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.

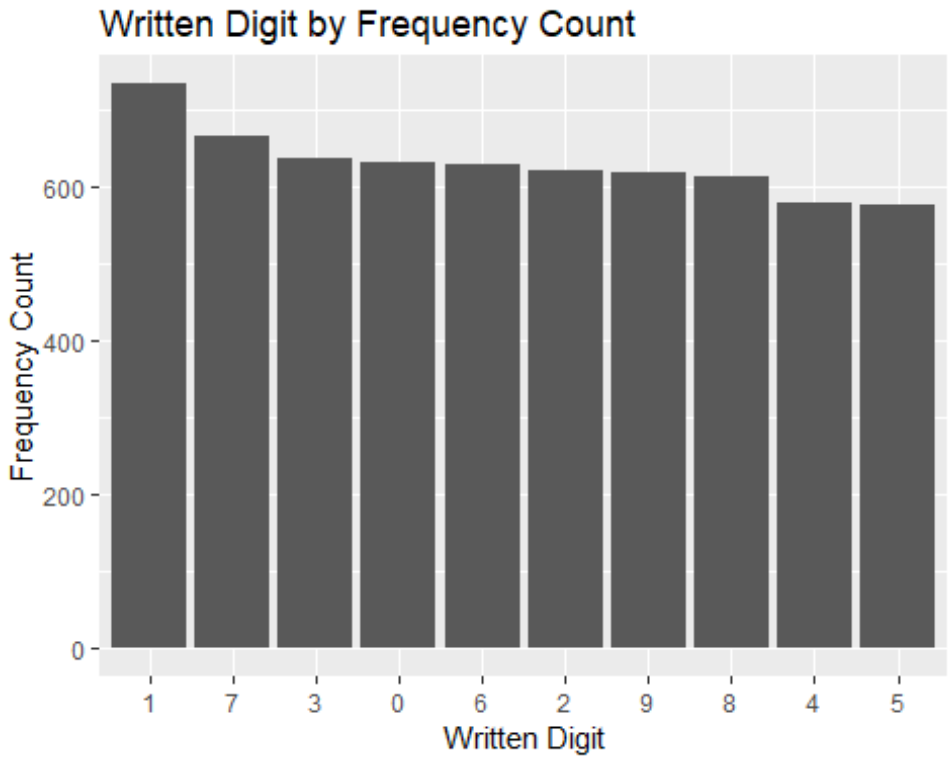
```
# Discretizing at 87%
binarized_trainset <- trainset
for (col in colnames(binarized_trainset)) {
  if (col != "label") {
    binarized_trainset[, c(col)] <- ifelse(binarized_trainset[, c(col)] >
131, 1, 0)
  }
}
for (col in colnames(binarized_trainset)) {
  if (col != "label") {
    binarized_trainset[, 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:

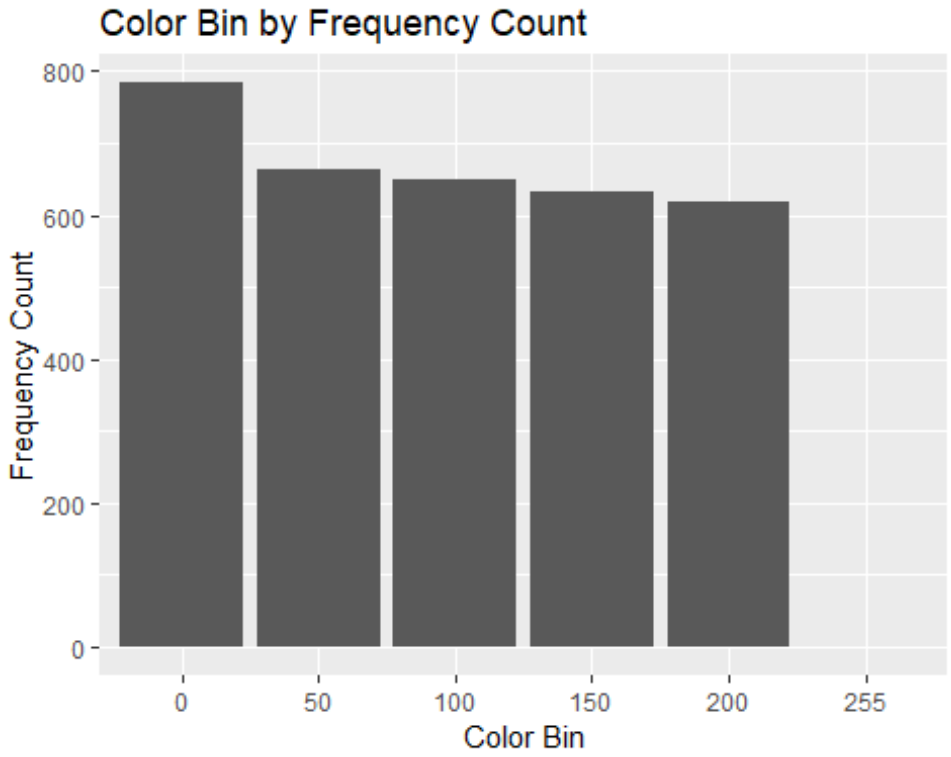
```
digit_freq <- sqldf("SELECT label, COUNT(label) as count
                     FROM trainset
                     GROUP BY label")
```

```
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")
```



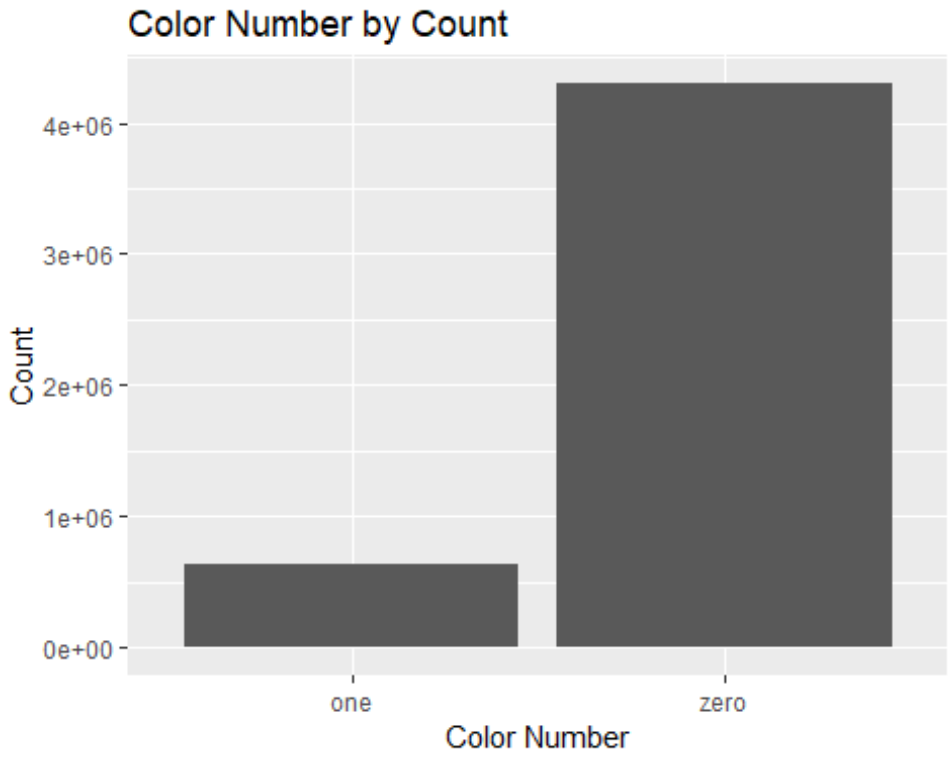

```
zero <- 0
fifty <- 0
one_hundred <- 0
one_hundred_fifty <- 0
two_hundred <- 0
two_hundred_fifty_five <- 0
for (col in colnames(trainset)) {
  if (col != "label") {
    #binarized_trainset[,c(col)] <- ifelse(binarized_trainset[,c(col)] > 131,
1, 0)
    ifelse(trainset[,c(col)] == 0, zero <- zero + 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_results$$pred$$)$$get_accuracy_rate(table(all_results$$orig, all_results$pred), length(all_results$$pred$$))

```
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