Hand Writting Recognision

Tajudeen Abdulazeez

March 8, 2019

Load Packages

The following packages will be used for the analysis

```
## Packages
library(ggplot2)
library(RWeka)
library(e1071)
library(rpart)
library(randomForest)

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

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

library(caret)

## Loading required package: lattice

library(rbokeh)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(readr)
library(grid)

library(descr)
```

Data Pre-processing

The datasets used is from a competition in Kaggle, each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image. Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is lo The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column. cated on row i and column j of a 28 x 28 matrix, (indexing by zero). Accuracy will be used as the evaluation metrics in selecting the best model for digit recognizer.

Evaluation Matrics

The target label is balance. The evauation metric that will be used to determing the model performance is the accuracy score. The forllowing model will used to create a classifier and the performance of each model will be evaluated using accuracy.

Model

1. Random Forest
2. KNN
3. SVM

Load Data

```
digit_train <- read.csv('../data/wk7-data/Kaggle-digit-train.csv')
digit_test <- read.csv('../data/wk7-data/Kaggle-digit-test.csv')

#digit_train$label[1:10]
#summary(digit_train)
dim(digit_train)
```

```
## [1] 42000   785
```

```
cat('The dimension of the training data is : ', dim(digit_train), '\n')
```

```
## The dimension of the training data is :  42000 785
```

```
cat("The dimension of the testing data is : ", dim(digit_test))
```

```
## The dimension of the testing data is :  28000 785
```

Transform the label feature to ordered factor

```
# change the label to factor
digit_train$label <- as.ordered(digit_train$label)

cat('The distribution of the target label is shown below: ')

## The distribution of the target label is shown below:

table(digit_train$label)

##
##    0    1    2    3    4    5    6    7    8    9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188
```

From the abov distribution, we can see that the training datasets label is balance across all classes.

Data quality check

```
dim(digit_train[, colSums(digit_train != 0) > 0])

## [1] 42000   709

dim(digit_test[, colSums(digit_test != 0 ) > 0 ])

## [1] 28000   702
```

Create a small subset of the dataset for modeling

```
ind <- sample(2, nrow(digit_train), replace = TRUE, prob = c(0.9, 0.1))

sub90 <- digit_train[ind==1,]
sub10 <- digit_train[ind==2,]

cat('sub90 dimension: ', dim(sub90), '\n\n\n')

## sub90 dimension:  37897 785

cat('sub10 dimension:', dim(sub10))

## sub10 dimension: 4103 785
```

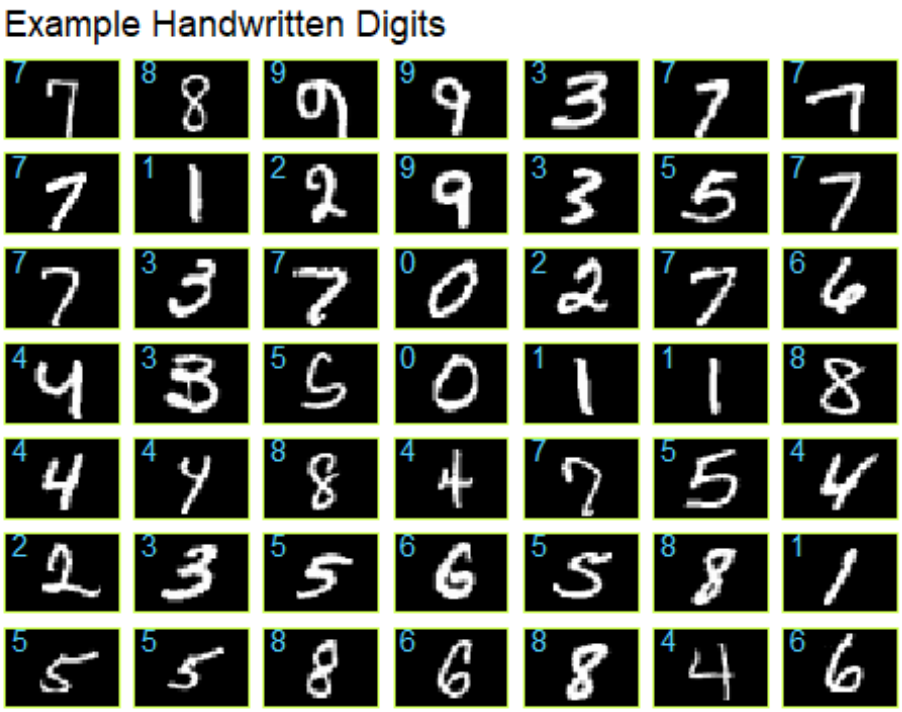
check the label distribution for sub10

```
table(sub10$label)

##
##   0   1   2   3   4   5   6   7   8   9
## 425 448 414 412 406 392 401 425 348 432
```

Visualize the datasets

```
#  References https://www.kaggle.com/jameshirschorn/example-handwritten-digits


labels   <- digit_train[,1]
features <- digit_train[,-1]

# Uncomment for reproducability
# set.seed(1)
rowsToPlot <- sample(1:nrow(digit_train), 49)

rowToMatrix <- function(row) {
  intensity <- as.numeric(row)/max(as.numeric(row))
  return(t(matrix((rgb(intensity, intensity, intensity)), 28, 28)))
}

geom_digit <- function(digits)
{
  layer(geom = GeomRasterDigit, stat = "identity", position = "identity", data = NULL,
        params = list(digits=digits))
}

GeomRasterDigit <- ggproto("GeomRasterDigit",
                           ggplot2::GeomRaster,
                           draw_panel = function(data, panel_scales, coordinates, digits = digits) {
                             if (!inherits(coordinates, "CoordCartesian")) {
                               stop("geom_digit only works with Cartesian coordinates",
                                    call. = FALSE)
                             }
                             corners <- data.frame(x = c(-Inf, Inf), y = c(-Inf, Inf))
                             bounds <- coordinates$transform(corners, panel_scales)
                             x_rng <- range(bounds$x, na.rm = TRUE)
                             y_rng <- range(bounds$y, na.rm = TRUE)

rasterGrob(as.raster(rowToMatrix(digits[data$rows,])),
                                        x = mean(x_rng), y = mean(y_rng),
                                        default.units = "native", just = c("center","center"),
                                        interpolate = FALSE)
                           })

p <- ggplot(data.frame(rows=rowsToPlot, labels=labels[rowsToPlot]),
            aes(x=0.1, y=.9, rows=rows, label=labels)) +
```

```
        geom_blank() + xlim(0,1) + ylim(0,1) + xlab("") + ylab("") +
        facet_wrap(~ rows, ncol=7) +
        geom_digit(features) +
        geom_text(colour="#53cfff") +
        theme(panel.background = element_rect(fill = 'black'),
              panel.border = element_rect(fill = NA, colour = "#cfff53"),
              panel.grid = element_blank(),
              strip.background = element_blank(),
              strip.text.x = element_blank(),
              axis.text.x = element_blank(),
              axis.text.y = element_blank(),
              axis.ticks = element_blank(),
              axis.line = element_blank()) +
        ggtitle("Example Handwritten Digits")

plot(p)
```



Split the datasets and prepared it for modeling

The sub10 will be splited into two

```
ind <- sample(2, nrow(sub10), replace = TRUE, prob = c(0.7, 0.3))
sub70 <- sub10[ind==1,]
sub30 <- sub10[ind==2,]

cat('sub70 : ', dim(sub70), '\n\n')
```

```
## sub70 :  2867 785
```

```
cat('sub30: ', dim(sub30))
```

```
## sub30:  1236 785
```

label distribution

```
table(sub70$label)
```

```
##
##   0   1   2   3   4   5   6   7   8   9
## 283 315 289 302 292 265 290 294 254 283
```

```
table(sub30$label)
```

```
##
##   0   1   2   3   4   5   6   7   8   9
## 142 133 125 110 114 127 111 131  94 149
```

The label distribution is balance.

Create train and test set

The sub30 contains about 1200 samples and the labeldistribution is propotional. In other to save computer time, this will be used to train and sub 70 will be used to test and evaluate the model

```
X_train <- sub30[,-1]
y_train <- sub30$label

X_test <- sub70[, -1]
y_test <- sub70$label

X_train_with_label <- sub30
X_test_with_label <- sub70
```

```
cat('X_train: ', dim(X_train), '\n')
```

```
## X_train:  1236 784
```

```
dim(X_test)
```

```
## [1] 2867  784
```

```
dim(X_train_with_label)
```

```
## [1] 1236  785
```

MODELING

Random Forest

Random forest is developed by aggregation of trees. This can be used for both regression and classification problems. It can deal with large number of features and avoid overfitting. `params : ntree > default = 500, mtry > default = sq.root(p) for classification, p/3 for regression.` p is the number of features.

Model 1

The first random forest model is build with the default parameters using the randomForest packages in R.

```
(rf_model1 <- randomForest(label ~ ., data = X_train_with_label))
```

```
##
## Call:
##  randomForest(formula = label ~ ., data = X_train_with_label)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 28
##
##         OOB estimate of  error rate: 10.52%
## Confusion matrix:
##     0   1   2  3  4   5   6   7  8   9 class.error
## 0 137   0   0  0  0   0   2   0  3   0 0.035211268
## 1   0 132   1  0  0   0   0   0  0   0 0.007518797
## 2   3   1 110  0  2   1   2   3  1   2 0.120000000
## 3   3   1   5 84  0   8   0   3  2   4 0.236363636
## 4   0   0   0  0 98   0   2   0  1  13 0.140350877
## 5   3   0   0  4  2 111   3   1  1   2 0.125984252
## 6   3   0   0  0  1   2 105   0  0   0 0.054054054
## 7   1   0   1  0  2   0   0 123  0   4 0.061068702
## 8   0   5   1  6  3   4   0   0 70   5 0.255319149
## 9   1   0   1  3  4   2   0   2  0 136 0.087248322
```

```
print(rf_model1)
```

```
##
## Call:
##  randomForest(formula = label ~ ., data = X_train_with_label)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 28
##
##         OOB estimate of  error rate: 10.52%
## Confusion matrix:
##     0   1   2  3  4   5   6   7  8   9 class.error
## 0 137   0   0  0  0   0   2   0  3   0 0.035211268
```

```
## 1   0 132   1  0  0   0   0   0  0   0 0.007518797
## 2   3   1 110  0  2   1   2   3  1   2 0.120000000
## 3   3   1   5 84  0   8   0   3  2   4 0.236363636
## 4   0   0   0  0 98   0   2   0  1  13 0.140350877
## 5   3   0   0  4  2 111   3   1  1   2 0.125984252
## 6   3   0   0  0  1   2 105   0  0   0 0.054054054
## 7   1   0   1  0  2   0   0 123  0   4 0.061068702
## 8   0   5   1  6  3   4   0   0 70   5 0.255319149
## 9   1   0   1  3  4   2   0   2  0 136 0.087248322
```

```
rf_model1$mtry
```

```
## [1] 28
```

Make prediction on new datasets

```
prediction  <- predict(rf_model1, X_test)
```

The confusion matrix of the model performance. The method is from the package caret.

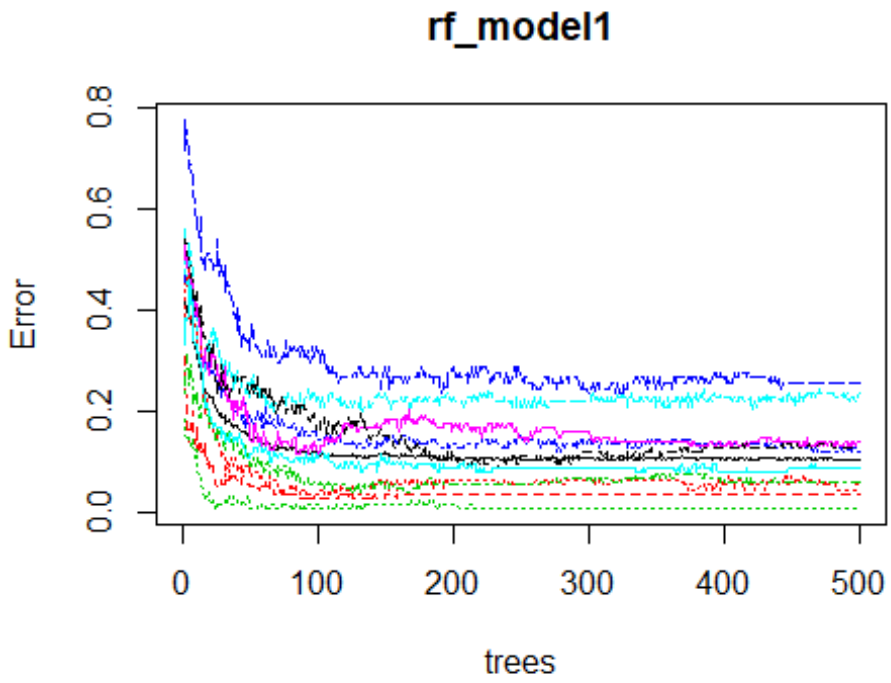
```
confusionMatrix(prediction, y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 275   0   1   0   0   1   4   1   0   5
##          1   0 307   1   4   4   9   0   7   7   5
##          2   0   2 267   5   3   1   0  10   8   1
##          3   0   0   2 261   0   6   0   1  13   4
##          4   0   2   4   1 263   1   5   1   4  10
##          5   1   1   1   8   0 232   7   0  11   4
##          6   4   0   5   0   1   3 272   0   2   0
##          7   1   1   5   6   0   0   1 271   1   9
##          8   2   0   0   2   1   3   1   0 196   0
##          9   0   2   3  15  20   9   0   3  12 245
##
## Overall Statistics
##
##                Accuracy : 0.903
##                  95% CI : (0.8916, 0.9136)
##     No Information Rate : 0.1099
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8922
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97173   0.9746  0.92388  0.86424  0.90068  0.87547
## Specificity           0.99536   0.9855  0.98836  0.98986  0.98913  0.98732
```

```
## Pos Pred Value         0.95819   0.8924  0.89899  0.90941  0.90378  0.87547
## Neg Pred Value         0.99690   0.9968  0.99144  0.98411  0.98874  0.98732
## Prevalence             0.09871   0.1099  0.10080  0.10534  0.10185  0.09243
## Detection Rate         0.09592   0.1071  0.09313  0.09104  0.09173  0.08092
## Detection Prevalence   0.10010   0.1200  0.10359  0.10010  0.10150  0.09243
## Balanced Accuracy      0.98354   0.9801  0.95612  0.92705  0.94491  0.93139
##                       Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.93793  0.92177  0.77165  0.86572
## Specificity            0.99418  0.99067  0.99656  0.97523
## Pos Pred Value         0.94774  0.91864  0.95610  0.79288
## Neg Pred Value         0.99302  0.99106  0.97821  0.98514
## Prevalence             0.10115  0.10255  0.08859  0.09871
## Detection Rate         0.09487  0.09452  0.06836  0.08546
## Detection Prevalence   0.10010  0.10290  0.07150  0.10778
## Balanced Accuracy      0.96606  0.95622  0.88410  0.92048
```

Error rate

```
plot(rf_model1)
```

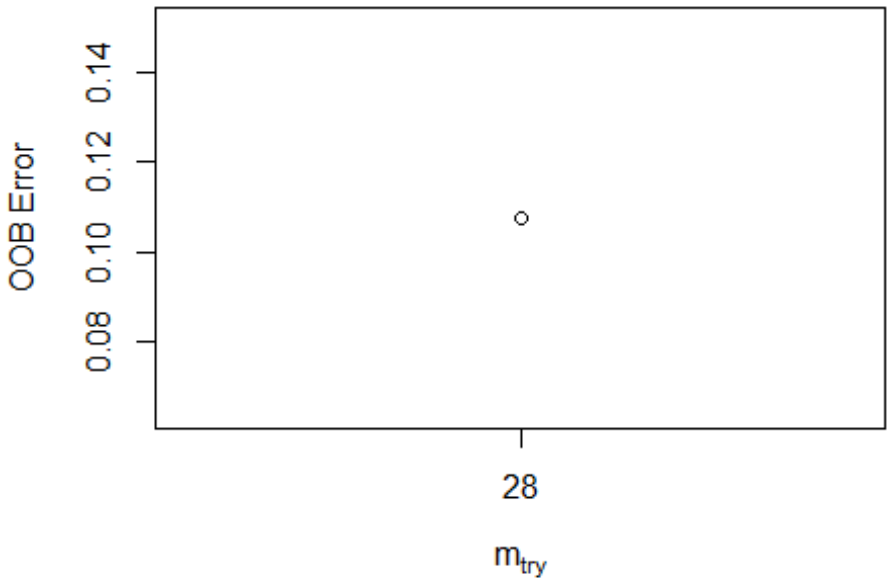


When number of tree is around 200, the error rate become contant and no significatnt improvement.

Tune the model parameters

ntreeTry = 200

```
tuneRF(X_train, y_train,
       stepFactor = 1,
       plot = TRUE,
       ntreeTry = 200,
       trace = TRUE,
       improve = 0.1)

## mtry = 28  OOB error = 10.76%
## Searching left ...
## Searching right ...
```




```
##        mtry  OOBError
## 28.OOB   28 0.1076052
```

Random Forest Parameters

ntree = 200 mtry = 28

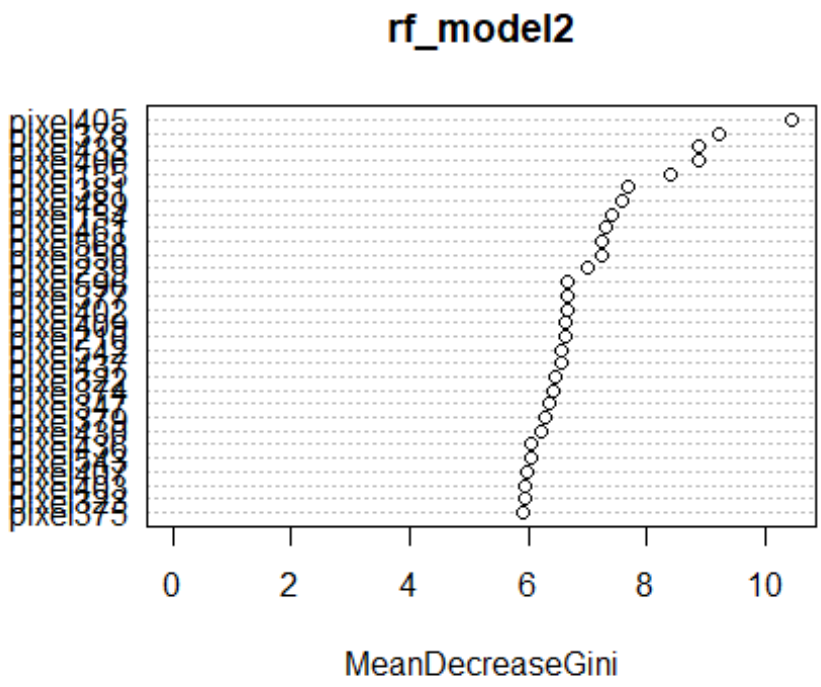
Build a random forest model with ntree = 200, mtry =28

Model 2

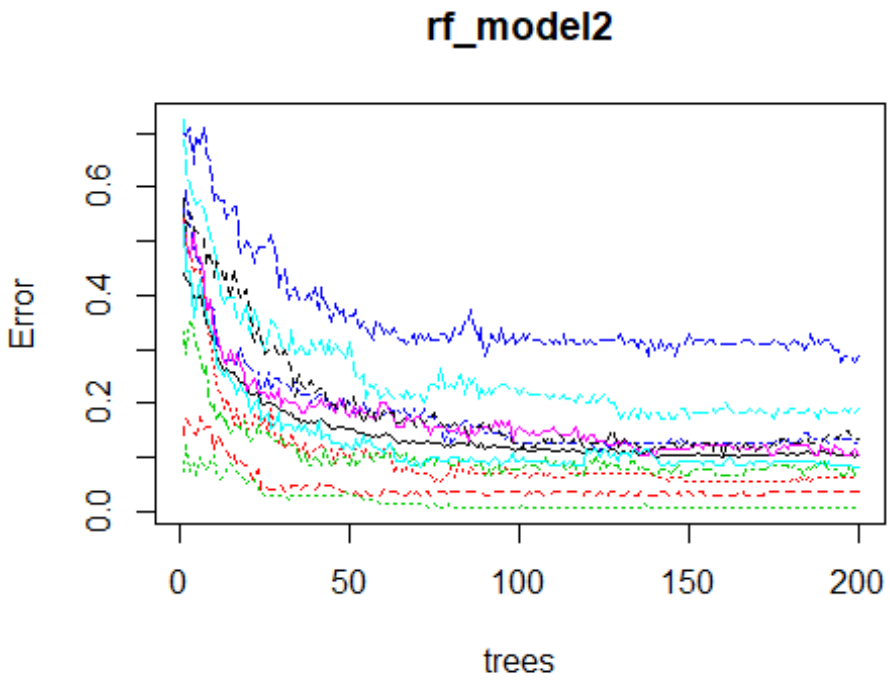
```
rf_model2 <- randomForest(label~.,data = X_train_with_label, ntree=200,
mtry=28)

print(rf_model2)
```

```
##
## Call:
##  randomForest(formula = label ~ ., data = X_train_with_label,      ntree =
200, mtry = 28)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 28
##
##         OOB estimate of  error rate: 10.36%
## Confusion matrix:
##     0   1   2  3   4   5   6   7  8   9 class.error
## 0 137   0   0  0   0   0   3   0  2   0 0.035211268
## 1   0 132   1  0   0   0   0   0  0   0 0.007518797
## 2   3   2 109  1   2   1   3   3  1   0 0.128000000
## 3   3   2   2 89   0   8   0   3  1   2 0.190909091
## 4   0   0   1  0 102   0   1   1  1   8 0.105263158
## 5   4   1   1  5   3 110   1   2  0   0 0.133858268
## 6   4   0   0  0   0   3 104   0  0   0 0.063063063
## 7   0   0   2  0   2   1   0 121  0   5 0.076335878
## 8   0   5   1  8   2   5   1   0 67   5 0.287234043
## 9   1   0   1  2   4   2   0   2  0 137 0.080536913
```

```
varImpPlot(rf_model2)
```

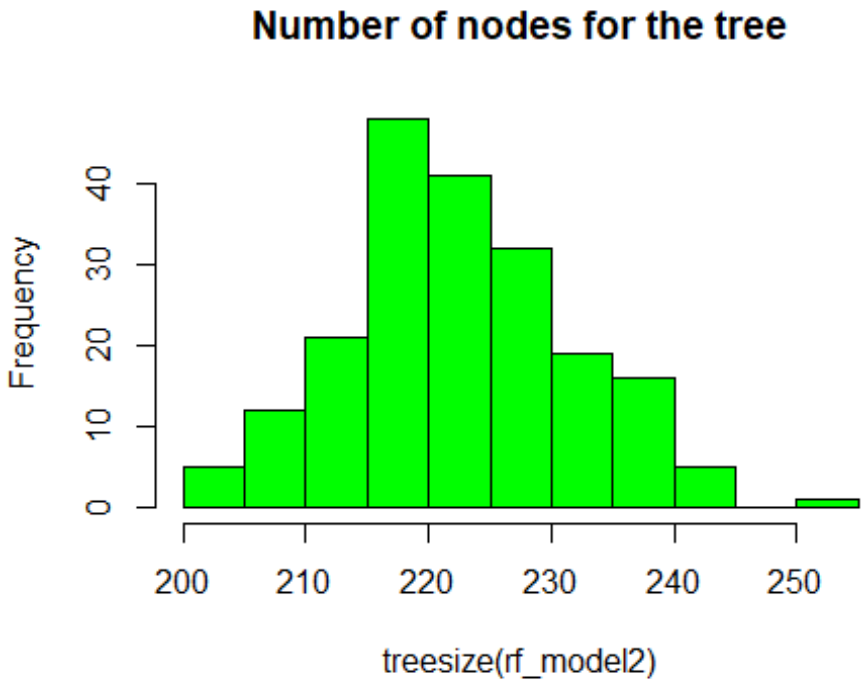



```
plot(rf_model2)
```

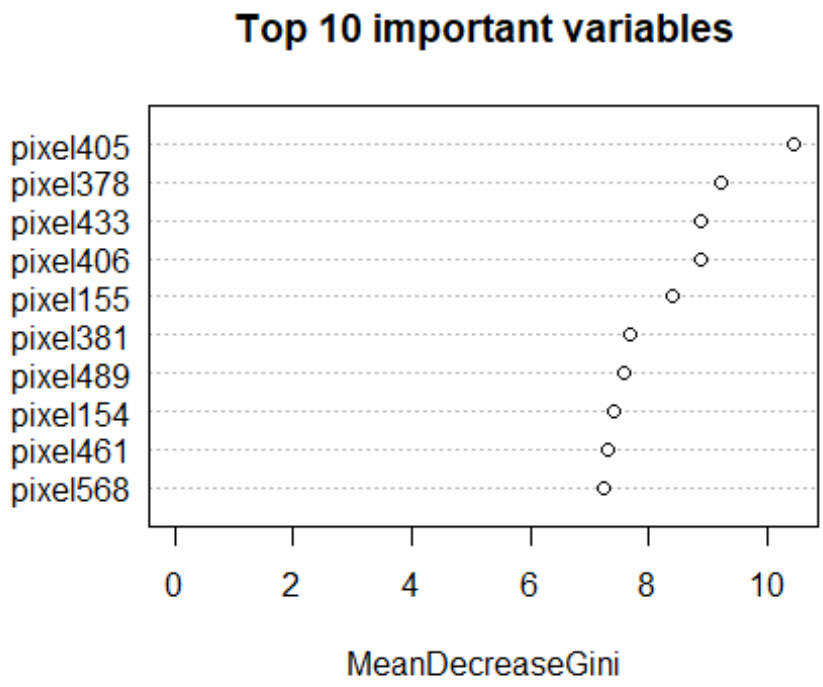


Tree size

```
hist(treesize(rf_model2), main = "Number of nodes for the tree", col='green')
```



```
#h <- figure(width = 600, height = 400, xlab = 'Tree Size') %>%
  #ly_hist(treesize(rf_model2), breaks = 40, freq = FALSE) %>%
 # ly_density(treesize(rf_model2))
#h

varImpPlot(rf_model2, sort = TRUE, n.var = 10, main = "Top 10 important
variables")
```



Prediction

```
p2 <- predict(rf_model2, X_test)

confusionMatrix(p2, y_test)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 275   0   1   0   0   1   5   1   0   5
##          1   0 306   3   5   5   9   1   7   7   5
##          2   0   2 262   7   3   2   0   8   7   2
##          3   0   1   3 256   0   8   0   1  13   2
##          4   0   2   5   1 258   1  10   3   3  13
##          5   1   0   0  13   0 230   4   0  10   5
##          6   4   0   7   0   3   5 267   0   2   0
##          7   1   1   4   5   0   0   1 270   1   9
##          8   2   0   1   0   1   2   2   0 198   0
##          9   0   3   3  15  22   7   0   4  13 242
```

```
##
## Overall Statistics
##
##                Accuracy : 0.8943
##                  95% CI : (0.8825, 0.9053)
##     No Information Rate : 0.1099
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8825
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.97173   0.9714  0.90657  0.84768  0.88356  0.86792
## Specificity           0.99497   0.9835  0.98798  0.98908  0.98524  0.98732
## Pos Pred Value        0.95486   0.8793  0.89420  0.90141  0.87162  0.87452
## Neg Pred Value        0.99690   0.9964  0.98951  0.98219  0.98678  0.98656
## Prevalence            0.09871   0.1099  0.10080  0.10534  0.10185  0.09243
## Detection Rate        0.09592   0.1067  0.09138  0.08929  0.08999  0.08022
## Detection Prevalence  0.10045   0.1214  0.10220  0.09906  0.10324  0.09173
## Balanced Accuracy     0.98335   0.9775  0.94727  0.91838  0.93440  0.92762
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           0.92069  0.91837  0.77953  0.85512
## Specificity           0.99185  0.99145  0.99694  0.97407
## Pos Pred Value        0.92708  0.92466  0.96117  0.78317
## Neg Pred Value        0.99108  0.99068  0.97896  0.98397
## Prevalence            0.10115  0.10255  0.08859  0.09871
## Detection Rate        0.09313  0.09418  0.06906  0.08441
## Detection Prevalence  0.10045  0.10185  0.07185  0.10778
## Balanced Accuracy     0.95627  0.95491  0.88823  0.91460
```

prepared submission to kaggle

```
prediction <- predict(rf_model2, digit_test)

submission<-data.frame(ImageId=1:nrow(digit_test),Label=prediction)


write.csv(submission, file="rfsubmission.csv", row.names=F)
```

Save the model

```
saveRDS(rf_model2, './randomForest_model2_digit.rds')
```

KNN

KNN is a supervised learning algorithm, which means we are given a labelled training sample with its corresponding training features to train the model.

Model 1

```
k <- round(sqrt(nrow(X_train)))

knn_Model <- class::knn(train = X_train, test = X_test, cl=y_train, k=k, prob = TRUE)
```

Model Performance

```
table(knn_Model, y_test)
```

```
##          y_test
## knn_Model   0   1   2   3   4   5   6   7   8   9
##         0 265   0   8   1   0   2   8   1   6   2
##         1   1 313  47  35  24  30  16  27  45  12
##         2   0   0 199   1   0   0   0   1   1   0
##         3   0   0   6 237   0  14   0   0  16   2
##         4   1   0   5   0 186   2  10   1   5   1
##         5   0   0   0   8   0 194   5   0   8   1
##         6   9   0   6   0   4   5 251   0   3   0
##         7   2   1  12   6   2   1   0 253   2  14
##         8   1   0   2   1   0   2   0   0 139   0
##         9   4   1   4  13  76  15   0  11  29 251
```

```
CrossTable(x = y_test, y = knn_Model,prop.chisq=F)
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
================================================================================
==================
##           knn_Model
## y_test        0       1       2       3       4       5       6       7
8       9   Total
## --------------------------------------------------------------------------
---------------------
## 0           265       1       0       0       1       0       9       2
1       4     283
##           0.936   0.004   0.000   0.000   0.004   0.000   0.032   0.007
0.004   0.014   0.099
##           0.904   0.002   0.000   0.000   0.005   0.000   0.032   0.007
0.007   0.010
##           0.092   0.000   0.000   0.000   0.000   0.000   0.003   0.001
0.000   0.001
## --------------------------------------------------------------------------
---------------------
```

```
## 1             0     313       0       0       0       0       0       1
0       1     315
##           0.000   0.994   0.000   0.000   0.000   0.000   0.000   0.003
0.000   0.003   0.110
##           0.000   0.569   0.000   0.000   0.000   0.000   0.000   0.003
0.000   0.002
##           0.000   0.109   0.000   0.000   0.000   0.000   0.000   0.000
0.000   0.000
## ------------------------------------------------------------------------
---------------------
## 2             8      47     199       6       5       0       6      12
2       4     289
##           0.028   0.163   0.689   0.021   0.017   0.000   0.021   0.042
0.007   0.014   0.101
##           0.027   0.085   0.985   0.022   0.024   0.000   0.022   0.041
0.014   0.010
##           0.003   0.016   0.069   0.002   0.002   0.000   0.002   0.004
0.001   0.001
## ------------------------------------------------------------------------
---------------------
## 3             1      35       1     237       0       8       0       6
1      13     302
##           0.003   0.116   0.003   0.785   0.000   0.026   0.000   0.020
0.003   0.043   0.105
##           0.003   0.064   0.005   0.862   0.000   0.037   0.000   0.020
0.007   0.032
##           0.000   0.012   0.000   0.083   0.000   0.003   0.000   0.002
0.000   0.005
## ------------------------------------------------------------------------
---------------------
## 4             0      24       0       0     186       0       4       2
0      76     292
##           0.000   0.082   0.000   0.000   0.637   0.000   0.014   0.007
0.000   0.260   0.102
##           0.000   0.044   0.000   0.000   0.882   0.000   0.014   0.007
0.000   0.188
##           0.000   0.008   0.000   0.000   0.065   0.000   0.001   0.001
0.000   0.027
## ------------------------------------------------------------------------
---------------------
## 5             2      30       0      14       2     194       5       1
2      15     265
##           0.008   0.113   0.000   0.053   0.008   0.732   0.019   0.004
0.008   0.057   0.092
##           0.007   0.055   0.000   0.051   0.009   0.898   0.018   0.003
0.014   0.037
##           0.001   0.010   0.000   0.005   0.001   0.068   0.002   0.000
0.001   0.005
## ------------------------------------------------------------------------
---------------------
```

```
## 6             8      16       0       0      10       5     251       0
0       0     290
##           0.028   0.055   0.000   0.000   0.034   0.017   0.866   0.000
0.000   0.000   0.101
##           0.027   0.029   0.000   0.000   0.047   0.023   0.903   0.000
0.000   0.000
##           0.003   0.006   0.000   0.000   0.003   0.002   0.088   0.000
0.000   0.000
## ------------------------------------------------------------------------
---------------------
## 7             1      27       1       0       1       0       0     253
0      11     294
##           0.003   0.092   0.003   0.000   0.003   0.000   0.000   0.861
0.000   0.037   0.103
##           0.003   0.049   0.005   0.000   0.005   0.000   0.000   0.863
0.000   0.027
##           0.000   0.009   0.000   0.000   0.000   0.000   0.000   0.088
0.000   0.004
## ------------------------------------------------------------------------
---------------------
## 8             6      45       1      16       5       8       3       2
139      29     254
##           0.024   0.177   0.004   0.063   0.020   0.031   0.012   0.008
0.547   0.114   0.089
##           0.020   0.082   0.005   0.058   0.024   0.037   0.011   0.007
0.959   0.072
##           0.002   0.016   0.000   0.006   0.002   0.003   0.001   0.001
0.048   0.010
## ------------------------------------------------------------------------
---------------------
## 9             2      12       0       2       1       1       0      14
0     251     283
##           0.007   0.042   0.000   0.007   0.004   0.004   0.000   0.049
0.000   0.887   0.099
##           0.007   0.022   0.000   0.007   0.005   0.005   0.000   0.048
0.000   0.621
##           0.001   0.004   0.000   0.001   0.000   0.000   0.000   0.005
0.000   0.088
## ------------------------------------------------------------------------
---------------------
## Total       293     550     202     275     211     216     278     293
145     404    2867
##           0.102   0.192   0.070   0.096   0.074   0.075   0.097   0.102
0.051   0.141
##
========================================================================
=================
```

Accuracy

```
cm = as.matrix(table(Actual = y_test, Predicted = knn_Model))

n = sum(cm) # number of instances
diag = diag(cm) # number of correctly classified instances per class



accuracy = sum(diag) / n

cat("The accuracy for the KNN  model is : ", accuracy, '\n', 'With k = ', k)
```

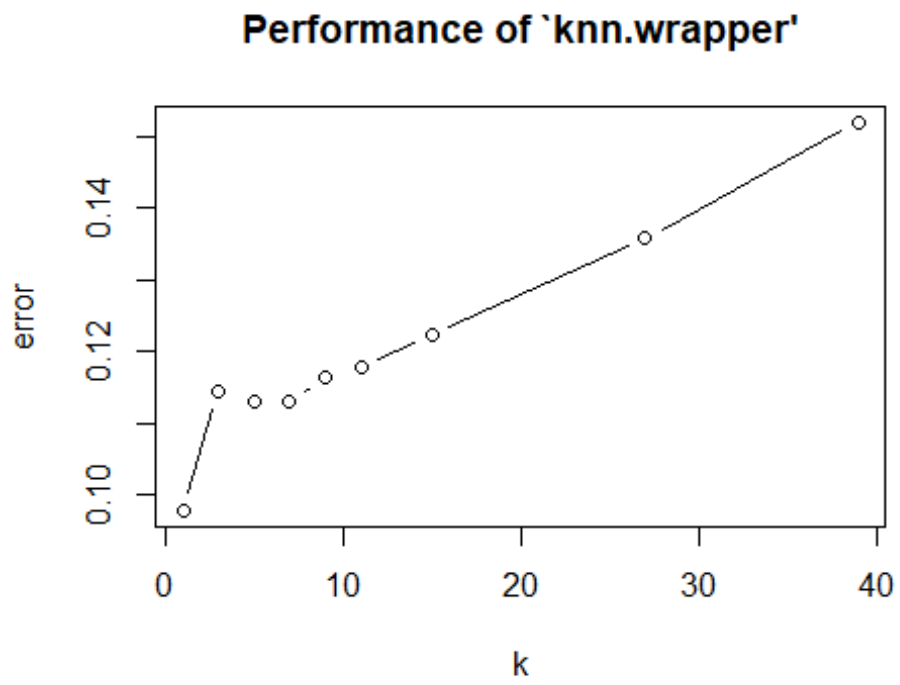
```
## The accuracy for the KNN  model is :  0.7980467
##  With k =  35
```

Tune K

```
k <- c(1,3,5,7,9,11,15,27,39)
(knnmodel <- tune.knn(X_test,  y_test , k = k, tunecontrol =
tune.control(sampling = "boot")))
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: bootstrapping
##
## - best parameters:
##  k
##  1
##
## - best performance: 0.09786604
```

```
plot(knnmodel)
```



The error rate did not converge,due to time constraints, the whole training set provided can not be used.

SVM

Surport vector machine is machine learning algorithm that seperate data using hyperplane.it can be used and perform well with data's that has non-regularity. i.e the distribution is not known.

The svm used for this analysis is from e1071 packages

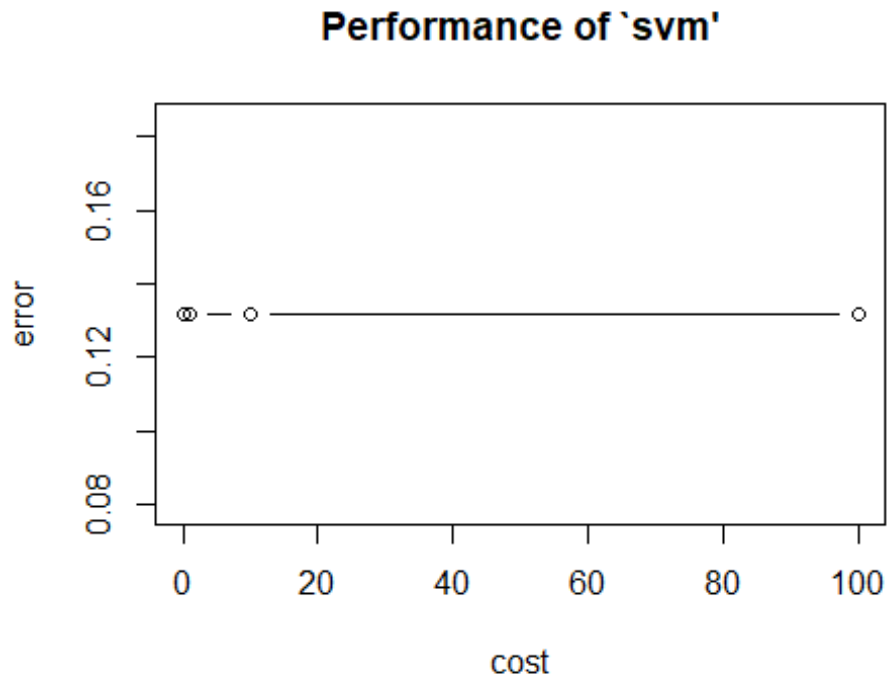
Lineal Kernel

```
tuned_cost <- tune(svm,label~., data=X_train_with_label,
                   kernel="linear",
                   ranges=list(cost=c(.01,.1,1,10,100,100)))
```

```
print(tuned_cost)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.1318516
```

```
plot(tuned_cost)
```



Polinomial Kernel

```
SVM_model_P <- svm(label~., data=X_test_with_label,
                   kernel="polynomial", cost=.1,
                   scale=FALSE)

print(SVM_model_P)
```

```
##
## Call:
```

```
## svm(formula = label ~ ., data = X_test_with_label, kernel = "polynomial",
##     cost = 0.1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.1
##      degree:  3
##       gamma:  0.00127551
##      coef.0:  0
##
## Number of Support Vectors:  1169

predict_svm <- predict(SVM_model_P, X_test)
```

Confusion Matrix

```
confusionMatrix(predict_svm, y_test)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 283   0   0   0   0   0   0   0   0   0
##          1   0 315   0   0   0   0   0   0   0   0
##          2   0   0 289   0   0   0   0   0   0   0
##          3   0   0   0 302   0   0   0   0   0   0
##          4   0   0   0   0 292   0   0   0   0   0
##          5   0   0   0   0   0 265   0   0   0   0
##          6   0   0   0   0   0   0 290   0   0   0
##          7   0   0   0   0   0   0   0 294   0   0
##          8   0   0   0   0   0   0   0   0 254   0
##          9   0   0   0   0   0   0   0   0   0 283
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9987, 1)
##     No Information Rate : 0.1099
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
```

```
## Sensitivity           1.00000   1.0000   1.0000   1.0000   1.0000  1.00000
## Specificity           1.00000   1.0000   1.0000   1.0000   1.0000  1.00000
## Pos Pred Value        1.00000   1.0000   1.0000   1.0000   1.0000  1.00000
## Neg Pred Value        1.00000   1.0000   1.0000   1.0000   1.0000  1.00000
## Prevalence            0.09871   0.1099   0.1008   0.1053   0.1018  0.09243
## Detection Rate        0.09871   0.1099   0.1008   0.1053   0.1018  0.09243
## Detection Prevalence  0.09871   0.1099   0.1008   0.1053   0.1018  0.09243
## Balanced Accuracy     1.00000   1.0000   1.0000   1.0000   1.0000  1.00000
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            1.0000   1.0000  1.00000  1.00000
## Specificity            1.0000   1.0000  1.00000  1.00000
## Pos Pred Value         1.0000   1.0000  1.00000  1.00000
## Neg Pred Value         1.0000   1.0000  1.00000  1.00000
## Prevalence             0.1012   0.1025  0.08859  0.09871
## Detection Rate         0.1012   0.1025  0.08859  0.09871
## Detection Prevalence   0.1012   0.1025  0.08859  0.09871
## Balanced Accuracy      1.0000   1.0000  1.00000  1.00000
```

Radial Kernel

```
SVM_model_P <- svm(label~., data=X_test_with_label,
                   kernel="radial", cost=.1,
                   scale=FALSE)

print(SVM_model_P)

##
## Call:
## svm(formula = label ~ ., data = X_test_with_label, kernel = "radial",
##     cost = 0.1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##       gamma:  0.00127551
##
## Number of Support Vectors:  2867

predict_svm <- predict(SVM_model_P, X_test)

confusionMatrix(predict_svm, y_test)
```

Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0   0   0   0   0   0   0   0   0   0   0
##          1 283 315 289 302 292 265 290 294 254 283
##          2   0   0   0   0   0   0   0   0   0   0
##          3   0   0   0   0   0   0   0   0   0   0
##          4   0   0   0   0   0   0   0   0   0   0
##          5   0   0   0   0   0   0   0   0   0   0
##          6   0   0   0   0   0   0   0   0   0   0
##          7   0   0   0   0   0   0   0   0   0   0
##          8   0   0   0   0   0   0   0   0   0   0
##          9   0   0   0   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.1099
##                  95% CI : (0.0987, 0.1219)
##     No Information Rate : 0.1099
##     P-Value [Acc > NIR] : 0.5088
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.00000   1.0000   0.0000   0.0000   0.0000  0.00000
## Specificity           1.00000   0.0000   1.0000   1.0000   1.0000  1.00000
## Pos Pred Value            NaN   0.1099      NaN      NaN      NaN      NaN
## Neg Pred Value        0.90129      NaN   0.8992   0.8947   0.8982  0.90757
## Prevalence            0.09871   0.1099   0.1008   0.1053   0.1018  0.09243
## Detection Rate        0.00000   0.1099   0.0000   0.0000   0.0000  0.00000
## Detection Prevalence  0.00000   1.0000   0.0000   0.0000   0.0000  0.00000
## Balanced Accuracy     0.50000   0.5000   0.5000   0.5000   0.5000  0.50000
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.0000   0.0000  0.00000  0.00000
## Specificity            1.0000   1.0000  1.00000  1.00000
## Pos Pred Value            NaN      NaN      NaN      NaN
## Neg Pred Value         0.8988   0.8975  0.91141  0.90129
## Prevalence             0.1012   0.1025  0.08859  0.09871
## Detection Rate         0.0000   0.0000  0.00000  0.00000
## Detection Prevalence   0.0000   0.0000  0.00000  0.00000
## Balanced Accuracy      0.5000   0.5000  0.50000  0.50000
```

Model	Accuracy	Parameters with Best Performance	Parameter Try
Random Forest	0.95	ntree=200, mtry =28	ntree= 0 -500
KNN	0.78	K=35	`k <- c(1,3,5,7,9,11,15,27,39)`
SVM Lineal Kernel	0.13	C = 0.1	`cost=c(.01,.1,1,10,100,100))`
SVM Polynomial Kernel	1	C = 0.1	`c(.01,.1,1,10,100,100)`
SVM Polynomial Kernel	0.1	C=0.1	

The best performing model for the hand writing recognition is the SVM with polynomial kernel and C=0.1 with and accuracy of 1 for a subset of the unseen data and kappa measured of 1.

The lineal kernel and the radial kernel are unable to accurately predict the hand writing with kappa measure of 0.

Random forest also did very well in predicting the hand writing digit with an accuracy of about 95% for unseen datasets and accuracy of 96% on Kaggle submission.