Homework 7 Tips

knitr::opts\_chunk$set(echo = TRUE)

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

# Introduction

In a previous HWs, a couple models were tested. Those models were Decision Trees and Naive Bayes. The ultimate goal of that trial was to investigate those models.

This investigation will be a follow up to that previous one attempting three new kinds of models: kNN, SVM, and Random Forest. These will be tested in a similar way with the same goal to find the pros and cons of each.

The same data set will be used to test these new models: the MNIST handwritten digit recognition data set.

# Analysis and Models

Please describe and introduce the models here ….

## About the data

Discuss the data. Feel free to reuse the EDA section from previous submissions if it included this data set.

First – load data and format. Next – organize sets for crossvalidation.

trainset <- read.csv("digit\_train.csv")  
trainset$label <- factor(trainset$label)  
  
  
#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  
}

## Data preprocessing.

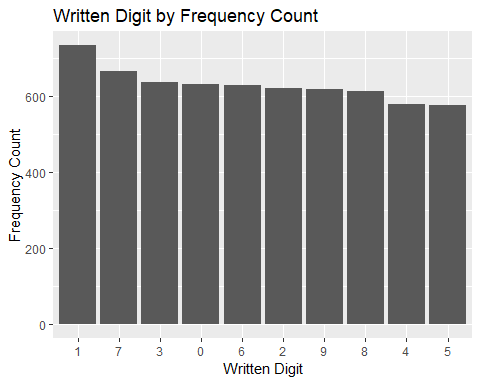
In this example, we binarize the data.

# Discretizing at 87%  
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)])  
 }  
}

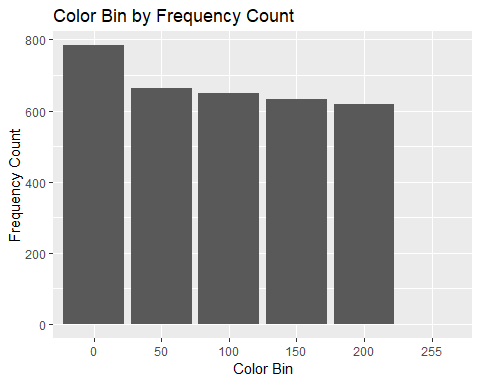
This version of the MNIST data set is made of 1,400 individual observations. Each observation is characterized by 785 columns 784 of which are the gray scale values (from 0 to 255) of each pixel of each number in the whole data set. The 784 pixels together form a 28 x 28 square grid which make up the 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 gray scale 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")

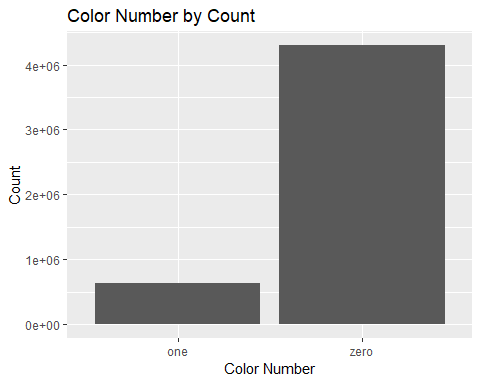


zero <- 0  
fifty <- 0  
one\_hundred <- 0  
one\_hundred\_fifty <- 0  
two\_hundred <- 0  
two\_hundred\_fifty\_five <- 0  
for (col in colnames(trainset)) {  
 if (col != "label") {  
 #binarized\_trainset[,c(col)] <- ifelse(binarized\_trainset[,c(col)] > 131, 1, 0)  
 ifelse(trainset[,c(col)] == 0, 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")



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")



## Models

As a quick note before beginning, all models will be tested in the same fashion: k- fold cross validation. Model results will be presented a confusion matrices along with an accuracy rating as a percentage.

### 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 = 7# round(sqrt(nrow(trainset)))  
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, test=new\_test, 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  
## 0 619 1 0 1 0 2 5 1 0 2  
## 1 0 726 1 0 2 0 0 3 0 1  
## 2 15 29 523 6 4 2 3 26 9 4  
## 3 2 9 1 591 0 10 2 8 7 7  
## 4 0 16 0 0 509 0 5 1 1 45  
## 5 5 7 0 15 3 518 13 1 2 12  
## 6 9 6 0 0 2 5 607 0 0 0  
## 7 0 28 0 0 3 0 1 621 0 13  
## 8 3 23 6 23 5 22 8 5 504 13  
## 9 2 9 0 8 12 2 0 26 1 558

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

## [1] 91.68254

#### k = 3

#### k = 5

#### k = 8

Whic result is the best of all of the kNN models???

### 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)]

The first attempt will be a straight baseline using the data preprocessed for SVM.

# Baseline SVM - no changes to data  
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$label, pred=pred))  
}  
table(all\_results$orig, all\_results$pred)

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 0 631 0 0 0 0 0 0 0 0  
## 1 0 733 0 0 0 0 0 0 0 0  
## 2 0 621 0 0 0 0 0 0 0 0  
## 3 0 637 0 0 0 0 0 0 0 0  
## 4 0 577 0 0 0 0 0 0 0 0  
## 5 0 576 0 0 0 0 0 0 0 0  
## 6 0 629 0 0 0 0 0 0 0 0  
## 7 0 666 0 0 0 0 0 0 0 0  
## 8 0 612 0 0 0 0 0 0 0 0  
## 9 0 618 0 0 0 0 0 0 0 0

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

## [1] 11.63492

What is the accuracy of the above experiment? How can we compute this from the confusion matrix??

#```{r, echo=TRUE, message=FALSE, warning=FALSE} # 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 on new data

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\_resultsorig, all\_resultspred))

Did accurracy improve here? Which sets of parameters seem to work best for SVMs on this data set??? lets try some more.   
  
#### Polynomial Kernel  
#```{r, echo=TRUE, message=FALSE, warning=FALSE}  
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$pred)  
get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

#### Radial Kernel

#### Sigmoid Kernel

#### Try varying Cost

#### SVM summary

Which Kernel seems to do the best???

### Random Forest

Next – Lets try a Random Forest Model.

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  
## 0 617 0 0 0 1 0 6 2 5 0  
## 1 0 719 1 1 4 0 0 5 2 1  
## 2 7 7 566 7 8 1 8 11 3 3  
## 3 3 4 10 574 3 14 6 5 9 9  
## 4 2 1 3 0 545 0 4 0 3 19  
## 5 13 8 1 15 1 520 10 0 3 5  
## 6 10 3 1 0 4 6 601 0 4 0  
## 7 1 7 7 0 6 0 1 626 4 14  
## 8 1 7 4 17 1 9 9 1 548 15  
## 9 3 3 3 8 13 3 0 18 5 562

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

## [1] 93.30159

How are the results??? Lets try the binarized version of the data …

Lets try varying the number of trees. Tree selection will be automated moving up in multiples of 5.

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  
 }  
 }  
}   
paste("Best Number of Trees:", best\_number\_trees, "- Best Result:", best\_result, sep=" ")

## [1] "Best Number of Trees: 10 - Best Result: 93.2698412698413"

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

##   
## 0 1 2 3 4 5 6 7 8 9  
## 0 617 0 1 0 1 2 3 1 6 0  
## 1 0 721 1 1 4 0 0 3 2 1  
## 2 5 6 570 7 8 1 8 11 3 2  
## 3 5 4 9 570 2 16 6 4 11 10  
## 4 2 1 3 0 542 0 4 0 3 22  
## 5 10 9 0 11 2 522 11 2 3 6  
## 6 8 3 3 0 5 5 602 0 3 0  
## 7 2 6 8 0 5 1 0 628 5 11  
## 8 1 5 5 17 2 9 9 2 548 14  
## 9 4 4 3 10 16 2 1 16 6 556

# Results

Which model and parameter set did the best?? Did binarizing the data help in all cases???

# Conclusions