# Introduction

This research aims to perform classification algorithms on a dataset which contains handwriting images. The dataset is from a Kaggle competition – Digit Recognizer Competition. Kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

The handwriting images dataset is from MNIST ("Modified National Institute of Standards and Technology"). Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. The goal is to correctly identify digits from a dataset of tens of thousands of handwritten images using classification algorithms such as decision tree, naïve Bayes, SVM, kNN, and Random Forest algorithms. The researcher will review the result of the classification for each algorithm and compare the results.

# Analysis and Models

## Load Libraries

library(sqldf)

## Loading required package: gsubfn

## Loading required package: proto

## Warning in doTryCatch(return(expr), name, parentenv, handler): unable to load shared object '/Library/Frameworks/R.framework/Resources/modules//R\_X11.so':  
## dlopen(/Library/Frameworks/R.framework/Resources/modules//R\_X11.so, 6): Library not loaded: /opt/X11/lib/libSM.6.dylib  
## Referenced from: /Library/Frameworks/R.framework/Versions/3.6/Resources/modules/R\_X11.so  
## Reason: image not found

## Could not load tcltk. Will use slower R code instead.

## Loading required package: RSQLite

library(ggplot2)  
library(class)  
library(e1071)  
library(randomForest)

## randomForest 4.6-14

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

##   
## Attaching package: 'randomForest'

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

library(readr)  
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(RSNNS)

## Loading required package: Rcpp

library(tidyr)  
library(gridExtra)

##   
## Attaching package: 'gridExtra'

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

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

library(randomForest)  
library(readr)  
library(RColorBrewer)  
library(foreign) # For reading and writing data stored by statistical packages such as Minitab,S,SAS,SPSS  
library(tree)  
library(maptree)

## Loading required package: cluster

## Loading required package: rpart

library(rpart) # Recursive Partitioning and Regression Trees (RPart)  
library(RWeka) # Weka

##   
## Attaching package: 'RWeka'

## The following objects are masked from 'package:foreign':  
##   
## read.arff, write.arff

library(FNN) # Fast k-Nearest Neighbors (kNN)

##   
## Attaching package: 'FNN'

## The following objects are masked from 'package:class':  
##   
## knn, knn.cv

library(e1071) # Support Vector Machine (SVM)  
library(tidyverse) # metapackage with lots of helpful functions

## Registered S3 method overwritten by 'cli':  
## method from  
## print.tree tree

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ tibble 3.0.1 ✓ stringr 1.4.0  
## ✓ purrr 0.3.4 ✓ forcats 0.5.0

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## x gridExtra::combine() masks dplyr::combine(), randomForest::combine()  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## x randomForest::margin() masks ggplot2::margin()

library(rpart) #Decision Tree  
library(gbm) #Gradient Boosting

## Loaded gbm 2.1.5

library(rpart.plot) #Model Visualization  
library(factoextra)

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

library(rpart)  
library(rpart.plot)  
library(rattle)

## Rattle: A free graphical interface for data science with R.  
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

##   
## Attaching package: 'rattle'

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

library(factoextra)  
library(FNN)  
library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

## The following objects are masked from 'package:RSNNS':  
##   
## confusionMatrix, train

library(randomForest)  
library(data.table)

##   
## Attaching package: 'data.table'

## The following object is masked from 'package:purrr':  
##   
## transpose

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

library(caTools)

##   
## Attaching package: 'caTools'

## The following object is masked from 'package:RWeka':  
##   
## LogitBoost

library(caret)  
library(randomForest)

## Load Data

The dataset is from Kaggle. (<https://www.kaggle.com/c/digit-recognizer/data>). The researcher used the dataset provided for a Kaggle competition. We transformed the column type for a label to factor and reviewed the structure of the dataset.

#First load the training data in csv format, and then convert "label" to nominal variable.  
filename <-"digit\_train.csv"  
trainset <- read.csv(filename, header = TRUE, stringsAsFactors = TRUE)  
trainset$label<-as.factor(trainset$label)  
dim(trainset)

## [1] 42000 785

#Create a random sample of n% of train data set  
percent <- .25  
set.seed(275)  
DigitSplit <- sample(nrow(trainset),nrow(trainset)\*percent)  
DigitDF <- trainset[DigitSplit,]  
dim(DigitDF)

## [1] 10500 785

filename <-"digit\_test.csv"  
testset <- read.csv(filename, header = TRUE, stringsAsFactors = TRUE)  
#testset$label<-as.factor(testset$label)  
# Wont use test data, instead crossvalidation on train.  
dim(testset)

## [1] 28000 784

## EDA

In this section, the researcher will review the data quality and present initial exploratory data analysis.

sum(is.na(trainset))

## [1] 0

sum(is.na(testset))

## [1] 0

levels(trainset[, 1])

## [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"

digit <- matrix(as.numeric(trainset[8,-1]), nrow = 28) #look at one digit  
image(digit, col = grey.colors(255))

A picture containing clock

Description automatically generated

Below present some handwriting digital images.

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

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

The below graph presents the average intensity of each digit. From left to right, it started from zero to nine. The researcher found that digit zero has the highest average intensity. Digit one has the least average intensity.

trainset$intensity <- apply(trainset[,-1], 1, mean) #takes the mean of each row in train  
  
intbylabel <- aggregate (trainset$intensity, by = list(trainset$label), FUN = mean)  
  
plot <- ggplot(data=intbylabel, aes(x=Group.1, y = x)) +  
 geom\_bar(stat="identity")  
plot + scale\_x\_discrete(limits=0:9) + xlab("digit label") +   
 ylab("average intensity")

A picture containing drawing

Description automatically generated

Below graphs present the intensity distribution for digit 1, 4, 7, and 9. The researcher observed data skewness in the plots. The samples show the distributions are right-skewed. As a result, the mean is higher than the median.

p1 <- qplot(subset(trainset, label ==1)$intensity, binwidth = .75,   
 xlab = "Intensity Histogram for 1")  
  
p2 <- qplot(subset(trainset, label ==4)$intensity, binwidth = .75,  
 xlab = "Intensity Histogram for 4")  
  
p3 <- qplot(subset(trainset, label ==7)$intensity, binwidth = .75,  
 xlab = "Intensity Histogram for 7")  
  
p4 <- qplot(subset(trainset, label ==9)$intensity, binwidth = .75,  
 xlab = "Intensity Histogram for 9")  
  
grid.arrange(p1, p2, p3,p4, ncol = 2)

A close up of text on a black background

Description automatically generated

Below graphs present some samples from the dataset.

train4 <- trainset[trainset$label == 4, ]  
train7 <- trainset[trainset$label == 7, ]  
  
flip <- function(matrix){  
 apply(matrix, 2, rev)  
}  
  
par(mfrow=c(3,3))  
for (i in 20:28){  
 digit <- flip(matrix(rev(as.numeric(train4[i,-c(1, 786)])), nrow = 28)) #look at one digit  
 image(digit, col = grey.colors(255))  
}

These are different handwriting of number four.

A screenshot of a cell phone

Description automatically generated

par(mfrow=c(3,3))  
for (i in 10:18){  
 digit <- flip(matrix(rev(as.numeric(train7[i,-c(1, 786)])), nrow = 28)) #look at one digit  
 image(digit, col = grey.colors(255))  
}

These are different handwriting of number seven.

A screenshot of a cell phone

Description automatically generated

## Models - Decision Tree, Naïve Bayes, kNN, SVM, and Random Forest

### Data preprocessing

First, to ensure the analysis starts with a clean and correct dataset. The researcher re-loaded the dataset and performed a basic review of the dataset.

trainset <- read.csv("digit\_train.csv")  
trainset$label <- factor(trainset$label)  
testset <- read.csv("digit\_test.csv")  
  
dim(trainset)

## [1] 42000 785

dim(testset)

## [1] 28000 784

#Create a random sample of n% of train data set  
percent <- .15  
dimReduce <- .10  
set.seed(275)  
DigitSplit <- sample(nrow(trainset),nrow(trainset)\*percent)  
  
trainset <- trainset[DigitSplit,]  
dim(trainset)

## [1] 6300 785

# Setting static variables used throughout the Models section  
N <- nrow(trainset)  
kfolds <- 2  
set.seed(30)  
holdout <- split(sample(1:N), 1:kfolds)  
  
# Function for model evaluation  
get\_accuracy\_rate <- function(results\_table, total\_cases) {  
 diagonal\_sum <- sum(c(results\_table[[1]], results\_table[[12]], results\_table[[23]], results\_table[[34]],  
 results\_table[[45]], results\_table[[56]], results\_table[[67]], results\_table[[78]],  
 results\_table[[89]], results\_table[[100]]))  
 (diagonal\_sum / total\_cases)\*100  
}

Binarizing the data.

binarized\_trainset <- trainset  
for (col in colnames(binarized\_trainset)) {  
 if (col != "label") {  
 binarized\_trainset[, c(col)] <- ifelse(binarized\_trainset[, c(col)] > 131, 1, 0)  
 }  
}  
for (col in colnames(binarized\_trainset)) {  
 if (col != "label") {  
 binarized\_trainset[, c(col)] <- as.factor(binarized\_trainset[, c(col)])  
 }  
}

This version of the MNIST data set is made of 1,400 individual observations. Each observation is characterized by 785 columns 784 of which are the grayscale values (from 0 to 255) of each pixel of each number in the whole data set. The 784 pixels together form a 28 x 28 square grid, which makes up the drawing of that particular number. The final column not yet discussed is the label, which is the actual digit 0 to 9.

Below are two bar charts displaying the distribution of each of the written digits and the spread of grayscale values:

digit\_freq <- sqldf("SELECT label, COUNT(label) as count  
 FROM trainset  
 GROUP BY label")  
ggplot(digit\_freq, aes(x=reorder(label, -count), y=count)) + geom\_bar(stat="identity") + xlab("Written Digit") + ylab("Frequency Count") + ggtitle("Written Digit by Frequency Count")

This graph presents the count of each digit sample from the dataset.

A screenshot of a cell phone

Description automatically generated

zero <- 0  
fifty <- 0  
one\_hundred <- 0  
one\_hundred\_fifty <- 0  
two\_hundred <- 0  
two\_hundred\_fifty\_five <- 0  
for (col in colnames(trainset)) {  
 if (col != "label") {  
 #binarized\_trainset[,c(col)] <- ifelse(binarized\_trainset[,c(col)] > 131, 1, 0)  
 ifelse(trainset[,c(col)] == 0, zero <- zero + 1, ifelse(  
 trainset[,c(col)] < 51, fifty <- fifty + 1, ifelse(  
 trainset[,c(col)] < 101, one\_hundred <- one\_hundred + 1, ifelse(  
 trainset[,c(col)] < 151, one\_hundred\_fifty <- one\_hundred\_fifty + 1, ifelse(  
 trainset[,c(col)] < 201, two\_hundred <- two\_hundred + 1, two\_hundred\_fifty\_five + 1  
 )  
 )  
 )  
 )  
 )  
 }  
}  
  
color\_bins <- data.frame("color\_bin"=c("0", "50", "100", "150", "200", "255"),  
 "count"=c(zero, fifty, one\_hundred, one\_hundred\_fifty, two\_hundred, two\_hundred\_fifty\_five))  
ggplot(color\_bins, aes(x=reorder(color\_bin, -count), y=count)) + geom\_bar(stat="identity") + xlab("Color Bin") + ylab("Frequency Count") + ggtitle("Color Bin by Frequency Count")

A screenshot of a cell phone

Description automatically generated

Finally, below is another bar chart showing the distribution of final color values in the binarized data:

color\_freq <- data.frame("0"=c(), "1"=c())  
for (col in colnames(binarized\_trainset)) {  
 if (col != "label") {  
 zero <- c(length(which(binarized\_trainset[,c(col)] == 0)))  
 one <- c(length(which(binarized\_trainset[,c(col)] == 1)))  
 color\_freq <- rbind(color\_freq, data.frame("0"=zero, "1"=one))  
 }  
}  
colnames(color\_freq) <- c("zero", "one")  
color\_freq <- data.frame("number"=c("zero", "one"), "count"=c(sum(color\_freq$zero), sum(color\_freq$one)))  
  
ggplot(color\_freq, aes(x=number, y=count)) + geom\_bar(stat="identity") + xlab("Color Number") + ylab("Count") + ggtitle("Color Number by Count")

A screenshot of a cell phone

Description automatically generated

### Decision Tree

Below the researcher prepared three ratios to perform decision tree analysis. Each ratio presents the tree graph, number of splits, complexity parameter, confusion table, and accuracies.

Training ratio/test ratio 80/20

formula = label ~ .  
set.seed(1256)  
train <- sample(1:nrow(trainset),size = ceiling(0.80\*nrow(trainset)),replace = FALSE)  
tree\_train <- trainset[train,]  
tree\_test <- trainset[-train,]  
tree = rpart(formula = formula, data = tree\_train, method = "class")  
#summary(tree)

rpart.plot(tree)

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

A picture containing text

Description automatically generated

barplot(tree$variable.importance)

A screenshot of a cell phone

Description automatically generated

predicted= predict(tree, tree\_test, type="class")  
rsq.rpart(tree)

##   
## Classification tree:  
## rpart(formula = formula, data = tree\_train, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] pixel211 pixel270 pixel290 pixel297 pixel347 pixel350 pixel381 pixel432  
## [9] pixel436 pixel461 pixel489 pixel542 pixel598  
##   
## Root node error: 4440/5040 = 0.88095  
##   
## n= 5040   
##   
## CP nsplit rel error xerror xstd  
## 1 0.105856 0 1.00000 1.00000 0.0051781  
## 2 0.088514 1 0.89414 0.89459 0.0065342  
## 3 0.080631 2 0.80563 0.80698 0.0072486  
## 4 0.066892 3 0.72500 0.72050 0.0076990  
## 5 0.045946 5 0.59122 0.60405 0.0079782  
## 6 0.034910 6 0.54527 0.55000 0.0079909  
## 7 0.026351 7 0.51036 0.52320 0.0079702  
## 8 0.015541 8 0.48401 0.49279 0.0079250  
## 9 0.014640 9 0.46847 0.47928 0.0078974  
## 10 0.013964 10 0.45383 0.47252 0.0078818  
## 11 0.012613 11 0.43986 0.45788 0.0078440  
## 12 0.010811 12 0.42725 0.44234 0.0077977  
## 13 0.010000 14 0.40563 0.43423 0.0077710

## Warning in rsq.rpart(tree): may not be applicable for this method

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

plotcp(tree)

A screenshot of a cell phone

Description automatically generated

fancyRpartPlot(tree)

A picture containing screenshot

Description automatically generated

confMat <- table(tree\_test$label,predicted)  
confusionMatrix(confMat)

## Confusion Matrix and Statistics  
##   
## predicted  
## 0 1 2 3 4 5 6 7 8 9  
## 0 92 2 0 2 0 6 2 5 10 0  
## 1 0 143 5 2 0 6 0 1 0 0  
## 2 0 15 46 7 11 7 11 9 6 4  
## 3 0 12 0 88 2 6 0 5 9 11  
## 4 1 4 2 1 69 8 3 12 7 18  
## 5 5 7 0 8 9 65 3 4 15 4  
## 6 3 5 4 10 3 19 58 9 10 2  
## 7 15 2 9 0 2 1 0 79 7 19  
## 8 0 13 2 11 1 11 11 2 51 8  
## 9 3 2 2 3 4 6 4 11 4 84  
##   
## Overall Statistics  
##   
## Accuracy : 0.6151   
## 95% CI : (0.5876, 0.642)  
## No Information Rate : 0.1627   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5713   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.77311 0.6976 0.65714 0.66667 0.68317 0.48148  
## Specificity 0.97634 0.9867 0.94118 0.96011 0.95168 0.95111  
## Pos Pred Value 0.77311 0.9108 0.39655 0.66165 0.55200 0.54167  
## Neg Pred Value 0.97634 0.9438 0.97902 0.96096 0.97181 0.93860  
## Prevalence 0.09444 0.1627 0.05556 0.10476 0.08016 0.10714  
## Detection Rate 0.07302 0.1135 0.03651 0.06984 0.05476 0.05159  
## Detection Prevalence 0.09444 0.1246 0.09206 0.10556 0.09921 0.09524  
## Balanced Accuracy 0.87472 0.8421 0.79916 0.81339 0.81743 0.71630  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.63043 0.5766 0.42857 0.56000  
## Specificity 0.94435 0.9510 0.94829 0.96486  
## Pos Pred Value 0.47154 0.5896 0.46364 0.68293  
## Neg Pred Value 0.97010 0.9485 0.94087 0.94195  
## Prevalence 0.07302 0.1087 0.09444 0.11905  
## Detection Rate 0.04603 0.0627 0.04048 0.06667  
## Detection Prevalence 0.09762 0.1063 0.08730 0.09762  
## Balanced Accuracy 0.78739 0.7638 0.68843 0.76243

Training ratio/test ratio 70/30

formula = label ~ .  
set.seed(1256)  
train <- sample(1:nrow(trainset),size = ceiling(0.70\*nrow(trainset)),replace = FALSE)  
tree\_train <- trainset[train,]  
tree\_test <- trainset[-train,]  
tree = rpart(formula = formula, data = tree\_train, method = "class")  
#summary(tree)

rpart.plot(tree)

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

A picture containing text

Description automatically generated

barplot(tree$variable.importance)

A screenshot of a cell phone

Description automatically generated

predicted= predict(tree, tree\_test, type="class")  
rsq.rpart(tree)

##   
## Classification tree:  
## rpart(formula = formula, data = tree\_train, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] pixel211 pixel270 pixel290 pixel297 pixel347 pixel350 pixel354 pixel432  
## [9] pixel436 pixel461 pixel489 pixel542 pixel627  
##   
## Root node error: 3894/4410 = 0.88299  
##   
## n= 4410   
##   
## CP nsplit rel error xerror xstd  
## 1 0.108115 0 1.00000 1.00000 0.0054816  
## 2 0.087057 1 0.89188 0.89163 0.0069787  
## 3 0.079866 2 0.80483 0.80534 0.0077296  
## 4 0.065614 3 0.72496 0.73087 0.0081587  
## 5 0.045198 5 0.59373 0.59836 0.0085132  
## 6 0.035953 6 0.54854 0.55470 0.0085252  
## 7 0.026451 7 0.51258 0.51900 0.0084972  
## 8 0.016179 8 0.48613 0.49718 0.0084633  
## 9 0.015922 9 0.46995 0.48947 0.0084482  
## 10 0.014381 10 0.45403 0.47381 0.0084125  
## 11 0.011813 11 0.43965 0.45198 0.0083515  
## 12 0.010914 12 0.42784 0.44093 0.0083155  
## 13 0.010000 14 0.40601 0.42476 0.0082564

## Warning in rsq.rpart(tree): may not be applicable for this method

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

plotcp(tree)

A screenshot of a cell phone

Description automatically generated

fancyRpartPlot(tree)

A screenshot of a cell phone

Description automatically generated

confMat <- table(tree\_test$label,predicted)  
confusionMatrix(confMat)

## Confusion Matrix and Statistics  
##   
## predicted  
## 0 1 2 3 4 5 6 7 8 9  
## 0 141 2 0 4 0 6 2 7 18 3  
## 1 0 219 5 3 1 10 0 1 2 0  
## 2 1 18 61 9 19 14 14 12 7 7  
## 3 6 16 0 132 3 11 1 7 17 12  
## 4 2 8 3 1 105 17 4 17 10 19  
## 5 13 9 0 13 11 96 4 8 21 3  
## 6 4 9 4 12 6 31 102 10 12 3  
## 7 17 4 11 0 2 5 0 125 12 18  
## 8 0 17 4 14 2 21 12 4 81 13  
## 9 6 2 2 5 5 14 4 18 9 115  
##   
## Overall Statistics  
##   
## Accuracy : 0.6228   
## 95% CI : (0.6005, 0.6447)  
## No Information Rate : 0.1608   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5797   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.74211 0.7204 0.67778 0.68394 0.68182 0.42667  
## Specificity 0.97529 0.9861 0.94389 0.95698 0.95334 0.95075  
## Pos Pred Value 0.77049 0.9087 0.37654 0.64390 0.56452 0.53933  
## Neg Pred Value 0.97129 0.9485 0.98322 0.96380 0.97124 0.92465  
## Prevalence 0.10053 0.1608 0.04762 0.10212 0.08148 0.11905  
## Detection Rate 0.07460 0.1159 0.03228 0.06984 0.05556 0.05079  
## Detection Prevalence 0.09683 0.1275 0.08571 0.10847 0.09841 0.09418  
## Balanced Accuracy 0.85870 0.8533 0.81083 0.82046 0.81758 0.68871  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.71329 0.59809 0.42857 0.59585  
## Specificity 0.94791 0.95895 0.94885 0.96170  
## Pos Pred Value 0.52850 0.64433 0.48214 0.63889  
## Neg Pred Value 0.97584 0.95047 0.93728 0.95439  
## Prevalence 0.07566 0.11058 0.10000 0.10212  
## Detection Rate 0.05397 0.06614 0.04286 0.06085  
## Detection Prevalence 0.10212 0.10265 0.08889 0.09524  
## Balanced Accuracy 0.83060 0.77852 0.68871 0.77878

Training ratio/test ratio 60/40

formula = label ~ .  
set.seed(1256)  
train <- sample(1:nrow(trainset),size = ceiling(0.60\*nrow(trainset)),replace = FALSE)  
tree\_train <- trainset[train,]  
tree\_test <- trainset[-train,]  
tree = rpart(formula = formula, data = tree\_train, method = "class")  
#summary(tree)

rpart.plot(tree)

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

A close up of text on a white background

Description automatically generated

barplot(tree$variable.importance)

A screenshot of a cell phone

Description automatically generated

predicted= predict(tree, tree\_test, type="class")  
rsq.rpart(tree)

##   
## Classification tree:  
## rpart(formula = formula, data = tree\_train, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] pixel211 pixel290 pixel297 pixel346 pixel350 pixel430 pixel436 pixel461  
## [9] pixel489 pixel569 pixel627 pixel654  
##   
## Root node error: 3329/3780 = 0.88069  
##   
## n= 3780   
##   
## CP nsplit rel error xerror xstd  
## 1 0.109643 0 1.00000 1.00000 0.0059867  
## 2 0.085611 1 0.89036 0.89126 0.0075883  
## 3 0.083809 2 0.80475 0.82097 0.0082648  
## 4 0.060529 3 0.72094 0.72424 0.0088765  
## 5 0.044157 5 0.59988 0.60859 0.0092103  
## 6 0.040553 6 0.55572 0.57975 0.0092322  
## 7 0.028237 7 0.51517 0.53560 0.0092194  
## 8 0.023430 8 0.48693 0.51427 0.0091932  
## 9 0.016221 9 0.46350 0.48994 0.0091471  
## 10 0.012917 10 0.44728 0.46530 0.0090827  
## 11 0.012016 11 0.43436 0.45089 0.0090366  
## 12 0.011415 12 0.42235 0.44908 0.0090303  
## 13 0.010000 13 0.41093 0.42866 0.0089529

## Warning in rsq.rpart(tree): may not be applicable for this method

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

plotcp(tree)

A screenshot of a cell phone

Description automatically generated

fancyRpartPlot(tree)

A close up of a map

Description automatically generated

confMat <- table(tree\_test$label,predicted)  
confusionMatrix(confMat)

## Confusion Matrix and Statistics  
##   
## predicted  
## 0 1 2 3 4 5 6 7 8 9  
## 0 179 2 0 6 0 6 3 0 34 1  
## 1 0 277 7 7 0 9 0 2 4 0  
## 2 4 26 127 13 7 18 7 10 7 3  
## 3 4 21 6 192 1 12 0 13 19 11  
## 4 4 11 2 1 160 11 5 19 10 36  
## 5 15 11 4 24 5 86 18 14 40 18  
## 6 7 13 18 16 36 35 94 5 14 19  
## 7 24 5 17 0 2 2 2 168 21 25  
## 8 0 25 11 16 1 24 1 14 125 8  
## 9 9 2 1 5 9 10 3 32 17 152  
##   
## Overall Statistics  
##   
## Accuracy : 0.619   
## 95% CI : (0.5998, 0.6381)  
## No Information Rate : 0.156   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.576   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 0.72764 0.7048 0.65803 0.68571 0.72398 0.40376  
## Specificity 0.97713 0.9864 0.95917 0.96116 0.95694 0.93541  
## Pos Pred Value 0.77489 0.9052 0.57207 0.68817 0.61776 0.36596  
## Neg Pred Value 0.97073 0.9476 0.97128 0.96073 0.97302 0.94442  
## Prevalence 0.09762 0.1560 0.07659 0.11111 0.08770 0.08452  
## Detection Rate 0.07103 0.1099 0.05040 0.07619 0.06349 0.03413  
## Detection Prevalence 0.09167 0.1214 0.08810 0.11071 0.10278 0.09325  
## Balanced Accuracy 0.85239 0.8456 0.80860 0.82344 0.84046 0.66958  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.70677 0.60650 0.42955 0.55678  
## Specificity 0.93171 0.95631 0.95514 0.96084  
## Pos Pred Value 0.36576 0.63158 0.55556 0.63333  
## Neg Pred Value 0.98277 0.95164 0.92767 0.94693  
## Prevalence 0.05278 0.10992 0.11548 0.10833  
## Detection Rate 0.03730 0.06667 0.04960 0.06032  
## Detection Prevalence 0.10198 0.10556 0.08929 0.09524  
## Balanced Accuracy 0.81924 0.78140 0.69235 0.75881

### Naïve Bayes

#Run training and Testing for each of the k-folds  
AllResults<-list()  
AllLabels<-list()  
for (k in 1:kfolds){  
   
 DigitDF\_Test <- trainset[holdout[[k]], ]  
 DigitDF\_Train=trainset[-holdout[[k]], ]  
 ## View the created Test and Train sets  
 (head(DigitDF\_Train))  
 (table(DigitDF\_Test$Label))  
 ## Make sure you take the labels out of the testing data  
 (head(DigitDF\_Test))  
 DigitDF\_Test\_noLabel<-DigitDF\_Test[-c(1)]  
 DigitDF\_Test\_justLabel<-DigitDF\_Test$label  
 (head(DigitDF\_Test\_noLabel))  
   
 #### Naive Bayes prediction ussing e1071 package  
 #Naive Bayes Train model  
 train\_naibayes<-naiveBayes(label~., data=DigitDF\_Train, na.action = na.pass)  
 train\_naibayes  
 #summary(train\_naibayes)  
   
 #Naive Bayes model Prediction   
 nb\_Pred <- predict(train\_naibayes, DigitDF\_Test\_noLabel)  
 nb\_Pred  
   
   
 #Testing accurancy of naive bayes model with Kaggle train data sub set  
 (confusionMatrix(nb\_Pred, DigitDF\_Test$label))  
   
 # Accumulate results from each fold, if you like  
 AllResults<- c(AllResults,nb\_Pred)  
 AllLabels<- c(AllLabels, DigitDF\_Test\_justLabel)  
}  
### end crossvalidation -- present results for all folds   
table(unlist(AllResults),unlist(AllLabels))

##   
## 1 2 3 4 5 6 7 8 9 10  
## 1 538 0 92 70 16 72 13 13 13 11  
## 2 4 749 70 115 27 64 53 66 157 46  
## 3 1 0 96 3 1 2 3 0 4 0  
## 4 0 0 28 170 0 7 0 3 5 1  
## 5 0 0 0 1 54 4 1 2 1 2  
## 6 0 0 4 5 6 42 5 1 8 1  
## 7 28 0 140 40 40 49 523 8 12 4  
## 8 0 0 2 4 1 1 0 145 3 1  
## 9 14 6 109 205 62 244 6 32 308 17  
## 10 20 2 19 64 413 72 19 405 102 530

plot(nb\_Pred, ylab = "Density", main = "Naive Bayes Plot")

A close up of a logo

Description automatically generated

get\_accuracy\_rate(table(unlist(AllResults),unlist(AllLabels)), length(AllLabels))

## [1] 50.07937

### kNN

The first algorithm will be kNN. This model requires a k value which is arbitrarily chosen. The first k value will just be the rounded square root of the number of rows in the training data set: 37.

k\_guess = 3  
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
  
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
  
 pred <- knn(train=new\_train[-1], test=new\_test[-1], cl=new\_train$label, k=k\_guess, prob=FALSE)  
  
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
 }  
 table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 599 0 0 0 0 2 3 0 1 0  
## 1 0 751 1 1 1 0 0 3 0 0  
## 2 14 21 494 7 1 2 2 13 5 1  
## 3 2 12 9 620 0 15 3 5 7 4  
## 4 3 13 2 0 562 0 6 2 0 32  
## 5 11 9 0 24 5 484 15 0 2 7  
## 6 9 2 0 1 1 6 602 0 2 0  
## 7 1 18 3 3 7 1 0 627 0 15  
## 8 11 19 5 35 5 13 5 7 502 11  
## 9 8 5 2 7 24 2 1 27 1 536

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 91.69841

k\_guess = 5  
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
  
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
  
 pred <- knn(train=new\_train[-1], test=new\_test[-1], cl=new\_train$label, k=k\_guess, prob=FALSE)  
  
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
 }  
 table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 597 0 0 1 1 2 2 0 2 0  
## 1 0 751 1 1 1 0 0 3 0 0  
## 2 10 25 491 6 1 1 2 18 5 1  
## 3 1 12 9 625 0 10 2 7 6 5  
## 4 1 14 1 0 568 0 7 2 0 27  
## 5 10 11 0 24 4 486 12 0 4 6  
## 6 8 3 0 0 1 5 604 0 2 0  
## 7 1 20 4 0 9 1 0 623 0 17  
## 8 8 22 4 31 6 19 3 7 501 12  
## 9 7 5 1 8 20 0 1 30 2 539

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 91.8254

k\_guess = 7  
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
  
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
  
 pred <- knn(train=new\_train[-1], test=new\_test[-1], cl=new\_train$label, k=k\_guess, prob=FALSE)  
  
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
 }  
 table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 596 0 0 0 0 2 5 0 1 1  
## 1 0 752 1 1 0 0 0 3 0 0  
## 2 10 30 477 7 3 2 5 18 6 2  
## 3 1 13 9 623 0 9 4 5 6 7  
## 4 3 15 1 0 560 0 7 3 0 31  
## 5 8 11 0 24 3 490 13 0 2 6  
## 6 8 5 0 0 1 7 599 0 3 0  
## 7 1 21 3 0 9 0 0 618 0 23  
## 8 6 22 3 33 4 17 2 6 506 14  
## 9 5 5 1 9 17 0 1 31 3 541

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 91.46032

k\_guess = 8  
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
  
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
  
 pred <- knn(train=new\_train[-1], test=new\_test[-1], cl=new\_train$label, k=k\_guess, prob=FALSE)  
  
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
 }  
 table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 596 0 1 1 0 2 3 0 1 1  
## 1 0 752 1 1 0 0 0 3 0 0  
## 2 11 32 471 7 4 2 4 20 7 2  
## 3 1 13 9 624 0 8 5 6 5 6  
## 4 2 16 1 0 565 0 6 3 0 27  
## 5 8 12 0 27 4 483 12 0 4 7  
## 6 7 6 0 0 1 9 598 0 2 0  
## 7 1 27 3 0 8 1 0 614 0 21  
## 8 5 26 2 32 4 19 3 6 500 16  
## 9 7 5 1 8 27 0 1 35 3 526

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 90.93651

### SVM

Next try the SVMs. Remember to experiment with different cost values and different kernels. See some examples below.

cols\_to\_remove = c()  
for (col in colnames(trainset)) { if (col != "label") {  
 if (length(unique(trainset[, c(col)])) == 1) {  
 cols\_to\_remove <- c(cols\_to\_remove, col)  
 } }  
}  
svm\_trainset <- trainset[-which(colnames(trainset) %in% cols\_to\_remove)]

all\_results <- data.frame(orig=c(), pred=c())   
for (k in 1:kfolds) {  
 new\_test <- svm\_trainset[holdout[[k]], ]  
 new\_train <- svm\_trainset[-holdout[[k]], ]  
 new\_test\_no\_label <- new\_test[-c(1)]   
 new\_test\_just\_label <- new\_test[c(1)]  
 test\_model <- svm(label ~ ., new\_train, na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred)) }

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):  
## Variable(s) 'pixel12' and 'pixel13' and 'pixel14' and 'pixel15' and 'pixel59'  
## and 'pixel60' and 'pixel61' and 'pixel80' and 'pixel87' and 'pixel167' and  
## 'pixel170' and 'pixel195' and 'pixel198' and 'pixel225' and 'pixel252' and  
## 'pixel253' and 'pixel280' and 'pixel281' and 'pixel307' and 'pixel308' and  
## 'pixel309' and 'pixel337' and 'pixel447' and 'pixel475' and 'pixel697' and  
## 'pixel751' and 'pixel753' and 'pixel777' and 'pixel778' constant. Cannot scale  
## data.

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):  
## Variable(s) 'pixel62' and 'pixel63' and 'pixel115' and 'pixel138' and 'pixel223'  
## and 'pixel334' and 'pixel362' and 'pixel394' and 'pixel531' and 'pixel559' and  
## 'pixel642' and 'pixel674' and 'pixel703' constant. Cannot scale data.

table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 0 605 0 0 0 0 0 0 0 0  
## 1 0 757 0 0 0 0 0 0 0 0  
## 2 0 560 0 0 0 0 0 0 0 0  
## 3 0 677 0 0 0 0 0 0 0 0  
## 4 0 620 0 0 0 0 0 0 0 0  
## 5 0 557 0 0 0 0 0 0 0 0  
## 6 0 623 0 0 0 0 0 0 0 0  
## 7 0 675 0 0 0 0 0 0 0 0  
## 8 0 613 0 0 0 0 0 0 0 0  
## 9 0 613 0 0 0 0 0 0 0 0

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 12.01587

# Binarizing preprocessed SVM trainset   
binarized\_svm\_trainset <- svm\_trainset  
for (col in colnames(binarized\_svm\_trainset)) {   
 if (col != "label") {   
 binarized\_svm\_trainset[, c(col)] <- ifelse(binarized\_svm\_trainset[, c(col)] > 131, 1, 0) } }   
for (col in colnames(binarized\_svm\_trainset)) { if (col != "label") { binarized\_svm\_trainset[, c(col)] <- as.factor(binarized\_svm\_trainset[, c(col)]) } }  
  
cols\_to\_remove = c()   
for (col in colnames(binarized\_svm\_trainset)) {   
 if (col != "label") {   
 if (length(unique(binarized\_svm\_trainset[, c(col)])) == 1) {   
 cols\_to\_remove <- c(cols\_to\_remove, col) } } }  
  
binarized\_svm\_trainset <- binarized\_svm\_trainset[-which(colnames(binarized\_svm\_trainset) %in% cols\_to\_remove)]

Testing SVM with kernals

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- svm(label ~ ., new\_train, na.action=na.pass)   
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 582 0 2 0 0 9 6 0 6 0  
## 1 0 748 0 2 2 0 0 3 2 0  
## 2 4 7 464 10 16 4 18 15 19 3  
## 3 1 7 7 590 1 36 3 7 19 6  
## 4 2 3 3 1 559 1 7 1 2 41  
## 5 8 12 1 25 9 482 9 3 7 1  
## 6 8 7 2 0 8 14 581 0 3 0  
## 7 2 17 9 1 12 2 1 607 1 23  
## 8 4 17 3 27 5 27 5 2 507 16  
## 9 7 8 0 7 28 8 1 28 7 519

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 89.50794

Polynomial Kernel

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- svm(label ~ ., new\_train, kernel="polynomial", na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 77 516 0 11 0 0 0 1 0 0  
## 1 0 757 0 0 0 0 0 0 0 0  
## 2 0 560 0 0 0 0 0 0 0 0  
## 3 0 677 0 0 0 0 0 0 0 0  
## 4 0 620 0 0 0 0 0 0 0 0  
## 5 0 557 0 0 0 0 0 0 0 0  
## 6 0 623 0 0 0 0 0 0 0 0  
## 7 0 675 0 0 0 0 0 0 0 0  
## 8 0 605 0 3 0 0 0 3 2 0  
## 9 0 613 0 0 0 0 0 0 0 0

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 13.26984

Sigmoid Kernel

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- svm(label ~ ., new\_train, kernel="sigmoid", na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 568 0 3 1 1 16 9 1 5 1  
## 1 0 749 0 2 1 0 0 3 2 0  
## 2 3 12 448 13 19 4 23 17 18 3  
## 3 1 12 7 572 2 38 3 9 23 10  
## 4 2 6 2 1 549 2 7 1 1 49  
## 5 6 15 1 35 9 466 11 5 6 3  
## 6 10 8 3 0 10 14 575 0 3 0  
## 7 2 22 6 2 11 2 1 603 2 24  
## 8 2 24 2 35 5 30 5 3 488 19  
## 9 8 8 0 7 28 7 1 36 7 511

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 87.7619

Radial Kernel

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- binarized\_svm\_trainset[holdout[[k]], ]  
 new\_train <- binarized\_svm\_trainset[-holdout[[k]], ]  
   
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 test\_model <- svm(label ~ ., new\_train, kernel="radial", na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
   
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 582 0 2 0 0 9 6 0 6 0  
## 1 0 748 0 2 2 0 0 3 2 0  
## 2 4 7 464 10 16 4 18 15 19 3  
## 3 1 7 7 590 1 36 3 7 19 6  
## 4 2 3 3 1 559 1 7 1 2 41  
## 5 8 12 1 25 9 482 9 3 7 1  
## 6 8 7 2 0 8 14 581 0 3 0  
## 7 2 17 9 1 12 2 1 607 1 23  
## 8 4 17 3 27 5 27 5 2 507 16  
## 9 7 8 0 7 28 8 1 28 7 519

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

## [1] 89.50794

### Random Forest

all\_results <- data.frame(orig=c(), pred=c())  
for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]   
 new\_train <- trainset[-holdout[[k]], ]  
 new\_test\_no\_label <- new\_test[-c(1)]   
 new\_test\_just\_label <- new\_test[c(1)]  
 test\_model <- randomForest(label ~ ., new\_train, na.action=na.pass)   
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
 all\_results <- rbind(all\_results,  
 data.frame(orig=new\_test\_just\_label$label, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 593 0 1 0 0 2 4 0 5 0  
## 1 0 749 2 1 2 0 0 3 0 0  
## 2 4 3 512 5 8 1 7 10 8 2  
## 3 3 2 6 617 2 20 2 8 11 6  
## 4 3 1 2 0 586 0 8 1 3 16  
## 5 7 6 0 20 2 509 5 1 4 3  
## 6 10 2 3 0 4 11 591 0 2 0  
## 7 3 5 10 1 8 0 0 633 3 12  
## 8 2 9 4 16 2 7 4 2 554 13  
## 9 7 3 0 8 12 6 2 18 6 551

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred),  
 length(all\_results$pred))

## [1] 93.57143

prev\_result <- 0  
best\_result <- 0   
best\_number\_trees <-0   
for (trees in 5:15) {  
 if (trees %% 5 == 0){  
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds) {  
 new\_test <- trainset[holdout[[k]], ]  
 new\_train <- trainset[-holdout[[k]], ]  
 new\_test\_no\_label <- new\_test[-c(1)]   
 new\_test\_just\_label <- new\_test[c(1)]  
 test\_model <- randomForest(label ~ ., new\_train, replace=TRUE,  
 na.action=na.pass)  
 pred <- predict(test\_model, new\_test\_no\_label, type=c("class"))  
 all\_results <- rbind(all\_results, data.frame(orig=new\_test\_just\_label$label, pred=pred))  
 }  
 #table(all\_results$orig, all\_results$pred)  
 new\_result <- get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))  
 if (new\_result > prev\_result) {  
 prev\_result <- new\_result  
 } else {  
 best\_number\_trees <- trees   
 best\_result <- new\_result   
 break  
 } }  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 590 0 2 0 0 2 6 0 5 0  
## 1 0 750 1 2 2 0 0 2 0 0  
## 2 5 5 513 5 6 2 5 8 9 2  
## 3 5 2 6 614 2 22 2 6 11 7  
## 4 2 0 2 0 587 0 7 1 3 18  
## 5 6 6 2 21 2 503 5 1 6 5  
## 6 10 3 2 0 3 8 594 0 3 0  
## 7 3 4 10 2 7 0 0 632 3 14  
## 8 2 10 4 13 2 6 6 1 557 12  
## 9 7 3 0 10 12 3 1 19 7 551

get\_accuracy\_rate(table(all\_results$orig, all\_results$pred),  
 length(all\_results$pred))

## [1] 93.50794

# Results

Decision Tree Model:

A Decision Tree is a supervised learning predictive model that uses a set of binary rules to calculate a target value. The decision tree classifies and predicts what the digits might be by analyzing the possibilities of how each digit was written. Based on the modeling in each split train and test ratio, we can summarize the result as below.

The resubstituting rate is a measure of error. It is the proportion of original observations that were misclassified by various subsets of the original tree. The lower number does not mean better. The largest tree will always yield the lowest resubstituting error rate. However, choosing the tree with the lowest resubstituting rate is not the optimal choice, as this tree will have a bias. Large trees will put random variation in the predictions as they overfit outliers.

X-fold cross-validation is used to obtain a cross-validated error rate, from which the optimal tree is selected instead of selecting a tree based on the resubstituting error rate. The X-fold cross-validation involves creating X-random subsets of the original data, setting one portion aside as a test set, constructing a tree for the remaining X-1 portions, and evaluating the tree using the test portion. This is repeated for all portions, and an estimate of the error is evaluated. Adding up the error across the X portions represents the cross-validated error rate. The tree yielding the lowest cross-validated error rate (xerror) is selected as the tree that best fits the data. Standard Error (xstd) is the standard deviation of error across the cross-validation sets.

80% train data and 20% test data

* Root node error: 4440/5040 = 0.88095 (This is the error rate for a single node tree)
* Complexity Table:

## CP nsplit rel error xerror xstd  
## 1 0.105856 0 1.00000 1.00000 0.0051781  
## 2 0.088514 1 0.89414 0.89459 0.0065342  
## 3 0.080631 2 0.80563 0.80698 0.0072486  
## 4 0.066892 3 0.72500 0.72050 0.0076990  
## 5 0.045946 5 0.59122 0.60405 0.0079782  
## 6 0.034910 6 0.54527 0.55000 0.0079909  
## 7 0.026351 7 0.51036 0.52320 0.0079702  
## 8 0.015541 8 0.48401 0.49279 0.0079250  
## 9 0.014640 9 0.46847 0.47928 0.0078974  
## 10 0.013964 10 0.45383 0.47252 0.0078818  
## 11 0.012613 11 0.43986 0.45788 0.0078440  
## 12 0.010811 12 0.42725 0.44234 0.0077977  
## 13 0.010000 14 0.40563 0.43423 0.0077710

* Confusion matrix

## Confusion Matrix and Statistics  
##   
## predicted  
## 0 1 2 3 4 5 6 7 8 9  
## 0 92 2 0 2 0 6 2 5 10 0  
## 1 0 143 5 2 0 6 0 1 0 0  
## 2 0 15 46 7 11 7 11 9 6 4  
## 3 0 12 0 88 2 6 0 5 9 11  
## 4 1 4 2 1 69 8 3 12 7 18  
## 5 5 7 0 8 9 65 3 4 15 4  
## 6 3 5 4 10 3 19 58 9 10 2  
## 7 15 2 9 0 2 1 0 79 7 19  
## 8 0 13 2 11 1 11 11 2 51 8  
## 9 3 2 2 3 4 6 4 11 4 84  
##   
## Overall Statistics  
##   
## Accuracy : 0.6151   
## 95% CI : (0.5876, 0.642)  
## No Information Rate : 0.1627   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5713   
##   
## Mcnemar's Test P-Value : NA

70% train data and 30% test data

* Root node error: 3894/4410 = 0.88299 (This is the error rate for a single node tree)
* Node number 2: 32 observations
* Complexity Table:

## CP nsplit rel error xerror xstd  
## 1 0.108115 0 1.00000 1.00000 0.0054816  
## 2 0.087057 1 0.89188 0.89163 0.0069787  
## 3 0.079866 2 0.80483 0.80534 0.0077296  
## 4 0.065614 3 0.72496 0.73087 0.0081587  
## 5 0.045198 5 0.59373 0.59836 0.0085132  
## 6 0.035953 6 0.54854 0.55470 0.0085252  
## 7 0.026451 7 0.51258 0.51900 0.0084972  
## 8 0.016179 8 0.48613 0.49718 0.0084633  
## 9 0.015922 9 0.46995 0.48947 0.0084482  
## 10 0.014381 10 0.45403 0.47381 0.0084125  
## 11 0.011813 11 0.43965 0.45198 0.0083515  
## 12 0.010914 12 0.42784 0.44093 0.0083155  
## 13 0.010000 14 0.40601 0.42476 0.0082564

* Confusion matrix

## Confusion Matrix and Statistics  
##   
## predicted  
## 0 1 2 3 4 5 6 7 8 9  
## 0 141 2 0 4 0 6 2 7 18 3  
## 1 0 219 5 3 1 10 0 1 2 0  
## 2 1 18 61 9 19 14 14 12 7 7  
## 3 6 16 0 132 3 11 1 7 17 12  
## 4 2 8 3 1 105 17 4 17 10 19  
## 5 13 9 0 13 11 96 4 8 21 3  
## 6 4 9 4 12 6 31 102 10 12 3  
## 7 17 4 11 0 2 5 0 125 12 18  
## 8 0 17 4 14 2 21 12 4 81 13  
## 9 6 2 2 5 5 14 4 18 9 115  
##   
## Overall Statistics  
##   
## Accuracy : 0.6228   
## 95% CI : (0.6005, 0.6447)  
## No Information Rate : 0.1608   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5797   
##   
## Mcnemar's Test P-Value : < 2.2e-16

60% train data and 40% test data

* Root node error: 3329/3780 = 0.88069 (This is the error rate for a single node tree)
* Complexity Table:

## CP nsplit rel error xerror xstd  
## 1 0.109643 0 1.00000 1.00000 0.0059867  
## 2 0.085611 1 0.89036 0.89126 0.0075883  
## 3 0.083809 2 0.80475 0.82097 0.0082648  
## 4 0.060529 3 0.72094 0.72424 0.0088765  
## 5 0.044157 5 0.59988 0.60859 0.0092103  
## 6 0.040553 6 0.55572 0.57975 0.0092322  
## 7 0.028237 7 0.51517 0.53560 0.0092194  
## 8 0.023430 8 0.48693 0.51427 0.0091932  
## 9 0.016221 9 0.46350 0.48994 0.0091471  
## 10 0.012917 10 0.44728 0.46530 0.0090827  
## 11 0.012016 11 0.43436 0.45089 0.0090366  
## 12 0.011415 12 0.42235 0.44908 0.0090303  
## 13 0.010000 13 0.41093 0.42866 0.0089529

* Confusion matrix

## Confusion Matrix and Statistics  
##   
## predicted  
## 0 1 2 3 4 5 6 7 8 9  
## 0 179 2 0 6 0 6 3 0 34 1  
## 1 0 277 7 7 0 9 0 2 4 0  
## 2 4 26 127 13 7 18 7 10 7 3  
## 3 4 21 6 192 1 12 0 13 19 11  
## 4 4 11 2 1 160 11 5 19 10 36  
## 5 15 11 4 24 5 86 18 14 40 18  
## 6 7 13 18 16 36 35 94 5 14 19  
## 7 24 5 17 0 2 2 2 168 21 25  
## 8 0 25 11 16 1 24 1 14 125 8  
## 9 9 2 1 5 9 10 3 32 17 152  
##   
## Overall Statistics  
##   
## Accuracy : 0.619   
## 95% CI : (0.5998, 0.6381)  
## No Information Rate : 0.156   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.576   
##   
## Mcnemar's Test P-Value : < 2.2e-16

Based on the accuracy rate, the researcher found that the model with 70% training data and 30% test data had the highest accuracy – 0.6228. The Kappa statistic (or value) is a metric that compares an Observed Accuracy with an Expected Accuracy (random chance). The kappa statistic is how closely the instances classified by the machine learning classifier matched the data labeled as ground truth, controlling for the accuracy of a random classifier as measured by the expected accuracy. The model with 70% training data and 30% test data also had the highest Kappa value. In other words, accuracy is the highest among the three models.

Naïve Bayes Model:

Naïve Bayes Model is a classification technique based on Bayes’ Theorem, which assumes independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. The researcher converted the data set into a frequency table, created the likelihood table by finding the probabilities. Then, the researcher used the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction. The accuracy rate is 50.07937.

kNN’s:

The k-nearest neighbors (KNN) algorithm is a supervised machine learning algorithm that can be used to solve both classification and regression problems. It is more widely used in classification problems in the industry. A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is assigned to the class of its nearest neighbor. Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise, but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. The researcher ran kNN models with K=3, K=5, K=7, K=8. In this case, the researcher found that the model had the highest accuracy when K was 5.

K = 3

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 599 0 0 0 0 2 3 0 1 0  
## 1 0 751 1 1 1 0 0 3 0 0  
## 2 14 21 494 7 1 2 2 13 5 1  
## 3 2 12 9 620 0 15 3 5 7 4  
## 4 3 13 2 0 562 0 6 2 0 32  
## 5 11 9 0 24 5 484 15 0 2 7  
## 6 9 2 0 1 1 6 602 0 2 0  
## 7 1 18 3 3 7 1 0 627 0 15  
## 8 11 19 5 35 5 13 5 7 502 11  
## 9 8 5 2 7 24 2 1 27 1 536

* Accuracy

91.69841

K = 5

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 597 0 0 1 1 2 2 0 2 0  
## 1 0 751 1 1 1 0 0 3 0 0  
## 2 10 25 491 6 1 1 2 18 5 1  
## 3 1 12 9 625 0 10 2 7 6 5  
## 4 1 14 1 0 568 0 7 2 0 27  
## 5 10 11 0 24 4 486 12 0 4 6  
## 6 8 3 0 0 1 5 604 0 2 0  
## 7 1 20 4 0 9 1 0 623 0 17  
## 8 8 22 4 31 6 19 3 7 501 12  
## 9 7 5 1 8 20 0 1 30 2 539

* Accuracy

91.8254

K = 7

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 596 0 0 0 0 2 5 0 1 1  
## 1 0 752 1 1 0 0 0 3 0 0  
## 2 10 30 477 7 3 2 5 18 6 2  
## 3 1 13 9 623 0 9 4 5 6 7  
## 4 3 15 1 0 560 0 7 3 0 31  
## 5 8 11 0 24 3 490 13 0 2 6  
## 6 8 5 0 0 1 7 599 0 3 0  
## 7 1 21 3 0 9 0 0 618 0 23  
## 8 6 22 3 33 4 17 2 6 506 14  
## 9 5 5 1 9 17 0 1 31 3 541

* Accuracy

91.46032

K = 8

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 596 0 1 1 0 2 3 0 1 1  
## 1 0 752 1 1 0 0 0 3 0 0  
## 2 11 32 471 7 4 2 4 20 7 2  
## 3 1 13 9 624 0 8 5 6 5 6  
## 4 2 16 1 0 565 0 6 3 0 27  
## 5 8 12 0 27 4 483 12 0 4 7  
## 6 7 6 0 0 1 9 598 0 2 0  
## 7 1 27 3 0 8 1 0 614 0 21  
## 8 5 26 2 32 4 19 3 6 500 16  
## 9 7 5 1 8 27 0 1 35 3 526

* Accuracy

90.93651

SVMs:

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outlier detection. The advantages of support vector machines are

1. Effective in high dimensional spaces.
2. Still effective in cases where the number of dimensions is greater than the number of samples.
3. It uses a subset of training points in the decision function (called support vectors), which is also memory efficient.
4. Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include

1. If the number of features is greater than the number of samples, avoid over-fitting in choosing Kernel functions, and regularization terms are crucial.
2. SVMs do not directly provide probability estimates; these are calculated using an expensive five-fold cross-validation.

The researcher ran the SVMs models with different kernel options and observed the accuracy rates. Below summarized how each kernel is commonly used.

* Polynomial kernel is popular in image processing. Equation is:

Polynomial kernel equation

where d is the degree of the polynomial.

* Sigmoid kernel can be used as the proxy for neural networks. Equation is:

 Sigmoid kernel equation

* Gaussian radial basis function (RBF) is a general-purpose kernel; used when there is no prior knowledge about the data. Equation is:

Gaussian radial basis function (RBF)

The accuracy rates and results from using different kernels are summarized below.

Without a kernel specified

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 582 0 2 0 0 9 6 0 6 0  
## 1 0 748 0 2 2 0 0 3 2 0  
## 2 4 7 464 10 16 4 18 15 19 3  
## 3 1 7 7 590 1 36 3 7 19 6  
## 4 2 3 3 1 559 1 7 1 2 41  
## 5 8 12 1 25 9 482 9 3 7 1  
## 6 8 7 2 0 8 14 581 0 3 0  
## 7 2 17 9 1 12 2 1 607 1 23  
## 8 4 17 3 27 5 27 5 2 507 16  
## 9 7 8 0 7 28 8 1 28 7 519

* Accuracy

89.50794

Kernel = Polynomial

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 77 516 0 11 0 0 0 1 0 0  
## 1 0 757 0 0 0 0 0 0 0 0  
## 2 0 560 0 0 0 0 0 0 0 0  
## 3 0 677 0 0 0 0 0 0 0 0  
## 4 0 620 0 0 0 0 0 0 0 0  
## 5 0 557 0 0 0 0 0 0 0 0  
## 6 0 623 0 0 0 0 0 0 0 0  
## 7 0 675 0 0 0 0 0 0 0 0  
## 8 0 605 0 3 0 0 0 3 2 0  
## 9 0 613 0 0 0 0 0 0 0 0

* Accuracy

13.26984

Kernel = Sigmoid

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 568 0 3 1 1 16 9 1 5 1  
## 1 0 749 0 2 1 0 0 3 2 0  
## 2 3 12 448 13 19 4 23 17 18 3  
## 3 1 12 7 572 2 38 3 9 23 10  
## 4 2 6 2 1 549 2 7 1 1 49  
## 5 6 15 1 35 9 466 11 5 6 3  
## 6 10 8 3 0 10 14 575 0 3 0  
## 7 2 22 6 2 11 2 1 603 2 24  
## 8 2 24 2 35 5 30 5 3 488 19  
## 9 8 8 0 7 28 7 1 36 7 511

* Accuracy

87.7619

Kernel = Radial

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 582 0 2 0 0 9 6 0 6 0  
## 1 0 748 0 2 2 0 0 3 2 0  
## 2 4 7 464 10 16 4 18 15 19 3  
## 3 1 7 7 590 1 36 3 7 19 6  
## 4 2 3 3 1 559 1 7 1 2 41  
## 5 8 12 1 25 9 482 9 3 7 1  
## 6 8 7 2 0 8 14 581 0 3 0  
## 7 2 17 9 1 12 2 1 607 1 23  
## 8 4 17 3 27 5 27 5 2 507 16  
## 9 7 8 0 7 28 8 1 28 7 519

* Accuracy

89.50794

The researcher found that using a Radial kernel had the highest accuracy rate – 89.50794. The result was the same as not having a kernel specified. The general-purpose method could classify the images better than using other kernels tested.

Random Forest Model:

Random Forest classifier is an ensemble tree-based learning algorithm. The Random Forest Classifier is a set of decision trees from a randomly selected subset of the training set. It aggregates the votes from different decision trees to decide the final class of the test object. The difference between the Random Forest algorithm and the decision tree algorithm is that in Random Forest, finding the root node and splitting the feature nodes will run randomly. Below, the researcher ran the random forest model with data prepared with K-fold and data that was binarized. k-Fold Cross-Validation is a resampling procedure used to evaluate machine learning models on a limited data sample. Binary data is data whose unit can take on only two possible states.

Data with K-fold

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 593 0 1 0 0 2 4 0 5 0  
## 1 0 749 2 1 2 0 0 3 0 0  
## 2 4 3 512 5 8 1 7 10 8 2  
## 3 3 2 6 617 2 20 2 8 11 6  
## 4 3 1 2 0 586 0 8 1 3 16  
## 5 7 6 0 20 2 509 5 1 4 3  
## 6 10 2 3 0 4 11 591 0 2 0  
## 7 3 5 10 1 8 0 0 633 3 12  
## 8 2 9 4 16 2 7 4 2 554 13  
## 9 7 3 0 8 12 6 2 18 6 551

* Accuracy

93.57143

Binarized data

* Confusion Matrix

## 0 1 2 3 4 5 6 7 8 9  
## 0 590 0 2 0 0 2 6 0 5 0  
## 1 0 750 1 2 2 0 0 2 0 0  
## 2 5 5 513 5 6 2 5 8 9 2  
## 3 5 2 6 614 2 22 2 6 11 7  
## 4 2 0 2 0 587 0 7 1 3 18  
## 5 6 6 2 21 2 503 5 1 6 5  
## 6 10 3 2 0 3 8 594 0 3 0  
## 7 3 4 10 2 7 0 0 632 3 14  
## 8 2 10 4 13 2 6 6 1 557 12  
## 9 7 3 0 10 12 3 1 19 7 551

* Accuracy

93.50794

The researcher found that the accuracy rates from using two different datasets were similar. Using random forest classification modeling on data with k-fold cross validation had a slightly higher accuracy – 93.57143.

# Conclusions

The research purpose of this paper is to compare the classification results using different classification models. The researcher reviewed data with structured exploratory data analysis to understand the dataset. The accuracy rates from each model used were analyzed and concluded in the result section. The researcher found that the Random Forest Classification had the best classification result. Random forest adds additional randomness to the model while growing the trees. Instead of searching for the most crucial feature while splitting a node, it searches for the best feature among a random subset of features. It results in a wide diversity that generally results in a better model. The result also matched what the researcher expected with his experience.

# References

1. Explanation of the Decision Tree Model. Retrieved from https://webfocusinfocenter.informationbuilders.com/wfappent/TLs/TL\_rstat/source/DecisionTree47.htm -2020
2. The Nature of Statistical Learning Theory - Vapnik – 1995
3. Support Vector Machines: Training and Applications - Osuna, Freund, et al. – 1997
4. A Probabilistic Theory of Pattern Recognition - Devroye, Gyorfi, et al. - 1996
5. Multiple randomized classifiers: Mrcl - Amit, Blanchard, et al. - 1999