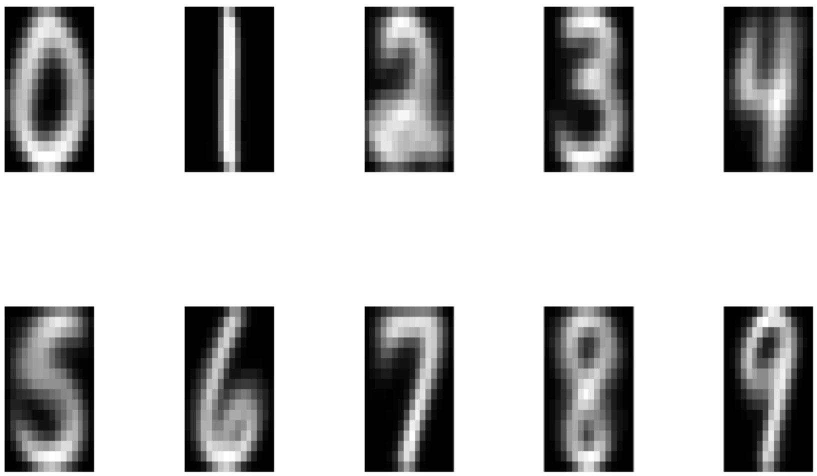
Kevin Wang
STA 141A
#912861670

Final Project

2.

a. Display graphically what each digit (0 through 9) looks like on average.

To get what each digit looks like on average, I subset data according to their label digit and took average of each pixel. Then graphed each digit based on its average pixel.



The graph above shows what each digit looks like on average.

b. Which pixels seem the most likely to be useful for classification?

To see which pixels seem to be the most useful for classification, variance of pixels will be great help. For example, some pixels in the matrix represent black space in the digit, thus, they will be always -1 in the matrix, so the variance is small. Thus, if the variance is larger, it's most useful for classification. Below is the table of three most useful pixel

Largest Variance	Pixel
0.7995005	230
0.7786657	219
0.7761626	105

c. Which pixels seem the least likely to be useful for classification?

If the variance is small, it's least useful for classification. Below is the table of three least most useful pixel.

Least Variance	Pixel
0.002221925	241
0.002674205	1
0.004363451	256

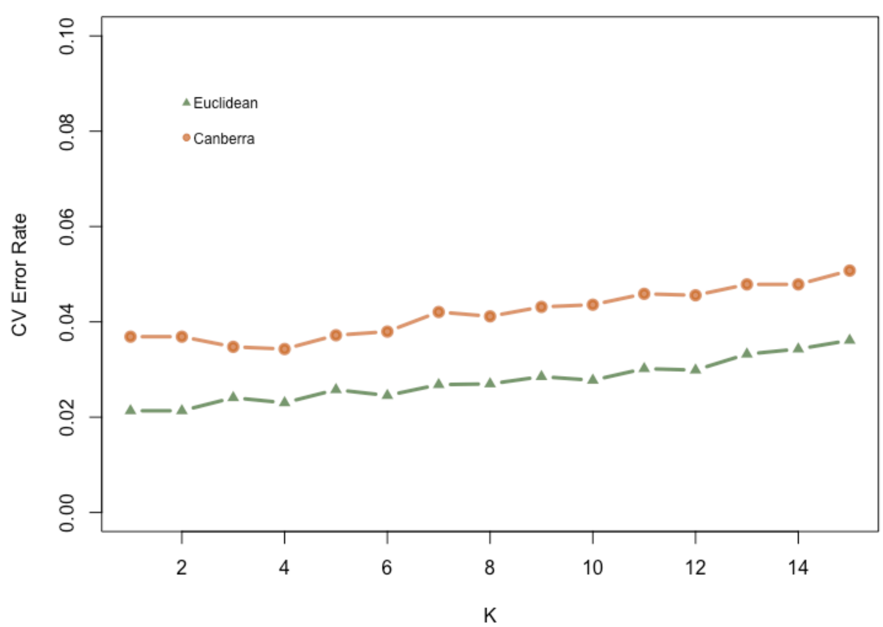
4. Write a function cv_error_knn() that uses 10-fold cross-validation to estimate the error rate for k-nearest neighbors. Briefly discuss the strategies you used to make your function run efficiently.

In question 4, my initial intuition is to use the predict KNN function in question 3.
I first randomized the input dataset by its index, separate them into 10 fold, and wrote a loop to change the predicting points and training position in the dataset. Lastly, I used the function in question 3 to find the predict label and then find the error rate. This function worked, but it took about more than 10 minutes to calculate the error rate for the "train" dataset.

I realized it was not a good function because it took way too long to calculate the error rate. Instead of using the KNN function, I move the distance calculation outside of the loop to avoid the excessive running time. In the loop, I simply rearranged the distance to different training and predicting points. With this improvement, the running time decreased from 10 minutes to 70 seconds.

5. In one plot, display 10-fold CV error rates for all combinations of k = 1…15 and two different distance metrics. Which combination of k and distance metric works best for this data set? Would it be useful to consider additional values of k?

In this problem, I display the 10-fold CV error rates for "Euclidean" distance metric and "Canberra" distance metric.

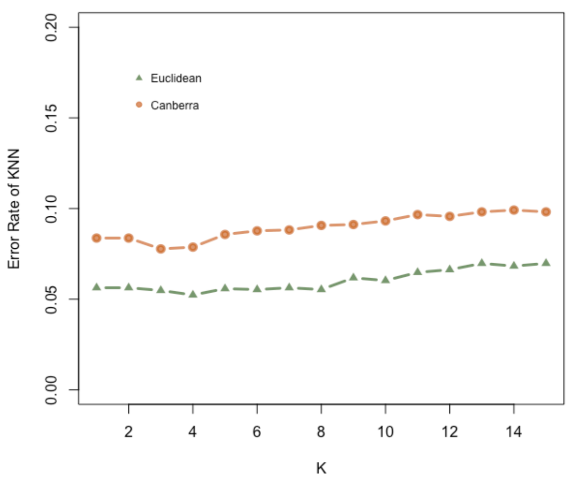


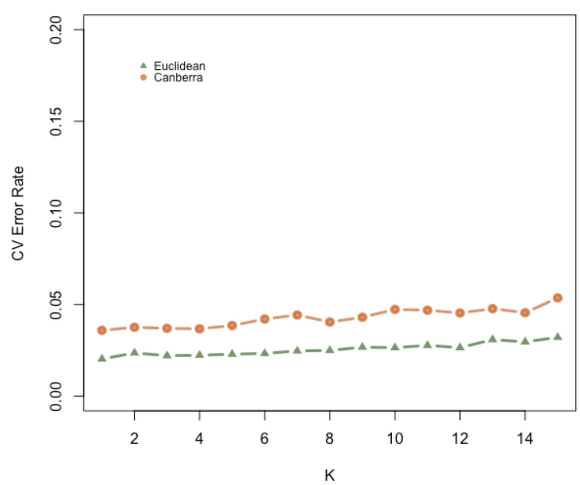
As we can see from the plot, the line of error rate of "Euclidean" distance metric is below that of "Canberra" distance metric. Thus, the Euclidean distance metric can help us get a lower error rate then "Canberra" distance metric.

When we take a closer look at the error rate, we can see that as **the k increases, the error rate tends to increase** as well. The best combination of k and distance metric is when k = 1, 2 ,4 and distance metric is Euclidean, which will give the three lowest error rate 0.021338212, 0.021338214 and 0.02301478. This result is logical because as k increases, there will be more **bias** which lead to higher error rate. **It would not be useful to consider additional value of k** because as k increases, the error rates tend to increase, it would not be meaningful to find additional biased error rates.

6. In one plot, display the test set error rates for all combinations of k = 1…15 and two different distance metrics. How does the result compare to the 10-fold CV error rates? What kinds of digits does the "best" model tend to get wrong?

In this problem, I display both KNN and the 10-fold CV error rates for "Euclidean" distance metric and "Canberra" distance metric.





As we can see from the graphs on the left, error rate of Canberra distance is also higher than Euclidean in KNN error rate. We can also observe an increasing pattern of error rate as k increases. The best combination is when k = 4, 3 ,6 and distance metric is Euclidean, which will give the three lowest error rate 0.052316893, 0.054808176 and 0.05530643.

Comparing KNN to Cross Validation, from the graphs above we can see Cross Validation can have lower error rate than KNN. This makes sense because KNN method only have training on one train data set and predict on one test data set; however, Cross Validation have 10 training set and 10 test set and take an average of all the error rate. In other words, the error rate of Cross Validation is lower because the error and bias got reduced by repeated training and testing.

Digit	Error
8	0.10843
5	0.09375
4	0.09

The highest error is for digit 8 with 0.10843373 error rate, digit 5 with 0.09375 error rate and digit 4 with 0.09 error rate. Thus, the best model tends to get wrong on digit 8, 5 and 4.

Citation

Q2:
https://stackoverflow.com/questions/23050928/error-in-plot-new-figure-margins-too-large-scatter-plot #**Fix the error: figure margins too large**
https://stackoverflow.com/questions/5638462/r-image-of-a-pixel-matrix
#**Change color to grey**
Q3:
https://www.tutorialspoint.com/r/r_mean_median_mode.htm #**Function mode**
Q5/Q6: https://www.r-graph-gallery.com/119-add-a-legend-to-a-plot/
#Plot and legend

R Appendix:

```
#HW6
setwd("~/Desktop/STA141A")
#1. Write a function read_digits() that reads a digits file into R. Your function must
#allow users to specify the path to the file (training or test) that they want to read. Your
#function must return a data frame with columns that have appropriate data types. No
#written answer is necessary for this question.

read_digits = function(file) {
  post = read.table(file, header = F, sep = "")
  return(post)
}

test = read_digits("digits/test.txt")
train = read_digits("digits/train.txt")
class(test)


#2. Explore the digits data:
#• Display graphically what each digit (0 through 9) looks like on average.
#• Which pixels seem the most likely to be useful for classification?
#• Which pixels seem the least likely to be useful for classification? Why?

dim(test)
dim(train)

par("mar")
par(mar=c(1,1,1,1))
#https://stackoverflow.com/
#questions/23050928/error-in-plot-new-figure-margins-too-large-scatter-plot
#Fix the error: figure margins too large

sample = test[1,c(2:257)]
# first row column 2 to column 257
sample = matrix(as.numeric(sample),ncol = 16, byrow=T)#change to 16*16
image(t(sample[16:1,]),col = grey(seq(0, 1, length = 256)),axes=F)# use t to ratate
#https://stackoverflow.com/questions/5638462/r-image-of-a-pixel-matrix
#change color to grey

# it worked now make a function that read number of row and text
digit_image = function(n,text){
  sample = text[n,c(2:257)]
  sample = matrix(as.numeric(sample),ncol = 16, byrow=T)
  image(t(sample[16:1,]),col = grey(seq(0, 1, length = 256)),axes=F)
}
```

```
digit_image(1,test)

# write a function that read a singe row a matrix into digit image
digit_picture = function(text){
  sample = text[c(2:257)]
  sample = matrix(as.numeric(sample),ncol = 16, byrow=T)
  image(t(sample[16:1,]),col = grey(seq(0, 1, length = 256)),axes=F)
}

#test on number one
one = subset(test, test$V1== 1)
one = as.matrix(one)
one = colSums(one)/nrow(one)

#test on number two
two = subset(test, test$V1== 3)
two = as.matrix(two)
two = colSums(two)/nrow(two)

digit_picture(two)

# it worked, applied it to all the number
par(mfrow=c(2,5))

for (i in 0:9){
  num = subset(test, test$V1== i)
  num = as.matrix(num)
  num = colSums(num)/nrow(num)
  digit_picture(num)
}


# Find variance of each pixel
variance = matrix(NA,1,256)

for (i in 2:257){
  variance[i] = var(train[,i])
}

variance = as.numeric(variance)
View(variance)
# smallest is 0.00178 #second least likely to be useful
# greatest is 0.81984 #230th, most useful
```

```
#3. Write a function predict_knn() that uses k-nearest neighbors to predict the label for a
#point or collection of points. At a minimum, your function must have parameters for the
#prediction point(s), the training points, a distance metric, and k. Use the training set
#to check that your function works correctly, but do not predict for the test set yet. No
#written answer is necessary for this question.

total =rbind(train,test)
total = total[,c(2:257)]

dis = dist(total,method = "euclidean",diag = FALSE, upper = FALSE)
dis = as.matrix(dis)
numtrain =nrow(train)
numtrain
numtest = nrow(test)
numtest
ncol(test)

dis[7292,1863]#test
mat = dis[7292,c(1:7291)] #get the dis at first in test.
mat = order(mat)#get dis from low to high
View(mat)
label = train[mat[c(1:77)],1]
View(label)
index = mat[c(1:2)]# for example k =2
label = train[index,1]

# Create the function mode#
#https://www.tutorialspoint.com/r/r_mean_median_mode.htm
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}
label = getmode(label)#Get the label!!


#Now time to write a function
predict_knn = function(predict,train,distant,k){
  total =rbind(train,predict)
  total = total[,c(2:ncol(train))]
  dis = dist(total, method = distant,diag = FALSE, upper = FALSE)
  dis = as.matrix(dis)
  numtrain =nrow(train)
  numpredict = nrow(predict)
  pred=c()

  for (i in 1:numpredict){
```

```
    mat = dis[numtrain+i,c(1:numtrain)]
    mat = order(mat)
    index = mat[c(1:k)]
    label = train[index,1]
    pred[i] = getmode(label)
  }
  return(pred)
}

#test
train1 = train[1:1000,]

train2 = train[1001:2000,]

predict_knn(train1,train2,"euclidean",3)


#4. Write a function cv_error_knn() that uses 10-fold cross-validation to estimate the error
#rate for k-nearest neighbors. Briefly discuss the strategies you used to make your function
#run efficiently.

#Initial Version
cv_error_knn = function(data,d,k){
  total_length = nrow(data)
  length_of_test = total_length/10
  sindex = sample(total_length,replace = FALSE)# mixed index
  sample_mixed = data[sindex,]

  total_error = 0
  for (i in 0:9){
    test = sample_mixed[(i*length_of_test+1:length_of_test*(1+i)),]
    train = sample_mixed[-(i*length_of_test+1:length_of_test*(1+i)),]
    predicts = predict_knn(test,train,d,k)
    real = test[,1]

    error = real - predicts
    error2 = subset(error,error!=0)
    error_rate2 = length(error2)/length(error)
    total_error = total_error+error_rate2
  }
  aver_error = (total_error)/10
  return(aver_error)
}


##Final version!
```

```
cv_error_knn = function(data,distance,k){
  total_length = nrow(data)
  length_of_test = as.integer(total_length/10)
  sindex = sample(total_length,replace = FALSE)# mixed index
  sample_mixed = data[sindex,]

  dis = dist(sample_mixed[,c(2:257)], method = distance)
  dis = as.matrix(dis)
  total_error = 0
  for (i in 0:9){
    test = sample_mixed[sindex[(i*length_of_test+1:length_of_test*(1+i))],]
    train = sample_mixed[sindex[-(i*length_of_test+1:length_of_test*(1+i))],]
    numtrain =nrow(train)
    numpredict = nrow(test)
    total = rbind(train,test)
    l=c()
    for (j in 1:numpredict){
      mat = dis[sindex[(i*length_of_test+1:length_of_test*(1+i))][j],sindex[-
(i*length_of_test+1:length_of_test*(1+i))]]
      mat = order(mat)
      index = mat[c(1:k)]
      label = train[index,1]
      l[j] = getmode(label)
    }
    real = test[,1]
    error = real-l
    error2 = subset(error,error!=0)
    error_rate = length(error2)/length(error)
    total_error = total_error+ error_rate
  }
  aver_error = (total_error)/10
  return(aver_error)
}


test_only = train[1:1000,]

cv_error_knn(test_only,"euclidean",15)



#5. In one plot, display 10-fold CV error rates for all combinations of k = 1...15 and two
#different distance metrics. Which combination of k and distance metric works best for
#this data set? Would it be useful to consider additional values of k?
```

```
cv_error_knn_change = function(data,distance,k){
  total_length = nrow(data)
  length_of_test = as.integer(total_length/10)
  sindex = sample(total_length,replace = FALSE)# mixed index
  sample_mixed = data[sindex,]

  dis = dist(sample_mixed[,c(2:257)], method = distance)
  dis = as.matrix(dis)
  total_error = c()
  x = c()
  for(h in 1:15){
    for (i in 0:9){
      test = sample_mixed[sindex[(i*length_of_test+1:length_of_test*(1+i))],]
      train = sample_mixed[sindex[-(i*length_of_test+1:length_of_test*(1+i))],]
      numtrain =nrow(train)
      numpredict = nrow(test)
      total = rbind(train,test)
      l=c()
      for (j in 1:numpredict){
        mat = dis[sindex[(i*length_of_test+1:length_of_test*(1+i))][j],sindex[-
(i*length_of_test+1:length_of_test*(1+i))]]
        mat = order(mat)
        index = mat[c(1:h)]
        label = train[index,1]
        l[j] = getmode(label)
      }
      real = test[,1]
      error = real-l
      error2 = subset(error,error!=0)
      error_rate = length(error2)/length(real)
      total_error[i] = error_rate

    }
    aver_error = mean(total_error)
    x[h] = aver_error
  }
  x
}

par(mfrow=c(1,1))

test_only = train[1:2000,]

eu1 = cv_error_knn_change(train,"euclidean",15)
eu1= as.data.frame(eu1)
```

```
can1= cv_error_knn_change(train,"canberra",15)
can1= as.data.frame(can1)

plot(eu1$eu1,type="b",col=rgb(0.2,0.4,0.1,0.7),lwd=3 , pch=17,xlab = "K",ylab = "CV Error
Rate",ylim=c(0,0.1))
lines(can1$can1,col=rgb(0.8,0.4,0.1,0.7),lwd=3 , pch=19 , type ="b")
legend("topleft",
       legend = c("Euclidean", "Canberra"),col = c(rgb(0.2,0.4,0.1,0.7),rgb(0.8,0.4,0.1,0.7)),
       pch = c(17,19),bty = "n",pt.cex = 0.75,cex = 0.75,text.col = "black",horiz = F ,inset = c(0.1,
0.1))


#https://www.r-graph-gallery.com/119-add-a-legend-to-a-plot/
#Plot and legend

#6. In one plot, display the test set error rates for all combinations of k = 1... 15 and two
#different distance metrics. How does the result compare to the 10-fold CV error rates?
#What kinds of digits does the “best” model tend to get wrong?

predict_knn_change = function(predict,train,distant,k){
  total =rbind(train,predict)
  total = total[,c(2:ncol(train))]
  dis = dist(total, method = distant,diag = FALSE, upper = FALSE)
  dis = as.matrix(dis)
  numtrain =nrow(train)
  numpredict = nrow(predict)
  p=c()
  x=c()
  for(h in 1:15){
    for (i in 1:numpredict){
      mat = dis[numtrain+i,c(1:numtrain)]
      mat = order(mat)
      index = mat[c(1:h)]
      label = train[index,1]
      p[i] = getmode(label)
    }

    real = predict[,1]
    error = real - p
    error2 = subset(error,error!=0)
    error_rate = length(error2)/length(error)
    x[h] = error_rate
  }
  x
}
```

```
train1 = train[1:500,]
train2 = train[501:1000,]
predict_knn_change(train1,train2,"euclidean",15)


eu2 = predict_knn_change(test,train,"euclidean",15)
eu2= as.data.frame(eu2)


can2= predict_knn_change(test,train,"canberra",15)
can2= as.data.frame(can2)

plot(eu2$eu2,type="b",col=rgb(0.2,0.4,0.1,0.7),lwd=3 , pch=17,xlab = "K",ylab = "Error Rate of KNN",ylim=c(0,0.1))
lines(can2$can2,col=rgb(0.8,0.4,0.1,0.7),lwd=3 , pch=19 , type ="b")
legend("bottomleft",
       legend = c("Euclidean", "Canberra"),col = c(rgb(0.2,0.4,0.1,0.7),rgb(0.8,0.4,0.1,0.7)),
       pch = c(17,19),bty = "n",pt.cex = 0.75,cex = 0.75,text.col = "black",horiz = F ,inset = c(0.1, 0.1))

#https://www.r-graph-gallery.com/119-add-a-legend-to-a-plot/
#Plot and legend

#Best combination is k = 1, eu

check = subset(test,test$V1 ==2)
predict_knn_change(check,train,"euclidean",1)


all =c()
for (i in 0:9){
  check = subset(test,test$V1 ==i)
  all[i] = predict_knn_change(check,train,"euclidean",1)
}
View(all)

# The highest error is for digit 8: 0.10843373,  5: 0.09375, 4: 0.09
```