files

library(tree):

Classification and regression trees

library(rpart):

Recursive partitioning for classification, regression and survival trees.

library(maptree):

Mapping, pruning, and graphing tree models

library(e1071):

Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, naive Bayes classifier, ...

library(class):

Various functions for classification, including k-nearest neighbour, Learning Vector Quantization and Self-Organizing Maps.

library(RWeka):

An R interface to Weka (Version 3.9.0). Weka is a collection of machine learning algorithms for data mining tasks written in Java, containing tools for data pre-processing, classification, regression, clustering, association rules, and visualization. Package 'RWeka' contains the interface code, the Weka jar is in a separate package 'RWekajars'.

Library(randomForest):

Classification and regression based on a forest of trees using random inputs.

I installed all these packages using a function.

```
> ##install the packages
> packages <- function(pkg){
+ new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
+ if (length(new.pkg))
+ install.packages(new.pkg, dependencies = TRUE, repos='http://cran.rstudio.com/')
+ sapply(pkg, require, character.only = TRUE)
+ }
> packages(c("RColorBrewer", "ElemStatLearn", "foreign", "tree", "RWeka",
+ "rpart", "maptree", "e1071", "cluster", "class", "FNN", "randomForest"))
 RColorBrewer ElemStatLearn       foreign          tree         RWeka         rpart       maptree         e1071
         TRUE          TRUE          TRUE          TRUE          TRUE          TRUE          TRUE          TRUE
      cluster         class           FNN  randomForest
         TRUE          TRUE          TRUE          TRUE
```

Set the colors, pattern and custom colors for visualization of digits.

```
> ## Set the colors for visualization of digits
> digit_colors <- c("red", "white")
> #"colorRampPalette": return functions that interpolate a set of given colors to create new color palettes
> more_colors <- colorRampPalette(colors = digit_colors)
> colors.plot <- colorRampPalette(brewer.pal(10, "Set3"))
```

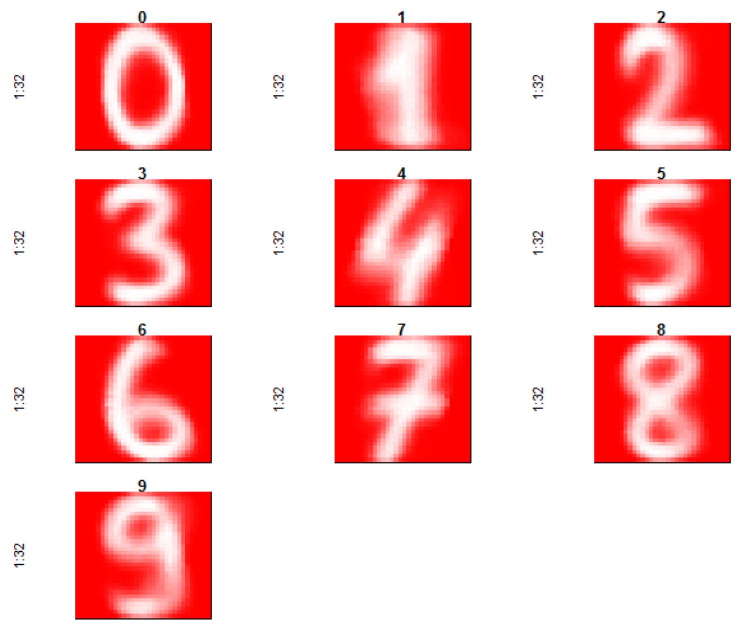
Descriptive Statistics

Now, display the digits of training set and validation set respectively.

```
> ###
> ### Display digits of training data set by calculating the average of each digit
> ###
> par(mfrow = c(4, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> digits.0_9 <- array(dim = c(10, 32 * 32))
> for (dig in 0:9) {
+ print(dig)
+ digits.0_9[dig + 1, ] <- apply(tra[tra[, 1] == dig, -1], 2, sum)
+ digits.0_9[dig + 1, ] <- digits.0_9[dig + 1, ]/max(digits.0_9[dig + 1, ]) * 1023
+ z <- array(digits.0_9[dig + 1, ], dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = dig, col = more_colors(1024))
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

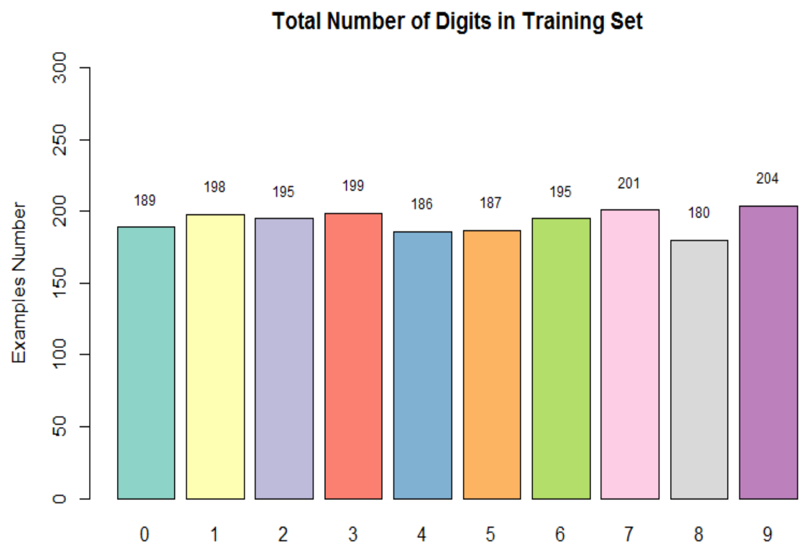
par can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to par in tag = value form, or by passing them as a list of tagged values. We can define rows and columns using **mfrow** and margins using **mar.**

Here I calculated the average of each digit for displaying.

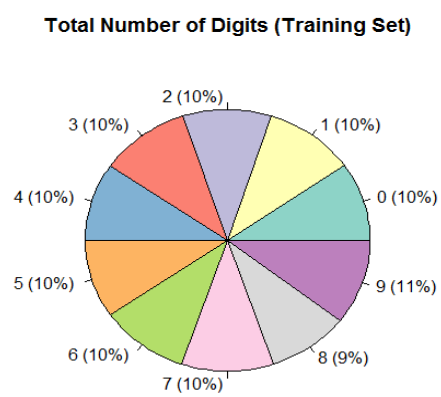


Let's see the total numbers in training set and validation set by plotting a bar chart.

```
> dig_lable <- table(tra$Y)
> par(mfrow = c(1, 1))
> par(mar = c(5, 4, 4, 2) + 0.1)  # increase y-axis margin.
> plot <- plot(tra$Y, col = colors.plot(10), main = "Total Number of Digits in Training Set",
+ ylim = c(0, 300), ylab = "Examples Number")
> text(x = plot, y = dig_lable+20, labels = dig_lable, cex = 0.75)
```

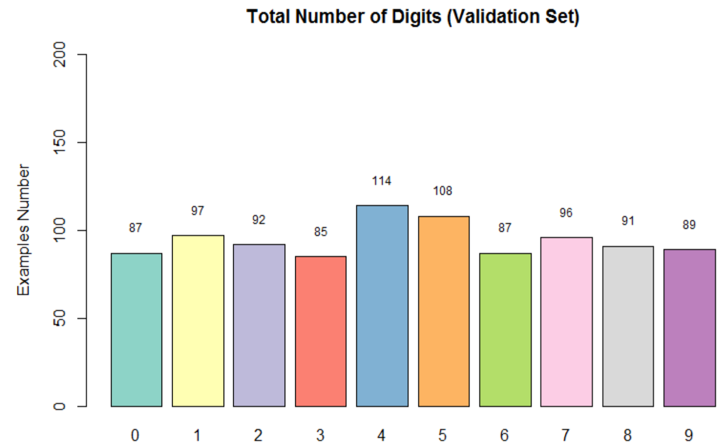


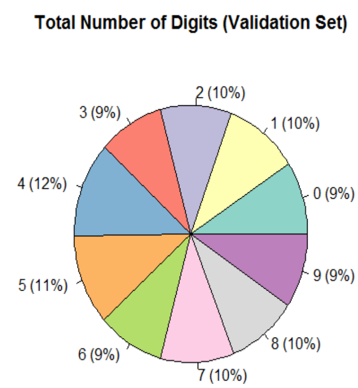

```
> par(mfrow = c(1, 1))
> percentage <- round(dig_lable/sum(dig_lable) * 100)
> labels <- paste0(row.names(dig_lable), " (", percentage, "%) ")  # add percents to labels
> pie(dig_lable, labels = labels, col = colors.plot(10), main = "Total Number of Digits (Training Set)")
> ## Total numbers in validation dataset
```



For validation set,

```
> ## Total numbers in validation dataset
> dig_lable <- table(cv$Y)
> par(mfrow = c(1, 1))
> par(mar = c(5, 4, 4, 2) + 0.1)  # increase y-axis margin.
> plot <- plot(cv$Y, col = colors.plot(10), main = "Total Number of Digits (Validation Set)",
+ ylim = c(0, 200), ylab = "Examples Number")
> text(x = plot, y = dig_lable + 15, labels = dig_lable, cex = 0.75)
> par(mfrow = c(1, 1))
> percentage <- round(dig_lable/sum(dig_lable) * 100)
> labels <- paste0(row.names(dig_lable), " (", percentage, "%) ")  # add percents to labels
> pie(dig_lable, labels = labels, col = colors.plot(10), main = "Total Number of Digits (Validation Set)")
```





Data Modeling / Predictive Statistics

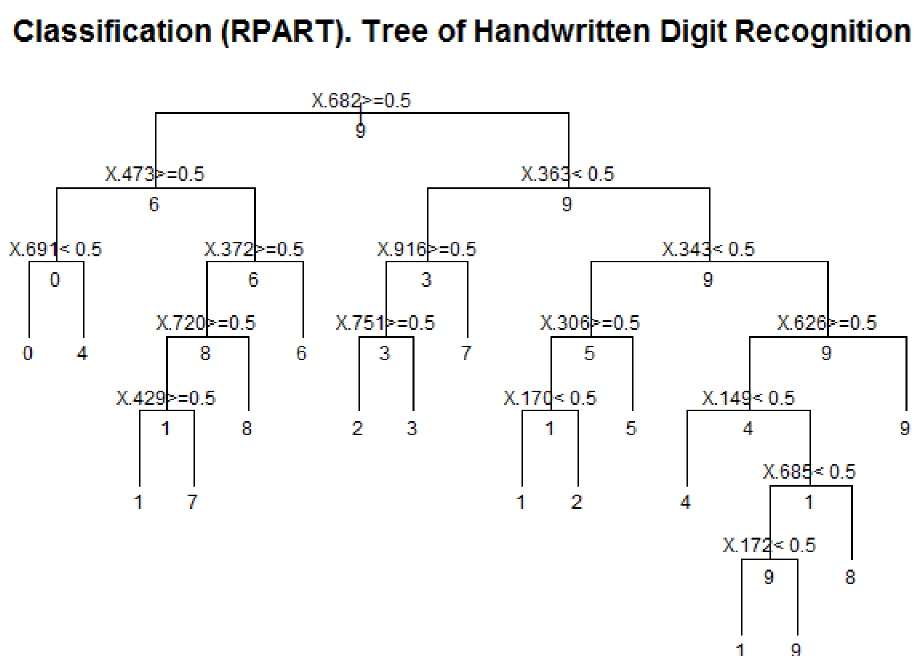
Now, our data is ready for processing. In this step we will apply various models on our datasets and see the results and accuracy of each model and its predictions.

MODEL 1

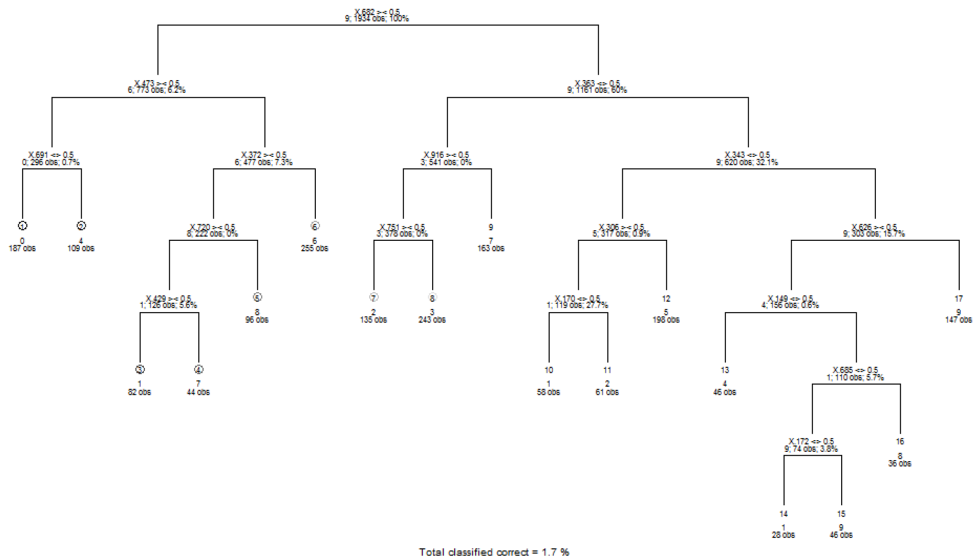
Recursive Partitioning and Regression Trees

```
> #####  MODEL 1  #####
> ##Classification. Predictive Model. RPart
> #proc.time() determines how much real and CPU time (in seconds) the currently running R process has already taken.
> x <- proc.time()
> fit.rpart <- rpart(tra$Y ~ ., method = "class", data = tra)
> proc.time() - x
   user  system elapsed 
   3.53    0.00    8.38 
> #printcp() Displays the cp table for fitted rpart object
> printcp(fit.rpart)

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

Variables actually used in tree construction:
 [1] X.149 X.170 X.172 X.306 X.343 X.363 X.372 X.429 X.473 X.626 X.682 X.685 X.691 X.720 X.751 X.916

Root node error: 1730/1934 = 0.89452

n= 1934 

         CP nsplit rel error  xerror      xstd
1  0.108671      0   1.00000 1.02197 0.0071207
2  0.106358      1   0.89133 0.93468 0.0094104
3  0.091908      2   0.78497 0.81329 0.0113182
4  0.089595      3   0.69306 0.70520 0.0122675
5  0.076879      4   0.60347 0.62543 0.0126200
6  0.058382      5   0.52659 0.54046 0.0127032
7  0.040462      7   0.40983 0.42023 0.0123125
8  0.027168      8   0.36936 0.38728 0.0120959
9  0.016474      9   0.34220 0.35954 0.0118737
10 0.016185     11   0.30925 0.34855 0.0117753
11 0.010790     12   0.29306 0.31792 0.0114677
12 0.010000     16   0.24855 0.30405 0.0113115
```

```
> plot(fit.rpart, uniform = TRUE, main = "Classification (RPART). Tree of Handwritten Digit Recognition ")
> text(fit.rpart, all = TRUE, cex = 0.75)
```




```
> draw.tree(fit.rpart, cex = 0.5, nodeinfo = TRUE, col = gray(0:8/8))
```



Now, make the prediction on validation dataset and create confusion matrix.

```
> #Confusion Matrix (RPart)
> prediction.rpart <- predict(fit.rpart, newdata = cv, type = "class")
> table(`Actual` = cv$Y, `Predicted` = prediction.rpart)
      Predicted
Actual  0  1  2  3  4  5  6  7  8  9
     0 82  0  0  0  1  0  2  0  2  0
     1  0 47 25 10  1  2  2  3  6  1
     2  0  0 60  4  0  3  5  9  6  5
     3  0  0  4 74  0  1  0  2  0  4
     4  1 14  0  1 70  9 14  2  0  3
     5  0  2  4  2  0 89  1  7  1  2
     6  0  0  0  0  1  2 84  0  0  0
     7  2  0  0  1  5  0  0 86  1  1
     8  0 19  6  1  2  0  8  1 49  5
     9  0  2  2  6  1  1  0  4  0 73
```

Calculate the accuracy of the model.

```
> error.rate.rpart <- sum(cv$Y != prediction.rpart)/nrow(cv)
> print(paste0("Accuracy: ", 1 - error.rate.rpart))
[1] "Accuracy: 0.754756871035941"
```

Accuracy is 75%.

Predict the digit for example 1 using Rpart. Let's see the actual digit and predicted digit. And plot the respective digit. Here actual digit is 5 and predicted digit is 9.

```
> #Predict Digit for Example 1 (RPart)
> row <- 1
> prediction.digit <- as.vector(predict(fit.rpart, newdata = cv[row, ], type = "class"))
> print(paste0("Actual Digit: ", as.character(cv$Y[row])))
[1] "Actual Digit: 5"
> print(paste0("Predicted Digit: ", prediction.digit))
[1] "Predicted Digit: 9"
> z <- array(as.vector(as.matrix(cv[row, -1])), dim = c(32, 32))
> z <- z[, 32:1]  ##right side up
> par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> image(1:32, 1:32, z, main = cv[row, 1], col = more_colors(1024))
```

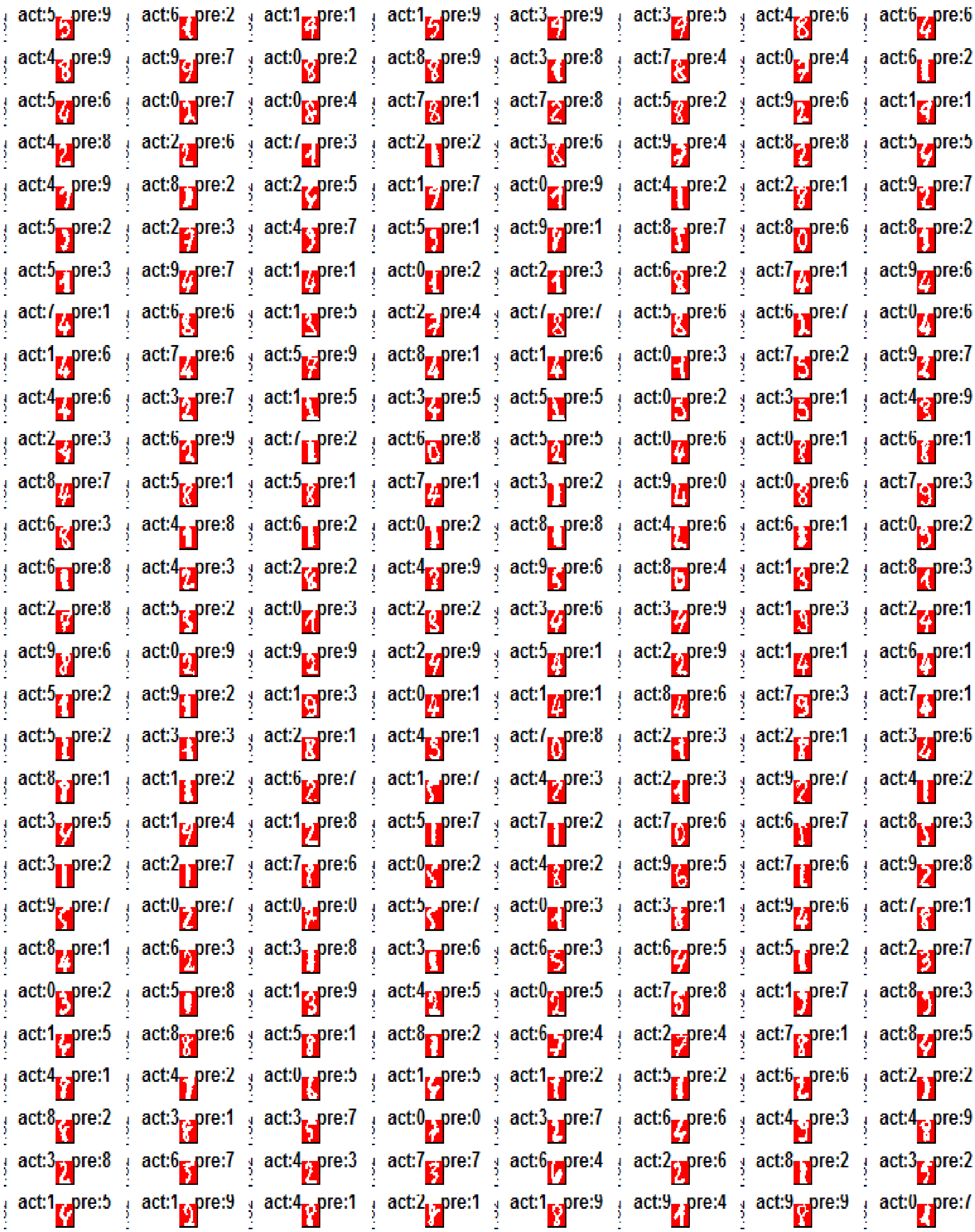
As the error rate is approx. 25 % our predicted digit is not matched with actual digit. Here, we can say that the tree model using RPart is not a good model as the accuracy is only 25%.

We can actually see the error numbers in our model and visualize that numbers. In this case error numbers are very large i.e. 232.

```
> ##Errors with tree based methods (rpart)
> errors <- as.vector(which(cv$Y != prediction.rpart))
> print(paste0("Error Numbers: ", length(errors)))
[1] "Error Numbers: 232"
> predicted <- as.vector(prediction.rpart)
> par(mfrow = c(29, 8), pty = "s", mar = c(.5, .5, .5, .5), xaxt = "n", yaxt = "n")
> for (i in 1:length(errors)) {
+ z <- array(as.vector(as.matrix(cv[errors[i], -1])), dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = paste0("act:", as.character(cv$Y[i]),
+ " - pre:", predicted[errors[i]]), col = more_colors(1024))
+ }
```

Error Numbers: 232

The below plot shows the actual digit (act) and predicted digit (pre).


act:5 pre:9
act:6 pre:2
act:1 pre:1
act:1 pre:9
act:3 pre:9
act:3 pre:5
act:4 pre:6
act:6 pre:6
act:4 pre:9
act:9 pre:7
act:0 pre:2
act:8 pre:9
act:3 pre:8
act:7 pre:4
act:0 pre:4
act:6 pre:2
act:5 pre:6
act:0 pre:7
act:0 pre:4
act:7 pre:1
act:7 pre:8
act:5 pre:2
act:9 pre:6
act:1 pre:1
act:4 pre:8
act:2 pre:6
act:7 pre:3
act:2 pre:2
act:3 pre:6
act:9 pre:4
act:8 pre:8
act:5 pre:5
act:4 pre:9
act:8 pre:2
act:2 pre:5
act:1 pre:7
act:0 pre:9
act:4 pre:2
act:2 pre:1
act:9 pre:7
act:5 pre:2
act:2 pre:3
act:4 pre:7
act:5 pre:1
act:9 pre:1
act:8 pre:7
act:8 pre:6
act:8 pre:2
act:5 pre:3
act:9 pre:7
act:1 pre:1
act:0 pre:2
act:2 pre:3
act:6 pre:2
act:7 pre:1
act:9 pre:6
act:7 pre:1
act:6 pre:6
act:1 pre:5
act:2 pre:4
act:7 pre:7
act:5 pre:6
act:6 pre:7
act:0 pre:6
act:1 pre:6
act:7 pre:6
act:5 pre:9
act:8 pre:1
act:1 pre:6
act:0 pre:3
act:7 pre:2
act:9 pre:7
act:4 pre:6
act:3 pre:7
act:1 pre:5
act:3 pre:5
act:5 pre:5
act:0 pre:2
act:3 pre:1
act:4 pre:9
act:2 pre:3
act:6 pre:9
act:7 pre:2
act:6 pre:8
act:5 pre:5
act:0 pre:6
act:0 pre:1
act:6 pre:1
act:8 pre:7
act:5 pre:1
act:5 pre:1
act:7 pre:1
act:3 pre:2
act:9 pre:0
act:0 pre:6
act:7 pre:3
act:6 pre:3
act:4 pre:8
act:6 pre:2
act:0 pre:2
act:8 pre:8
act:4 pre:6
act:6 pre:1
act:0 pre:2
act:6 pre:8
act:4 pre:3
act:2 pre:2
act:4 pre:9
act:9 pre:6
act:8 pre:4
act:1 pre:2
act:8 pre:3
act:2 pre:8
act:5 pre:2
act:0 pre:3
act:2 pre:2
act:3 pre:6
act:3 pre:9
act:1 pre:3
act:2 pre:1
act:9 pre:6
act:0 pre:9
act:9 pre:9
act:2 pre:9
act:5 pre:1
act:2 pre:9
act:1 pre:1
act:6 pre:1
act:5 pre:2
act:9 pre:2
act:1 pre:3
act:0 pre:1
act:1 pre:1
act:8 pre:6
act:7 pre:3
act:7 pre:1
act:5 pre:2
act:3 pre:3
act:2 pre:1
act:4 pre:1
act:7 pre:8
act:2 pre:3
act:2 pre:1
act:3 pre:6
act:8 pre:1
act:1 pre:2
act:6 pre:7
act:1 pre:7
act:4 pre:3
act:2 pre:3
act:9 pre:7
act:4 pre:2
act:3 pre:5
act:1 pre:4
act:1 pre:8
act:5 pre:7
act:7 pre:2
act:7 pre:6
act:6 pre:7
act:8 pre:3
act:3 pre:2
act:2 pre:7
act:7 pre:6
act:0 pre:2
act:4 pre:2
act:9 pre:5
act:7 pre:6
act:9 pre:8
act:9 pre:7
act:0 pre:7
act:0 pre:0
act:5 pre:7
act:0 pre:3
act:3 pre:1
act:9 pre:6
act:7 pre:1
act:8 pre:1
act:6 pre:3
act:3 pre:8
act:3 pre:6
act:6 pre:3
act:6 pre:5
act:5 pre:2
act:2 pre:7
act:0 pre:2
act:5 pre:8
act:1 pre:9
act:4 pre:5
act:0 pre:5
act:7 pre:8
act:1 pre:7
act:8 pre:3
act:1 pre:5
act:8 pre:6
act:5 pre:1
act:8 pre:2
act:6 pre:4
act:2 pre:4
act:7 pre:1
act:8 pre:5
act:4 pre:1
act:4 pre:2
act:0 pre:5
act:1 pre:5
act:1 pre:2
act:5 pre:2
act:6 pre:6
act:2 pre:2
act:8 pre:2
act:3 pre:1
act:3 pre:7
act:0 pre:0
act:3 pre:7
act:6 pre:6
act:4 pre:3
act:4 pre:9
act:3 pre:8
act:6 pre:7
act:4 pre:3
act:7 pre:7
act:6 pre:4
act:2 pre:6
act:8 pre:2
act:3 pre:2
act:1 pre:5
act:1 pre:9
act:4 pre:1
act:2 pre:1
act:1 pre:9
act:9 pre:4
act:9 pre:9
act:0 pre:7

MODEL 2

Naïve Bays Algorithm

```
> #####  MODEL 2  #####
> ##
> ##Classification. Predictive Model. Naive Bayes Algorithm
> ##
> #Naive Bays Algorithm{e1071}: Computes the conditional a-posterior probabilities of a
> #categorical class variable given independent predictor variables using the Bayes rule.
> x <- proc.time()
> fit.naiveBayes <- naiveBayes(tra$Y ~ ., data = tra)
> proc.time() - x
   user  system elapsed
   1.25    0.02   32.55
> summary(fit.naiveBayes)
        Length Class  Mode
apriori   10   table  numeric
tables  1024   -none- list
levels    10   -none- character
call       4   -none- call
```

Now, make the prediction on validation dataset and create confusion matrix.

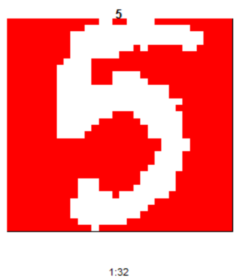
```
> #Confusion Matrix (naiveBayes)
> prediction.naiveBayes <- predict(fit.naiveBayes, newdata = cv, type = "class")
> table(`Actual` = cv$Y, `Predicted` = prediction.naiveBayes)
      Predicted
Actual  0  1  2  3  4  5  6  7  8  9
     0 86  0  0  0  0  0  0  0  1  0
     1  0 72  2  0  0  0  1  0 20  2
     2  0  0 81  1  0  0  0  0 10  0
     3  0  0  0 73  0  0  0  1  9  2
     4  1  1  0  0 92  0  4  1 13  2
     5  0  0  0  4  0 93  1  0  7  3
     6  0  0  0  0  0  0 86  0  1  0
     7  0  0  0  0  0  0  0 93  3  0
     8  0  0  0  0  0  0  1  0 90  0
     9  0  1  0 14  1  0  0  2 18 53
> error.rate.naiveBayes <- sum(cv$Y != prediction.naiveBayes)/nrow(cv)
> print(paste0("Accuracy: ", 1 - error.rate.naiveBayes))
[1] "Accuracy: 0.865750528541226"
```

Accuracy is 87%.

This model is Little better than previous model.

Predict the digit for example 1 using Naïve Bays. Let's see the actual digit and predicted digit. And plot the respective digit. Here actual digit is 5 and predicted digit is 5.

```
> ##Predict Digit for Example 1 (naiveBayes)
> # All Prediction for Row 1
> row <- 1
> prediction.digit <- as.vector(predict(fit.naiveBayes, newdata = cv[row, ], type = "class"))
> print(paste0("Actual Digit: ", as.character(cv$Y[row])))
[1] "Actual Digit: 5"
> print(paste0("Predicted Digit: ", prediction.digit)) #[1] "Predicted Digit: 5"
[1] "Predicted Digit: 5"
> z <- array(as.vector(as.matrix(cv[row, -1])), dim = c(32, 32))
> z <- z[, 32:1]  ##right side up
> par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> image(1:32, 1:32, z, main = cv[row, 1], col = more colors(1024))
```



Our prediction is correct in this case. As both actual and predicted values are exact same.

We can actually see the error numbers in our model and visualize that numbers. In this case error numbers are very large i.e. 127. The number is reduced as the accuracy of the model increased.

```
> ##Errors with Naive Bayes
> errors <- as.vector(which(cv$Y != prediction.naiveBayes))
> print(paste0("Error Numbers: ", length(errors)))
[1] "Error Numbers: 127"
> predicted <- as.vector(prediction.naiveBayes)
> par(mfrow = c(16, 8), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> for (i in 1:length(errors)) {
+ z <- array(as.vector(as.matrix(cv[errors[i], -1])), dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = paste0("act:", as.character(cv$Y[i]),
+ " - pre:", predicted[errors[i]]), col = more_colors(1024))
+ }
```

Error Numbers: 127

The below plot shows the actual digit (act) and predicted digit (pre).

MODEL 3

Support Vector Machine

```
> #####  MODEL 3  #####
> ##
> ##Classification. Predictive Model. SVM (Support Vector Machine) Algorithm
> ##
> #svm {e1071}:is used to train a support vector machine.
> #It can be used to carry out general regression and classification (of nu and epsilon-type),
> #as well as density-estimation.
> x <- proc.time()
> fit.svm <- svm(tra$Y ~ ., method = "class", data = tra)

> proc.time() - x
   user  system elapsed 
  11.02    0.03   11.47 
> summary(fit.svm) #Number of Support Vectors:  1141

Call:
svm(formula = tra$Y ~ ., data = tra, method = "class")


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  radial 
       cost:  1 
      gamma:  0.0009765625 

Number of Support Vectors:  1141

 ( 79 93 109 87 101 122 135 130 160 125 )


Number of Classes:  10 

Levels: 
 0 1 2 3 4 5 6 7 8 9
```

Now, make the prediction on validation dataset and create confusion matrix.

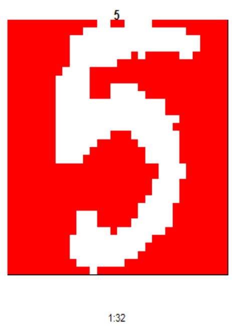
```
> ##Confusion Matrix (SVM)
> prediction.svm <- predict(fit.svm, newdata = cv, type = "class")
> table(`Actual` = cv$Y, `Predicted` = prediction.svm)
      Predicted
Actual   0   1   2   3   4   5   6   7   8   9
     0  87   0   0   0   0   0   0   0   0   0
     1   0  96   0   0   0   0   0   1   0   0
     2   0   0  87   0   0   0   0   0   4   1
     3   0   0   0  79   0   1   0   0   1   4
     4   1   1   0   0 109   0   0   0   2   1
     5   0   0   0   0   0 107   0   0   0   1
     6   0   0   0   0   1   0  86   0   0   0
     7   0   0   0   0   1   0   0  95   0   0
     8   0   1   0   0   0   0   1   0  89   0
     9   0   1   0   0   0   0   0   1   0  87
> error.rate.svm <- sum(cv$Y != prediction.svm)/nrow(cv)
> print(paste0("Accuracy (Precision): ", 1 - error.rate.svm))
[1] "Accuracy (Precision): 0.974630021141649"
```

Accuracy is 97%

Accuracy of the model increased by 10 using support vector machine algorithm compared to previous. This is a good model with error rate of only 3%.

Predict the digit for example 1 using SVM. Let's see the actual digit and predicted digit. And plot the respective digit. Here actual digit is 5 and predicted digit is 5.

```
> #Predict Digit for Example 1 (SVM)
> # All Prediction for Row 1
> row <- 1
> prediction.digit <- as.vector(predict(fit.svm, newdata = cv[row, ], type = "cl
> print(paste0("Actual Digit: ", as.character(cv$Y[row])))
[1] "Actual Digit: 5"
> print(paste0("Predicted Digit: ", prediction.digit))
[1] "Predicted Digit: 5"
> z <- array(as.vector(as.matrix(cv[row, -1])), dim = c(32, 32))
> z <- z[, 32:1]  ##right side up
> par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> image(1:32, 1:32, z, main = cv[row, 1], col = more_colors(1024))
```



Our prediction is correct in this case. As both actual and predicted values are exact same.

We can actually see the error numbers in our model and visualize that numbers. In this case error numbers are less i.e. only 24. The number is reduced as the accuracy of the model increased.

```
> ##Errors with Support Vector Machine (SVM)
> errors <- as.vector(which(cv$Y != prediction.svm))
> print(paste0("Error Numbers: ", length(errors)))
[1] "Error Numbers: 24"
> predicted <- as.vector(prediction.svm)
> par(mfrow = c(6, 4), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> for (i in 1:length(errors)) {
+ z <- array(as.vector(as.matrix(cv[errors[i], -1])), dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = paste0("act:", as.character(cv$Y[i]),
+ " - prd:", predicted[errors[i]]), col = more_colors(1024))
+ }
```

Error Numbers: 24

The below plot shows the actual digit (act) and predicted digit (pre).

MODEL 4

Fast Nearest Neighbors

```
> #####  MODEL 4  #####
> ##
> ##Classification. Fast Nearest Neighbors (FNN) Algorithm
> ##
> x <- proc.time()
> # Avoid Name Collision (knn)
> fit.fnn <- FNN::knn(tra[, -1], cv[, -1], tra$Y,
+ k = 10, algorithm = "cover_tree")
> proc.time() - x
   user  system elapsed
   6.39    0.04   10.45
> summary(fit.fnn)
  0   1   2   3   4   5   6   7   8   9
 88 105  93  83 112 105  89  98  83  90
```

Now, make the prediction on validation dataset and create confusion matrix.

```
> ##Confusion Matrix (FNN)
> table(`Actual` = cv$Y, `Predicted` = fit.fnn)
      Predicted
Actual   0   1   2   3   4   5   6   7   8   9
     0  87   0   0   0   0   0   0   0   0   0
     1   0  96   0   0   0   0   0   1   0   0
     2   0   0  91   0   0   0   0   0   0   1
     3   0   0   2  81   0   0   0   0   1   1
     4   1   0   0   0 112   0   0   0   0   1
     5   0   0   0   1   0 105   1   0   0   1
     6   0   0   0   0   0   0  87   0   0   0
     7   0   0   0   0   0   0   0  96   0   0
     8   0   7   0   1   0   0   1   0  82   0
     9   0   2   0   0   0   0   0   1   0  86
> error.rate.fnn <- sum(cv$Y != fit.fnn)/nrow(cv)
> print(paste0("Accuracy: ", 1 - error.rate.fnn))
[1] "Accuracy: 0.97568710359408"
```

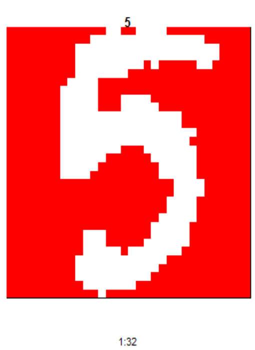
Accuracy is 97.6%

Accuracy of the model is increased by approximately ~ 0.5% compared to previous model.

This is a very good model with error rate of only ~ 2.5%.

Predict the digit for example 1 using FNN::KNN. Let's see the actual digit and predicted digit. And plot the respective digit. Here actual digit is 5 and predicted digit is 5.

```
> ##Predict Digit for Example 1 (FNN)
> row <- 1
> prediction.digit <- fit.fnn[row]
> print(paste0("Actual Digit: ", as.character(cv$Y[row])))
[1] "Actual Digit: 5"
> print(paste0("Predicted Digit: ", prediction.digit))
[1] "Predicted Digit: 5"
> par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n",
> z <- array(as.vector(as.matrix(cv[row, -1])), dim = c(32, 32))
> z <- z[, 32:1]  ##right side up
> image(1:32, 1:32, z, main = cv[row, 1], col = more_colors(1024))
```

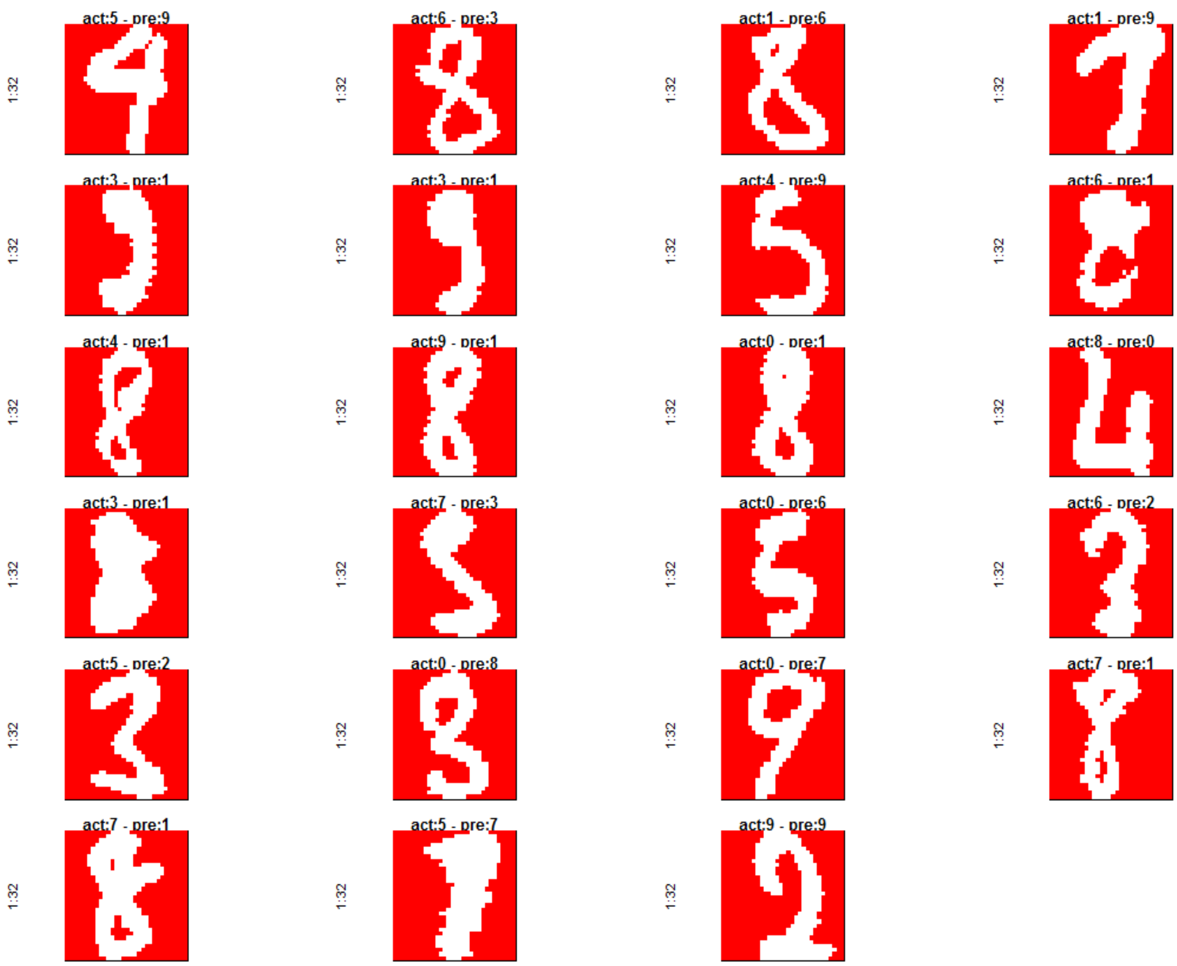


```
> ##Errors with Fast Nearest Neighbors (FNN)
> errors <- as.vector(which(cv$Y != fit.fnn))
> print(paste0("Error Numbers: ", length(errors)))
[1] "Error Numbers: 23"
> predicted <- as.vector(fit.fnn)
> par(mfrow = c(6, 4), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> for (i in 1:length(errors)) {
+ z <- array(as.vector(as.matrix(cv[errors[i], -1])), dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = paste0("act:", as.character(cv$Y[i]),
+ " - pre:", predicted[errors[i]]), col = more_colors(1024))
+ }
>
```

Error Numbers: 23.

Error numbers reduced by 1 as accuracy increased by 0.5%.

The below plot shows the actual digit (act) and predicted digit (pre).



MODEL 5

Random Forest

```
> #####  MODEL 5  #####
> ###
> ###Classification. Predictive Model. Random Forest Algorithm
> ###
> x <- proc.time()
> fit.randomForest <- randomForest(tra$Y ~ ., data = tra, method = "class", ntree=200)
> proc.time() - x
   user  system elapsed 
  50.18    0.00   50.27 
> summary(fit.randomForest)
                Length Class  Mode     
call                5  -none- call     
type                1  -none- character
predicted        1934  factor numeric  
err.rate         2200  -none- numeric  
confusion         110  -none- numeric  
votes           19340  matrix numeric  
oob.times        1934  -none- numeric  
classes            10  -none- character
importance       1024  -none- numeric  
importanceSD        0  -none- NULL     
localImportance     0  -none- NULL     
proximity           0  -none- NULL     
ntree               1  -none- numeric  
mtry                1  -none- numeric  
forest             14  -none- list     
y                1934  factor numeric  
test                0  -none- NULL     
inbag               0  -none- NULL     
terms               3  terms  call     
```

Now, make the prediction on validation dataset and create confusion matrix.

```
> ##Confusion Matrix Random Forest
> prediction.randomForest <- predict(fit.randomForest, newdata = cv, type = "class")
> table(`Actual` = cv$Y, `Predicted` = prediction.randomForest)
      Predicted
Actual   0   1   2   3   4   5   6   7   8   9
     0  87   0   0   0   0   0   0   0   0   0
     1   0  96   0   0   0   0   0   1   0   0
     2   0   0  89   1   0   0   0   1   0   1
     3   0   0   0  82   0   1   0   0   1   1
     4   1   0   0   0 112   0   0   0   0   1
     5   0   0   0   0   0 107   0   0   0   1
     6   0   0   0   0   1   0  86   0   0   0
     7   0   0   0   0   0   0   0  96   0   0
     8   0   1   1   0   0   0   1   0  88   0
     9   0   1   0   0   0   1   0   1   0  86
> error.rate.randomForest <- sum(cv$Y != prediction.randomForest)/nrow(cv)
> print(paste0("Accuracy: ", 1 - error.rate.randomForest))
[1] "Accuracy: 0.982029598308668"
```

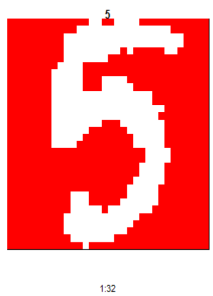
Accuracy is 98.2%.

Again, accuracy is increased by ~ 0.5% comparing to previous model.

This is a very very good model with error rate of only ~ 1.8%.

Predict the digit for example 1 using Random Forest. Let's see the actual digit and predicted digit. And plot the respective digit. Here actual digit is 5 and predicted digit is 5.

```
> #Predict Digit for Example 1 (Random Forest)
> # All Prediction for Row 1
> row <- 1
> prediction.digit <- as.vector(predict(fit.randomForest, newdata = cv[row, ], type = "class"))
> print(paste0("Actual Digit: ", as.character(cv$Y[row])))
[1] "Actual Digit: 5"
> print(paste0("Predicted Digit: ", prediction.digit))
[1] "Predicted Digit: 5"
> z <- array(as.vector(as.matrix(cv[row, -1])), dim = c(32, 32))
> z <- z[, 32:1]  ##right side up
> par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> image(1:32, 1:32, z, main = cv[row, 1], col = more_colors(1024))
```




```
> ##Errors with Random Forest
> errors <- as.vector(which(cv$Y != prediction.randomForest))
> print(paste0("Error Numbers: ", length(errors)))
[1] "Error Numbers: 17"
> predicted <- as.vector(prediction.randomForest)
> par(mfrow = c(5, 4), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> for (i in 1:length(errors)) {
+ z <- array(as.vector(as.matrix(cv[errors[i], -1])), dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = paste0("act:", as.character(cv$Y[i]),
+ " - pre:", predicted[errors[i]]), col = more_colors(1024))
+ }
```

Error Numbers: 17.

Error numbers are reduced as accuracy is further increased by 0.5%.

The below plot shows the actual digit (act) and predicted digit (pre).


act:5 - pre:9
act:6 - pre:9
act:1 - pre:3
act:1 - pre:6
act:3 - pre:9
act:3 - pre:1
act:4 - pre:2
act:6 - pre:0
act:4 - pre:1
act:9 - pre:5
act:0 - pre:8
act:8 - pre:7
act:3 - pre:7
act:7 - pre:5
act:0 - pre:7
act:6 - pre:4
act:5 - pre:9

MODEL 6

K Nearest Neighbor (KNN)

```
> #####  MODEL 6  #####
> ##
> ##Classification. k-Nearest Neighbors (kNN) Algorithm
> ##
> #IBk(RWeka): provides a k-nearest neighbors classifier
> x <- proc.time()
> ##Knn is also provided by Weka as a class "IBk"
> fit.knn <- IBk(tra$Y ~ ., data = tra) #IBk(): R interfaces to Weka lazy learners
> proc.time() - x
   user  system elapsed
  13.81    0.00   92.26
> summary(model.knn) ##Correctly Classified Instances=1934(100%)

=== Summary ===

Correctly Classified Instances        1934              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.0009
Root mean squared error                  0.0015
Relative absolute error                  0.5145 %
Root relative squared error              0.5144 %
Total Number of Instances             1934

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   j   <-- classified as
 189   0   0   0   0   0   0   0   0   0 |   a = 0
   0 198   0   0   0   0   0   0   0   0 |   b = 1
   0   0 195   0   0   0   0   0   0   0 |   c = 2
   0   0   0 199   0   0   0   0   0   0 |   d = 3
   0   0   0   0 186   0   0   0   0   0 |   e = 4
   0   0   0   0   0 187   0   0   0   0 |   f = 5
   0   0   0   0   0   0 195   0   0   0 |   g = 6
   0   0   0   0   0   0   0 201   0   0 |   h = 7
   0   0   0   0   0   0   0   0 180   0 |   i = 8
   0   0   0   0   0   0   0   0   0 204 |   j = 9
```

Now, make the prediction on validation dataset and create confusion matrix.

```
> #Confusion Matrix (kNN)
> prediction.knn <- predict(fit.knn, newdata = cv, type = "class")
> table(`Actual` = cv$Y, `Predicted` = prediction.knn)
      Predicted
Actual   0   1   2   3   4   5   6   7   8   9
     0  87   0   0   0   0   0   0   0   0   0
     1   0  96   0   0   0   0   0   1   0   0
     2   0   0  92   0   0   0   0   0   0   0
     3   0   0   1  82   0   0   0   0   1   1
     4   0   0   0   0 114   0   0   0   0   0
     5   0   0   0   1   1 105   0   0   0   1
     6   0   0   0   0   0   0  87   0   0   0
     7   0   0   0   0   0   0   0  96   0   0
     8   0   2   1   1   0   0   1   0  86   0
     9   0   0   0   0   0   1   0   0   0  88
> error.rate.knn <- sum(cv$Y != prediction.knn)/nrow(cv)
> print(paste0("Accuracy: ", 1 - error.rate.knn))
[1] "Accuracy: 0.986257928118393"
```

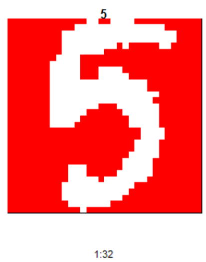
Accuracy is 98.6% => Highest among all the models.

Again, accuracy is increased by ~ 0.5% comparing to previous model.

This is the best model with error rate of only ~ 1.4%.

Predict the digit for example 1 using Random Forest. Let's see the actual digit and predicted digit. And plot the respective digit. Here actual digit is 5 and predicted digit is 5.

```
> ##Predict Digit for Example 1 (kNN)
> row <- 1
> prediction.digit <- as.vector(predict(fit.knn, newdata = cv[row, ], type = "class"))
> print(paste0("Actual Digit: ", as.character(cv$Y[row])))
[1] "Actual Digit: 5"
> print(paste0("Predicted Digit: ", prediction.digit))
[1] "Predicted Digit: 5"
> par(mfrow = c(1, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> z <- array(as.vector(as.matrix(cv[row, -1])), dim = c(32, 32))
> z <- z[, 32:1]  ##right side up
> image(1:32, 1:32, z, main = cv[row, 1], col = more_colors(1024))
```

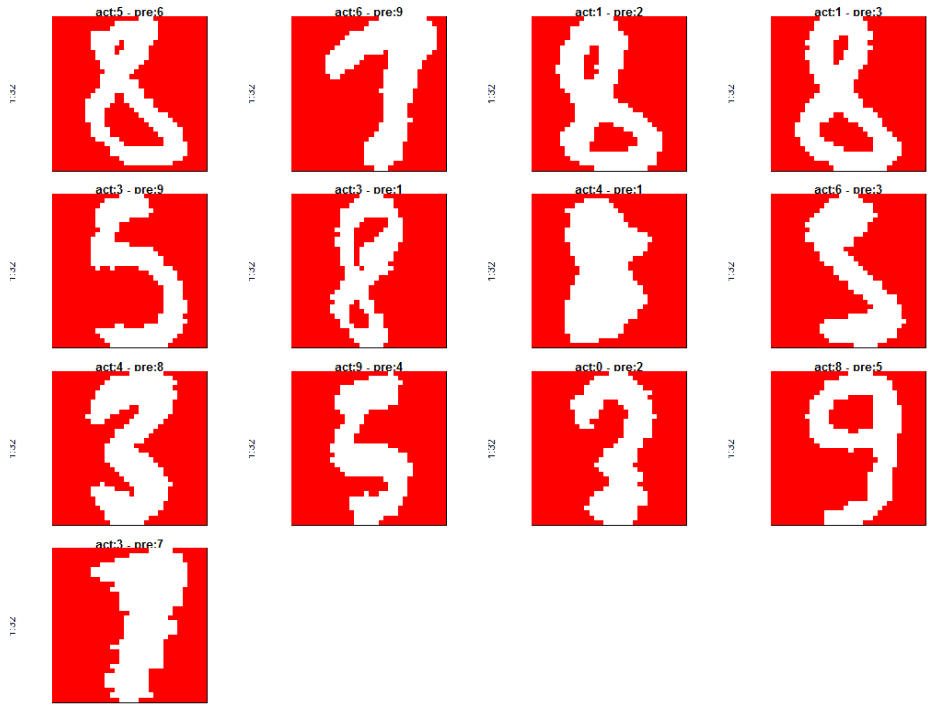



```
> ##Errors with K Nearest Neighbours (KNN)
> errors <- as.vector(which(cv$Y != prediction.knn))
> print(paste0("Error Numbers: ", length(errors)))
[1] "Error Numbers: 13"
> predicted <- as.vector(prediction.knn)
> par(mfrow = c(4, 4), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
> for (i in 1:length(errors)) {
+ z <- array(as.vector(as.matrix(cv[errors[i], -1])), dim = c(32, 32))
+ z <- z[, 32:1]  ##right side up
+ image(1:32, 1:32, z, main = paste0("act:", as.character(cv$Y[i]),
+ " - pre:", predicted[errors[i]]), col = more_colors(1024))
+ }
```

Error Numbers: 13 => The lowest error numbers among all models.

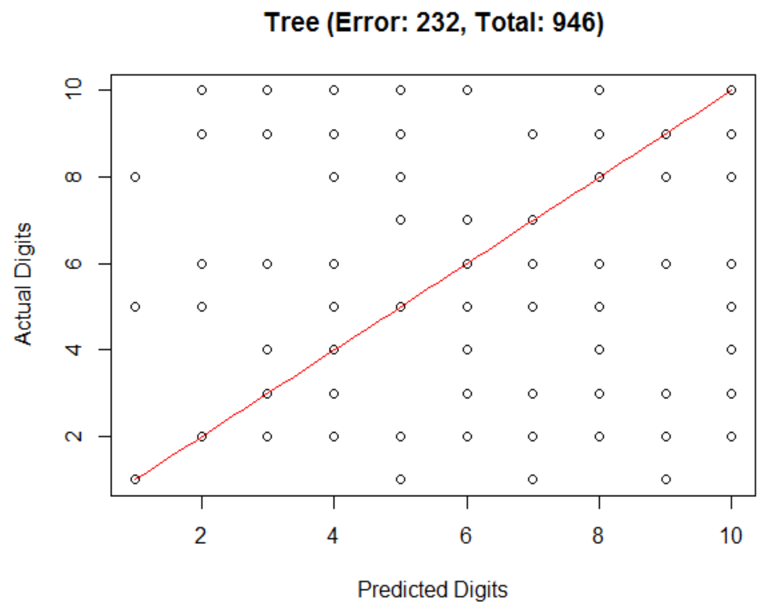
Error numbers are reduced as accuracy is further increased by 0.5%.

The below plot shows the actual digit (act) and predicted digit (pre).

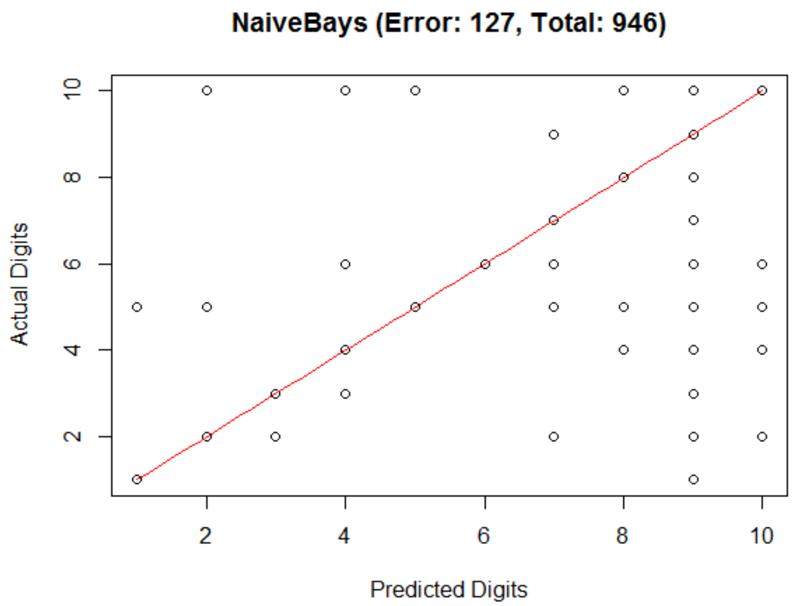


Plots: Actual Vs Predicted

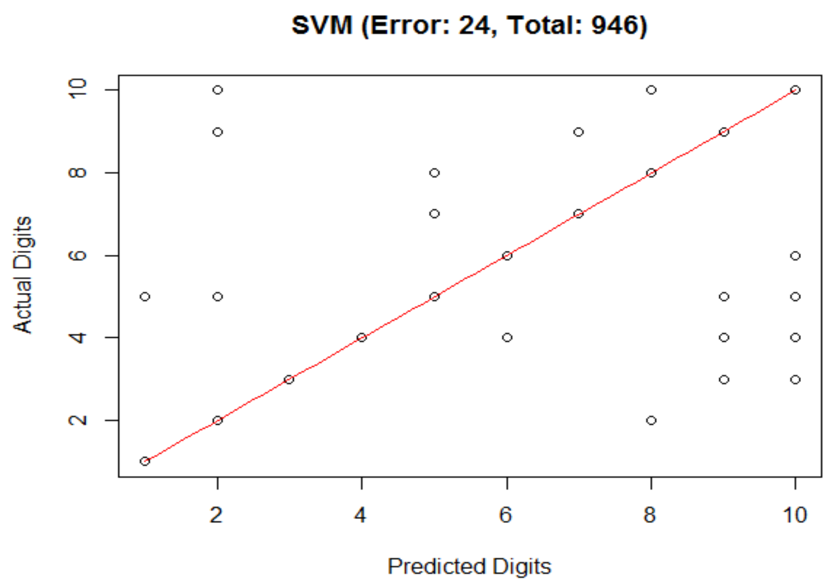
```
> ###PLOTS: Actual vs Predicted ####
>
> ### Model1: rpart
> prediction.rpart1 <- as.data.frame(prediction.rpart)
> View(prediction.rpart1)
> actual.rpart <- as.data.frame(cv$Y)
> View(actual.rpart)
> rpart <- cbind(actual.rpart, prediction.rpart1)
> View(rpart)
> write.csv(rpart, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_
files-selected\\rpartResult.csv")
> with(rpart, scatter.smooth(rpart$prediction.rpart, rpart$`cv$Y`, main="Tree
(Error: 232, Total: 946)",  xlab = "Predicted Digits", ylab = "Actual Digits"
, lpars = list(col = "red", lwd = 1, lty = 1)))
```



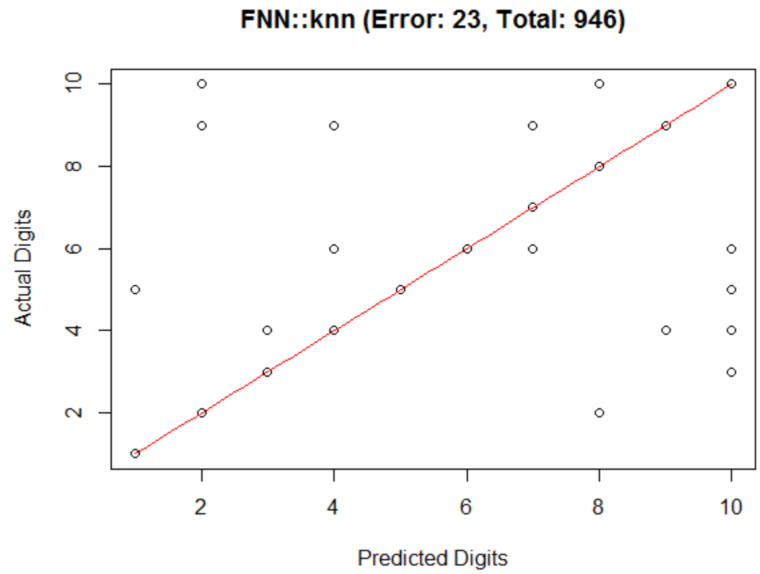

```
>
> ### Model2: Naive Bayes Algorithm
> prediction.naiveBayes1 <- as.data.frame(prediction.naiveBayes)
> View(prediction.naiveBayes1)
> actual.naiveBayes <- as.data.frame(cv$Y)
> View(actual.naiveBayes)
> naiveBayes <- cbind(actual.naiveBayes, prediction.naiveBayes1)
> View(naiveBayes)
> write.csv(rpart, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_
files-selected\\naiveBayesResult.csv")
> with(naiveBayes, scatter.smooth(naiveBayes$prediction.naiveBayes, naiveBaye
s$`cv$Y`, main="NaiveBays (Error: 127, Total: 946)",  xlab = "Predicted Digit
s", ylab = "Actual Digits", lpars = list(col = "red", lwd = 1, lty = 1)))
```



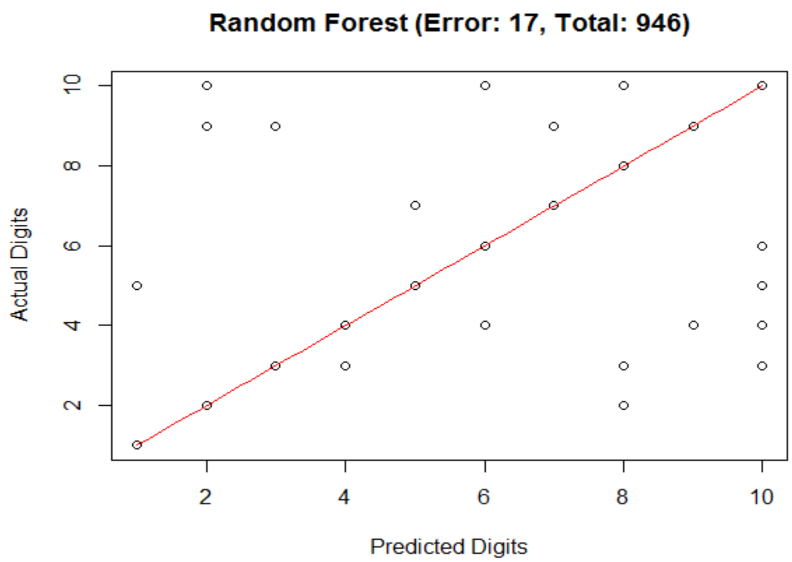

```
>
> ### Model3: Support Vector Machine
> prediction.svm1 <- as.data.frame(prediction.svm)
> View(prediction.svm1)
> actual.svm <- as.data.frame(cv$Y)
> View(actual.svm)
> svm <- cbind(actual.svm, prediction.svm1)
> View(svm)
> write.csv(knn, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_fi
les-selected\\svmResult.csv")
> with(svm, scatter.smooth(svm$prediction.svm, svm$`cv$Y`, main="SVM (Error:
24, Total: 946)",  xlab = "Predicted Digits", ylab = "Actual Digits", lpars =
list(col = "red", lwd = 1, lty = 1)))
```



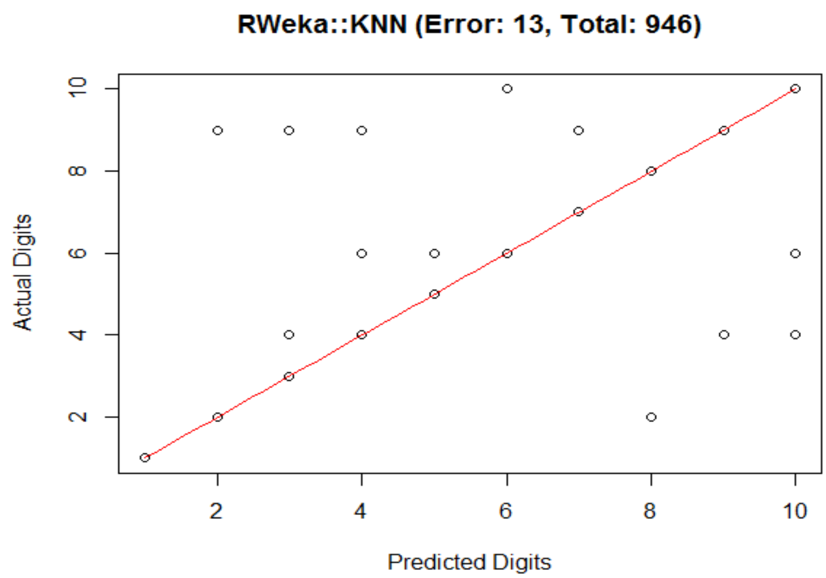

```
> ### Model4: FNN::KNN
> prediction.digit1 <- as.data.frame(prediction.digit)
> View(prediction.digit1)
> actual.fnn <- as.data.frame(cv$Y)
> View(actual.fnn)
> fnn <- cbind(actual.fnn, prediction.digit1)
> View(fnn)
> write.csv(fnn, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_fi
les-selected\\fnnResult.csv")
> scatter.smooth(fnn$`cv$Y`, fnn$prediction.digit)
> with(fnn, scatter.smooth(fnn$prediction.digit, fnn$`cv$Y`, main="FNN::knn (
Error: 23, Total: 946)",  xlab = "Predicted Digits", ylab = "Actual Digits",
lpars = list(col = "red", lwd = 1, lty = 1)))
```




```
> ### Model5: Random Forest
> prediction.randomForest1 <- as.data.frame(prediction.randomForest)
> View(prediction.randomForest1)
> actual.randomForest <- as.data.frame(cv$Y)
> View(actual.randomForest)
> randomForest <- cbind(actual.randomForest, prediction.randomForest1)
> View(randomForest)
> write.csv(randomForest, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticu
m\\uci_files-selected\\randomForestResult.csv")
> scatter.smooth(randomForest$`cv$Y`, randomForest$prediction.randomForest)
> with(randomForest, scatter.smooth(randomForest$prediction.randomForest, ran
domForest$`cv$Y`, main="Random Forest (Error: 17, Total: 946)",  xlab = "Pred
icted Digits", ylab = "Actual Digits", lpars = list(col = "red", lwd = 1, lty
= 1)))
```




```
> ### Model6: RWeka::IBk
> prediction.knn1 <- as.data.frame(prediction.knn)
> View(prediction.knn1)
> actual.knn <- as.data.frame(cv$Y)
> View(actual.knn)
> knn <- cbind(actual.knn, prediction.knn1)
> View(knn)
> write.csv(knn, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_fi
les-selected\\knnResult.csv")
> scatter.smooth(knn$`cv$Y`, knn$prediction.knn)
> with(knn, scatter.smooth(knn$prediction.knn, knn$`cv$Y`, main="RWeka::KNN (
Error: 13, Total: 946)",  xlab = "Predicted Digits", ylab = "Actual Digits",
lpars = list(col = "red", lwd = 1, lty = 1)))
```




```
>
> HandwricttenDiditsPrediction <- cbind(cv$Y, rpart$prediction.rpart, naiveBayes$prediction.naiveBayes, svm$prediction.svm, fnn$prediction.digit, randomForest$prediction.randomForest, knn$prediction.knn)
> colnames(HandwricttenDiditsPrediction)  <-  c("ActualDigits", "rprtPrediction", "naiveBayesPrediction", "svmPrediction", "FNNPrediction", "randomForestPrediction", "kNNPrediction")
> write.csv(HandwricttenDiditsPrediction, file = "C:\\Users\\Varsha\\Music\\AnalyticsPracticum\\uci_files-selected\\HandwricttenDiditsPrediction.csv")
```

Summary Of Models used

Models Used for Digits Recognition	R Package	% Accuracy of a Model	% Error of a Model	Misclassified Digits Out of Total 946	Time to Train the Model in Sec.	Model Prediction Time in Sec.
Recursive Partitioning and Regression Trees	rpart	75.48 %	24.52 %	232	3.72	0.21
Naïve Bays Algorithm	e1071	86.58 %	13.42 %	127	1.06	23.95
Support Vector Machine	e1071	97.46 %	2.54 %	24	10.87	2.08
Fast Nearest Neighbor Search Algorithm	FNN	97.57 %	2.43 %	23	6.51	6.51
Random Forest	random Forest	98.20 %	1.8 %	17	56.55	0.22
K Nearest Neighbor Algorithm	RWeka	98.63 %	1.37 %	13	13.89	11.14