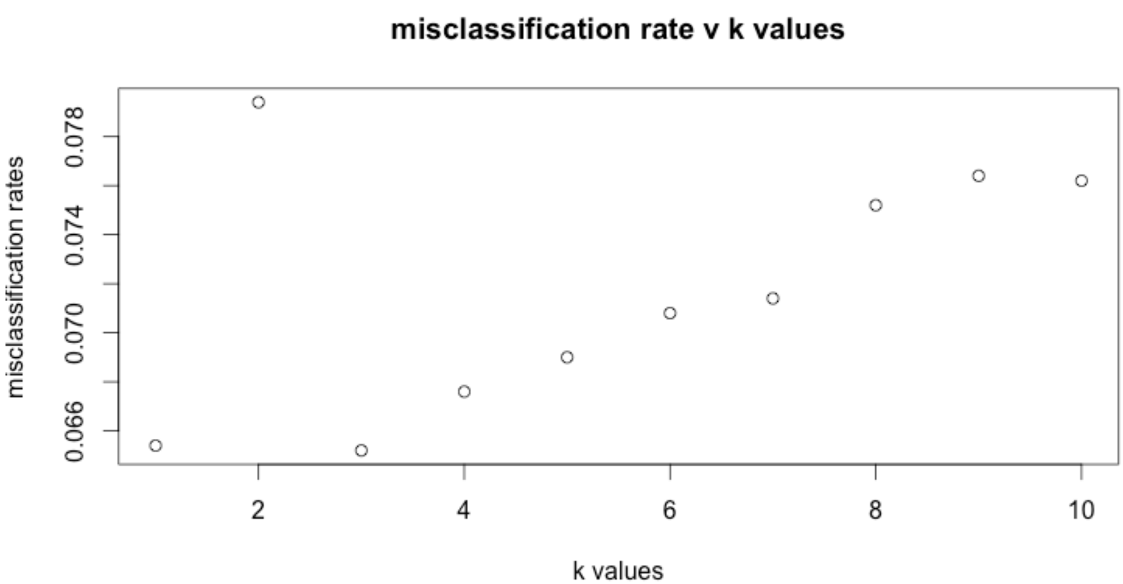
Hannah Kosinovsky
Worked with Chad Pickering, Sierra Tevlin, Rico Lin, Janice Luong, and Richard Safran
Resources: Chris Murphey(took this class last year), stackoverflow, rblogger, piazza, office hours
Section: Friday 8 am, Olson
11/3/15

Assignment 3

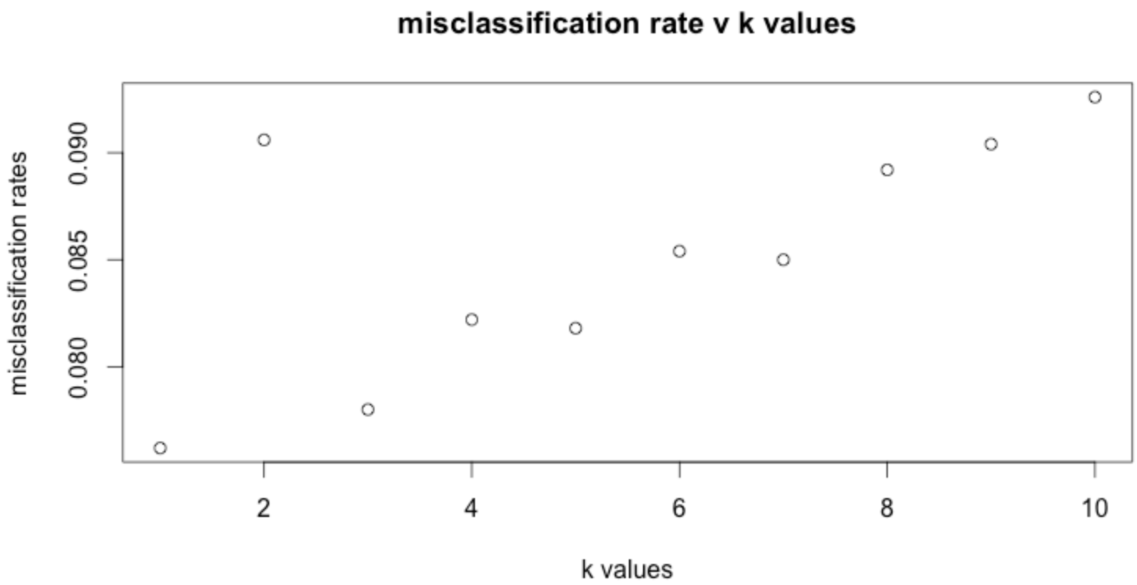
Part 1: k nearest neighbors:

1. The metric is euclidean, k is 3 is the best model because it has the lowest misclassification rate of 6.52%. I got 7.62% for the manhattan metric and 6.78% for the canberra metric.

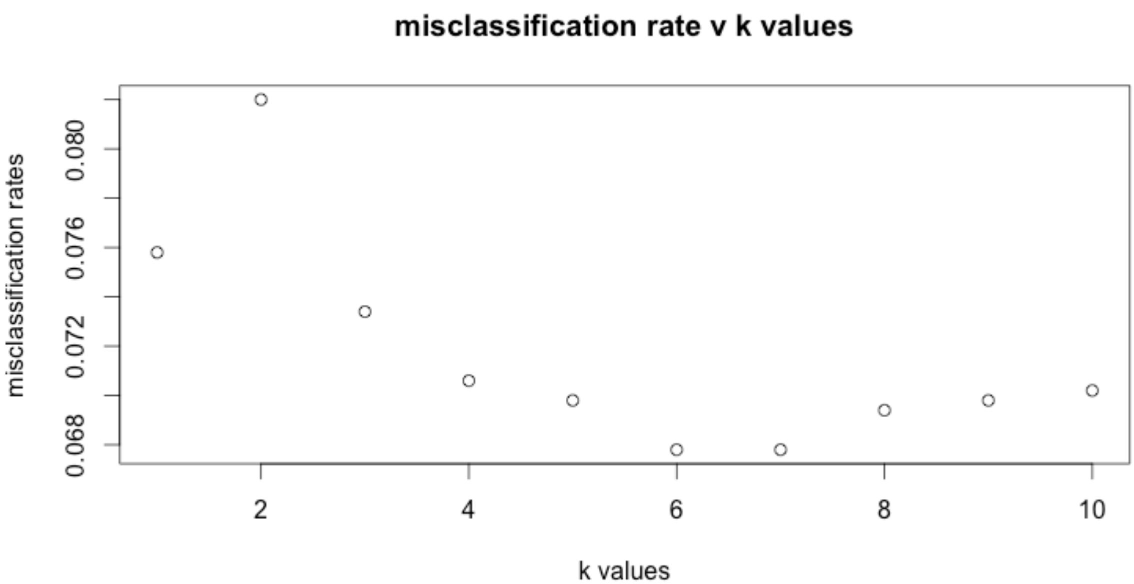
2. For euclidean, we see in the plot that the lowest k value is 3.



k = 1 is lowest for manhattan.



k = 6 is lowest for canberra.



3: The diagonal values classify correctly because the diagonal values have the same value for the row and column. Each cell shows if the row value was mistaken for the column value.

> confusion

	0	1	2	3	4	5	6	7	8	9
0	487	0	0	0	0	0	3	0	0	0
1	0	586	1	1	2	0	1	2	0	1
2	5	17	467	4	2	1	1	12	3	2
3	2	3	5	500	1	8	0	1	0	3

4	1	11	2	0	432	0	2	2	0	26
5	6	3	0	20	1	404	6	0	3	5
6	6	3	1	0	2	2	480	0	2	0
7	0	14	2	1	2	1	0	524	0	10
8	4	10	4	16	2	17	1	4	357	11
9	4	3	3	6	16	0	0	9	1	437

4: Zeros are misclassified the least amount of times so they're the best. Eights are misclassified the most amount of times so they're the worst.

5. 4s and 9s are confused 26 times and 5s and 3s are confused 20 times.

6. Here are some of the 5s that were misclassified as 3s

It is apparent how a machine could confuse the patterns in the images above for a 3 because there is a similar bottom curve and the top portion varies by handwriting and could be mistaken. A human, however, can easily see that these are 5s.

Part 2:

Using distance to average,

My euclidean distance misclassification rate is 18.48%
My manhattan distance misclassification rate is 33.86%
And my canberra distance misclassification rate is 45.02%

Knn nearest neighbors method is best because it has much smaller error rates than the distance to average method. Euclidean distance with k nearest neighbors is overall the best method because it has the smallest error rate overall.

Appendix:

```
#Worked with Chad Pickering, Sierra Tevlin, Rico Lin, Janice Luong, and Richard
Safran
setwd("/Users/hannah/desktop/stat141")

digitsdata=read.csv("digitsTrain.csv", header= TRUE)
labels= digitsdata[,1]
just_pixels= digitsdata[, -1]
dim(digitsdata)
#5000  785

#checking the code from the assignment
getImage =
 function(vals)
 {
  matrix(as.integer(vals), 28, 28, byrow = TRUE)
 }

draw = function(vals, colors = rgb((255:0)/255, (255:0)/255, (255:0)/255), ...)
{
 if(!is.matrix(vals))
  vals = getImage(vals)

 m = t(vals)  # transpose the image
 m = m[,nrow(m):1]  # turn up-side-down

 image(m, col = colors, xaxt = "n", yaxt = "n")
}

draw(digitsdata[1, -1]) #draws a single row w/out first col

sapply(digitsdata[, -1], max) #gives max value for each panel

par(mfrow = c(10, 10), mar = rep(0,4))
invisible(sapply(1:100, function(i) draw(digitsdata[i, -1]))) # draw first 10x10

by_label = split(digitsdata, labels) #split by label


par(mfrow = c(1,1))

getmeans = function(i) {
 draw(sapply(by_label[[i]][, -1], mean))
}
sapply(1:10, getmeans)

stdevs = function(i) {
```

```
  draw(sapply(by_label[[i]][, -1], sd))
}
sapply(1:10, stdevs)

#####  k nearest neighbors cross validation

#nonparametric test testing whether the rows are random
install.packages("randtests")
library(randtests)
runs.test(digitsdata[, 1])
# pvalue is 0.9812

# decide not to randomize based on nonparametric test
#the test rejects the null since the pvalue is very high so
#we conclude that they are not random
#so we shouldn't randomize.

#using different types of distance function and making each a matrix
euclidean_dist = as.matrix(dist(just_pixels, method = "euclidean"))
manhattan_dist = as.matrix(dist(just_pixels, method = "manhattan"))
canberra_dist = as.matrix(dist(just_pixels, method = "canberra"))

#create sets of training data w each of 3 distance metrics
training_data_euclidean = function(upperend)
{
  lower = upper - 999
  range = lower:upper
  test_train = euclidean_dist[range, -range]
  return(test_train)
}

training_data_manhattan = function(upperend)
{
  lower = upper - 999
  range = lower:upper
  test_train = manhattan_dist[range, -range]
  return(test_train)
}

training_data_canberra = function(upperend)
{
  lower = upper - 999
  range = lower:upper
  test_train = canberra_dist[range, -range]
  return(test_train)
}
```

```
#finding k nearest neighbors
kNN = function(row, k, distance, data)
{
  nearest = as.numeric(names(sort(distance[row, ])[1:k]))
  return(as.integer(names(which.max(table(data$label[nearest])))))
}

#create list of prediction values
predic_vals = function(k, metric) {
  list = c() #create empty list
  for (w in 1:5) # 5 folds
  {
    predictions = sapply(1:1000, function(y) kNN(y, k, metric[[w]], digitsdata))
    list = c(list, predictions)
  }
  return(list)
}

#find misclassification rates
misclassification_rates = function(k, metric) {
  predictions = lapply(1:k, function(x) predic_vals(x, metric))
  all_comparisons = data.frame(digitsdata[ , 1])
  all_comparisons$predictions = predictions[[k]]
  logical_table = table(all_comparisons[ , 1] == all_comparisons[ , 2])
  misclassification = logical_table[[1]]/(logical_table[[1]] + logical_table[[2]])
  misclassification
}

test_euclidean = lapply(c(1000, 2000, 3000, 4000, 5000), training_data_euclidean)
test_manhattan = lapply(c(1000, 2000, 3000, 4000, 5000),
training_data_manhattan)
test_canberra = lapply(c(1000, 2000, 3000, 4000, 5000), training_data_canberra)

par(mfrow = c(1,1))
par(mar = c(5.1, 4.1, 4.1, 2.1))

#questions 1 and 2
#final function to get the best kNN
best_kNN = function(k, metric)
{
  misclassifications = lapply(1:k, function(k) misclassification_rates(k, metric))
  unlist_misclass = unlist(misclassifications)
  order_misclass = order(unlist_misclass)
  best_k = as.numeric(head(order_misclass, 1))
```

```
  k_values = c(1:k)
  misclassed_k = data.frame(k_values, unlist_misclass)
  plot(misclassed_k[ , 1], misclassed_k[ , 2], xlab = "k values", ylab =
"misclassification rates", main = "misclassification rate v k values", )
  #this plot shows ks and misclassifications to see lowest k
  return_list = list(misclassifications[[best_k]][1], best_k)
  return(misclassifications[[best_k]][1])
}

#use function best_kNN to find the best k for each distance function type
#this checks through up to chose 10, but from the plots we can see the lowest k

best_k_euclidean = best_kNN(10, test_euclidean)
#0.0652
best_k_manhattan = best_kNN(10, test_manhattan)
#0.0762
best_k_canberra = best_kNN(10, test_canberra)
#0.0678

############## question 3-5: confusion matrix
predictions = predic_vals(k = 3, test_euclidean)
all_comparisons = data.frame(digitsdata[ , 1])
all_comparisons$predictions = predictions
confusion_matrix = table(all_comparisons[ , 1] , all_comparisons[ , 2])
confusion_matrix


#6: Show some of the misclassified digits, 5 and 3 were mistaken.
#they were confused 20 times (as seen in confusion matrix)
fivesandthrees = all_comparisons[(all_comparisons[ , 1] == 5 & all_comparisons[ , 2]
== 3) , ]
par(mfrow = c(2, 2), mar = c(5.1, 4.1, 4.1, 2.1))
drawingmistakes = function(row) {
  draw(digitsdata[row, -1])
}
#chose 4 times that 5 and 3 were confused
sapply(c(803, 1119, 2058, 4220), drawingmistakes)

#### step 2 ###### distance to average

#comparing rows 11-5010 to colums 1-10 , distance from
#image to average image for each label
label_0 = subset(digitsdata, digitsdata$label == 0)
avg_for_label_0 = as.matrix(sapply(label_0[-1], mean))

label_1 = subset(digitsdata, digitsdata$label == 1)
```

```
avg_for_label_1 = as.matrix(sapply(label_1[-1], mean))

label_2 = subset(digitsdata, digitsdata$label == 2)
avg_for_label_2 = as.matrix(sapply(label_2[-1], mean))

label_3 = subset(digitsdata, digitsdata$label == 3)
avg_for_label_3 = as.matrix(sapply(label_3[-1], mean))

label_4 = subset(digitsdata, digitsdata$label == 4)
avg_for_label_4 = as.matrix(sapply(label_4[-1], mean))

label_5 = subset(digitsdata, digitsdata$label == 5)
avg_for_label_5 = as.matrix(sapply(label_5[-1], mean))

label_6 = subset(digitsdata, digitsdata$label == 6)
avg_for_label_6 = as.matrix(sapply(label_6[-1], mean))

label_7 = subset(digitsdata, digitsdata$label == 7)
avg_for_label_7 = as.matrix(sapply(label_7[-1], mean))

label_8 = subset(digitsdata, digitsdata$label == 8)
avg_for_label_8 = as.matrix(sapply(label_8[-1], mean))

label_9 = subset(digitsdata, digitsdata$label == 9)
avg_for_label_9 = as.matrix(sapply(label_9[-1], mean))

avgs_each_label = cbind(avg_for_label_0, avg_for_label_1, avg_for_label_2,
                        avg_for_label_3, avg_for_label_4, avg_for_label_5,
                        avg_for_label_6, avg_for_label_7, avg_for_label_8,
                        avg_for_label_9)
avgs_each_label = as.data.frame(avgs_each_label)
label_names = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9")
colnames(avgs_each_label) = label_names

#Arrange the averages and the pixels into a single dataset
transpose_pixels = t(just_pixels)
avgs_pixels = cbind(avgs_each_label, transpose_pixels)
t_avgs_pixels = t(avgs_pixels)
dim(t_avgs_pixels)
#gives us 5010  784, which is good because we
#ultimately wanted 784 columns
#were taking the transpose in order to be able to use cbind

#apply euclidean distance function for the 5010 by 784 matrix.
euclidean_dist_avgs = dist(t_avgs_pixels, method = "euclidean")
euclidean_dist_avgs = as.matrix(euclidean_dist_avgs)
```

```
traintest_avgs = euclidean_dist_avgs[11:5010, 1:10]

kNN_mean = function(row, distance, data)
{
 x = as.numeric(names(sort(distance[row,])[1]))
 return(x)
}

predictions_avgs = sapply(1:5000, function(y) kNN_mean(y, traintest_avgs,
digitsdata))
euc_avgs_comp = data.frame(digitsdata[,1])
euc_avgs_comp$predictions = predictions_avgs
test_euc_avgs = table(euc_avgs_comp[,1] == euc_avgs_comp[,2])
false_euc_avgs = as.integer(test_euc_avgs[1])
true_euc_avgs = as.integer(test_euc_avgs[2])
euc_misclass_rate = false_euc_avgs/(false_euc_avgs+true_euc_avgs)
euc_misclass_rate
#0.1848

#manhattan
manhattan_dist_avgs = dist(t_avgs_pixels, method = "manhattan")
manhattan_dist_avgs = as.matrix(manhattan_dist_avgs)

traintest_avgs = manhattan_dist_avgs[11:5010, 1:10]
predictions_avgs = sapply(1:5000, function(y) kNN_mean(y, traintest_avgs,
digitsdata))
man_avgs_comp = data.frame(digitsdata[,1])
man_avgs_comp$predictions = predictions_avgs
test_man_avgs = table(man_avgs_comp[,1] == man_avgs_comp[,2])
false_man_avgs = as.integer(test_man_avgs[1])
true_man_avgs = as.integer(test_man_avgs[2])
man_misclass_rate = false_man_avgs/(false_man_avgs+true_man_avgs)
man_misclass_rate
#0.3386

#canberra
canberra_dist_avgs = dist(t_avgs_pixels, method = "canberra")
canberra_dist_avgs = as.matrix(canberra_dist_avgs)

traintest_avgs = canberra_dist_avgs[11:5010, 1:10]
predictions_avgs = sapply(1:5000, function(y) kNN_mean(y, traintest_avgs,
digitsdata))
can_avgs_comp = data.frame(digitsdata[,1])
can_avgs_comp$predictions = predictions_avgs
test_canberra_avgs = table(can_avgs_comp[,1] == can_avgs_comp[,2])
```

```
false_canberra_avgs = as.integer(test_canberra_avgs[1])
true_canberra_avgs = as.integer(test_canberra_avgs[2])
can_misclass_rate = false_canberra_avgs/(false_canberra_avgs+true_canberra_avgs)
can_misclass_rate
#0.4502
```