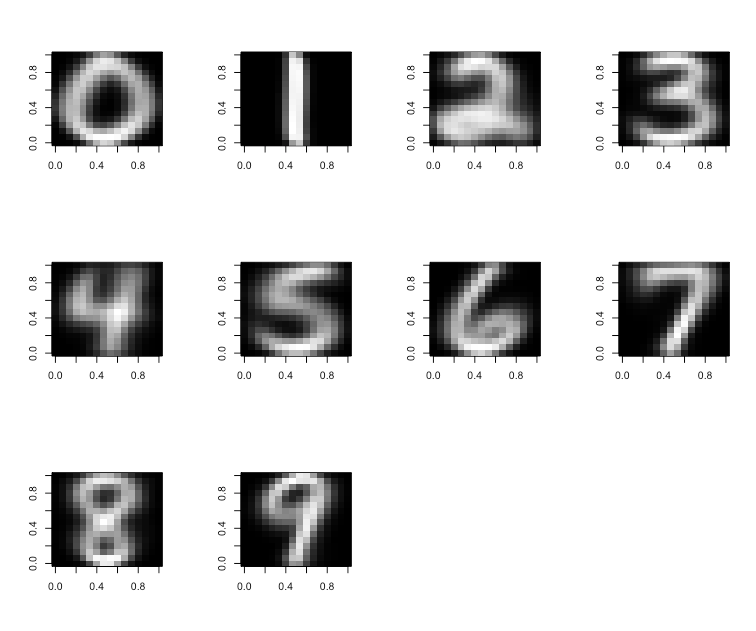
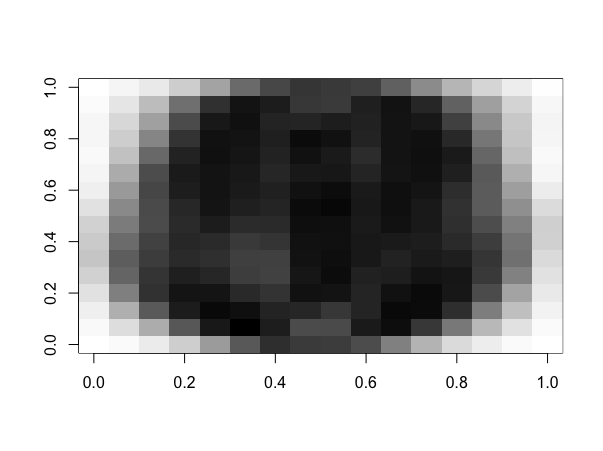
**Exploring the Digits data**

To explore what each digit looks like on average, I generated the following plot of the average pixel values for the training set. All of the digits are readable as the numbers they represent, and the digit for “one” is the most clear.



Next, to explore which pixels were the most and least useful for classification, I generated the variances for each pixel in the 16x16 matrix, and displayed this data below in a heatmap. The lighter areas correspond to less variable pixels, and the darker areas correspond to more variable ones. The outer pixels have less variation in them, because in the images themselves these outer pixels are mostly black. Although they have the least amount of variance, I would say they still aren’t the most useful for classification because they are always the same in every image. The most useful pixels would be those that are clustered towards the middle and have lighter shades than those surrounding them. This indicates that although these pixels are found in an area where many pixels relevant for classification are found (in the center), they have less variability, indicating that they are more likely to be associated with a particular zip code digit. 

**Cross Validation Of KNN model**

For my cross validation function, there were a few methods that I used in order to make my function run faster. Originally, I tried splitting the data frames by assigned group “m” inside my “shuffle” function, (after shuffling and assigning random groups of m), but I ran into difficulties rejoining them, and I realized that the code would run more efficiently just by using subset and indexing. Another way I saved code efficiency was by calculating the distance matrix outside of the for loop, inside of the predict\_knn function. I had the distance matrix as a parameter inside the predict\_knn function, that would default to calculating the distance matrix once if it wasn’t provided, based off of “predict” and “train”. Both of these ideas for code efficiency came from Nick’s comments on Piazza.

For this assignment, I was unable to completely finish debugging my two functions, predict\_knn and cv\_error\_knn before the deadline. I did spend many hours trying to optimize these functions, but was not able to complete them because I did not start early enough. I do believe I was very close to finishing my cross validation function: I was running into an error when I expanded it to the whole data frame, in that when I tried to append the randomized group labels for splitting (via subsetting) into predict and train data inside the for loop. I am sure there is an easy fix for this, but it is the error I could not solve as I hit the deadline. I did manage to complete my predict\_knn function, and it was returning predicted value labels, though since I did not manage to apply my cv\_error\_knn to the entire dataset, I cannot tell how successful my algorithm was. When I did apply my cv\_error\_knn function to a small subset of the data, it generated 75-100% error most of the time, indicating there were problematic issues with how I set up my predict\_knn function. One issue is that I did not include a case to deal with ties. Another is that I decided the winning label by counting the one with the highest votes, rather than the inverse distance model which would have been a more accurate estimation of the label based on nearest neighbor. I used the which.max function to return the first case with the highest votes, which introduced bias towards whichever label was alphabetized first.

I have annotated my code to the best of my ability, hoping that you will give me partial credit for the amount of work I did manage to complete, and that you will see that I did provide credible effort into this assignment. I also believe I should earn some credit for good coding skills, because I split up both of my functions into a few smaller ones which hits the “code modularity” standard. All of my smaller functions work well, but finishing the main functions predict\_knn and cv\_error\_knn was where I ran into the deadline and did not succeed in completing this assignment.

**Sources:**

I used many of the posts on piazza to help me write my code, below are the few that I saved:

<https://stackoverflow.com/questions/32443250/matrix-to-image-in-r>

<https://piazza.com/class/jmf7qwk0sf03ya?cid=534>

<https://piazza.com/class/jmf7qwk0sf03ya?cid=583>

<https://piazza.com/class/jmf7qwk0sf03ya?cid=577>

<https://piazza.com/class/jmf7qwk0sf03ya?cid=514>

<https://piazza.com/class/jmf7qwk0sf03ya?cid=508>

# Mira Mastoras

# Assignment 6

library(ggplot2)

#Question 1

# read digits function

read\_digits = function(file\_path) {

image\_df = read.table(file\_path, header = F, sep = "")

return (image\_df)

}

test\_table = read\_digits("/Users/miramastoras/sta141a/digits/test.txt")

train\_table = read\_digits("/Users/miramastoras/sta141a/digits/train.txt")

#Question 2

#What does each digit look like on average?

#combined data, but i ended up just using the training set

all\_image\_data = rbind.data.frame(test\_table, train\_table)

all\_image\_data

#rename V1 to be "class"

colnames(train\_table)[colnames(train\_table) == "V1"] = "class"

#split data by class

split\_data = split(train\_table, train\_table$class)

#find means in each column for pixels in each class

avg\_data = lapply(split\_data, colMeans)

# convert to a matrix

avg\_data = lapply(avg\_data, as.matrix)

#subset

test\_image\_zero = avg\_data$`0`[-1,1]

#convert to matrix

test\_zero = as.matrix(test\_image\_zero)

typeof(test\_zero)

# coerce into a 16X16 matrix

first = matrix(test\_zero, 16, 16, byrow=T)

# apply the image function

image(t(apply(first, 2, rev)), col=grey(seq(0,1,length=256)))

# Generalized image function

get\_images = function(DF\_element) { # input is avg\_data$'class'

test\_image = DF\_element[-1,1]

test\_matrix = as.matrix(test\_image)

first = matrix(test\_matrix,16, 16, byrow=T)

image(t(apply(first, 2, rev)), col=grey(seq(0,1,length=256))) # from Stack Overflow

}

par(mfrow = c(3,4))

lapply(avg\_data,get\_images)

# Which pixels seem to be the most useful ones for classification ?

# this is asking which pixel location in the matrix is the most or least variable

# need to get the variances across each of the pixel locations in the 16 x 16 matrix

# make a heat map of the variances

#remove class label

test\_tbl = subset(test\_table, select = -c(V1))

#get the variance

var\_data = lapply(test\_tbl, var)

var\_data= as.numeric(var\_data) # make it numeric before making the matrix - Piazza

var\_data = as.matrix(var\_data)

#coerce to 16x16 matrix

var\_data\_16X16 = matrix(var\_data,16, 16, byrow=T)

#variance heatmap

par(mfrow = c(1,1)) #resetting the grid

image(t(apply(var\_data\_16X16, 2, rev)), col=grey(seq(1,0,length=256)))

class(var\_data\_16X16)

# Question 3: KNN function

# need to calculate distance matrix between K neighbors - distances of all the points between the two

# 16x16 matrices

# this function iterates through and finds labels for the K nearest neighbors

# returns a vector of the labels

find\_k\_labels = function(k, row\_ordered, correct\_dist, combined, m){

x = 1

labels = vector(mode = "list", length = k) # preallocate a vector of labels based on K

while (x <= k){

positions = which(correct\_dist == row\_ordered[x], arr.ind=T) #getting row & col number in distance matrix with the k distance value

label = combined[(positions[2]) + m, 1] # gives column of original distance matrix, and position to draw label from

labels[x] = label

x = x + 1

}

return(labels)

}

# this function takes in vector of labels for K nearest neighbors and decides

# which one wins

decide\_label = function(labels) {

unlisted = unlist(labels)

tbl = table(unlisted)

label = as.numeric(names(which.max(tbl)))

#still need to deal with ties somehow

return(label)

}

predict\_knn = function(predict, train, k, distance = dist(rbind(predict, train))) {

m = nrow(predict)

p = nrow(train)

combined = rbind(predict, train)

distance = as.matrix(distance)

predicted\_labels = vector(mode = "list", length = m) # preallocating room for a vector of the predicted labels for each row

correct\_dist = distance[(1:m),(m + 1):(m + p)] #subsetted distance matrix

for (row in (1:m)) {

row\_ordered = sort(correct\_dist[row,], decreasing = T) # order the distances in the row for that point

labels = find\_k\_labels(k, row\_ordered, correct\_dist, combined, m) # gets a vector of k labels

label = decide\_label(labels) # finds the highest label

predicted\_labels[row] = label # add highest label to correct position in vector

}

return(predicted\_labels) # returns vector of predicted labels for predict data based off of train data

}

# KNN function testing:

train = train\_matrix[1:200,]

predict = test\_matrix[1:200,]

train\_matrix = as.matrix(train\_table)

test\_matrix = as.matrix(test\_table)

train = train\_matrix

predict = test\_matrix

ncol(predict)

predict\_labels = predict\_knn(predict, train, k = 6, metric = "euclidean")

predict\_labels = unlist(predict\_labels)

table(predict\_labels)

debug(predict\_knn)

# Question 4 Cross Validation

# for each fold, (m) calculate and compute an error estimate

# calculate the distance matrix outside of predict\_knn and then pass it into predict\_knn

train = train\_matrix[1:8,]

set.seed(141)

# this function shuffles the data and splits on m groups

shuffle = function (train, m) {

train\_df = as.data.frame(train) # convert to data frame

train\_rows = nrow(train\_df)

shuffled = train\_df[sample(nrow(train\_df)), ] # shuffle the data

groups = rep(c(1:m), (train\_rows / m )) # create vector of group labels

shuffled$Groups = groups # assign groups to data

return(shuffled) # returns a data frame with original train data split into m groups

}

# This function calculates error rate based on output from predict\_knn & actual labels

calc\_error\_rate = function (predicted\_labels, predict) {

length = nrow(predict)

sum\_incorrect = 0

for (row in (1:length)) {

if (predicted\_labels[row] != predict[row,1]) {

sum\_incorrect = sum\_incorrect + 1

}

}

error\_rate = sum\_incorrect / length

return(error\_rate)

}

cv\_error\_knn = function(train, m, k, metric) {

train\_shuffle = shuffle(train, m) # calls the function to shuffle & split data

sum\_error\_rates = 0

for (group in 1:m) {

predict = as.matrix(train\_shuffle[train\_shuffle$Groups == group,])

train = as.matrix(train\_shuffle[-(train\_shuffle$Groups == group),])

distance = dist(rbind(predict, train), method = metric)

predicted\_labels = predict\_knn(predict, train, k , distance )

group\_error\_rate = calc\_error\_rate(predicted\_labels, predict) #get error rate

sum\_error\_rates = sum\_error\_rates + group\_error\_rate # add error rate to sum

}

cv\_error\_rate = sum\_error\_rates / m

}

# Testing my CVV function

train = train\_matrix

predict = test\_matrix

debug(cv\_error\_knn)

cv\_error\_test = cv\_error\_knn(train, m = 5, k = 10)

cv\_error\_test

predict\_labels = predict\_knn(predict, train, k = 2)

train\_test = lapply(train\_split[-1], as.matrix)

train\_test = unsplit(train\_split[-1], train\_split[])

train\_test = train\_split[-1]

train\_split = split\_shuffle(train, m)

train\_test$'2'

debug(shuffle)

shuffledd = shuffle(train, m = 10)

shuffledd

group = 1

typeof(as.matrix((shuffled[shuffled$Groups == group,])))