forrest\_fallon\_hw7.R

Forrest Fallon

2021-11-29

#This week’s exercise is another chance to learn the inner workings of some more complex modeling algorithms at our disposal; specifically, the kMM, SVM, and RandomForest programs. We will be using the same dataset as last week, which is a collection of pixel-data from handwriting samples of the numbers one through nine.

#The nature of this data is nothing short of complex and variable heavy. If not queried correctly, this data will leave you waiting for hours to run certain models, and in some cases will even endlessly loop until you realize your PC is running at 100% with no end in sight.

#As shown below, the processing and cleaning of this data is a bit more involved than previous exercises. The need for splicing this data into a fraction of itself is so we can run this data without a super-computer at our disposal, and some models agree better with a manageable dataset. There are also some functions created to reduce input needed at later stages.

#Minor housekeeping note: accuracies of each test are bolded in this document, but they are also presented in list form before the conclusion.

#Load necessary packages  
library(sqldf)

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

#load the data  
trainset <- read.csv("C:/Users/forrest/Desktop/schoo/IST 707/HW6/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,]   
  
  
# 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   
}  
  
# 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)])   
 }   
}  
  
digit\_freq <- sqldf("SELECT label, COUNT(label) as count FROM trainset GROUP BY label")  
  
  
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   
 )   
 )   
 )   
 )   
 )   
 }  
}  
  
  
  
  
  
# 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))  
}   
  
get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

**## [1] 91.66667**

#new k\_guesses  
k\_guess = 3 # 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)  
get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

**## [1] 91.93651**

k\_guess = 5 # 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)  
get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

**## [1] 91.90476**

k\_guess = 8 # 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)  
get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

**## [1] 91.07937**  
  
# Out of all of the k\_guess values, 3 presented the highest accuracy of 91.936. The value of 5 was not far behind, holding an accuracy of 91.905.   
  
  
#SVM testing  
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)]  
  
   
# 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))   
}

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

**## [1] 12.01587**  
# After running the baseline SVM, we are presented with an error along the lines of “Cannot scale data.” This is likely what produces such a low accuracy as seen above. The confusion matrix that is produced here also points to the non-scaling nature of said data, as it shows only results under the column “1”. In order to remedy this, we will binarize the trainset.

# 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)]  
  
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),

**## [1] 89.50794**

#much better accuracy, it seems as though the SVM model works much better with this amount of data  
#when everything is binarized.

#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)

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

**## [1] 13.26984**

#This accuracy is likely explained from the limitations of the polynomial kernel.   
#Parametric model sizes are fixed, and therefore become saturated once too much data is introduced.  
  
  
#radial  
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)

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

**## [1] 89.50794**

#Sigmoid  
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)

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

**## [1] 87.7619**

# The radial kernel setting produced the highest accuracy results here, so we will fine-tune the cost in order to explore the accuracy bounds.   
  
  
  
#previous accuracy = 89.5, new cost = 0.1  
  
  
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, cost = 0.1)   
 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),

**## [1] 72.11111**

#new accuracy = 72.11, new cost = 10  
  
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, cost=10)   
 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),

**## [1] 91.57143**

#new accuracy = 91.57, final cost = 100  
  
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, cost = 100)   
 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),

**## [1] 90.98413**

# The cost variable has shown us some possibilities regarding the accuracy of our SVM model.  
# With a very low cost, our accuracy dropped down to 72% from our original 89.5%. This is   
# likely because the data within our margins is no longer penalized as being "too close" to  
# one another, and therefore produces results that are less than trustworthy. Meanwhile, a cost   
# of 100 produced only 1.5% higher accuracy than our original model. When taking into account   
# our cost of 10 model, this becomes another important lesson in producing the extremes on both sides  
# (cost = 0.1, cost = 100), and using these findings to help find the stable ground in the middle.  
  
  
#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, 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)

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

**## [1] 93.34921**

#tuning the randomForest algorithm, adjusting for best number of trees in hopes that   
#we will have better accuracies.   
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)  
get\_accuracy\_rate(table(all\_results$orig, all\_results$pred), length(all\_results$pred))

**## [1] 93.44444**

# Summary of all results:  
# kNN 7: 91.683  
# kNN 3: 91.936  
# kNN 5: 91.905  
# kNN 8: 91.079  
# SVM non-binary: 12.016  
# SVM binary: 89.510  
# SVM Polynomial: 13.270  
# SVM Radial: 89.508  
# SVM Sigmoid: 87.762  
# SVM Radial cost 0.1: 72.111  
# SVM Radial cost 10: 91.571  
# SVM Radial cost 100: 90.984  
# RandomForest: 93.492  
# RandomForest tuned trees: 93.508  
#   
# Decision Tree (last week): 85.290  
# Naive Bayes (last week): 50.630  
  
#Conclusion  
  
# This dataset has proved its worth once again by showing us the limitations   
# of models such as kMM, SVM, and RandomForest. The massive number of variables and   
# inputs in play does a great job of highlighting any weaknesses these algorithms have,   
# and provides many opportunities to flex the fine-tuning muscles.   
  
# As seen above, the accuracies (when data is fitted correctly) range from as low as   
# 72% to 93.5%. The lowest accuracy was a result of an improper cost value within the   
# SVM model parameters, the cost parameter being how hard we want the model to penalize the values  
# within our margin of difference between two seperate values.   
  
# Our highest accuracy was a product of the RandomForest algorithm, a program designed to   
# handle large data and complex variables. RandomForest essentially builds a model of many  
# different decision trees. It uses the result of running multiple different decision trees  
# and the difference between each as a means of building out a robust and accurate model.   
# Also worth noting, RandomForest uses random selections of variables to create splits, rather  
# than the "best" variable it can find to create splits. This allows for multiple trees of   
# different values to be created, and thus more models to create the final result with.   
# When comparing the results of RandomForest to our decsion tree created last week, the difference  
# of almost 10% in accuracy is worth paying attention to.   
  
# In summary, the above results should paint a better picture as to how to treat large, variable-diverse  
# datasets such as the one handled here. Many different factors must be paid attention to,   
# and poor accuracy results should not be chalked up to the test under-performing. Rather,   
# considerations such as "should we binarize the data?" or "are our parameters appropriate?"  
# need to be taken into account.