Kevin_pollard_HW8a

Kevin Pollard

5/30/2020

Contents

Overview and Intro

One of the biggest challenges in machine learning is computer vision, which is the field of training computers to correctly recognize imagery in a similar way a human would. In the case of machine learning there is typically a use case associated with computer vision. For example, in a bank one of the use cases is to train the computer to recognize fraudulent check images, both from the check's text and the check's pixels. In case of the later banks are looking for things that are out of the norm for checks they process for business to business transactions.

The computer vision challenge for the analysis to follow has a somewhat more common use case, which involves training the computer to recognize hand-written numbers. The data to train the machine is from the widely-known database called MNIST. It is a good database for people who want to try pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting.

In the analysis to follow, several machine learning methods will be tested using the MNIST data. Each will be compared to determine, which is the best for this specific task. Additionally, methods will be incorporated to address the high-dimensional nature of the data as it can create technical complications with some machine learning methods.

```
# Load Libraries - Use pacman to wrap library and package related functions and names them in an intuit

if (require(pacman) == FALSE) {
  install.packages("pacman")
}
```

```
## Loading required package: pacman
```

```
## Warning: package 'pacman' was built under R version 3.5.3
```

```
#pacman::p_unload("all")
#loadedNamespaces()

## Make sure your current packages are up to date
#update.packages()
## devtools is required
#install.packages("devtools")
require(devtools)
```

```
## Loading required package: devtools
```

```
## Warning: package 'devtools' was built under R version 3.5.3
```

```
## Loading required package: usethis
```

```
## Warning: package 'usethis' was built under R version 3.5.3
```

```
library(devtools)
#install_github("trinker/pacman")


pacman::p_load(tidyverse, lubridate, magrittr, data.table, #ddply, #sqldf,
               # calling python in r
               janitor, #,BiocManager,hclust,
               # styles to tables
               kableExtra, knitr, flextable,
               # visualizations
               #highcharter, leaflet, corrplot,
              # machine learning
               caret,caretEnsemble,Rtsne, randomForest,naivebayes,e1071, #modelr, fastDummies, doParall
               #plotting
               GGally,corrplot,ggExtra,ggthemes,cowplot,plotly,cowplot,
               #maps
               leaflet,ggmap,RColorBrewer,
               #R.utils
               R.utils, readr,FactoMineR,factoextra,rattle
```

```
                )


#as.data.frame(loadedNamespaces())
```

```
#require(caret)
#install.packages("caretEnsemble")
# library(caretEnsemble)
# library(doParallel)
# #install.packages("PerformanceAnalytics")
# library(PerformanceAnalytics)

# #install.packages("")
# library(FactoMineR)
# library(factoextra)
# #install.packages("caret", dependencies=c("Depends", "Suggests"))
# #install.packages("MLeval")
# library(MLeval)
# library(caret)
# library(cowplot)
#-------------------------------
# #install.packages("NbClust")
# #install.packages("ggsignif")
# library(ggsignif)
# library(NbClust)
# #install.packages("tidyverse")
# # flextable is good for knitting to word
# #install.packages("flextable")
# #install.packages("huxtable")
# library(magrittr)
# library(flextable)
# library(tidyverse)
# library(kableExtra)
# library(GGally)
# #library(sqldf)
# library(ggExtra)
# library(gridExtra)
#library('ggthemes') # visualization
# library('ggridges') # visualization
# library('corrplot') # visualisation
#====================================================
# install.packages("arulesViz")
# install.packages("arulesViz")
# install.packages("caTools")
#install.packages("arulesCBA", dependencies=c("Depends", "Suggests"))
# library(arules)
# library(caTools)
# library(arulesViz)
# library(arulesCBA)
```

```
# library(cluster)
#library(mclust)

#text
# library(wordcloud)
# library(tm)
# #NLP
# library(slam)
# #install.packages("quanteda")
# library(quanteda)
# #
# library(SnowballC)
# library(proxy)
#
# library(stringi)
# library(Matrix)
# library(tidytext)
#library(plyr)
#library(ggplot2)

 #theme_set(theme_cowplot())

#kmeans animation
#install.packages("animation")


# suppressPackageStartupMessages(library(DT))
#suppressPackageStartupMessages(library(rio))
#suppressPackageStartupMessages(library(car))
#install.packages("sf")
# #library(sf)
# #suppressPackageStartupMessages(library(sf))
# suppressPackageStartupMessages(library(sp))
# suppressPackageStartupMessages(library(maps))
# #suppressPackageStartupMessages(library(rnaturalearth))
# suppressPackageStartupMessages(library(rnaturalearthdata))
# #suppressPackageStartupMessages(library(spatstat))
# suppressPackageStartupMessages(library(tidyverse))
# suppressPackageStartupMessages(library(ggcorrplot))
# suppressPackageStartupMessages(library(circlize))
```

```
col34="dodgerblue4"
col2="blue"
col1="slateblue"
col="#E64141"
Covid19="E64141"

# caption used for certain the charts
#caption <- "Covid-19 analysis"

theme_1 <- function (base_size = 11, base_family = "") {
ggplot2::theme_set(theme_gray())
suppressWarnings(
```

```
theme_update(
  axis.text.x = element_text(size = 16)
, axis.text.y = element_text(size = 16)
, axis.title.x = element_text(size = 16)
, axis.title.y = element_text(size = 16)
, legend.title = element_text(size = 16)
, legend.text  = element_text(size = 14)
, panel.grid.major = element_blank()
, panel.grid.minor = element_blank()
, strip.background = element_blank()
, panel.margin = unit(0, "lines")
, legend.key.size = unit(.55, "cm")
, legend.key = element_rect(fill = "white")
, panel.margin.y = unit(0.5, "lines")
, panel.border = element_rect(
    colour = "black", fill = NA, size = 1)
, strip.text.x = element_text(
    size = 16, colour = "blue", face = "bold")
  )
)
}

#
# theme_2 <-
#     theme_gray() +
#     theme(
#       axis.title.y = element_blank(),
#       legend.title = element_blank(),
#       legend.position = "top")
#
# #set global ggtheme
# gbl_ggtheme <- theme_1()
# gbl_ggtheme <- theme_2
# #,panel.background = element_rect(fill = "#E64141") #Covid Red #E64141
#
#
# #https://www.rapidtables.com/web/color/RGB_Color.html
```

```
# gets the percentages for each column that has NA's
NASummaryInfoPct <- function(indf){
  a<- gather(indf %>% summarise_all(~(sum(is.na(.))/n())), key = "columns", value = "percent_missing")

 return(a)
}

# gets the percentages for each column that has NA's
NASummaryInfoCnt <- function(indf){
  b<- gather(indf %>% summarise_all(~(sum(is.na(.)))), key = "columns", value = "count_missing")

 return(b)
}
```

```r
#Return multiple data frames in one object using the S4 class system.
setClass(
    "twodf",
    representation(
        df1 = "vector",
        df2 = "vector",
        df3 = "vector"
    )
)

#Return multiple data frames in one object using the S4 class system.
setClass(
    "mdl",
    representation(
        trn = "vector",
        tst = "vector",
        hld = "vector"
    )
)

#Function modelData accepts one data frame and spits back a trainingset and testset
modelData <- function (df1 = df_ww) {
  z <- df1
    # get number of rows in the data frame
  len <- as.numeric(dim(z)[1])
  randIndex <- sample(as.numeric(rownames(z)[1:len]))
  # setting cutpoint to 2/3 of the input data frame
  cutPoint2_3 <- floor(2 * len/3)
  # setting cutpoint to 1/3 of the input data frame
  cutPoint1_3 <- floor(1 * len/3)
  trn1 <- z[randIndex[1:cutPoint2_3],]
  tst1 <- z[randIndex[cutPoint2_3+1:c(len-cutPoint2_3)],]
  hld1 <- z[randIndex[1: cutPoint1_3],]

  mdl <- new("mdl",
                 trn = trn1,
                 tst = tst1,
                 hld = hld1 )
  return(mdl)

}


modelData2 <- function (df1 = df_ww) {
  z <- df1
    # get number of rows in the data frame
  len <- as.numeric(dim(z)[1])
  randIndex <- sample(as.numeric(rownames(z)[1:len]))
  # setting cutpoint to 2/3 of the input data frame
  cutPoint80 <- floor(.8 * len)
  # setting cutpoint to 1/3 of the input data frame
  cutPoint20 <- floor(.2 * len)
  trn2 <- z[randIndex[1:cutPoint80],]
```

```
  tst2 <- z[randIndex[cutPoint80+1:c(len-cutPoint80)],]
  hld2 <- z[randIndex[1: cutPoint20],]

  mdl <- new("mdl",
                  trn = trn2,
                  tst = tst2,
                  hld = hld2  )
  return(mdl)

}


require(caret)
modelDataC <- function (df1 = df,trnpct=.8) {
  z <- df1
  len <- as.numeric(dim(z)[1])
    # get number of rows in the data frame
    inTrain<- createDataPartition(y=z$label, p=trnpct, list=FALSE)
    trn<-z[inTrain,]
    tst<-z[-inTrain,]
    #create holdout
    #trnpct <- 1-trnpct
    #cutPoint <- floor(trnpct * len)
    inTest<- createDataPartition(y=tst$label, p=trnpct, list=FALSE)
    tst2 <-z[inTest,]

  mdl <- new("mdl",
                  trn = trn,
                  tst = tst,
                  hld = tst2  )
  return(mdl)

}
```

Section 1: About the Data

Load Data

There are 42000 rows and 785 columns. The first column "label" shows what number the pixel data represents.

```
filename <-"train.csv"
DigitTotalDF<- read.csv(filename)
#nrow(DigitTotalDF)
DigitTotalDF$label<-as.factor(DigitTotalDF$label)
dim(DigitTotalDF)
```

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

```
#head(DigitTotalDF[,1:5])
```

```
#DigitTotalDF <- DigitTotalDF[1:20000,]

x <- modelDataC(DigitTotalDF,.25)
train <-x@trn
test <- x@tst
hold <- x@hld

trn <- as.data.frame(dim(train))
names(trn) <- c("dimension_training_train")
tst <-as.data.frame(dim(test))
names(tst) <- c("dimension_testing_test")
hld <-as.data.frame(dim(hold))
names(hld) <- c("dimension_testing_holdout")

DigitDF <- train
df <- DigitDF
head(df[,1:5])
```

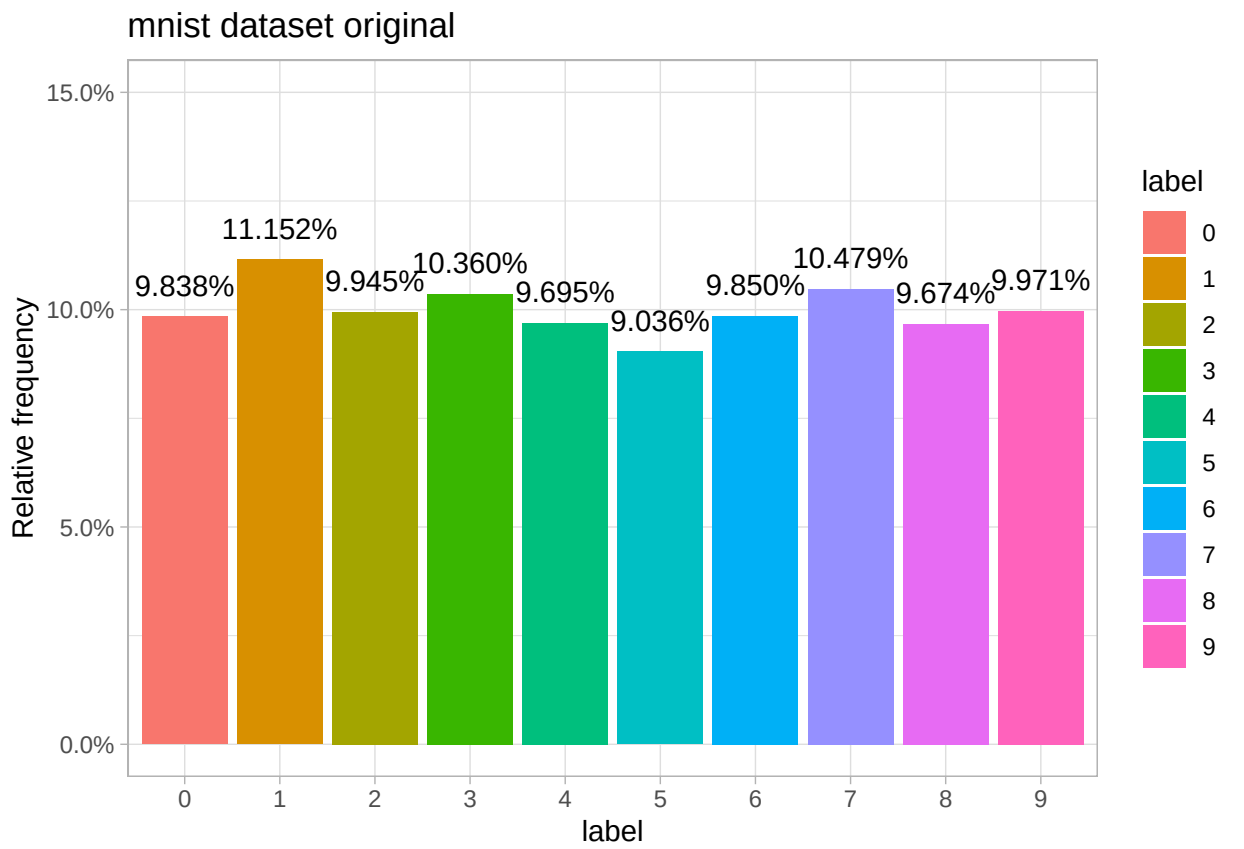
```
##    label pixel0 pixel1 pixel2 pixel3
## 7      7      0      0      0      0
## 10     3      0      0      0      0
## 17     2      0      0      0      0
## 19     7      0      0      0      0
## 26     3      0      0      0      0
## 29     9      0      0      0      0
```

```
#dim(DigitDF)
```

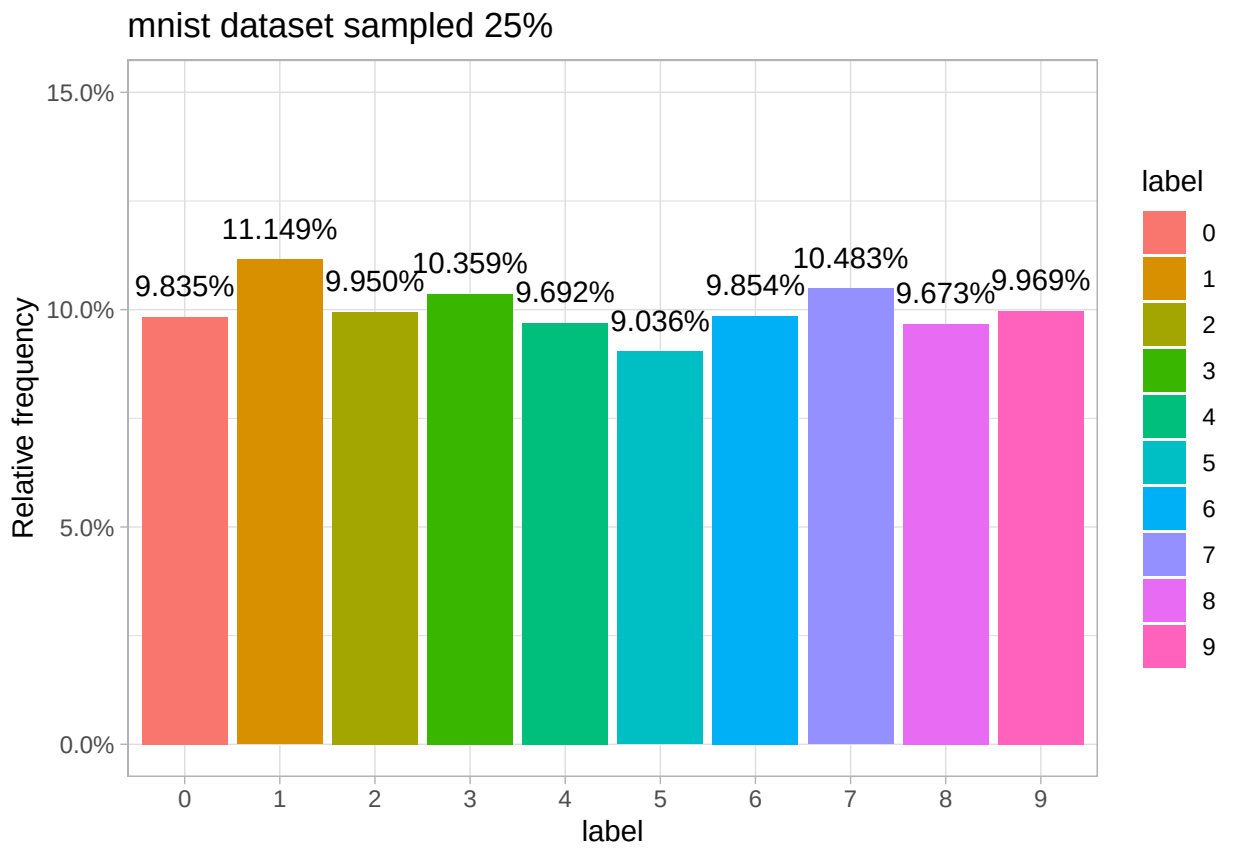
EDA

The original and the sampled 25% look very similar so we are safe using the sample to model digit predictions.

```
EDA <- DigitTotalDF

## Distribution of digits across all data sets

plot1 <- ggplot(EDA, aes(x = label, y = (..count..)/sum(..count..),fill = label)) + geom_bar() + theme_
                labs(y = "Relative frequency", title = "mnist dataset original") +
                scale_y_continuous(labels=scales::percent, limits = c(0 , 0.15)) +
                geom_text(stat = "count",
                          aes(label = scales:: percent((..count..)/sum(..count..)), vjust = -1))


plot1
```

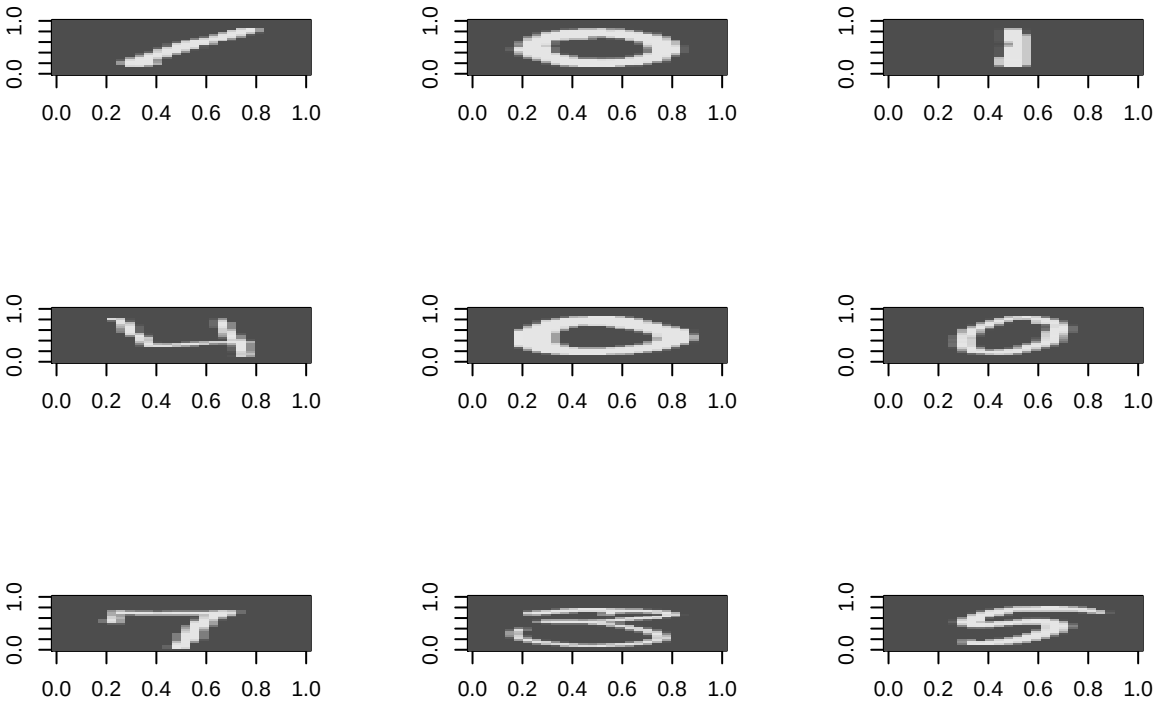


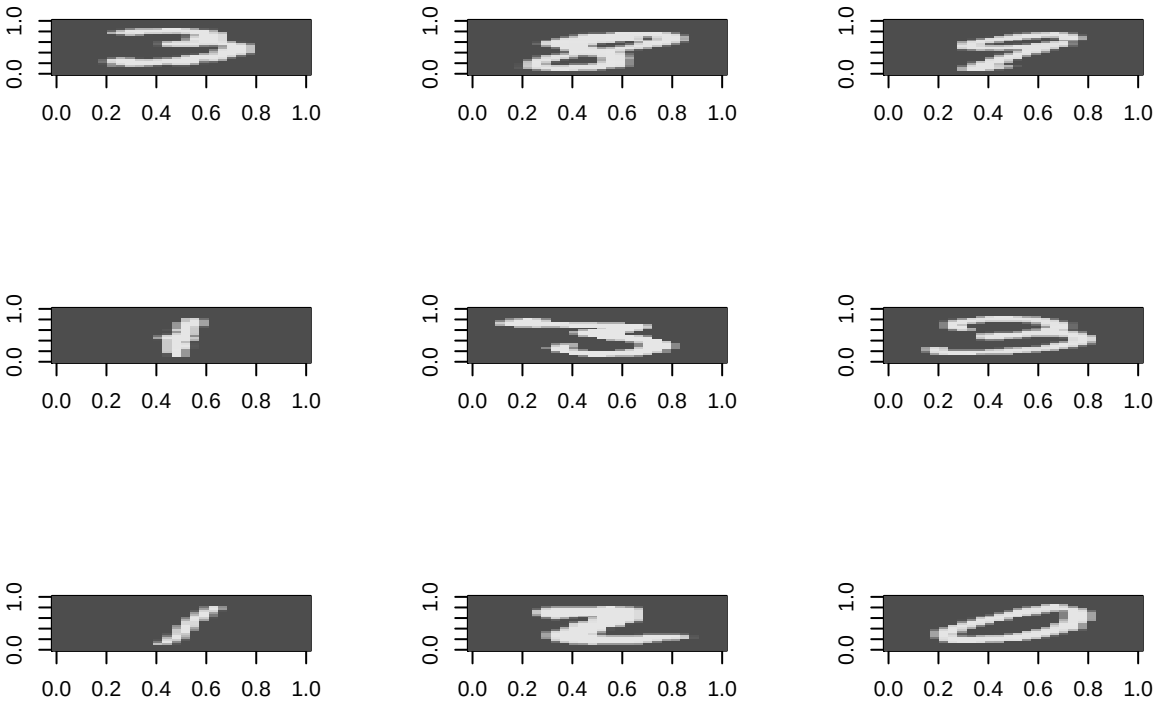

```
EDA2 <- df

## Distribution of digits across all data sets

plot2 <- ggplot(EDA2, aes(x = label, y = (..count..)/sum(..count..),fill = label)) + geom_bar() + theme_
                labs(y = "Relative frequency", title = "mnist dataset sampled 25%") +
                scale_y_continuous(labels=scales::percent, limits = c(0 , 0.15)) +
                geom_text(stat = "count",
                          aes(label = scales:: percent((..count..)/sum(..count..)), vjust = -1))


plot2
```

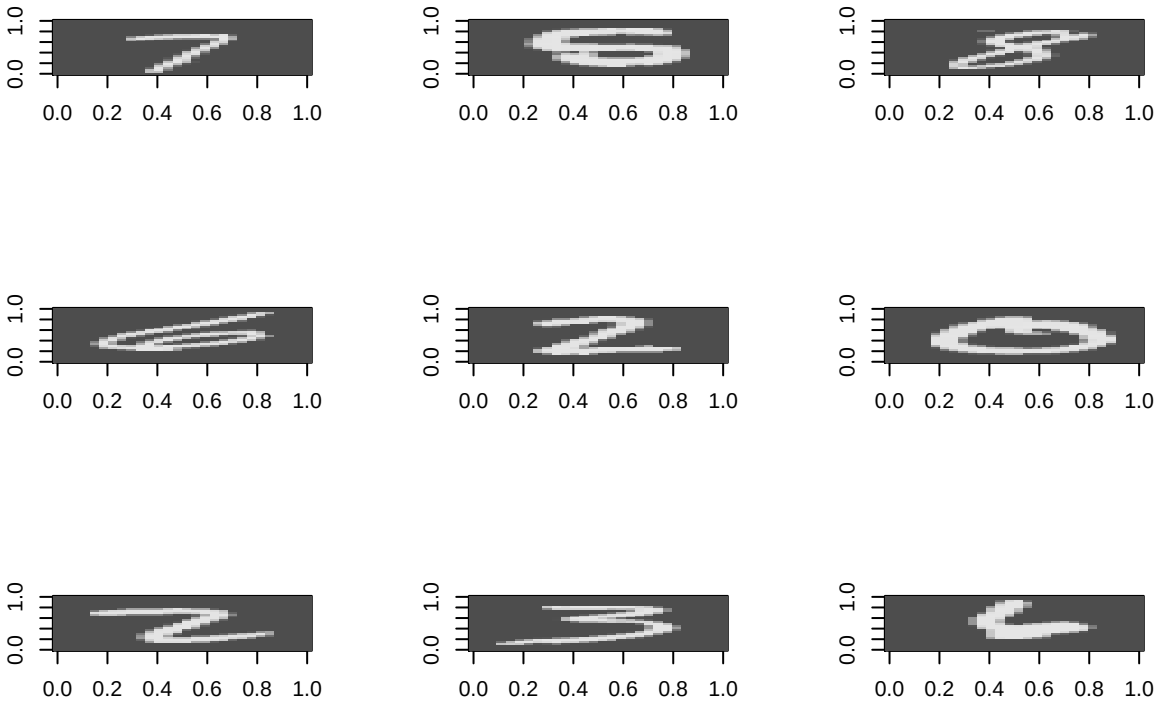


Plots of digit images using full data set.

```
flip <- function(matrix){
    apply(matrix, 2, rev)
}

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


0.0 1.0
0.0 0.2 0.4 0.6 0.8 1.0


0.0 1.0
0.0 0.2 0.4 0.6 0.8 1.0



Principal Component Analysis

Principal Component Analysis (PCA) is a multivariate technique that allows us to summarize the systematic patterns of variations in the data.

From a data analysis standpoint, PCA is used for studying one table of observations and variables with the main idea of transforming the observed variables into a set of new variables, the principal components, which are uncorrelated and explain the variation in the data. For this reason, PCA allows to reduce a "complex" data set to a lower dimension in order to reveal the structures or the dominant types of variations in both the observations and the variables.

```
#install.packages("Rtsne") # Install Rtsne package from CRAN
#require("Rtsne")
library("Rtsne") # Load package
tsne <- Rtsne::Rtsne(df[1:300,-1], dims = 2, perplexity=20, verbose=TRUE, max_iter = 500)
```

```
## Performing PCA
## Read the 300 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 20.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.03 seconds (sparsity = 0.278600)!
## Learning embedding...
## Iteration 50: error is 60.995978 (50 iterations in 0.05 seconds)
```

```
## Iteration 100: error is 60.653222 (50 iterations in 0.05 seconds)
## Iteration 150: error is 60.471431 (50 iterations in 0.05 seconds)
## Iteration 200: error is 60.392176 (50 iterations in 0.07 seconds)
## Iteration 250: error is 60.322964 (50 iterations in 0.06 seconds)
## Iteration 300: error is 1.048608 (50 iterations in 0.03 seconds)
## Iteration 350: error is 0.941828 (50 iterations in 0.03 seconds)
## Iteration 400: error is 0.913278 (50 iterations in 0.02 seconds)
## Iteration 450: error is 0.894376 (50 iterations in 0.04 seconds)
## Iteration 500: error is 0.883279 (50 iterations in 0.03 seconds)
## Fitting performed in 0.43 seconds.
```

```
require(plotly)
colors = rainbow(length(unique(df$label)))
names(colors) = unique(df$label)
# plot(tsne$Y, t='n', main="tsne")
# text(tsne$Y, labels=df$label, col=colors[df$label])

tsne_plot <- data.frame(x = tsne$Y[,1], y = tsne$Y[,2], col = tsne$Y[,1])
tplt <- ggplot(tsne_plot) + geom_point(aes(x=x, y=y, color=col))

 ggplotly(tplt)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

There are several methods for doing PCA using R, but we're going use princomp method, since its in base R stats.

Typical PCA results should consist of the following:

- A set of eigenvalues
- A table with the scores or Principal Components (PCs)
- A table of loadings (or correlations between variables and PCs).

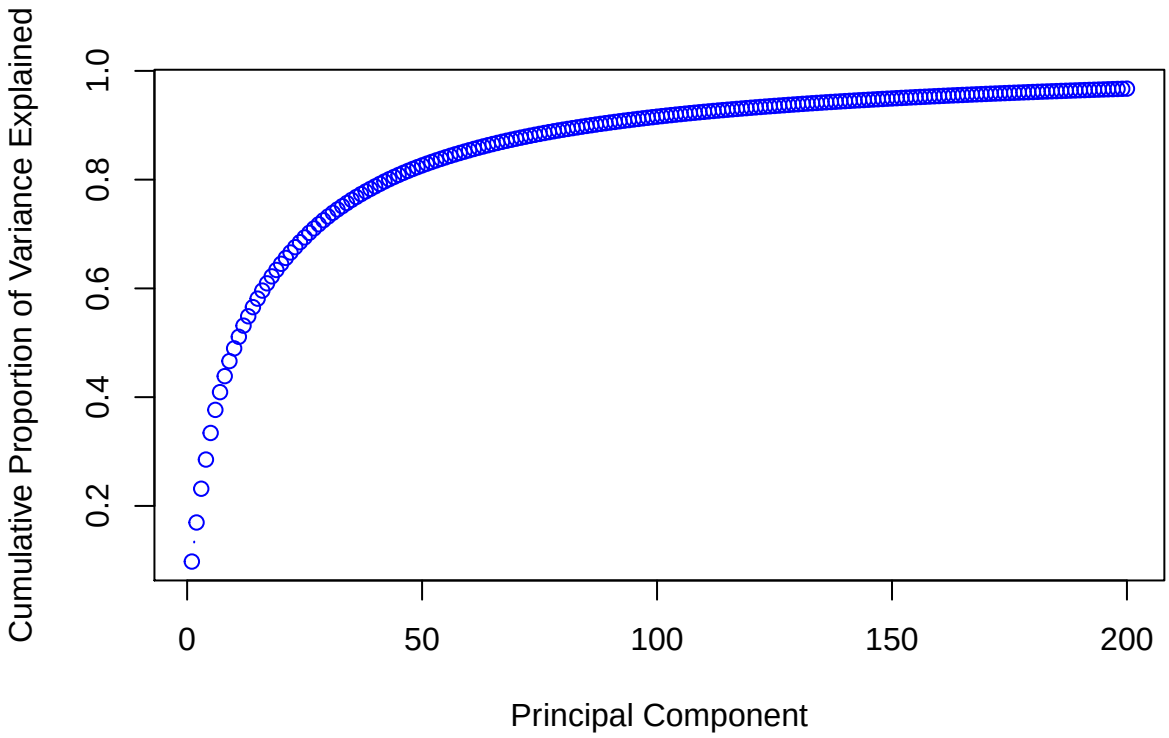
The eigenvalues provide information of the variability in the data. The scores provide information about the structure of the observations. The loadings (or correlations) allow you to get a sense of the relationships between variables, as well as their associations with the extracted PCs.

```
features<-df[,-1] #remove the label
pca<-princomp(features)
std_dev <- pca[1:260]$sdev
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)

res.pca <- pca
```

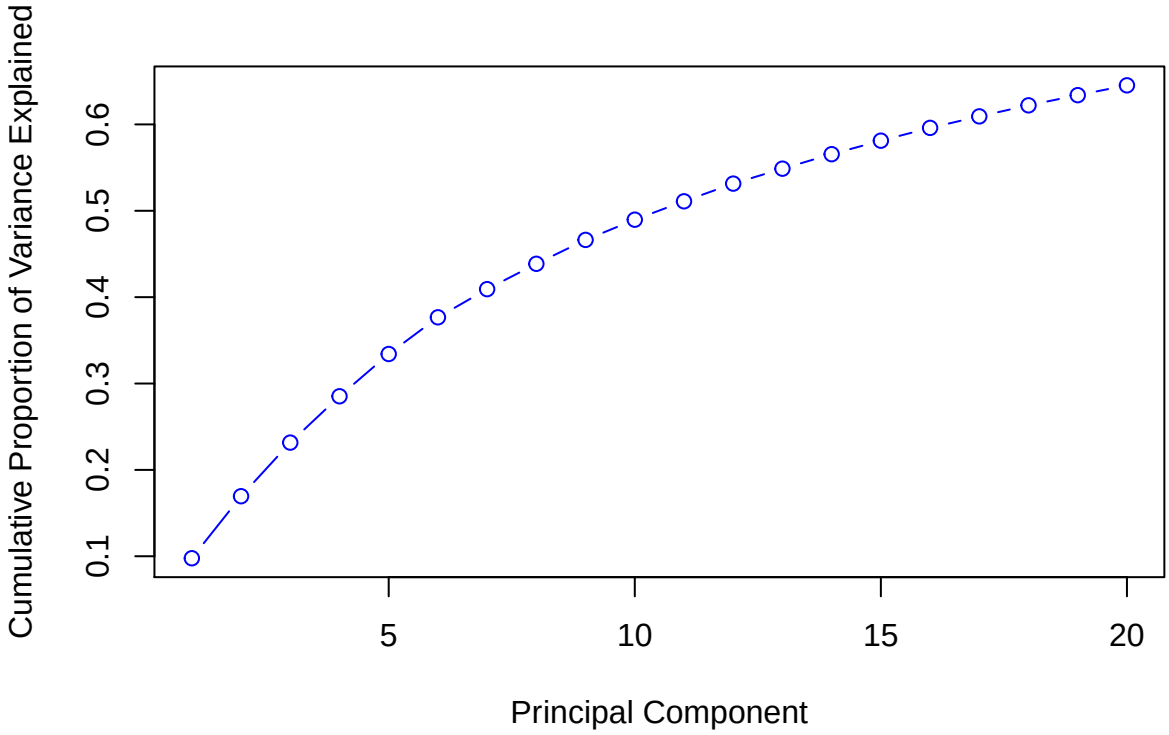
As seen in the plot below, nearly first 200 components explains most variation in the data. The second plot indicates 20 components might be adequate for the modeling data sets.

```
require(factoextra)
require(plotly)
plot(cumsum(prop_varex[1:200]), xlab = "Principal Component",
```

```
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b",
     col = 4)
```




```
plot(cumsum(prop_varex[1:20]), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b",
     col = 4)
```



The screeplot shows the first component is explaining 10% of the variances. The line drops of quickly from there, which bolsters the case that we can us far less than 784 features to get a good predcition.

```
require(factoextra)
require(plotly)
res.pca <- pca
fplt <- fviz_eig(res.pca)

ggplotly(fplt)
```

Section 2: Model Build

Model Data Sets

New Data Sets using PCA leverage the 20 components out of the original 784 dimensions. If the PCA analysis is successful, then predictions of 90% or better can still be obtained. The processing time should go down drastically when compared to training with all dimensions.

```
nDim <- 20

set.seed(275)
new_digit<-data.frame(label = df[, "label"], pca$scores)
df <- new_digit
dim(df)
```

```
## [1] 10503   785
```

```
pca_df<- new_digit[,1:nDim]
samp_size <-floor(0.80* nrow(pca_df))
train_ind <-sample(seq_len(nrow(pca_df)), size = samp_size)
train <- pca_df[train_ind,]
dim(train)
```

```
## [1] 8402   20
```

```
test  <-pca_df[-train_ind,]
dim(test)
```

```
## [1] 2101   20
```

Final Data Prep

Transforming label variable, so 1 = "one", 2 = "two" , etc. Splitting data into model data sets using 80/20 splits.

```
#original non-PCA data
df$label=dplyr::recode(df$label, `0`="zero", `1`="one", `2`="two",`3`="three",`4`="four",`5`="five",`6`=

#PCA limited data to 260 dims
pca_df$label=dplyr::recode(pca_df$label, `0`="zero", `1`="one", `2`="two",`3`="three",`4`="four",`5`="f

# head(df[,1:5],10)
# head(pca_df[,1:5],10)
# dim(pca_df)
```

```
require(kableExtra)
require(caret)
modelDataC <- function (df1 = df,trnpct=.8) {
  z <- df1
  len <- as.numeric(dim(z)[1])
    # get number of rows in the data frame
    inTrain<- createDataPartition(y=z$label, p=trnpct, list=FALSE)
    trn<-z[inTrain,]
    tst<-z[-inTrain,]
    #create holdout
    #trnpct <- 1-trnpct
    #cutPoint <- floor(trnpct * len)
    inTest<- createDataPartition(y=tst$label, p=trnpct, list=FALSE)
    tst2 <-z[inTest,]

  mdl <- new("mdl",
                  trn = trn,
                  tst = tst,
                  hld = tst2  )
  return(mdl)
```

```
}


#x <- modelDataC(df,.8) # used this for full data set runs

x <- modelDataC(pca_df,.8)
train <-x@trn
test <- x@tst
hold <- x@hld

trn <- as.data.frame(dim(train))
names(trn) <- c("dimension_training_train")
tst <-as.data.frame(dim(test))
names(tst) <- c("dimension_testing_test")
hld <-as.data.frame(dim(hold))
names(hld) <- c("dimension_testing_holdout")

# split <- cbind(trn,tst,hld)
# split #%>%kable() %>% kable_styling()

# train <- train
# dim(train)
# test <- test
# dim(test)
# hold <- hold
# dim(hold)
```

Model controls

Model controls will consist of cross validation set to 5-fold, and fitmetric set to "Accuracy."

```
seed <- 275
set.seed(seed)

#Metric Measurement for Model Performance
fitmetric <- "Accuracy"
#fitmetric <- "ROC"

#Run algorithms using 5-fold cross validation
maincontrol <- trainControl(method="cv", number=5,classProbs = T,savePredictions = T)

#Run algorithms using 10-fold cross validation
maincontrol1 <- trainControl(method="cv", number=10,classProbs = T,savePredictions = T)
```

Model Build

Reviewing the models and their different parameters which can be tuned.

- Decision Tree - A tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target feature or the arc leads to a subordinate decision node on a different input feature.

- Random Forest - An ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees.
- Naive Bayes - An algorithm that uses Bayes' theorem to classify objects. Naive Bayes classifiers assume strong, or naive, independence between attributes of data points.
- K Nearest Neighbors - KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.It estimates how likely a data point is to be a member of one group or the other depending on what group in which the data points nearest to it.The k-nearest-neighbor is an example of a "lazy learner" algorithm, meaning that it does not build a model using the training set until a query of the data set is performed.
- Support Vector Machines (radial) - A discriminative classifier formally defined by a separating hyperplane. The algorithm creates a line or a hyperplane, which separates the data into classes.

Caret will be used with method model training controls to run algorithms u5-fold cross validation and various parameter tunings. The most important features are printed after each model is trained for additional information.

```
require(caret)
x1<- modelLookup(model='rpart')
x2<-modelLookup(model='rf')
x3<-modelLookup(model='nb')
x4<-modelLookup(model='knn')
x5<-modelLookup(model='svmLinear')
x6<-modelLookup(model='svmRadial')
x7<-modelLookup(model='gbm')

#mlist <-rbind(x1,x2,x3,x4,x5,x6,x7)
mlist <-rbind(x1,x2,x3,x4,x5)
mlist
```

```
##       model parameter                          label forReg forClass probModel
## 1     rpart        cp          Complexity Parameter   TRUE     TRUE      TRUE
## 2        rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
## 3        nb        fL            Laplace Correction  FALSE     TRUE      TRUE
## 4        nb usekernel             Distribution Type  FALSE     TRUE      TRUE
## 5        nb    adjust          Bandwidth Adjustment  FALSE     TRUE      TRUE
## 6       knn         k                     #Neighbors   TRUE     TRUE      TRUE
## 7 svmLinear         C                           Cost   TRUE     TRUE      TRUE
```

Decision Tree

```
library(rpart)
#Ten Fold CV used with maincontrol2
control <- maincontrol
#Train Tree Model 2 CV
ptm <- proc.time()
fit.tree <- train(label~ ., data = train, method="rpart", metric=fitmetric, trControl=control,control=r
fit.tree
```

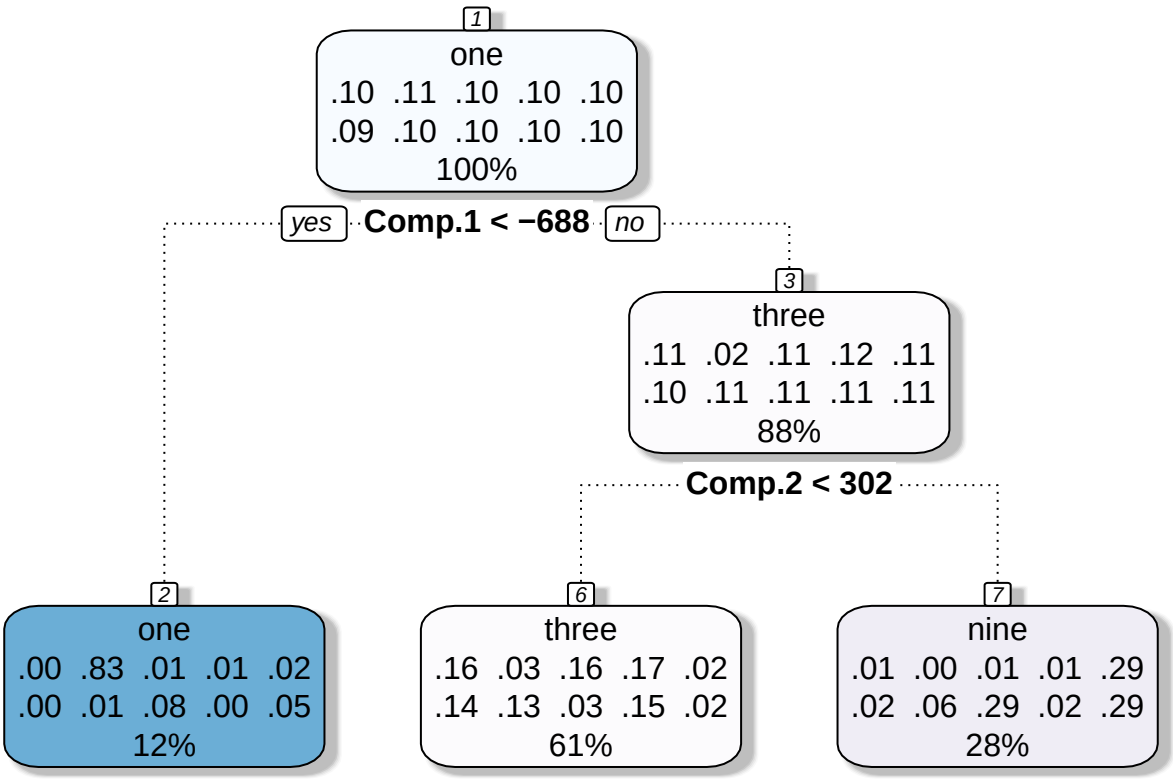
```
## CART
```

```
##
## 8406 samples
##   19 predictor
##   10 classes: 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6724, 6725, 6723, 6726, 6726
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.06948721  0.3281826  0.25102526
##   0.08890079  0.2561244  0.16992455
##   0.09733565  0.1628566  0.06195462
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06948721.
```

```
proc.time() - ptm
```

```
##    user  system elapsed
##    2.83    0.06    2.89
```

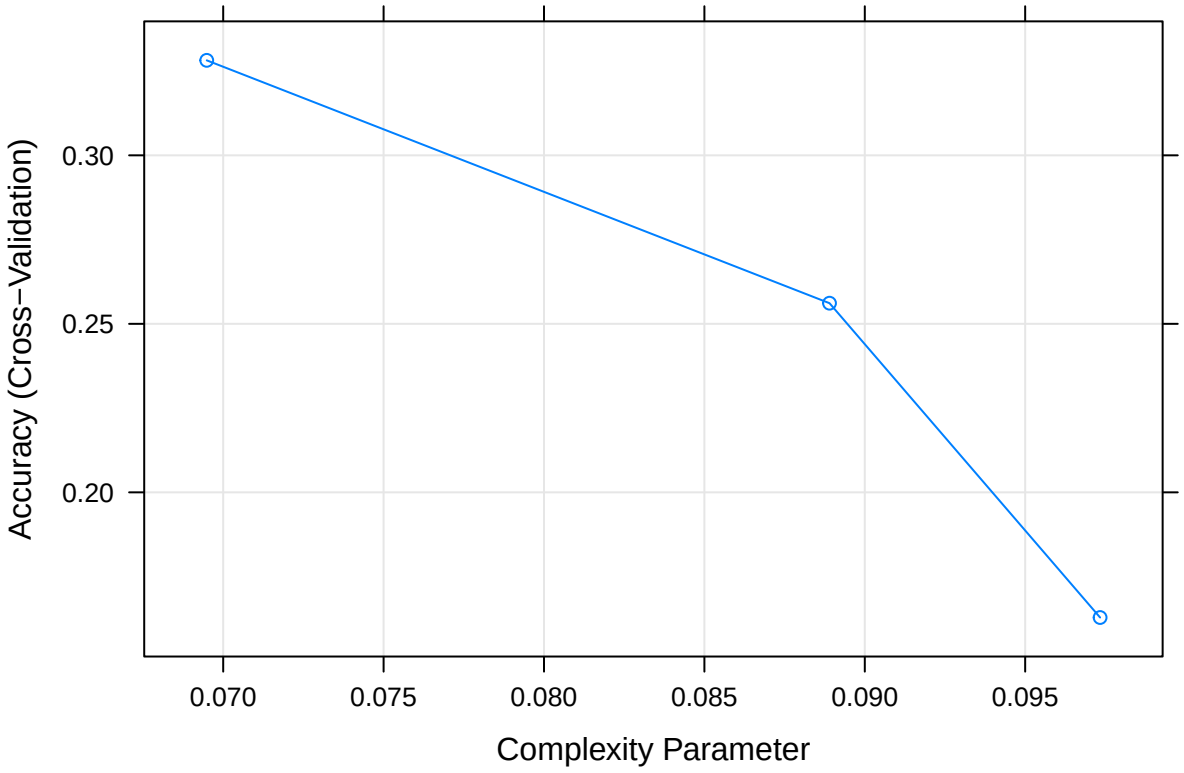
```
#fancy tree plots
suppressMessages(library(rattle))
fancyRpartPlot(fit.tree$finalModel)
```



```r
#summary(fit.tree2$finalModel)

#important variables
treeImp <- varImp(fit.tree)
treeImp
```

```
## rpart variable importance
##
##         Overall
## Comp.2   100.00
## Comp.1    97.49
## Comp.5    65.27
## Comp.4    50.60
## Comp.6    49.74
## Comp.13    0.00
## Comp.11    0.00
## Comp.18    0.00
## Comp.8     0.00
## Comp.3     0.00
## Comp.9     0.00
## Comp.14    0.00
## Comp.7     0.00
## Comp.15    0.00
## Comp.17    0.00
## Comp.16    0.00
## Comp.19    0.00
## Comp.12    0.00
## Comp.10    0.00
```

```r
#ggplot(varImp(treeImp))

#summary(fit.tree$finalModel)

plot(fit.tree)
```




```
res.tree <-as_tibble(fit.tree$results[which.min(fit.tree$results[,2]),])
res.tree
```

```
## # A tibble: 1 x 5
##       cp Accuracy  Kappa AccuracySD KappaSD
##    <dbl>    <dbl>  <dbl>      <dbl>   <dbl>
## 1 0.0973    0.163 0.0620     0.0469  0.0566
```

Random Forest

```
#Train Random Forest CV
control <- maincontrol
newGrid = expand.grid(mtry = c(2,4,8,9))
ptm <- proc.time()
fit.rf <- train(label~ ., data = train, method="rf",ntree=50, metric=fitmetric, trControl=control,tuneG
fit.rf
```

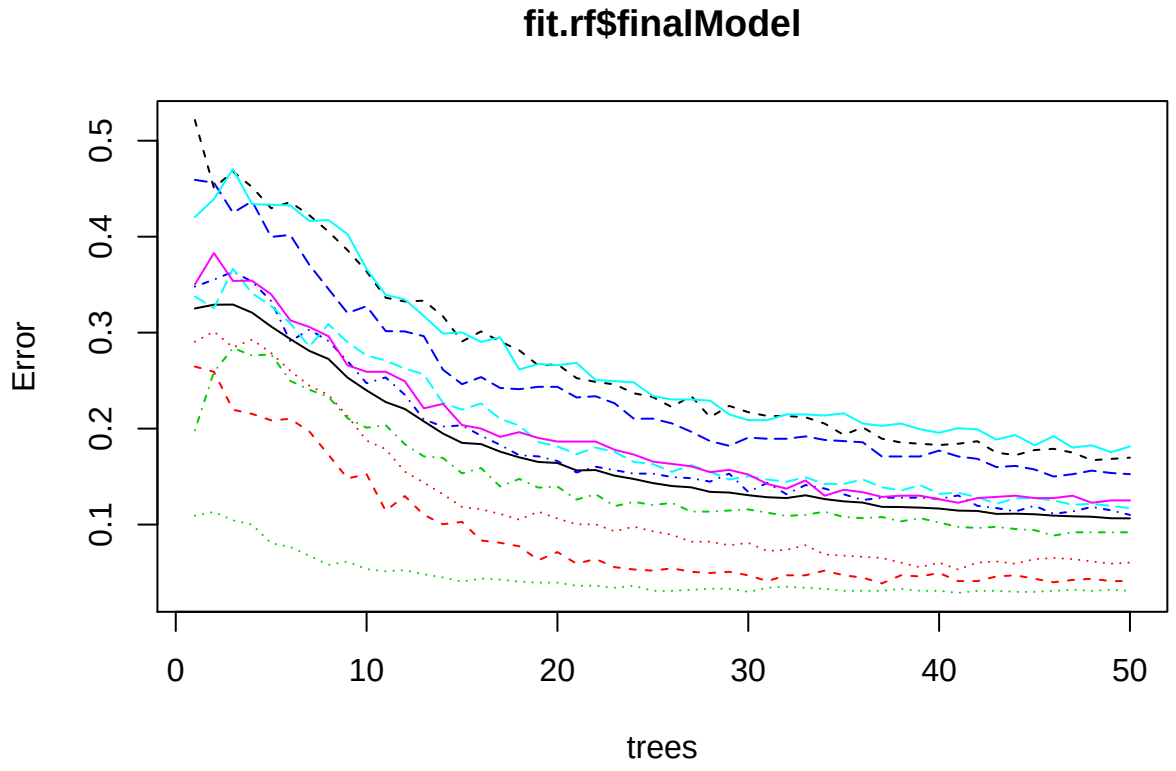
```
## Random Forest
##
## 8406 samples
##   19 predictor
##   10 classes: 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6725, 6724, 6726, 6724, 6725
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9107808  0.9008296
##   4     0.9074495  0.8971313
##   8     0.8954344  0.8837758
##   9     0.8976944  0.8862876
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
proc.time() - ptm
```

```
##    user  system elapsed
##   16.94    0.20   17.18
```

```
plot(fit.rf$finalModel)
```



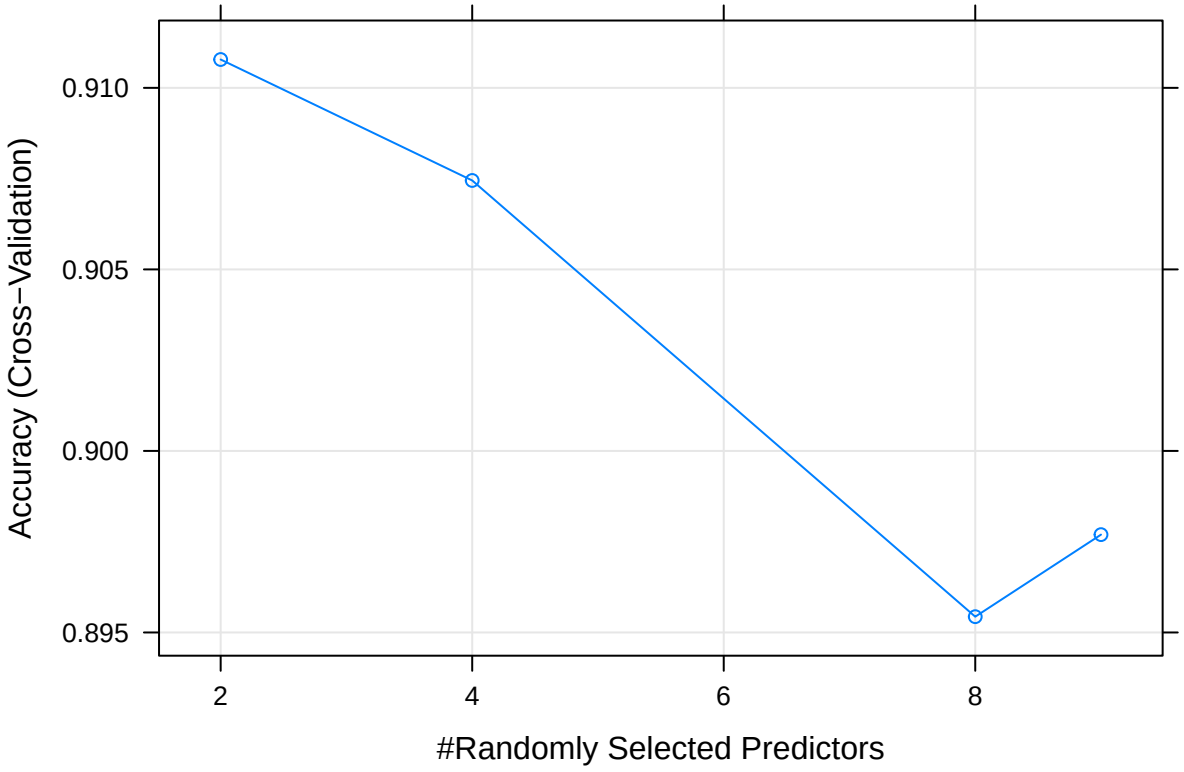

```
#summary(fit.rf$$finalModel)

rfImp <- varImp(fit.rf)
rfImp
```

```
## rf variable importance
##
##         Overall
## Comp.2  100.000
## Comp.1   83.841
## Comp.5   68.892
## Comp.4   56.780
## Comp.6   47.426
## Comp.3   38.667
## Comp.8   38.342
## Comp.7   33.239
## Comp.14  15.166
## Comp.12  14.601
## Comp.15  13.260
## Comp.10  12.704
## Comp.13  11.478
## Comp.9    7.609
## Comp.17   7.495
## Comp.16   6.436
## Comp.11   4.020
## Comp.19   3.505
## Comp.18   0.000
```

```
#ggplot(varImp(rfImp))

summary(fit.rf$finalModel)
```

```
##                 Length Class      Mode
## call               5  -none-     call
## type               1  -none-     character
## predicted       8406  factor     numeric
## err.rate         550  -none-     numeric
## confusion        110  -none-     numeric
## votes          84060  matrix     numeric
## oob.times       8406  -none-     numeric
## classes           10  -none-     character
## importance        19  -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               8406  factor     numeric
## test               0  -none-     NULL
## inbag              0  -none-     NULL
## xNames            19  -none-     character
## problemType        1  -none-     character
## tuneValue          1  data.frame list
## obsLevels         10  -none-     character
## param              1  -none-     list
```

```
plot(fit.rf)
```




```
res.rf <-as_tibble(fit.rf$results[which.min(fit.rf$results[,2]),])
res.rf
```

```
## # A tibble: 1 x 5
##    mtry Accuracy Kappa AccuracySD KappaSD
##   <dbl>    <dbl> <dbl>      <dbl>   <dbl>
## 1     8    0.895 0.884    0.00994  0.0110
```

Naive Bayes

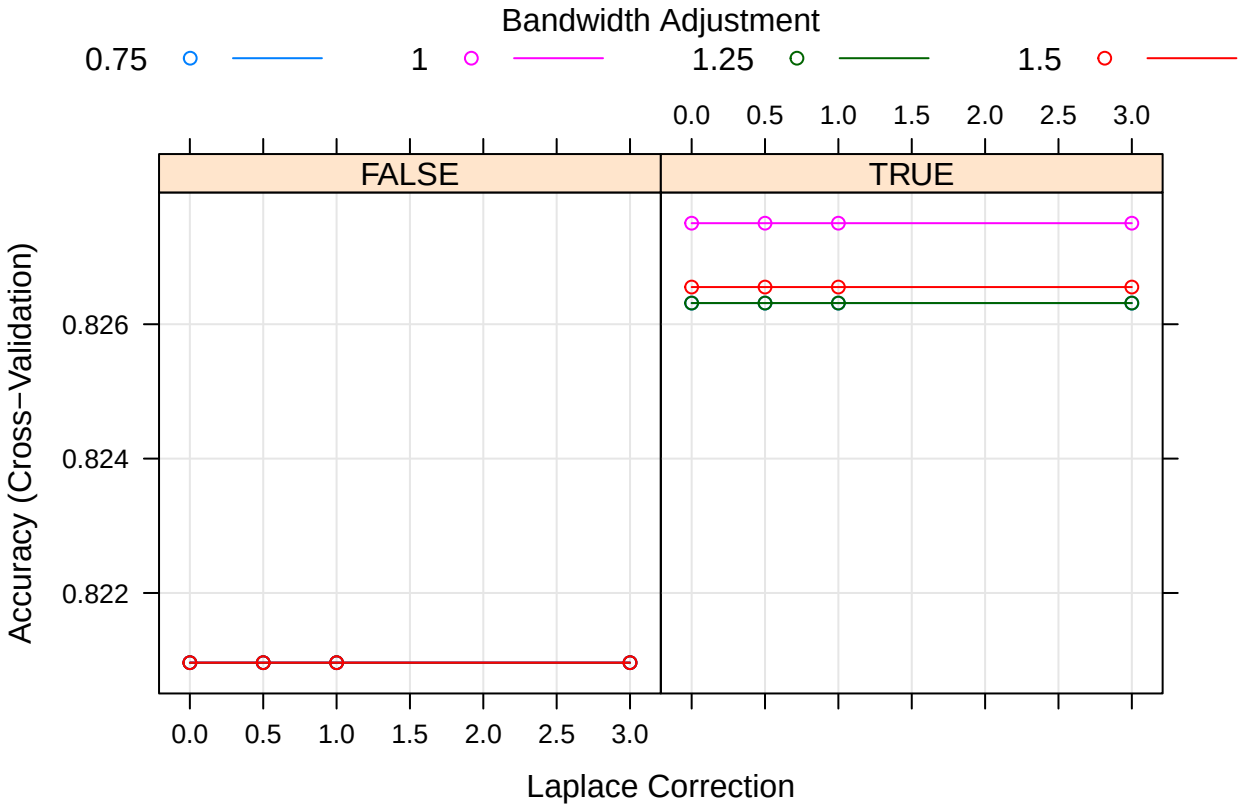
```
control <- maincontrol
#install.packages("naivebayes")
library(naivebayes)
library(e1071) #naiveBayes

# # Define tuning grid
nb_grid <- expand.grid(usekernel = c(TRUE, FALSE),laplace = c(0, 0.5, 1,3), adjust = c(0.75, 1, 1.25, 1

ptm <- proc.time()
fit.nb <- train(label~ ., data = train, method="naive_bayes", tuneGrid = nb_grid, trControl=control)
fit.nb
```

```
## Naive Bayes
##
## 8406 samples
##   19 predictor
##   10 classes: 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6725, 6726, 6724, 6725, 6724
## Resampling results across tuning parameters:
##
##   usekernel  laplace  adjust  Accuracy   Kappa
##   FALSE      0.0      0.75    0.8209624  0.8010308
##   FALSE      0.0      1.00    0.8209624  0.8010308
##   FALSE      0.0      1.25    0.8209624  0.8010308
##   FALSE      0.0      1.50    0.8209624  0.8010308
##   FALSE      0.5      0.75    0.8209624  0.8010308
##   FALSE      0.5      1.00    0.8209624  0.8010308
##   FALSE      0.5      1.25    0.8209624  0.8010308
##   FALSE      0.5      1.50    0.8209624  0.8010308
##   FALSE      1.0      0.75    0.8209624  0.8010308
##   FALSE      1.0      1.00    0.8209624  0.8010308
##   FALSE      1.0      1.25    0.8209624  0.8010308
##   FALSE      1.0      1.50    0.8209624  0.8010308
##   FALSE      3.0      0.75    0.8209624  0.8010308
##   FALSE      3.0      1.00    0.8209624  0.8010308
##   FALSE      3.0      1.25    0.8209624  0.8010308
##   FALSE      3.0      1.50    0.8209624  0.8010308
##    TRUE      0.0      0.75    0.8263149  0.8069877
##    TRUE      0.0      1.00    0.8275041  0.8083024
##    TRUE      0.0      1.25    0.8263154  0.8069734
##    TRUE      0.0      1.50    0.8265537  0.8072331
##    TRUE      0.5      0.75    0.8263149  0.8069877
##    TRUE      0.5      1.00    0.8275041  0.8083024
##    TRUE      0.5      1.25    0.8263154  0.8069734
##    TRUE      0.5      1.50    0.8265537  0.8072331
##    TRUE      1.0      0.75    0.8263149  0.8069877
##    TRUE      1.0      1.00    0.8275041  0.8083024
##    TRUE      1.0      1.25    0.8263154  0.8069734
##    TRUE      1.0      1.50    0.8265537  0.8072331
##    TRUE      3.0      0.75    0.8263149  0.8069877
##    TRUE      3.0      1.00    0.8275041  0.8083024
##    TRUE      3.0      1.25    0.8263154  0.8069734
##    TRUE      3.0      1.50    0.8265537  0.8072331
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 1.
```

```
proc.time() - ptm
```

```
##    user  system elapsed
##   35.66    0.03   35.70
```

```
#fit.nb$bestTune
# nbImp <- varImp(fit.nb)
# nbImp
# plot(nbImp)

plot(fit.nb)
```




```
res.nb<-as_tibble(fit.nb$results[which.min(fit.nb$results[,2]),])
res.nb
```

```
## # A tibble: 1 x 7
##   usekernel laplace adjust Accuracy Kappa AccuracySD KappaSD
##   <lgl>       <dbl>  <dbl>    <dbl> <dbl>      <dbl>   <dbl>
## 1 FALSE           0   0.75    0.821 0.801    0.00842 0.00933
```

K Nearest Neighbor

```
control <- maincontrol
# # Define tuning grid
knn_grid <- expand.grid(k= c(2,3,5,7))

ptm <- proc.time()
fit.knn <- train(label~ ., data = train, method="knn", tuneGrid = knn_grid, trControl=control)
fit.knn
```

```
## k-Nearest Neighbors
##
## 8406 samples
##   19 predictor
##   10 classes: 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6724, 6727, 6725, 6723, 6725
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   2  0.9281472  0.9201363
##   3  0.9413499  0.9348108
##   5  0.9413508  0.9348117
##   7  0.9408738  0.9342820
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

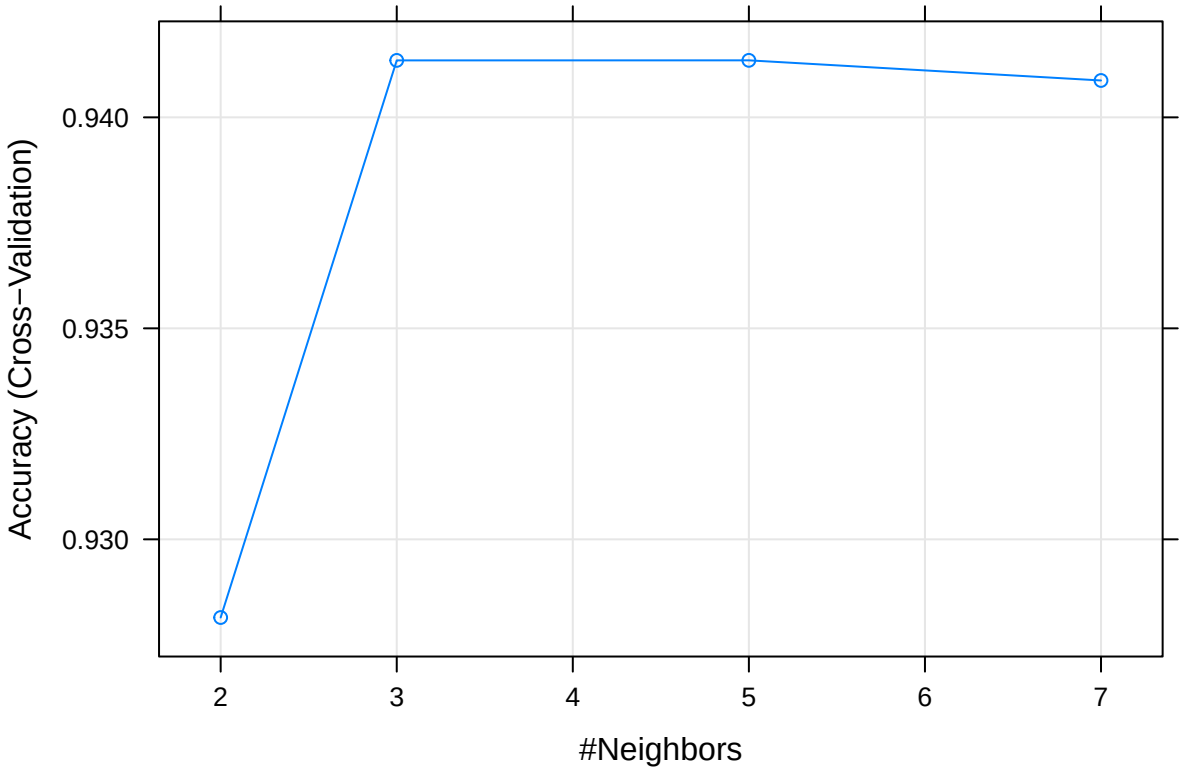
```
proc.time() - ptm
```

```
##    user  system elapsed
##    9.96    0.01   10.00
```

```
fit.knn$bestTune
```

```
##   k
## 3 5
```

```
#plot(fit.knn$finalModel)
# knnImp <- varImp(fit.knn)
# knnImp
# plot(knnImp)
plot(fit.knn)
```




```
res.knn<-as_tibble(fit.knn$results[which.min(fit.knn$results[,2]),])
res.knn
```

```
## # A tibble: 1 x 5
##       k Accuracy Kappa AccuracySD KappaSD
##   <dbl>    <dbl> <dbl>      <dbl>   <dbl>
## 1     2    0.928 0.920    0.00380 0.00422
```

```
 ### Support Vector Machines (Radial)

# #Training Support Vector Machine As a Challenger Model to Decision Tree.
# control <- maincontrol
# #SVMgrid = expand.grid(sigma = c(.01, 0.04, 0.1), C = c(0.01, 10))
# SVMgrid <- expand.grid(C = seq(.01,3, length = 10),sigma = seq(.01,.1, length = 10))
# SVMgrid2 <- expand.grid(C = seq(.01,3, length = 5))
#
# ptm <- proc.time()
#
# fit.svm <- train(label~ ., data = train, method="svmRadial", metric=fitmetric, trControl=control,tune
# #fit.svm <- train(label~ ., data = train, method="svmRadial", metric=fitmetric, trControl=control,pre
# fit.svm
# proc.time() - ptm
# #print(fit.svm)
# plot(fit.svm)
# res.svm<-as_tibble(fit.svm$results[which.min(fit.svm$results[,2]),])
```

```
# res.svm
# #plot(fit.svm$finalModel)
```

Support Vector Machines (Linear)

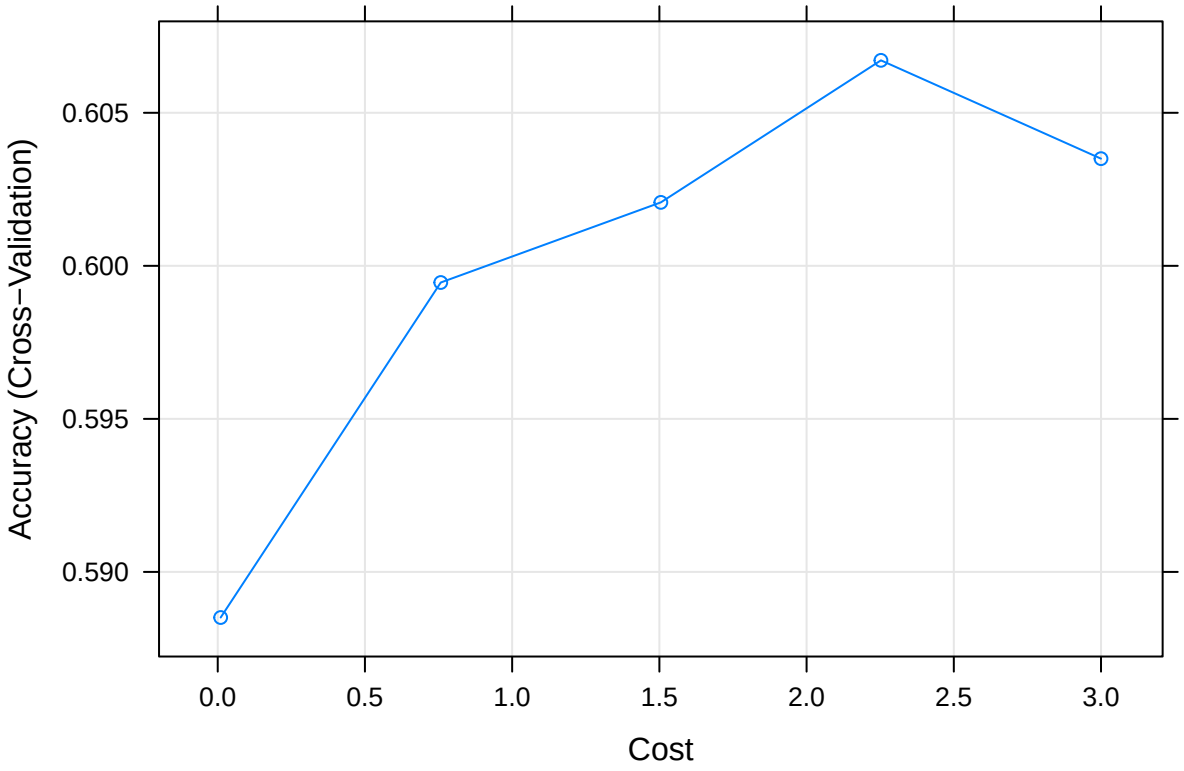
```
control <- maincontrol

SVMgrid_ln <- expand.grid(C = seq(.01,3, length = 5))

ptm <- proc.time()
fit.svm <- train(label~ ., data=train,method="svmLinear",metric=fitmetric,trControl=control,tuneGrid =S

fit.svm <-fit.svm
fit.svm
```

```
## Support Vector Machines with Linear Kernel
##
## 8406 samples
##   19 predictor
##   10 classes: 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6725, 6724, 6725, 6726, 6724
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa
##   0.0100  0.5885100  0.5428997
##   0.7575  0.5994552  0.5548384
##   1.5050  0.6020733  0.5577205
##   2.2525  0.6067132  0.5628718
##   3.0000  0.6035009  0.5593043
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 2.2525.
```

```
proc.time() - ptm
```

```
##    user  system elapsed
##   96.18    0.20   96.49
```

```
#print(fit.svm)
plot(fit.svm)
```




```
res.svm<-as_tibble(fit.svm$results[which.min(fit.svm$results[,2]),])
res.svm
```

```
## # A tibble: 1 x 5
##       C Accuracy Kappa AccuracySD KappaSD
##   <dbl>    <dbl> <dbl>      <dbl>   <dbl>
## 1  0.01    0.589 0.543    0.00697 0.00771
```

```
#plot(fit.svm$finalModel)
```

Section 3: Model Results

Model Fit Results

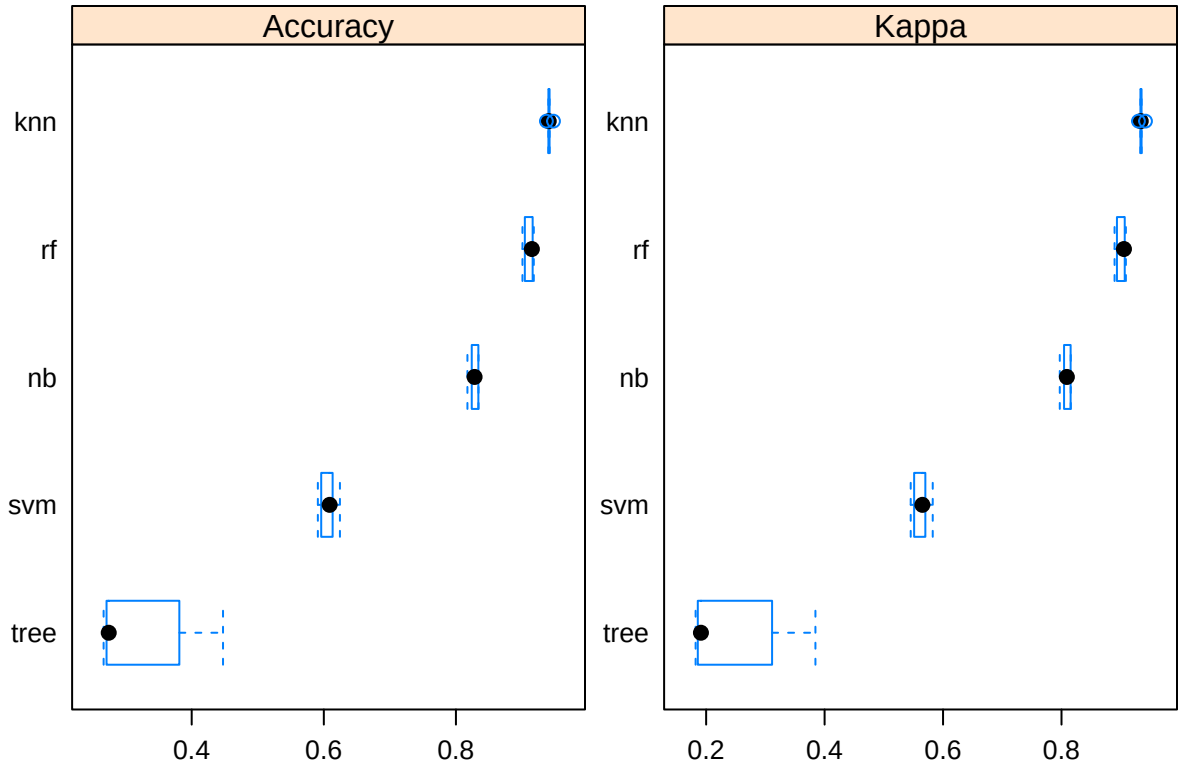
The graph below shows two evaluation metrics for the models.

- Accuracy is the percentage of correctly classifies instances out of all instances. It is more useful on a binary classification than multi-class classification problems because it can be less clear exactly how the accuracy breaks down across those classes (e.g. you need to go deeper with a confusion matrix). Learn more about Accuracy here.
- Kappa or Cohen's Kappa is like classification accuracy, except that it is normalized at the baseline of random chance on your dataset. It is a more useful measure to use on problems that have an imbalance

in the classes (e.g. 70-30 split for classes 0 and 1 and you can achieve 70% accuracy by predicting all instances are for class 0). Learn more about Kappa here.

The random forest algorithm denoted by *rf* is the better model according to these metrics.

```
##
## Call:
## summary.resamples(object = results)
##
## Models: rf, tree, nb, svm, knn
## Number of resamples: 5
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf   0.9007134 0.9042806 0.9149316 0.9107808 0.9161214 0.9178571    0
## tree 0.2666667 0.2711058 0.2744048 0.3281826 0.3813206 0.4474153    0
## nb   0.8173706 0.8239143 0.8281807 0.8275041 0.8339286 0.8341260    0
## svm  0.5909631 0.5960738 0.6087990 0.6067132 0.6133254 0.6244048    0
## knn  0.9369423 0.9399881 0.9404407 0.9413508 0.9417014 0.9476813    0
##
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf   0.8896346 0.8936015 0.9054479 0.9008296 0.9067698 0.9086941    0
## tree 0.1821943 0.1858780 0.1912283 0.2510253 0.3113250 0.3845007    0
## nb   0.7970749 0.8043383 0.8090451 0.8083024 0.8154038 0.8156500    0
## svm  0.5453767 0.5511973 0.5651083 0.5628718 0.5701204 0.5825564    0
## knn  0.9299152 0.9332985 0.9338023 0.9348117 0.9351983 0.9418442    0
```



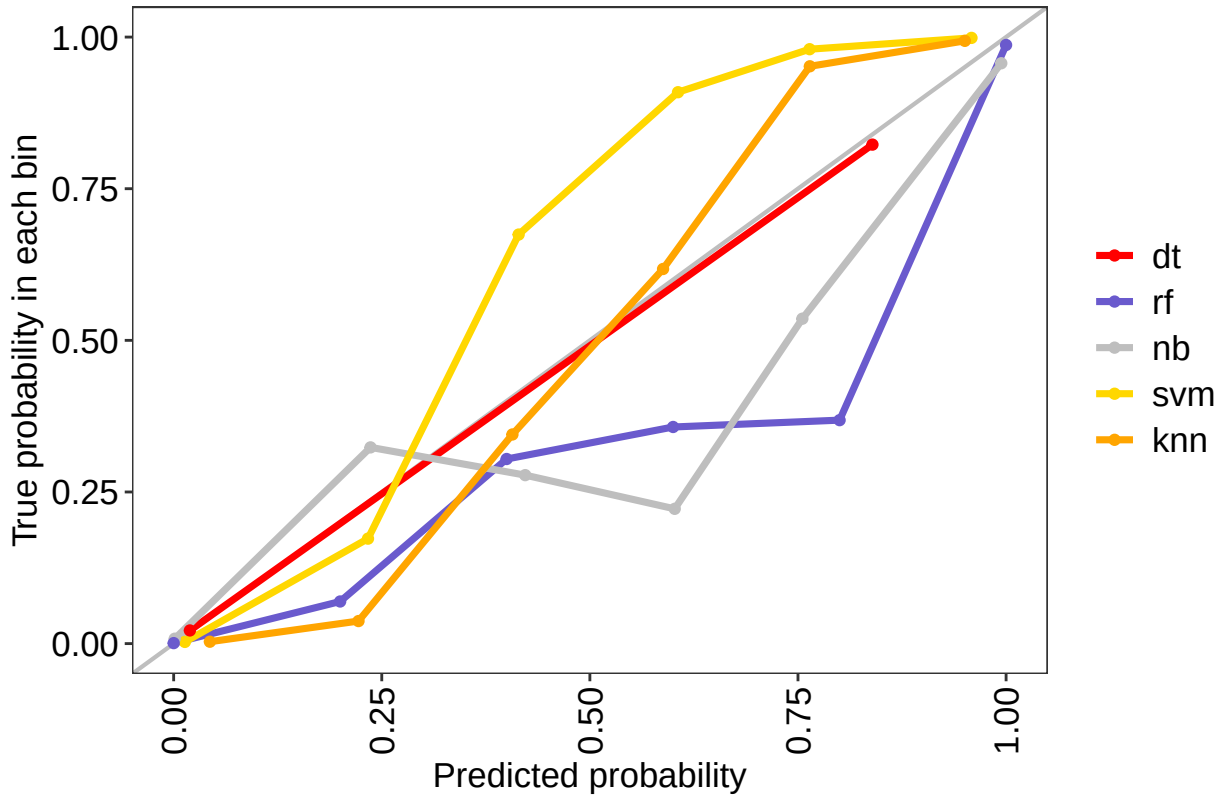
- **AUC-ROC** curve: This is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. **Higher the AUC, better the model** is at binary predictions.

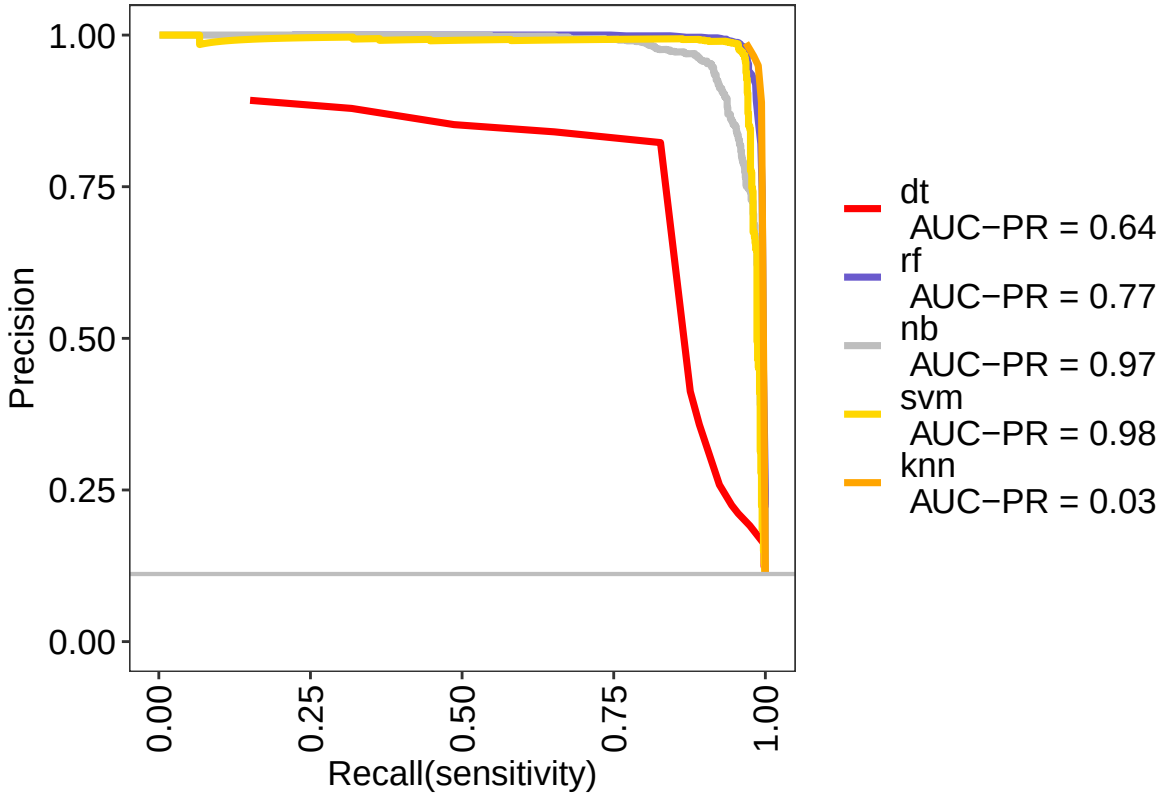
The model decision tree is fitting to the data well. Support vector machines is very good too, but random forest fits better overall. The last graph is plotting AUC-ROC and the random forest(rf) has more area under the curve as compared to the best tree model denoted by the purple line. This means the random forest is a better fit for the data than other models, judging by the AUC-ROC metric.

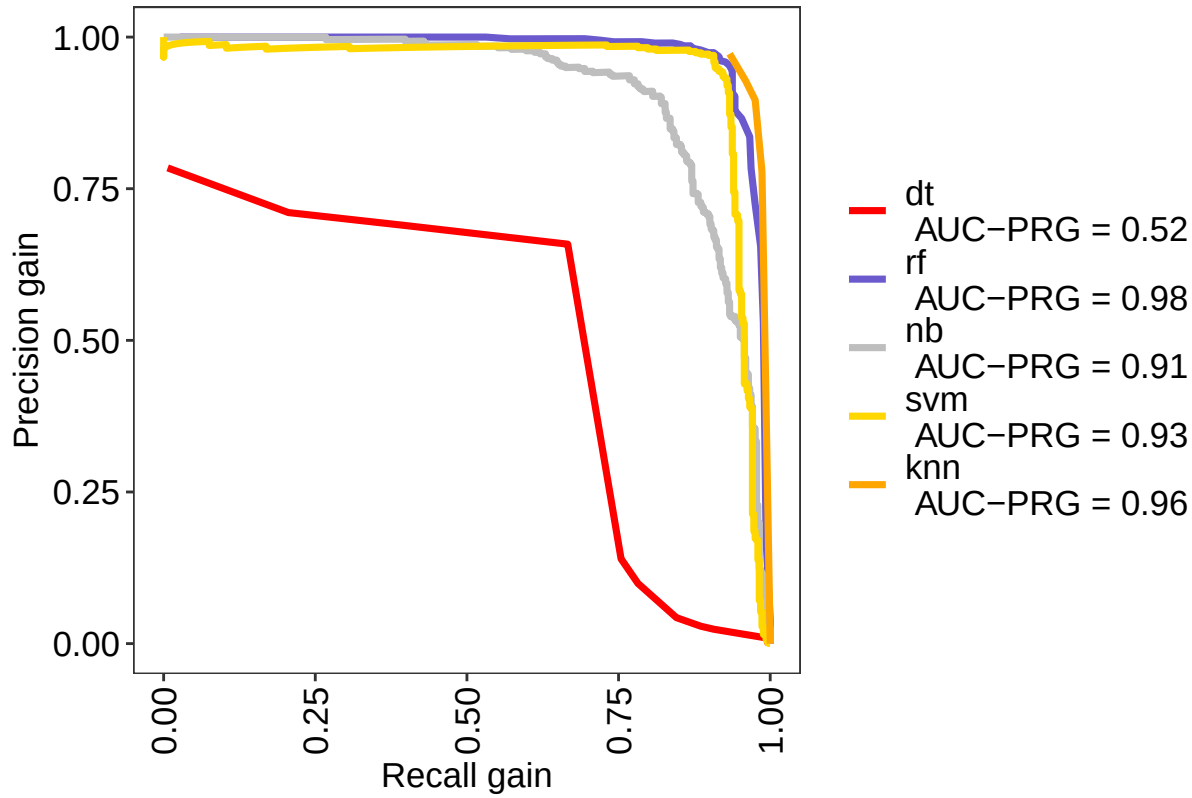
```
## run MLeval comparing tree 2 and 3
library(MLeval)
res <- evalm(list(fit.tree,fit.rf,fit.nb,fit.svm,fit.knn),gnames=c('dt','rf','nb','svm','knn'))
```

```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object

## Not averaging probs.

## Group 1 type: cv

## Group 2 type: cv
```

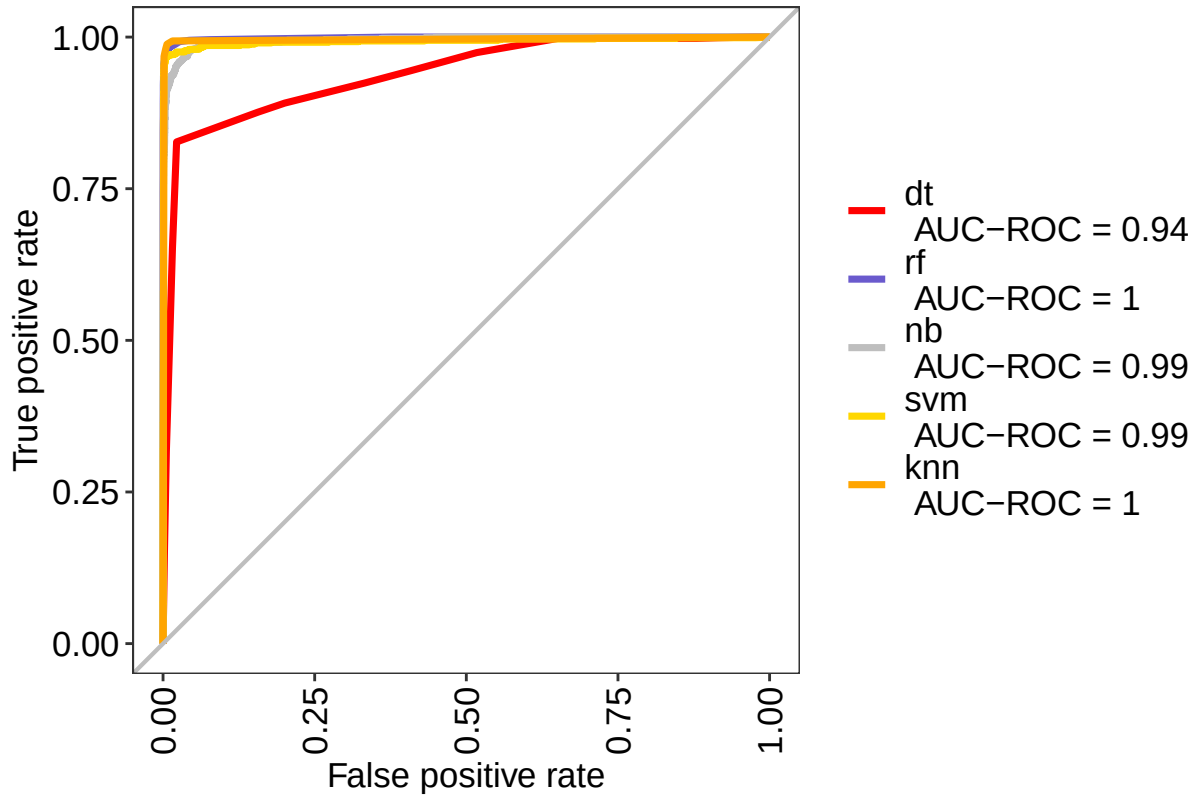
```
## Group 3 type: cv

## Group 4 type: cv

## Group 5 type: cv

## Observations: 42030

## Number of groups: 5

## Observations per group: 8406

## Positive: one

## Negative: zero

## Group: dt

## Positive: 937

## Negative: 827

## Group: rf

## Positive: 937

## Negative: 827

## Group: nb

## Positive: 937

## Negative: 827

## Group: svm

## Positive: 937

## Negative: 827

## Group: knn

## Positive: 937

## Negative: 827

## ***Performance Metrics***
```


1.00
0.75
0.50
0.25
0.00
True probability in each bin
0.00
0.25
0.50
0.75
1.00
Predicted probability
dt
rf
nb
svm
knn


1.00
0.75
0.50
0.25
0.00
Precision
0.00
0.25
0.50
0.75
1.00
Recall(sensitivity)
dt
AUC−PR = 0.64
rf
AUC−PR = 0.77
nb
AUC−PR = 0.97
svm
AUC−PR = 0.98
knn
AUC−PR = 0.03




```
## dt Optimal Informedness = 0.804748706607017

## rf Optimal Informedness = 0.973146636835312

## nb Optimal Informedness = 0.933677056915293

## svm Optimal Informedness = 0.962359681489609

## knn Optimal Informedness = 0.981699955690208

## dt AUC-ROC = 0.94

## rf AUC-ROC = 1

## nb AUC-ROC = 0.99

## svm AUC-ROC = 0.99

## knn AUC-ROC = 1
```



Model Prediction Results

Definition of the Terms related to measuring classification models:

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Accuracy: * Classification Rate or Accuracy is given by the relation: (TP + TN) / (TP + TN + FP + FN) * It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Recall: Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (a small number of FN). Recall is given by the relation: TP / (TP + FN)

AUC-ROC curve: This is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. **Higher the AUC, better the model** is at binary predictions.

Summary the test results for each of the following classifier models:

- Decision Tree - Accuracy : 0.28
- Random Forest - Accuracy : 0.90
- Naive Bayes - Accuracy : 0.83
- K Nearest Neighbors - Accuracy : 0.94
- Support Vector Machines - Accuracy : 0.59

```
test <- test
# head(test[,1:5])

require(alluvial)
```

```
## Loading required package: alluvial
```

```
## Warning: package 'alluvial' was built under R version 3.5.3
```

```
#library(mlbench)
#install.packages("alluvial")
library(alluvial)
```

Build Helper Functions

```
#alluvial plots
plotCM <- function(cm){
  cmdf <- as.data.frame(cm[["table"]])
  cmdf[["color"]] <- ifelse(cmdf[[1]] == cmdf[[2]], "blue", "red")

  alluvial::alluvial(cmdf[,1:2]
                     , freq = cmdf$Freq
                     , col = cmdf[["color"]]
                     , alpha = 0.5
                     , hide  = cmdf$Freq == 0
                     )
}


ggPlotCM <- function(cm){
## ggplotly version
cm_d <- as.data.frame(cm$table)
# confusion matrix statistics as data.frame
cm_st <-data.frame(cm$overall)
# round the values
cm_st$cm.overall <- round(cm_st$cm.overall,2)

# here we also have the rounded percentage values
cm_p <- as.data.frame(prop.table(cm$table))
cm_d$Perc <- round(cm_p$Freq*100,2)

# plotting the matrix
cm_d_p <-  ggplot(data = cm_d, aes(x = Prediction , y =  Reference, fill = Freq))+
  geom_tile() +
  #geom_text(aes(label = paste("",Freq,",",Perc,"%")), color = 'red', size = 3) +
```

```
  geom_text(aes(label = paste(Freq)), color = 'red', size = 3) +
  theme_light() +
  guides(fill=FALSE) + theme_1()

# plotting the stats
#cm_st_p <-  tableGrob(cm_st)

cm_d_ply <-ggplotly(cm_d_p)
return (cm_d_ply)
}
```

Decision Tree

```
predict.tree= predict(fit.tree, test, type="raw")
#predict.tree= predict(fit.tree, test, type="prob")

summary(predict.tree)
```

```
##  zero   one   two three  four  five   six seven eight  nine
##     0   245     0  1254     0     0     0     0     0   598
```

```
#caret confusion
cm <- confusionMatrix(factor(predict.tree), factor(test$label))
```

```
## Warning in levels(reference) != levels(data): longer object length is not a
## multiple of shorter object length
```

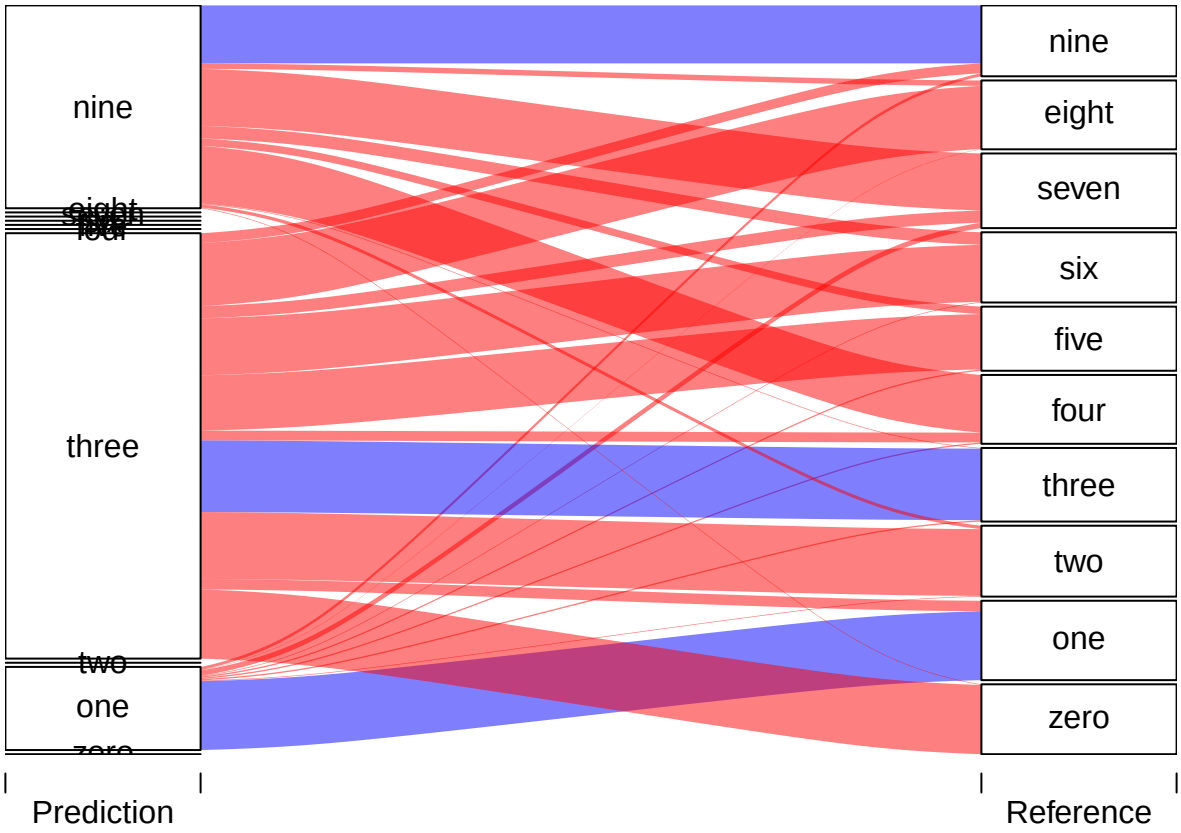
```
## Warning in confusionMatrix.default(factor(predict.tree), factor(test$label)):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction zero one two three four five six seven eight nine
##      zero     0   0   0     0    0    0   0     0     0    0
##      one      0 203   2     4    4    4   2    16     1    9
##      two      0   0   0     0    0    0   0     0     0    0
##      three  204  31 197   211   29  163 168    36   186   29
##      four     0   0   0     0    0    0   0     0     0    0
##      five     0   0   0     0    0    0   0     0     0    0
##      six      0   0   0     0    0    0   0     0     0    0
##      seven    0   0   0     0    0    0   0     0     0    0
##      eight    0   0   0     0    0    0   0     0     0    0
##      nine     2   0  10     2  170   22  37   168    16  171
##
## Overall Statistics
##
```

```
##                Accuracy : 0.279
##                  95% CI : (0.2599, 0.2987)
##     No Information Rate : 0.1116
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1959
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: zero Class: one Class: two Class: three Class: four
## Sensitivity              0.00000     0.8675    0.00000       0.9724      0.0000
## Specificity              1.00000     0.9775    1.00000       0.4452      1.0000
## Pos Pred Value               NaN     0.8286        NaN       0.1683         NaN
## Neg Pred Value           0.90176     0.9833    0.90033       0.9929      0.9032
## Prevalence               0.09824     0.1116    0.09967       0.1035      0.0968
## Detection Rate           0.00000     0.0968    0.00000       0.1006      0.0000
## Detection Prevalence     0.00000     0.1168    0.00000       0.5980      0.0000
## Balanced Accuracy        0.50000     0.9225    0.50000       0.7088      0.5000
##                      Class: five Class: six Class: seven Class: eight
## Sensitivity              0.00000    0.00000       0.0000       0.0000
## Specificity              1.00000    1.00000       1.0000       1.0000
## Pos Pred Value               NaN        NaN          NaN          NaN
## Neg Pred Value           0.90987    0.90129       0.8951       0.9032
## Prevalence               0.09013    0.09871       0.1049       0.0968
## Detection Rate           0.00000    0.00000       0.0000       0.0000
## Detection Prevalence     0.00000    0.00000       0.0000       0.0000
## Balanced Accuracy        0.50000    0.50000       0.5000       0.5000
##                      Class: nine
## Sensitivity              0.81818
## Specificity              0.77383
## Pos Pred Value           0.28595
## Neg Pred Value           0.97465
## Prevalence               0.09967
## Detection Rate           0.08155
## Detection Prevalence     0.28517
## Balanced Accuracy        0.79601
```

```
plotCM(cm)
```




```
ggPlotCM(cm)
```

```
## Warning: 'heatmap' objects don't have these attributes: 'showlegend'
## Valid attributes include:
## 'type', 'visible', 'opacity', 'name', 'uid', 'ids', 'customdata', 'meta', 'hoverinfo', 'hoverlabel',
```

SVM

```
#Test Results SVM
predict.svm= predict(fit.svm, test, type="raw")
#predict.sum= predict(fit.sum, test, type="prob")
summary(predict.svm)
```

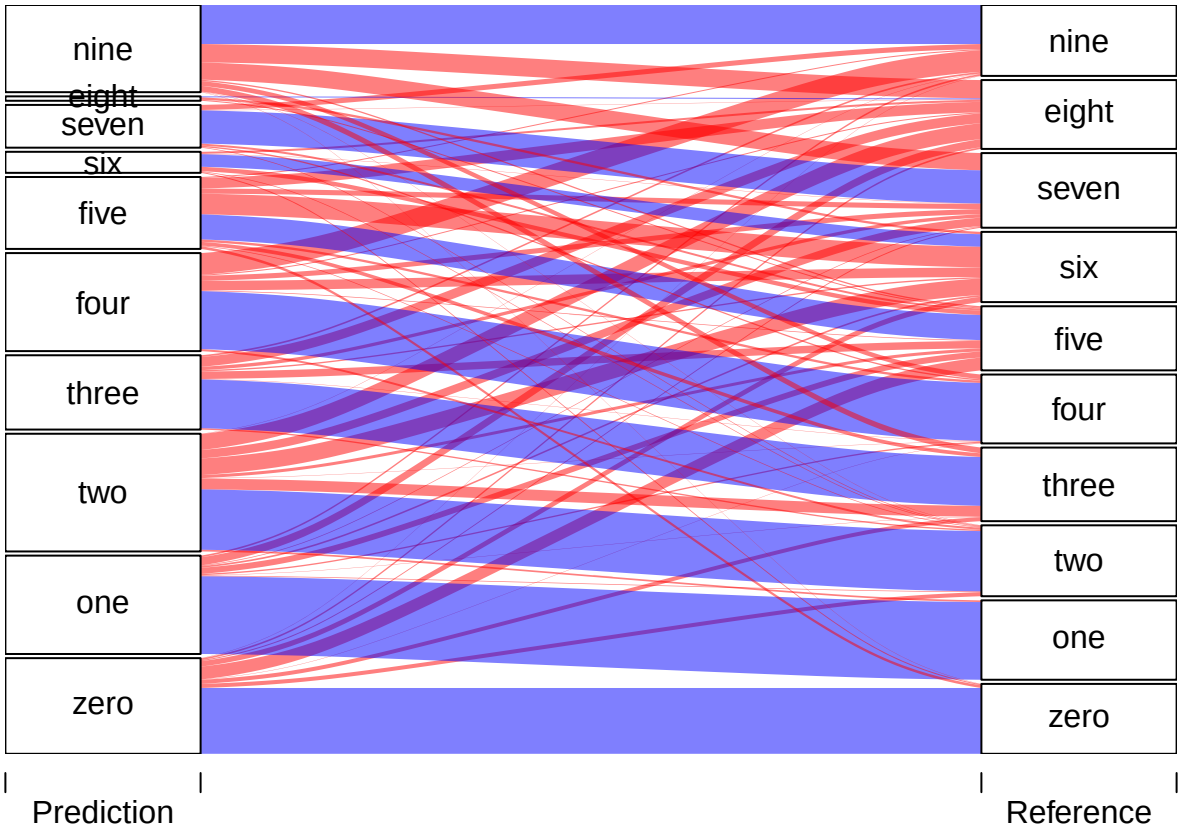
```
##  zero   one   two three  four  five   six seven eight  nine
##   282   290   347   219   289   212    62   126    13   257
```

```
#confusion matrix to find correct and incorrect predictions
#table(labelship=predict.sum, true=test$label)
#caret confusion
cm <-confusionMatrix(factor(predict.svm), factor(test$label))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction zero one two three four five six seven eight nine
##      zero   194   0  12    12    1   38  16     3     5    1
##      one      0 229   2     1    4   18   5     2    24    5
##      two      0   5 178    32    1    9  49    25    47    1
##      three    0   0   4   144    1   22   4    10    29    5
##      four     0   0   6     0  171    2  28    15     3   64
##      five     8   0   2    11    7   74  61    15    31    3
##      six      3   0   2     0    0   13  37     1     6    0
##      seven    0   0   2     0    4    6   0    98     1   15
##      eight    0   0   0     0    0    3   7     0     3    0
##      nine     1   0   1    17   14    4   0    51    54  115
##
## Overall Statistics
##
##                Accuracy : 0.5928
##                  95% CI : (0.5714, 0.6139)
##     No Information Rate : 0.1116
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5473
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: zero Class: one Class: two Class: three Class: four
## Sensitivity              0.94175     0.9786    0.85167      0.66359     0.84236
## Specificity              0.95346     0.9673    0.91049      0.96011     0.93770
## Pos Pred Value           0.68794     0.7897    0.51297      0.65753     0.59170
## Neg Pred Value           0.99339     0.9972    0.98229      0.96113     0.98230
## Prevalence               0.09824     0.1116    0.09967      0.10348     0.09680
## Detection Rate           0.09251     0.1092    0.08488      0.06867     0.08155
## Detection Prevalence     0.13448     0.1383    0.16547      0.10443     0.13782
## Balanced Accuracy        0.94761     0.9729    0.88108      0.81185     0.89003
##                      Class: five Class: six Class: seven Class: eight
## Sensitivity              0.39153    0.17874      0.44545     0.014778
## Specificity              0.92767    0.98677      0.98508     0.994720
## Pos Pred Value           0.34906    0.59677      0.77778     0.230769
## Neg Pred Value           0.93899    0.91646      0.93810     0.904031
## Prevalence               0.09013    0.09871      0.10491     0.096805
## Detection Rate           0.03529    0.01764      0.04673     0.001431
## Detection Prevalence     0.10110    0.02957      0.06009     0.006199
## Balanced Accuracy        0.65960    0.58276      0.71527     0.504749
##                      Class: nine
## Sensitivity              0.55024
## Specificity              0.92479
## Pos Pred Value           0.44747
## Neg Pred Value           0.94891
## Prevalence               0.09967
## Detection Rate           0.05484
## Detection Prevalence     0.12256
```

```
## Balanced Accuracy         0.73751
```

```
plotCM(cm)
```




```
ggPlotCM(cm)
```

```
## Warning: 'heatmap' objects don't have these attributes: 'showlegend'
## Valid attributes include:
## 'type', 'visible', 'opacity', 'name', 'uid', 'ids', 'customdata', 'meta', 'hoverinfo', 'hoverlabel',
```

Random Forest

```
predict.rf= predict(fit.rf, test, type="raw")
#predicted3= predict(fit.rf, test, type="prob")
summary(predict.rf)
```

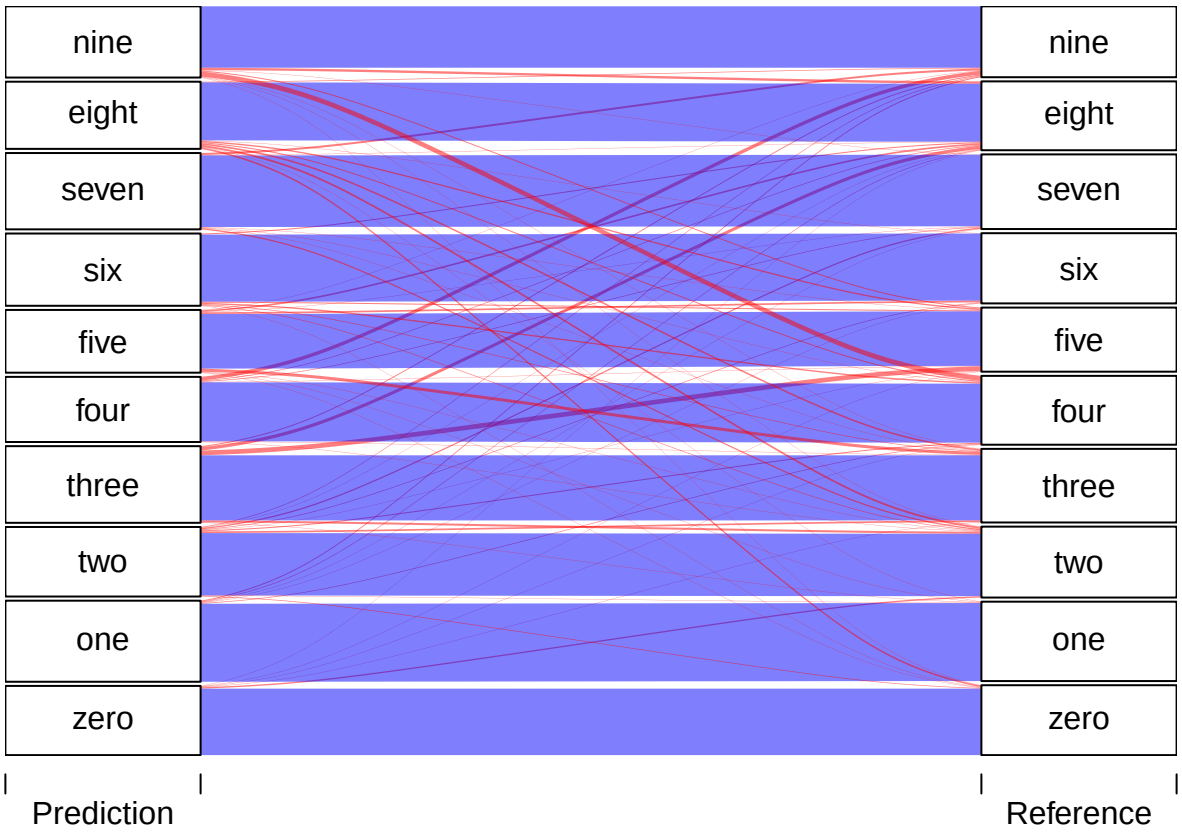
```
##  zero   one   two three  four  five   six seven eight  nine
##   204   239   205   226   192   185   213   225   198   210
```

```
#confusion matrix to find correct and incorrect predictions
#table(labelship=predict.rf, true=test$label)
#caret confusion
cm <- confusionMatrix(factor(predict.rf), factor(test$label))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction zero one two three four five six seven eight nine
##      zero   196   0   4     1    1    1   0     0     1    0
##      one      0 230   0     0    2    0   1     1     2    3
##      two      2   1 184     5    3    1   2     4     1    2
##      three    0   1   6   192    0   14   0     0    10    3
##      four     0   0   1     1  174    1   0     2     2   11
##      five     1   1   1     9    0  161   5     1     5    1
##      six      1   0   2     2    4    3 198     0     3    0
##      seven    0   0   4     1    1    1   0   211     1    6
##      eight    5   1   6     5    3    4   1     0   171    2
##      nine     1   0   1     1   15    3   0     1     7  181
##
## Overall Statistics
##
##                Accuracy : 0.9051
##                  95% CI : (0.8917, 0.9173)
##     No Information Rate : 0.1116
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8945
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: zero Class: one Class: two Class: three Class: four
## Sensitivity              0.95146     0.9829    0.88038      0.88479     0.85714
## Specificity              0.99577     0.9952    0.98888      0.98191     0.99050
## Pos Pred Value           0.96078     0.9623    0.89756      0.84956     0.90625
## Neg Pred Value           0.99472     0.9978    0.98679      0.98664     0.98478
## Prevalence               0.09824     0.1116    0.09967      0.10348     0.09680
## Detection Rate           0.09347     0.1097    0.08774      0.09156     0.08298
## Detection Prevalence     0.09728     0.1140    0.09776      0.10777     0.09156
## Balanced Accuracy        0.97361     0.9890    0.93463      0.93335     0.92382
##                      Class: five Class: six Class: seven Class: eight
## Sensitivity              0.85185    0.95652       0.9591      0.84236
## Specificity              0.98742    0.99206       0.9925      0.98574
## Pos Pred Value           0.87027    0.92958       0.9378      0.86364
## Neg Pred Value           0.98536    0.99522       0.9952      0.98315
## Prevalence               0.09013    0.09871       0.1049      0.09680
## Detection Rate           0.07678    0.09442       0.1006      0.08155
## Detection Prevalence     0.08822    0.10157       0.1073      0.09442
## Balanced Accuracy        0.91964    0.97429       0.9758      0.91405
##                      Class: nine
## Sensitivity              0.86603
## Specificity              0.98464
## Pos Pred Value           0.86190
## Neg Pred Value           0.98516
## Prevalence               0.09967
## Detection Rate           0.08631
## Detection Prevalence     0.10014
```

```
## Balanced Accuracy         0.92533
```

```
plotCM(cm)
```




```
ggPlotCM(cm)
```

```
## Warning: 'heatmap' objects don't have these attributes: 'showlegend'
## Valid attributes include:
## 'type', 'visible', 'opacity', 'name', 'uid', 'ids', 'customdata', 'meta', 'hoverinfo', 'hoverlabel',
```

K Nearest Neighbors

```
predict.knn= predict(fit.knn, test, type="raw")
#predicted3= predict(fit.knn, test, type="prob")
summary(predict.knn)
```

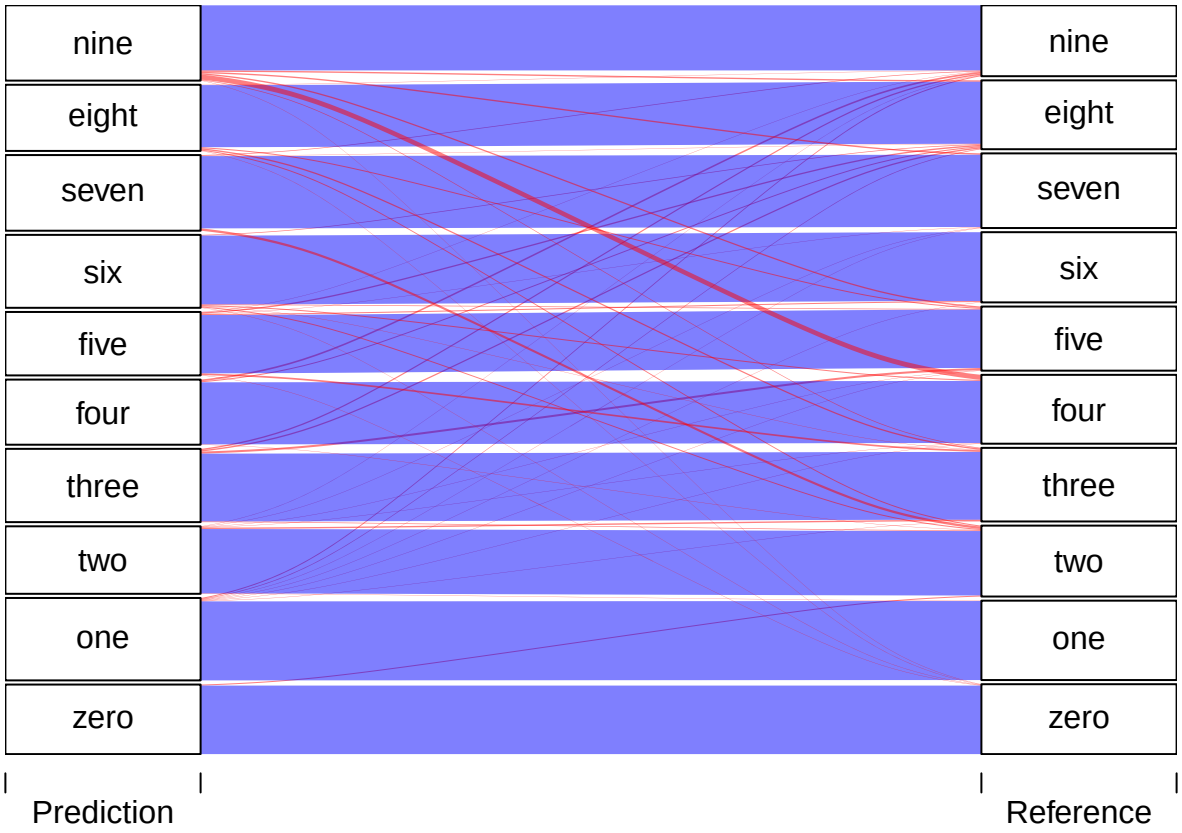
```
##  zero   one   two three  four  five   six seven eight  nine
##   205   243   199   216   192   187   215   223   195   222
```

```
#table(labelship=predict.rf, true=test$label)
#caret confusion

cm <- confusionMatrix(factor(predict.knn), factor(test$label))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction zero one two three four five six seven eight nine
##      zero   202   0   3     0    0    0   0     0     0    0
##      one      0 233   0     1    1    1   1     1     2    3
##      two      0   1 190     4    1    1   0     1     0    1
##      three    0   0   2   200    0    6   0     0     4    4
##      four     0   0   1     0  183    0   0     0     3    5
##      five     1   0   0     5    0  172   3     1     4    1
##      six      1   0   3     1    3    2 203     0     2    0
##      seven    0   0   7     0    0    0   0   213     1    2
##      eight    1   0   3     4    0    3   0     0   183    1
##      nine     1   0   0     2   15    4   0     4     4  192
##
## Overall Statistics
##
##                Accuracy : 0.9399
##                  95% CI : (0.9289, 0.9497)
##     No Information Rate : 0.1116
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9332
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: zero Class: one Class: two Class: three Class: four
## Sensitivity              0.98058     0.9957    0.90909      0.92166     0.90148
## Specificity              0.99841     0.9946    0.99523      0.99149     0.99525
## Pos Pred Value           0.98537     0.9588    0.95477      0.92593     0.95312
## Neg Pred Value           0.99789     0.9995    0.98999      0.99096     0.98950
## Prevalence               0.09824     0.1116    0.09967      0.10348     0.09680
## Detection Rate           0.09633     0.1111    0.09061      0.09537     0.08727
## Detection Prevalence     0.09776     0.1159    0.09490      0.10300     0.09156
## Balanced Accuracy        0.98950     0.9952    0.95216      0.95657     0.94836
##                      Class: five Class: six Class: seven Class: eight
## Sensitivity              0.91005    0.98068       0.9682      0.90148
## Specificity              0.99214    0.99365       0.9947      0.99366
## Pos Pred Value           0.91979    0.94419       0.9552      0.93846
## Neg Pred Value           0.99110    0.99787       0.9963      0.98948
## Prevalence               0.09013    0.09871       0.1049      0.09680
## Detection Rate           0.08202    0.09680       0.1016      0.08727
## Detection Prevalence     0.08918    0.10253       0.1063      0.09299
## Balanced Accuracy        0.95110    0.98716       0.9814      0.94757
##                      Class: nine
## Sensitivity              0.91866
## Specificity              0.98411
## Pos Pred Value           0.86486
## Neg Pred Value           0.99093
## Prevalence               0.09967
## Detection Rate           0.09156
## Detection Prevalence     0.10587
```

```
## Balanced Accuracy        0.95139
```

```
plotCM(cm)
```




```
ggPlotCM(cm)
```

```
## Warning: 'heatmap' objects don't have these attributes: 'showlegend'
## Valid attributes include:
## 'type', 'visible', 'opacity', 'name', 'uid', 'ids', 'customdata', 'meta', 'hoverinfo', 'hoverlabel',
```

Naive Bayes

```
predict.nb= predict(fit.nb, test, type="raw")
summary(predict.nb)
```

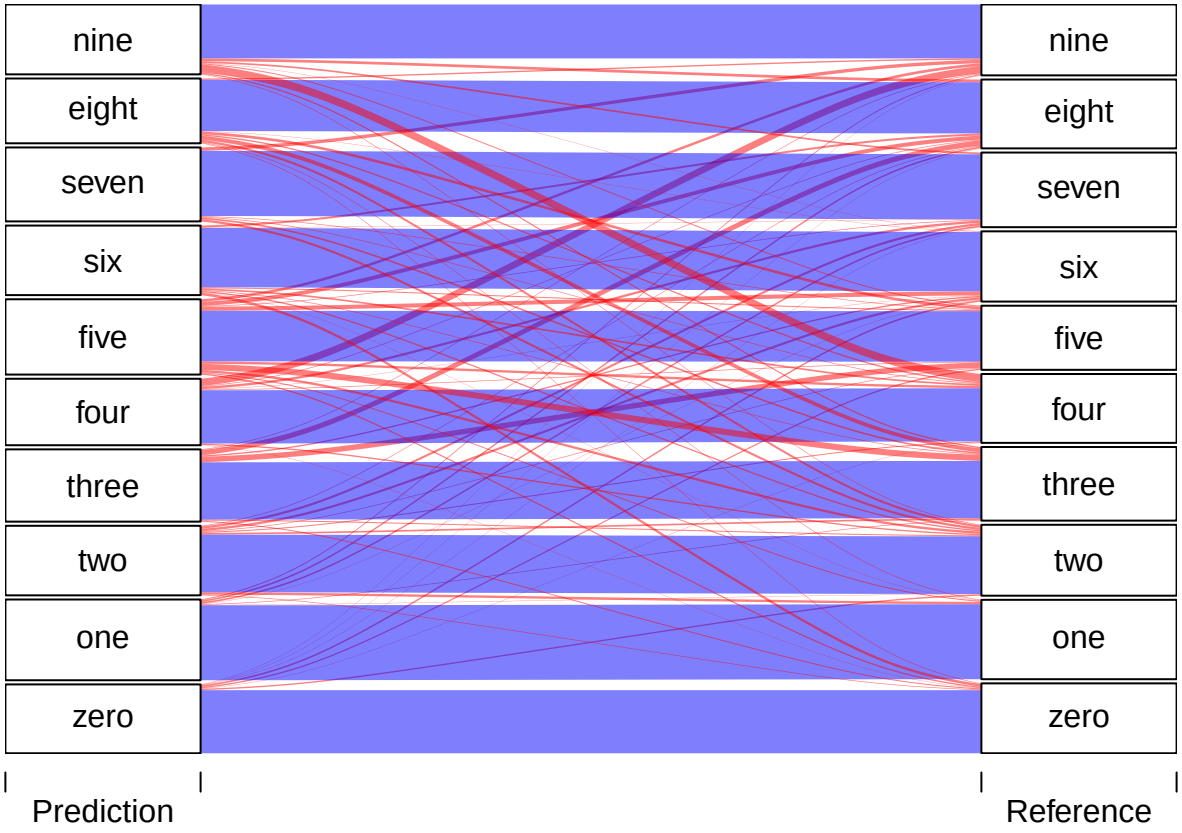
```
##  zero   one   two three  four  five   six seven eight  nine
##   203   238   205   213   196   222   205   218   191   206
```

```
#confusion matrix to find correct and incorrect predictions
#table(labelship=predict.nb, true=test$label)
cm <- confusionMatrix(factor(predict.nb), factor(test$label))
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction zero one two three four five six seven eight nine
##      zero   186   0   4     0    1    4   5     1     1    1
##      one      0 221   1     2    0    0   1     5     5    3
##      two      2   7 170     5    3    2   7     7     2    0
##      three    3   0   3   169    0   16   4     0    16    2
##      four     1   0   4     0  157    2   1     7     2   22
##      five     4   3   7    18    7  149  12     2    12    8
##      six      8   1   4     2    5    3 175     1     6    0
##      seven    0   0   7     5    2    2   0   192     0   10
##      eight    2   2   7    12    4    8   1     0   151    4
##      nine     0   0   2     4   24    3   1     5     8  159
##
## Overall Statistics
##
##                Accuracy : 0.8245
##                  95% CI : (0.8075, 0.8406)
##     No Information Rate : 0.1116
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.805
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: zero Class: one Class: two Class: three Class: four
## Sensitivity              0.90291     0.9444    0.81340      0.77880     0.77340
## Specificity              0.99101     0.9909    0.98146      0.97660     0.97941
## Pos Pred Value           0.91626     0.9286    0.82927      0.79343     0.80102
## Neg Pred Value           0.98944     0.9930    0.97939      0.97452     0.97580
## Prevalence               0.09824     0.1116    0.09967      0.10348     0.09680
## Detection Rate           0.08870     0.1054    0.08107      0.08059     0.07487
## Detection Prevalence     0.09680     0.1135    0.09776      0.10157     0.09347
## Balanced Accuracy        0.94696     0.9677    0.89743      0.87770     0.87640
##                      Class: five Class: six Class: seven Class: eight
## Sensitivity              0.78836    0.84541      0.87273      0.74384
## Specificity              0.96174    0.98413      0.98615      0.97888
## Pos Pred Value           0.67117    0.85366      0.88073      0.79058
## Neg Pred Value           0.97867    0.98309      0.98510      0.97272
## Prevalence               0.09013    0.09871      0.10491      0.09680
## Detection Rate           0.07105    0.08345      0.09156      0.07201
## Detection Prevalence     0.10587    0.09776      0.10396      0.09108
## Balanced Accuracy        0.87505    0.91477      0.92944      0.86136
##                      Class: nine
## Sensitivity              0.76077
## Specificity              0.97511
## Pos Pred Value           0.77184
## Neg Pred Value           0.97356
## Prevalence               0.09967
## Detection Rate           0.07582
## Detection Prevalence     0.09824
```

```
## Balanced Accuracy        0.86794
```

```
plotCM(cm)
```




```
ggPlotCM(cm)
```

```
## Warning: 'heatmap' objects don't have these attributes: 'showlegend'
## Valid attributes include:
## 'type', 'visible', 'opacity', 'name', 'uid', 'ids', 'customdata', 'meta', 'hoverinfo', 'hoverlabel',
```

Conclusions

We set out to demonstrate that machine learning methods can be used to train a machine to recognize hand-written digits. After this study, we can safely conclude that computer vision is very effective at the specific task at hand. However, the study also uncovered that not all predictive modeling approaches are created equal when it comes to successful computer vision outcomes. The models and the data need to be tuned to get the best outcomes.

The data associated with computer vision brings with it complexities of high dimensions. The dimensions in the case of the digits are all the pixels that make up the images of all hand written numbers. Images have a lot of pixels, which is what puts the MNIST data in the high dimension category. However, we can conclude methods such as principal component analysis can be employed to overcome challenges associated with this specific data problem.

In summary, machine learning is quite capable of making predictions associated with computer vision with very good outcomes. In the study here a 90% and greater prediction rates were achieved with the MNIST image data. There is a lot of work that goes into achieving the best results possible, and some of those involve refining machine learning parameters, and reducing the data size without compromising the predictive qualities of it.