STA141AFinalProj

Sam Tsoi, Brody Lowry

11/28/2017

1.

. Write a function read_digits() that loads a digits file into R. Your function should convert columns to appropriate data types and allow users to specify which file they want to load. No interpretation is necessary for this question.

Answer is attached in the code.

2.

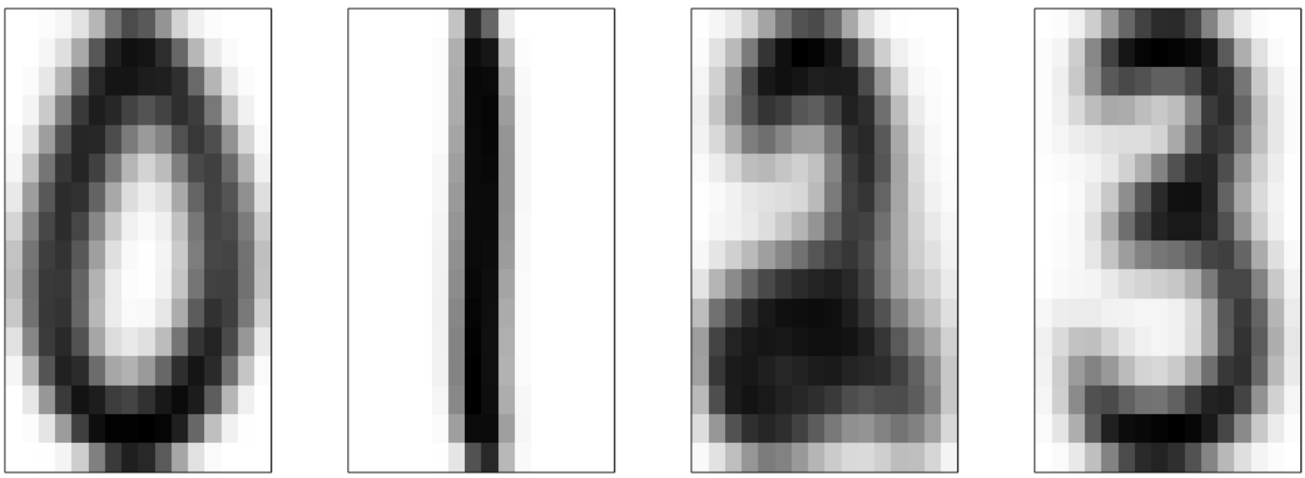
Write a function view_digit() that displays one observation (one digit) from the data set as a grayscale image. Your function should allow users to specify which observation they want to display. No interpretation is necessary for this question.

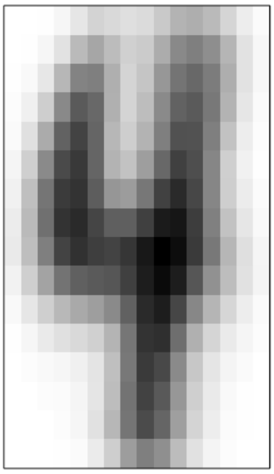
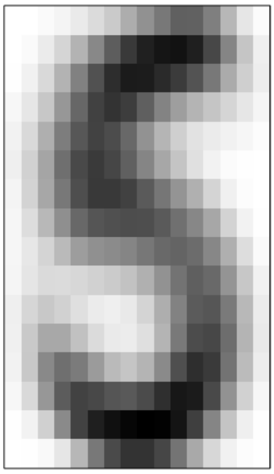
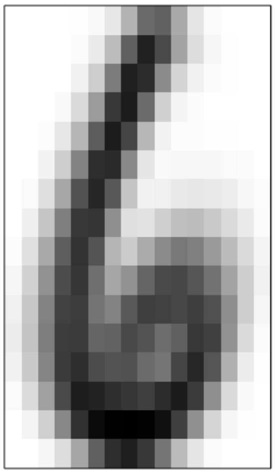
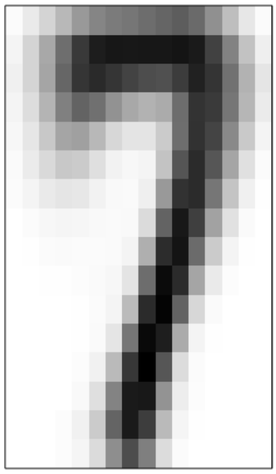
Answer is attached in the code.

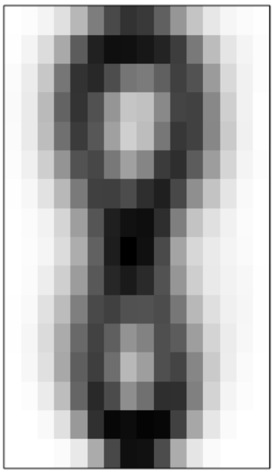
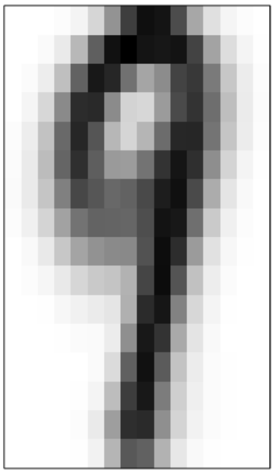
3.

Explore the digits data. In addition to your own explorations: (a) Display graphically what each digit (0 through 9) looks like on average.

Figure 1: Each digit (0 to 9) on average

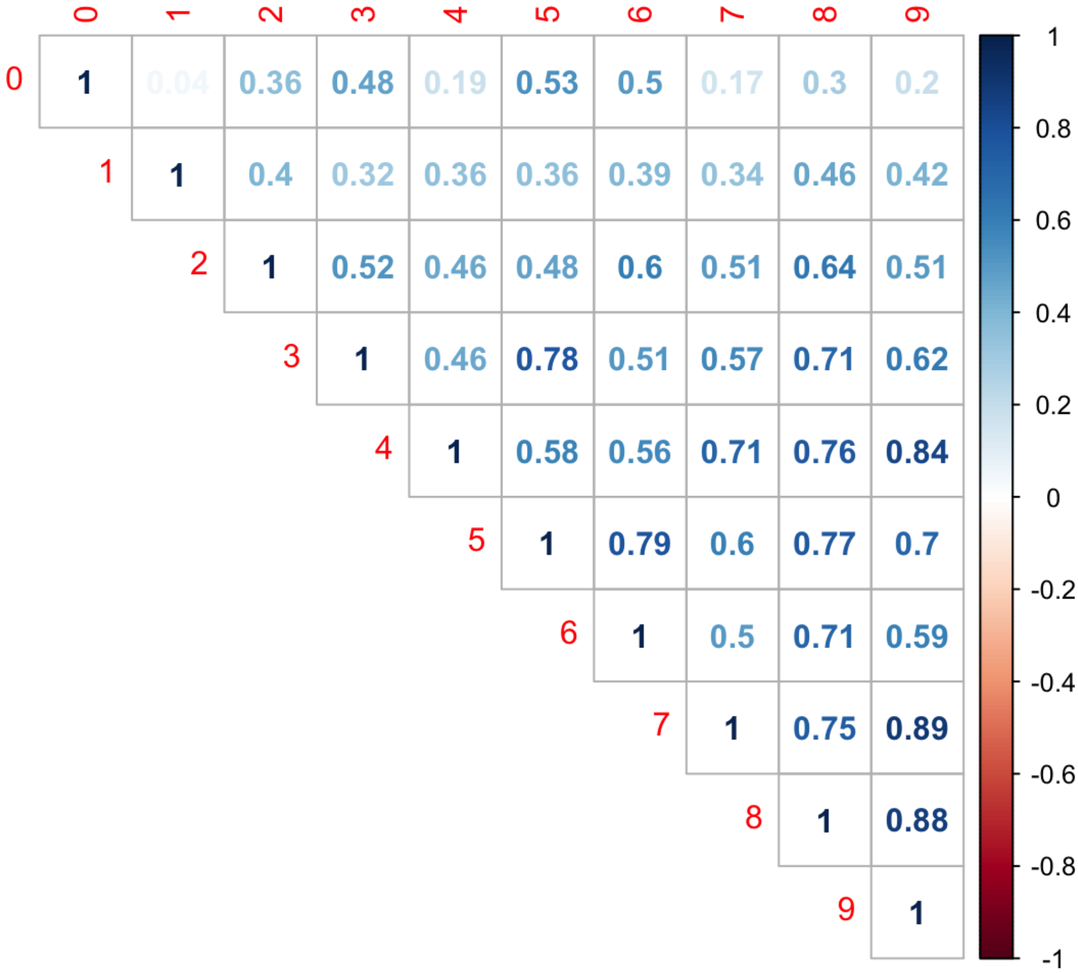


b. Which pixels seem the most likely to be useful for classification? Which pixels seem the least likely to be useful for classification? Why? Some of these will be presented in lecture/discussion.

Figure 2: Correlation plot of each digit compared to another digit



Positive correlations are displayed in blue and negative correlations in red color. Color intensity of the blue is proportional to the correlation coefficients, so the darker the more correlated the numbers are. By looking at our correlation matrix plot we can see that certain numbers are very similar to other numbers. For instance, 8 and 9 are very similar in appearance because of the loop that is on top, therefore their correlation is high but if you look at 1 and 3, their correlation is very low since 1 is a straight line, and 3 has two big curves and no straight lines. 0 has nearly zero correlation with 1 since zero is a circle and 1 is a line, which can be seen as opposites.

Figure 3: Pixels with the highest variance

```
##      V122      V186      V106      V220      V231
## 0.7698937 0.7760059 0.7761626 0.7786657 0.7995005
```

Figure 4: Pixels with the lowest variance

```
##         V242          V2        V257         V17         V18
## 0.002221925 0.002674205 0.004363451 0.005665296 0.010526655
```

Figure 3 and Figure 4 gives insight on which pixels are good for classification and which are not. By looking at the variance for each pixel, we can see if the observations change a lot or not. If the pixel has a low variance then it possibly signifies that the pixel doesn’t change much between observations and classes and would not be a good indicator as to what class it belongs to. A high variance would mean that the pixel changes a lot between observations and classes so therefore these higher variance pixels might be more useful in determining what classification the digit should belong to. So, Figure 3 illustrates the possible pixels that might be good for classification.

4.

Write a function predict_knn() that uses k-nearest neighbors to predict the label for a point or collection of points. At the least, your function should take the prediction point(s), the training points, a distance metric, and k as input. No interpretation is necessary for this question.

Answer is attached in the code.

5.

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.

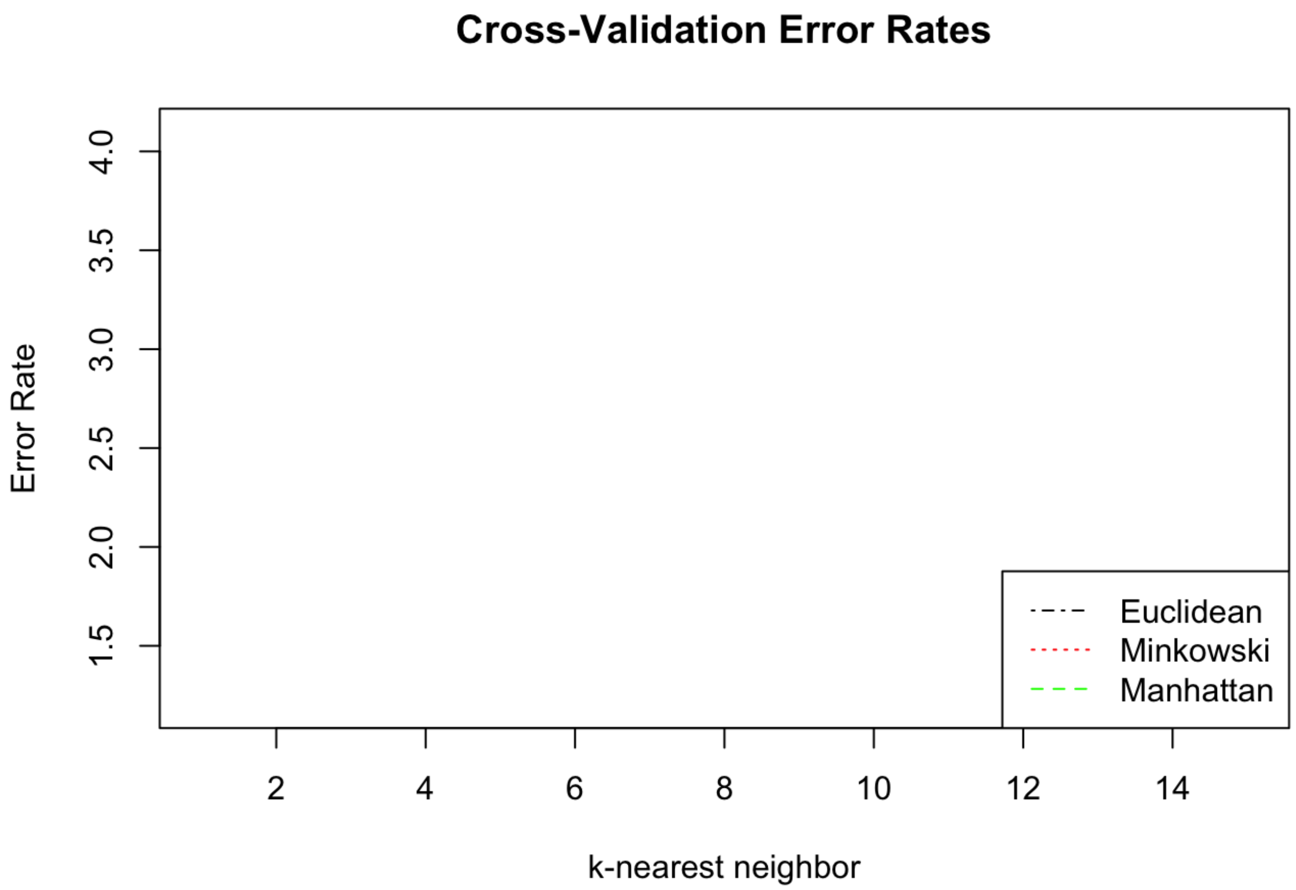
To make our cv_ error_knn() run efficiently, we had a single cv function that creates the folds for the full _cv function so that it doesn't have to do it again and again. Rather than doing the order 1 by 1, we order the matrix and cache it, we utilize caching methods and saving things so it doesn't have to do calculations multiple times.

Answer is attached in the code.

6.

Display 10-fold CV error rates for k = 1, . . . , 15 and at least 2 different distance metrics in one plot. Discuss your results. Which combination of k and distance metric is the best? Would it be useful to consider additional values of k?

Figure 5: 10-fold CV error rates for k = 1, . . . , 15



Based on Figure 5, Manhattan seems to have a very high error rate, compared with those of Minkowski and

Based on the figure, Manhattan seems to have a very high error rate, compared with those of Minkowski and Euclidean. As expected, the error rate increases as k increases. From the figure, it seems like k = 1 and k = 3 have the lowest error rates and the best accuracies for any of the distance metrics. Comparing the individual values for these two k's with these two distance matrices, the error rate for Minkowski when k = 3 is 1.828350, and the error rate for Euclidean when k = 3 is 1.839105. Since error rates seem to increase as k increases after k = 5, we think that it would be safe to say that we do not need to consider other K's based on this trend. It makes sense that error rate would be so high with so many nearest neighbors, and because you are not pooling the highest frequency and making a prediction from the most nearest neighbors. We think that it is pretty interesting that the error rate for all metrics shoot up when k=2. We predict that this might be because our algorithm choose the neighbors with the highest frequency, and chooses randomly if there is a tie in highest frequency. So, if we are only looking at 2 neighbors, and if the two neighbors are different numbers, then the frequency of each is one and our algorithm would choose between those two randomly. Additionally, it might be useful to consider other metrics, and examine on how the error rate changes between each metric.

7.

For each of the 3 best k and distance metric combinations, use 10-fold cross-validation to estimate the confusion matrix. Discuss your results. Does this change which combination you would choose as the best?

Euclidean. As expected, the error rate increases as k increases, specifically after k=5 in our case. From the figure, it seems like k = 1 and k = 3 have the lowest error rates and the best accuracies for any of the distance metrics. Comparing the individual values for these two k's with these two distance matrices, the error rate for Minkowski when k = 3 is 1.828350, and the error rate for Euclidean when k = 3 is 1.839105. Since error rates seem to increase as k increases after k = 5, we think that it would be safe to say that we do not need to consider other K's based on this trend. It makes sense that error rate would be so high with so many nearest neighbors, and because you are not pooling the highest frequency and making a prediction from the most nearest neighbors, for example if k=3. We think that it is pretty interesting that the error rate for all metrics shoot up when k=2. We predict that this might be because our algorithm choose the digit with the highest frequency, and chooses randomly amongst the digits that has the highest frequency if there is a tie in highest frequency. So, if we are only looking at 2 neighbors, and if the two neighbors are different numbers, then the frequency of each is one and our algorithm would choose between those two randomly. Additionally, it might be useful to consider other metrics, and examine on how the error rate changes between each metric.

7.

For each of the 3 best k and distance metric combinations, use 10-fold cross-validation to estimate the confusion matrix. Discuss your results. Does this change which combination you would choose as the best?

Figure 6: Confusion matrix for k = 1, using Euclidean

```
##    best1
##        0    1    2    3    4    5    6    7    8    9
##   0 1549    0    2    2    0    0    0    0    0    0
##   1    0 1265    0    0    3    0    1    0    0    0
##   2   12    3  901    7    2    0    1    3    0    0
##   3    2    0    7  804    2    9    0    0    0    0
##   4    1    7    4    2  831    1    3    2    0    1
##   5    1    0    4   11    2  685    7    2    3    1
##   6    1    0    1    0    3    7  822    0    0    0
##   7    0    0    0    0    6    1    0  781    1    3
##   8    0    0    0    3    1    9    3    4  684    4
##   9    0    0    0    0    1    0    0    9    2  809
```

Figure 7: Confusion matrix for k = 3, using Euclidean

```
##    best2
##        0    1    2    3    4    5    6    7    8    9
##   0 1545    0    4    3    0    0    0    1    0    0
##   1    0 1265    0    0    3    0    1    0    0    0
##   2   11    5  896    9    2    0    1    5    0    0
##   3    3    1    4  807    1    7    0    0    1    0
##   4    0    5    4    0  833    1    5    3    0    1
##   5    3    0    4    8    3  689    7    0    0    2
##   6    0    0    0    0    3    3  827    0    1    0
##   7    0    0    0    0    6    1    0  778    1    6
##   8    0    0    0    4    1    7    1    5  686    4
##   9    0    0    0    0    1    0    0   11    3  806
```

Figure 8: Confusion matrix for k = 3, using Minkowski

```
##    best3
##        0    1    2    3    4    5    6    7    8    9
##   0 1545    0    5    2    0    1    0    0    0    0
##   1    0 1266    0    0    3    0    0    0    0    0
##   2   11    6  895    9    1    1    1    5    0    0
##   3    2    1    4  807    2    7    0    0    1    0
##   4    0    5    4    0  833    1    5    3    0    1
##   5    3    0    4   10    3  687    6    0    1    2
##   6    0    0    0    0    3    5  826    0    0    0
##   7    0    0    1    0    6    0    0  778    1    6
##   8    0    0    0    5    0    7    1    6  686    3
##   9    0    0    0    0    1    0    0   12    3  805
```

Figure 6, 7, and 8 are confusion matrix of the 3 best k and distance metric combinations we found, based on the lowest error rate. The best k and distance metric combinations we found were k=1,3, and 5 for all 3 metrics we used. However, the Manhattan metric has the highest error rate compared with the other two metric systems. The error rate for both Minkowski and Euclidean at k=1 are 1.796085, and it is interesting that k =1 has the lowest error rate, as you are using one neighbor to predict your digit. For using 3 nearest neighbors, Minkowski has an error rate of 1.828350 while Euclidean has an error rate of 1.839105. The confusion matrix does not change what combination I would choose as the best because these figures represent the error rates pretty well. For example, predicting a 2 correctly when it is the digit 2 for k = 1 using Euclidean (in Figure 6) is highest compared with those of Figure 7 and Figure 8. These minor details might be why the error rate for both Minkowski and Euclidean at k=1 are 1.796085, just very slightly be lower than the error rates for k = 3. However, this would not change what combination I would choose. This confusion matrix gives a good insight on which digits are best at being guessed correctly and incorrectly. For instance, 7 and 9 seem to be confused more often than other digits.

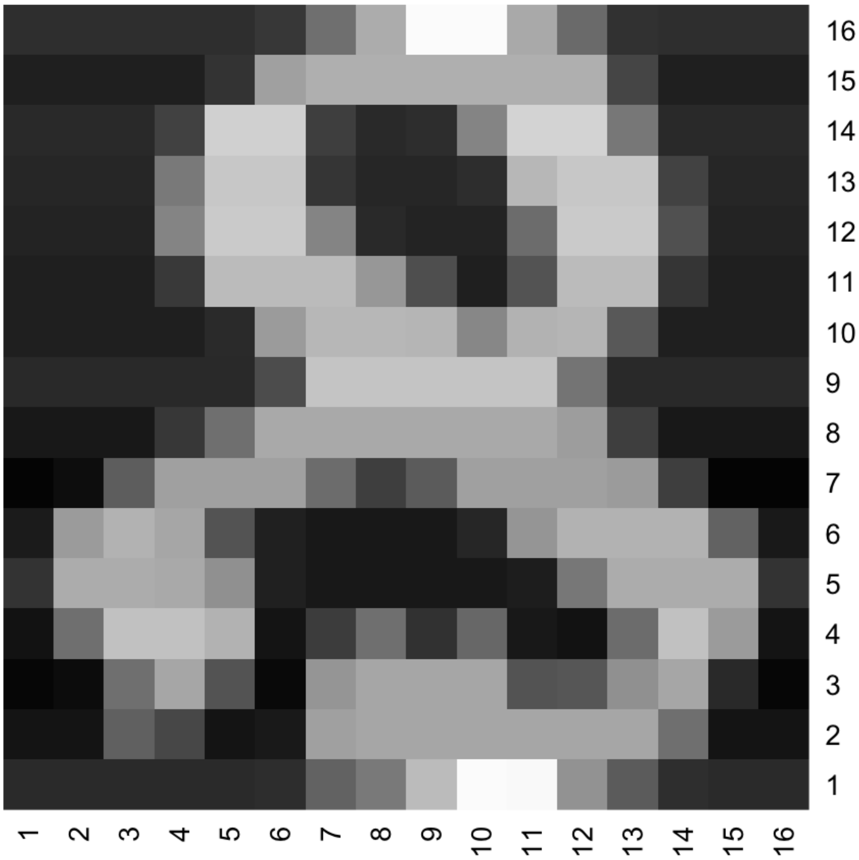
8.

For the best k and distance metric combination, explore the training set digits that were misclassified during cross-validation. Discuss what you can conclude about the classifier.
We used Minkowski at k = 3 for the best k and distance metric combination, as it has an error rate of 1.828350 with k = 3 neighbors. We did not use k = 1 because even though it has the lowest error rate. It only uses one neighbor so this might not be the best if a test digit was really far off. Additionally, k = 3 has a similar error rate.

Figure 9: rows/pixels that were misclassified using Minkowski for k = 3

```
##   [1]   18   28  123  132  135  146  161  199  254  266  280  301  302  340
##  [15]  350  475  483  485  510  517  528  529  550  654  699  794  795  806
##  [29]  811  896  915  971  991  994  995 1007 1008 1039 1047 1094 1105 1148
##  [43] 1214 1226 1358 1376 1395 1431 1432 1517 1545 1630 1653 1734 1814 1815
##  [57] 1816 1865 1872 1893 1952 1965 1978 2234 2284 2333 2390 2394 2438 2498
##  [71] 2504 2572 2771 2804 2917 2989 3059 3251 3495 3496 3534 3578 3750 3871
##  [85] 3891 3966 3998 4090 4096 4128 4175 4280 4347 4348 4350 4351 4367 4386
##  [99] 4473 4629 4671 4678 4821 4863 4883 4898 4961 4976 5007 5046 5047 5088
## [113] 5145 5193 5252 5313 5374 5457 5539 5645 5800 5910 5938 5953 6141 6154
## [127] 6269 6283 6352 6399 6462 6493 6820 6821 6913 6975 7047 7099 7132 7296
## [141] 7304 7435 7440 7546 7567 7687 7713 7761 7812 7837 7858 7884 8007 8008
## [155] 8097 8098 8127 8364 8403 8768 8886 8891 8893 8915 8978 8983 8994 9098
## [169] 9193 9242
```

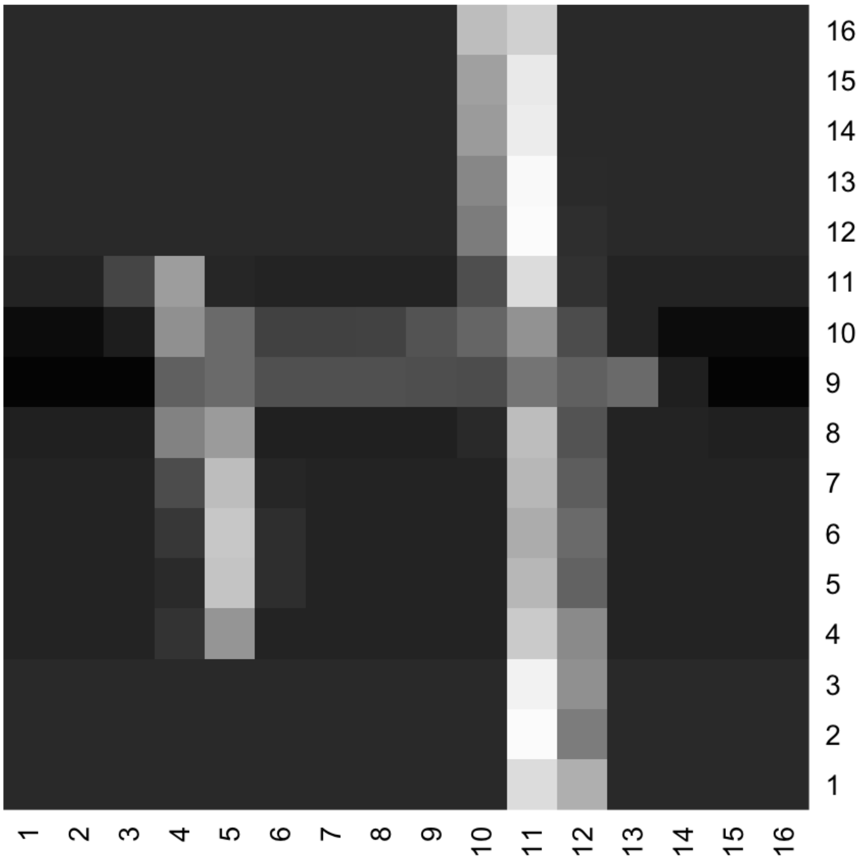
Examining some of the training set digits, such as observation row 18, the image is:



but our KNN classified it as a

```
## [1] "4"
```

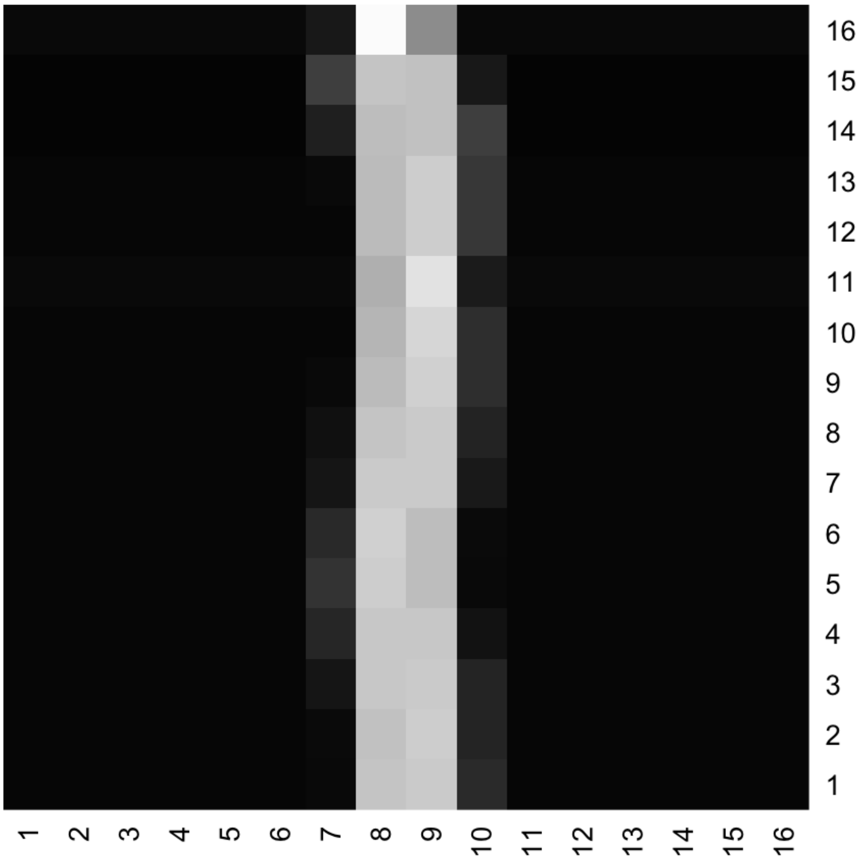
Another training set digit, such as observation row 301, the image is:



but our KNN classified it as a

```
## [1] "7"
```

Another training set digit, such as observation row 5046, the image is:



but our KNN classified it as a

```
## [1] "5"
```

For these training set digits, it seems like our KNN is classifying it wrong as these numbers do not look like 4, 7, and 5, respectively, to the human eye. Our accuracy rate seems pretty high, as the number of rows/pixels that are misclassified is 166 out of the 9,298 pixels we looked at, and this could be because we only have 10 digits to look over. This might mean that the error rate would shoot up if we were classifying other things, like letters for example. We can conclude that our KNN works well when a student doesn't have super terrible handwriting or when the digit is not smeared or damaged to some capacity, because for example, for observation row 18, it seems pretty obvious to the human eye that it is an 8, but our KNN classified the image as a 4. This is a similar case for the other rows we observed. Therefore, it means that one's handwriting cannot be terrible and that the handwriting cannot be damaged or smeared, and our classifier would work well.

9.

Display test set error rates for k = 1, . . . , 15 and at least 2 different distance metrics in one plot. Compare your results to the 10-fold CV error rates.

Figure 10: test set error rates for k = 1, . . . , 15

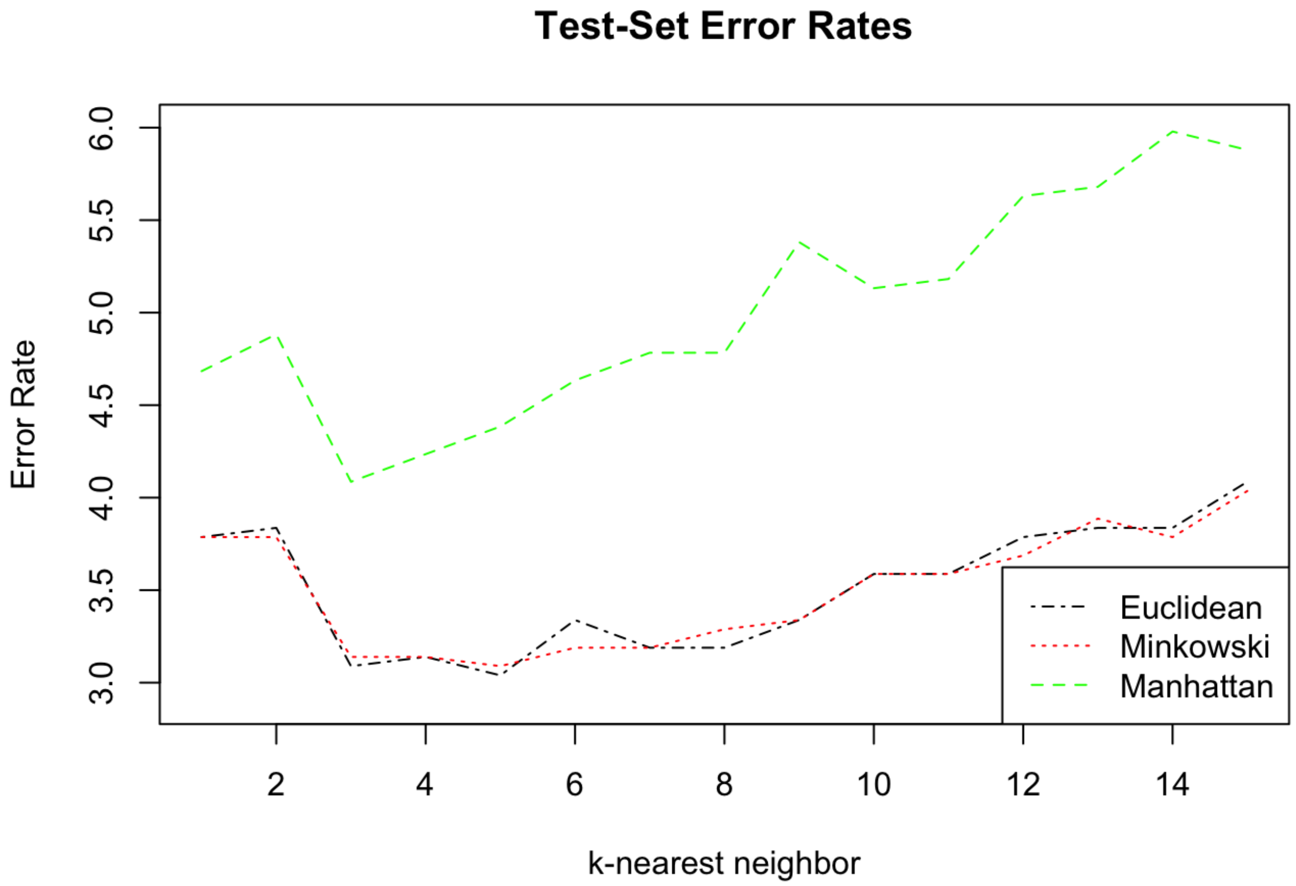


Figure 10 show that the test set error rates are higher than our cv_error rates by almost double, but that isn't too alarming because our cv error rates were very low. Manhattan is shown to still be a terrible distance metric for what we are doing. Looking at the test set error rates we can see that euclidean is shown to be better than Minkowski, but not by much, but enough to say that it is better. Unlike with cv error rates where it was hard to distinguish which was better. 3 and 5 are shown to be the optimal K, which makes sense, since those were promising during our cv error rates. 5 seems like an optimal k since we are getting enough neighbors to help classify our number, but not too many that are harmful or not useful to give noise to our selection. Using the information gained from both our test set and cv error rates we can infer that k = 3 or 5 and the Euclidean distance metric are our best combination in classifying numbers.

10.

Briefly summarize what each group member contributed to the group.

As mentioned in the email sent to the professor and TA Nick, Thomas Fay was unable to contribute due to family circumstances. To start off the project, Sam worked on the code for #1 - #2, and Broderick worked on for #3 - #5. Broderick and Sam worked on the code for #6-#9 together, and also split the work for the write up. /#1, #2, and #4 do not require interpretation. Sam worked on the interpretation for #3, #6 - #8, and Brody worked on interpretation for #5 and #9, and also helped out with #3b. We would say that this project was split pretty evenly in terms of work between the group members.

Citations:

Had to research the description for KNN

Looked on Google for corrplot() information.

Referenced Duncan's for view_digit(): http://eeyore.ucdavis.edu/stat141/Hws/drawCode.R

Referenced Piazza for help before getting started on each problem.

Code Appendix

```
library(corrplot)
###Q1

##convert columns to appropriate data types and allow users to specify which file t
hey want to load.
read_digits <- function(infile)
{
  dataname <- read.table(infile, header=FALSE)
  dataname[] <- lapply(dataname,function(x) as.numeric(x))

  return (dataname)
}

#load test and train data
test <-read_digits("~/Desktop/Downloads/digits/test.txt")
train <- read_digits("~/Desktop/Downloads/digits/train.txt")

### Q2

#displays one observation (one digit) from the data set as a grayscale image
view_digit = function(vals)
{
  m<-matrix(vals,16,16,byrow=TRUE)
  colors = rgb((255:0)/255, (255:0)/255, (255:0)/255)
  m = t(m)   # transpose the image
  m = m[,nrow(m):1]   # turn up-side-down
  image(m, col = colors, xaxt = "n", yaxt = "n")
}

#subsetting, the first column is the digit itself
test2 <- test[,(2:257)]
train2 <- train[,(2:257)]

###Q3a

digit_list<-split(train,train$V1)

#finding the average for each item
avg_digits<-lapply(1:length(digit_list),function(x)apply(digit_list[[x]],2,mean))
avg_digitsdf <- data.frame(avg_digits)
names(avg_digitsdf) <- avg_digitsdf["V1",] #change the names of the column to each
digit
temp <- avg_digitsdf[2:257,] #subset it so the digits are not included

#scaling images to fit to page
par(mfrow=c(1,4), mar=c(10,1,10,2))

#display the average of each digit
for(i in 1:length(digit_list)){
  view_digit(temp[[i]])
}
par(mfrow=c(1,1))
```

```
###Q3b

#cleaning up
digit_correlations<-cor(do.call(cbind,avg_digits))
rownames(digit_correlations)<-0:9
colnames(digit_correlations)<-0:9

#correlation plot
corrplot(digit_correlations,method="number",type="upper")

#finding variance, using tail to find highest variance
pixel_var<-apply(train2, 2,var)
tail(sort(pixel_var), n=5)
#using head to find smallest variance
head(sort(pixel_var), n=5)
### Q4
full<-rbind(test,train)
distmat<-as.matrix(dist(full))

#need to decide which nearest neighbor, because there could be ties for max freq
decide_NN<-function(NN_set){
  NN_table<-table(NN_set) #put into table, has its frequencies of the digits
  max_digits<-which(NN_table==max(NN_table)) #finds the digit that appears most in
the table. if there's a tie, which will get both the elements
  best_digits<-names(NN_table)[max_digits] #need to be done for ties
  knn_pred<-sample(best_digits,1) #if it's a tie, then randomly choose one
  return(knn_pred)
}

#uses k-nearest neighbors to predict the label for a point or collection of points.
predict_knn<-function(train,test,distance,k){
  ordmat<-apply(distance,1,order) #find the distance and order it by indices
  k_NN<-matrix(train[unlist(ordmat[1:k,]),1],nrow(test),k,byrow=TRUE) #put the mat
rix into vector and then back to the vector again
  knn_preds<-apply(k_NN,1,decide_NN)
  return(knn_preds)
}
### Q5
### to figure out the error

#creating fold, so it doesn't have do it numerous times
single_cv<-function(folds,i,k,distance_mat){
  distance<-distance_mat[folds==i,folds!=i]
  test2<-full[folds==i,]
  train2<-full[folds!=i,]
  knn_pred<-predict_knn(train2,test2,distance,k)
  return(knn_pred)
}


#using single_cv
full_cv<-function(n,k,distance_mat){
  folds<-sort(rep_len(1:n,nrow(full)))
  full_predictions<-unlist(sapply(1:n,function(i)single_cv(folds,i,k,distance_mat)
))
  return(full_predictions)
}
```

```
# uses 10-fold cross-validation to estimate the error rate for k-nearest neighbors
cv_error_knn<-function(folds,k,distance_mat){
  print(paste0("k=",k))
  temp_cv<-full_cv(folds,k,distance_mat)
  error<-(1-sum(diag(table(full$V1,temp_cv)))/nrow(full))*100
  return(error)
}


### Q6

#using different distance metric
distmat2 <- as.matrix(dist(full, method = "minkowski"))
distmat3 <- as.matrix(dist(full, method = "manhattan"))
dist_mats<-list(euclidean=distmat,minkowski=distmat2,manhattan=distmat3)

#finding 10-fold CV error rates for 1 to 15
ten_fold_rates<-mapply(function(i,j)cv_error_knn(10,i,dist_mats[[j]]),rep(1:15),re
p(names(dist_mats),each=15))

#subsetting, cleaning up data
euclid<-ten_fold_rates[1:15]
minkow<-ten_fold_rates[16:30]
manhat<-ten_fold_rates[31:45]
fold_rates<-data.frame(euclid,minkow,manhat)

#plotting 10-fold CV error rates for k = 1, . . . , 15
plot(fold_rates$euclid,col="black",type="l",lty=4,ylim=c(1.2,4.1),xlab="k-nearest n
eighbor",ylab= "Error Rate",main="Cross-Validation Error Rates")
lines(fold_rates$minkow,col="red",lty=3)
lines(fold_rates$manhat,col="green",lty=2)
legend("bottomright",legend=c("Euclidean","Minkowski","Manhattan"),col=c("black","
red","green"),lty=c(4,3,2))

###7
best1<-full_cv(10,1,dist_mats$euclidean)
best2<-full_cv(10,3,dist_mats$euclidean)
best3<-full_cv(10,3,dist_mats$minkowski)
table(full$V1,best1)
table(full$V1,best2)
table(full$V1,best3)
###8
which(full$V1!=best3)
numOfBad <- sum(which(full$V1!=best3))

view_digit2 <- function(rowToConvert)
{
  mat <- matrix(as.numeric(rowToConvert), ncol = sqrt(length(rowToConvert)), byrow
= TRUE)
  heatmap(mat,Rowv=NA,Colv=NA,col=paste("gray",1:99,sep=""))
}
view_digit2(train2[18,])
mis18 <- best3[18]
mis18
view_digit2(train2[301,])
mis280 <- best3[301]
```

```
mis280
view_digit2(train2[5046,])
mis5046 <- best3[5046]
mis5046

###9
test_error<-function(preds){
  conf_mat<-table(test$V1,preds)
  error_rate<-(1-sum(diag(conf_mat))/nrow(test))*100
  return(error_rate)
}
predict_knn2<-function(train,test,distance,k){
  print(paste0("k=",k))
  distance<-distance[1:nrow(test),-c(1:nrow(test))]
  ordmat<-apply(distance,1,order) #find the distance and order it by indices
  k_NN<-matrix(train[unlist(ordmat[1:k,]),1],nrow(test),k,byrow=TRUE) #put the mat
rix into vector and then back to the vector again
  knn_preds<-apply(k_NN,1,decide_NN)
  return(knn_preds)
}

test_set_preds<-mapply(function(i,j)predict_knn2(train,test,dist_mats[[j]],i),rep(
1:15,3),rep(names(dist_mats),each=15))

test_error_rates<-apply(test_set_preds,2,test_error)

test_error_df<-data.frame(method=rep(names(dist_mats),each=15),k=rep(1:15,3),error
=test_error_rates)

ts_euclid<-test_error_rates[1:15]
ts_minkow<-test_error_rates[16:30]
ts_manhat<-test_error_rates[31:45]
ts_error_rates<-data.frame(ts_euclid,ts_minkow,ts_manhat)
plot(ts_error_rates$ts_euclid,col="black",type="l",lty=4,ylim=c(2.9,6),xlab="k-near
est neighbor",ylab= "Error Rate",main="Test-Set Error Rates")
lines(ts_error_rates$ts_minkow,col="red",lty=3)
lines(ts_error_rates$ts_manhat,col="green",lty=2)
legend("bottomright",legend=c("Euclidean","Minkowski","Manhattan"),col=c("black","
red","green"),lty=c(4,3,2))
```