Katherine Olson

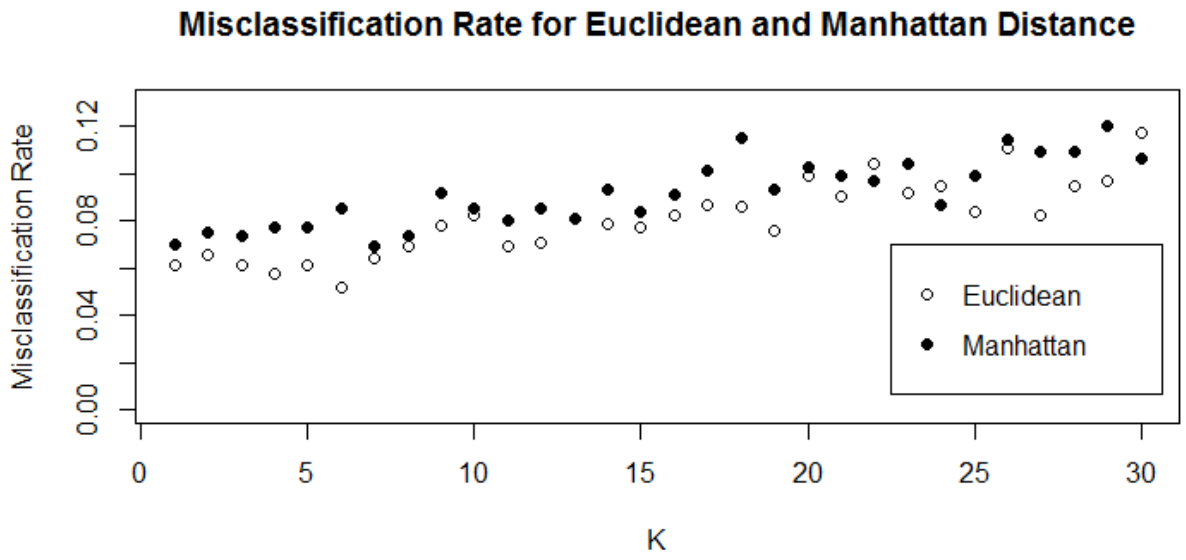
STA141 Assignment Three

Report

Task 1: K-nearest neighbors and cross valdidation

Question 1: Using cross-validations and comparing models with Euclidean and Manhattan distance for values of k from one to thirty, the best model is Euclidean distance with k equal to six.

Question 2: The plot can be seen below. The plot tells us that Euclidean and Manhattan have close misclassification rates, but Euclidean is better overall. Both metrics have a positive relationship between k and the misclassification rate. So as k increases, so does the misclassification rate, overall.



Question 3: Confusion matrix

Predictions

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	486	0	0	0	0	1	3	0	0	0
1	0	587	1	1	2	0	1	2	0	0
2	3	15	469	6	2	1	3	10	3	2
3	1	1	3	502	1	9	0	2	0	4
4	0	9	0	0	434	0	4	1	0	28
5	2	3	0	11	1	415	7	0	3	6
6	6	3	1	0	0	3	481	0	2	0

7	0	15	1	0	0	0	0	529	0	9
8	2	10	0	14	1	12	2	4	370	11
9	3	4	0	6	5	0	0	10	2	449

True labels are on the y-axis

Question 4:

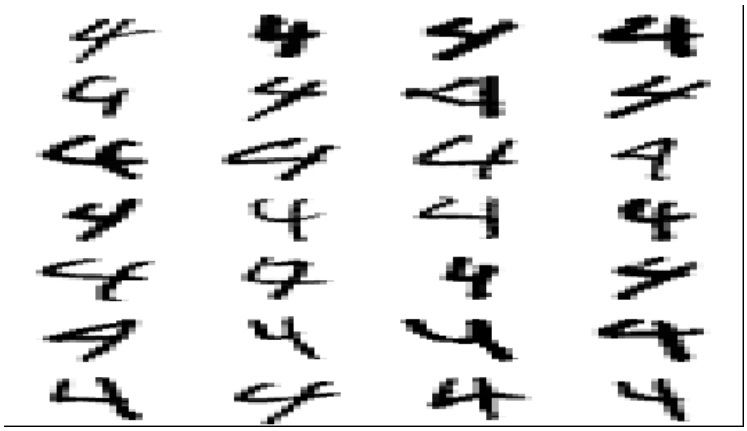
Looking at the confusion matrix, twos were classified really well, with 3 being the highest sum in each column and 4 misclassified overall. Ones were also classified really well, with 2 being the highest sum in each column and with 7 misclassified overall. The rest were not classified as well. The sixes, threes, sevens, nines and fives were in the middle, in that order, with 15 to 33 digits misclassified overall. Then the fours, twos, and eights were classified the worst with 42 to 56 digits misclassified overall.

Question 5:

Looking at counts of 10 and above, fours were misclassified as nines. Sevens were misclassified as ones. Eights were misclassified as ones, threes, fives and nines. Twos were misclassified as ones and sevens. Fives were misclassified as threes. Nines were misclassified as sevens.

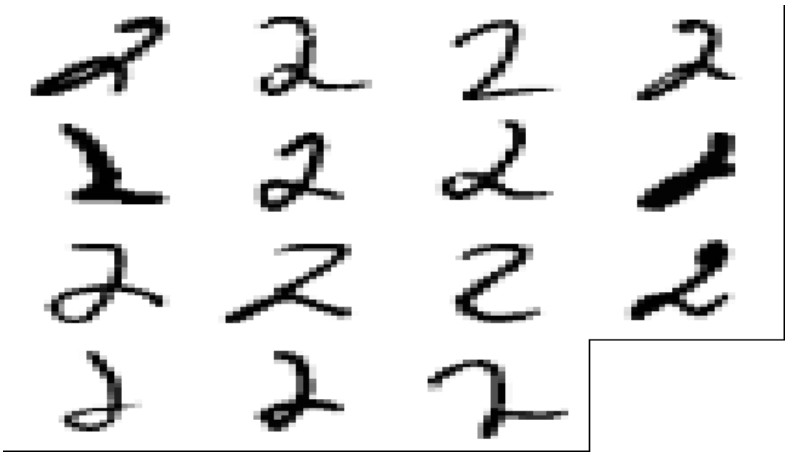
Question 6:

Fours that were misclassified as nines:



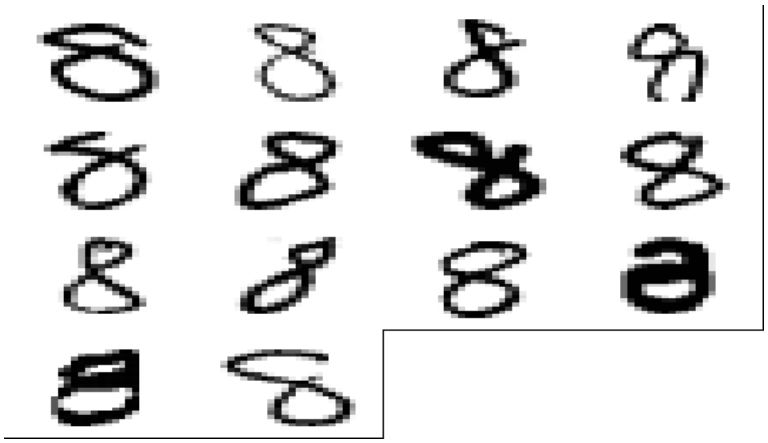
Looking at the plot on the left, these were probably misclassified because they are at a diagonal angle or the top of the four is connected like a nine.

Twos misclassified as ones:



Looking at the plot on the left, it is strange that these twos were misclassified as ones. Most of them are curly twos, which are might just be enough to make them more similar to ones.

Eights misclassified as threes:



Looking at the plot on the left, the eights that were misclassified as threes are either have very round loops or are at an angle.

Task 2: Distance to average and cross validation

Using this method, I got a misclassification rate of 0.192 for Euclidean and 0.343 for Manhattan. These misclassification rate are much higher than the rates from k-nearest neighbors. So k-nearest neighbors was a better method.

Code Appendix

```
# Task 1: k-nearest neighbors and Cross Validation
# ====================================
# Function that does knn for each fold and returns predictions (yhat):
eachGroup = function(distTest, k, digits){
        cnames = colnames(distTest) # Saving the colnames of the 4000 across

        predictions = t(apply(distTest, 1, function(row){
                row = order(row) #order that row, but not correct, the cols are not continuous(1,2,
                # 4, 5, 7)
                # and order gives us order with (1, 2, 3, 4, 5, 6, 7)
        row = as.numeric(cnames[row]) # Fixing that
        if (k == 1){
                yhat = digits[which(rownames(digits) == row[1]), 1] # Nearest neighbor is 1st
                # element
        }
        else{ # This needs to deal with ties
                labels = sapply(1:k, function(i) digits[which(rownames(digits) == row[i]), 1])
                # Getting the labels for all k nearest neighbors
                 highest = max(table(labels)) # Getting highest count
                labsMaxTF = (table(labels) == highest) # Finding T/F for labels with that count
                 labsMaxT = labsMaxTF[labsMaxTF == TRUE] # Only T for highest count
```

```
        highLabs = names(labsMaxT[labsMaxT == TRUE]) # Label(s) for highest count

        if (length(highLabs) == 1) {# (no ties - one label has highest count)
                yhat = as.numeric(highLabs)
        }

        else if (length(highLabs) == 2) { # Two way tie
                 a = which(labels == highLabs[1])
                 b = which(labels == highLabs[2])
                 asum = sum(a) # Summing the values of nearest neighbor for 1st tie
                bsum = sum(b)
                if (asum < bsum) {# THe two distances of label 1 are closer
                        yhat = as.numeric(highLabs[1])
                }
                else {
                        yhat = as.numeric(highLabs[2])
                }
        }

        else { # There is a tie, more than a two way tie
                 highLabs = as.numeric(highLabs)
                b = sapply(1:length(highLabs), function(i){ # For each tie
                        a = which(labels == highLabs[i])
                        asum = sum(a) # Sum how close each one is
                 })
                 almostyhat = highLabs[which(b == min(b))] # Want lowest sum
                if (length(almostyhat == 1)){
                         yhat = almostyhat
                }
                else { # Tie of tie sums
                        # Just going to pick the first one
                        yhat = almostyhat[1]
                  }
        }
}
if (length(yhat) > 1){ # For some reason some yhat are returning 2 values, just picking
# first one
        yhat = yhat[1]
 }
yhat
```

```
  }))
return(predictions)
}

# The KNN function, gives average of 5 (5-fold) decimals of how accurate the prediction was:
KNN = function(data, k, distance){
        # Shuffle data:
        digits = data[sample(nrow(data), replace = FALSE),] # has row.names

        # Split into 5 groups and send each section of distance to eachGroup

        # Group 1:
        g1 = as.numeric(row.names(digits[1:1000,])) # All row indices
        dist1 = distance[g1, -g1] # gives 1000 by 4000 matrix, 1000 = g1, 4000 = not g1
        # (rownums)
        pred1 = eachGroup(dist1, k, digits) # eachGroup does knn and returns decimal of how
        many predictions were correct
        mc1 = mean(digits[1:1000, 1] != pred1) # mc = missclassification rate

         # Group 2:
        g2 = as.numeric(row.names(digits[1001:2000,]))
        dist2 = distance[g2, -g2]
        pred2 = eachGroup(dist2, k, digits)
        mc2 = mean(digits[1001:2000, 1] != pred2)

        # Group 3:
        g3 = as.numeric(row.names(digits[2001:3000,]))
        dist3 = distance[g3, -g3]
        pred3 = eachGroup(dist3, k, digits)
        mc3 = mean(digits[2001:3000, 1] != pred3)

        # Group 4:
        g4 = as.numeric(row.names(digits[3001:4000,]))
        dist4 = distance[g4, -g4]
         pred4 = eachGroup(dist4, k, digits)
        mc4 = mean(digits[3001:4000, 1] != pred4)

        # Group 5:
         g5 = as.numeric(row.names(digits[4001:5000,]))
        dist5 = distance[g5, -g5]
```

```
        pred5 = eachGroup(dist5, k, digits)
        mc5 = mean(digits[4001:5000, 1] != pred5)

        return(mean(mc1, mc2, mc3, mc4, mc5))
}

# Calling the functions:
digits = read.csv("~/UC Davis/STA 141/HW 3/digitsTrain.csv") # 5000 by 785
# Creating the 2 distance matrices:
euclidean = as.matrix(dist(digits[, -1], method = "euclidean"))
manhattan = as.matrix(dist(digits[, -1], method = "manhattan"))
# CV for Euclidian:
first20 = lapply(1:20, function(i) KNN(digits, i, euclidean))
next10E = lapply(21:30, function(i) KNN(digits, i, euclidean))
lowestE = min(unlist(first20))
bestkE = which(first20 == lowestE)

# CV for Manhattan:
first30M = lapply(1:30, function(i) KNN(digits, i, manhattan))
lowestM = min(unlist(first30M))
bestkM = which(first20M == lowestM)

# Question 1:
# --------------
# Report the best model, i.e., value of k and metric.
if (lowestE < lowestM){
        # bestModel = KNN(digits, bestkE, euclidean)
        bestk = print(bestkE)
        bestMetric = print("euclidean")
}
else{
        bestModel = KNN(digits, bestkM, manhattan)
        bestk = bestkM
        bestk = print(bestkM)
        bestMetric = print("manhattan")
}

# Question 2:
# --------------
```

```
plot(1:30, c(first20, next10E), ylim = c(0, 0.13), main = "Misclassification Rate for Euclidean
        and Manhattan Distance", ylab = "Misclassification Rate", xlab = "K")
points(1:30, y = first30M, pch = 19) # pch = 1 for first points, these are manhattan
leg = c("Euclidean", "Manhattan")
legend(22.5, 0.07, legend = leg, pch = c(1, 19))

# Question 3:
# ---------------

# Modify KNN function:
KNNConf = function(digits, k, distance){
        # Don't need to shuffle data

        # Loop through each row of distance matrix and get prediction:
        predictions = sapply(1:5000, function(i){
                # Need to find that section in the distance matrix and remove the unwanted part:
                wanted = distance[i, -(i)]
                # Send to eachGroupConf which returns the prediction:
                yhat = eachGroupConf(wanted, k, digits)
        })

        confMat = table("True Digits" = digits[1:5000, 1], "Predictions" = predictions)

        return(confMat)
}

# Modifying eachGroup function:
eachGroupConf = function(distTest, k, digits){
        # distTest is now one row of dist matrix, not entire section
        row = distTest
        cnames = names(row) # Saving the colnames since order gets them wrong

        # Do the same thing as eachGroup, but outside of a loop
        row = order(row)
        row = as.numeric(cnames[row])

        # Now KNN:
        if (k == 1){
                yhat = digits[which(rownames(digits) == row[1]), 1] # Nearest neighbor is 1st
                # element
```

```
}
else{ # This needs to deal with ties
        labels = sapply(1:k, function(i) digits[which(rownames(digits) == row[i]), 1])
        #Getting the labels for all k nearest neighbors
        highest = max(table(labels)) # Getting highest count
        labsMaxTF = (table(labels) == highest) # Finding T/F for labels with that count
        labsMaxT = labsMaxTF[labsMaxTF == TRUE] # Only T for highest count
        highLabs = names(labsMaxT[labsMaxT == TRUE]) # Label(s) for highest count

        if (length(highLabs) == 1) {# (no ties - one label has highest count)
                yhat = as.numeric(highLabs)
        }

        else if (length(highLabs) == 2) { # Two way tie
                a = which(labels == highLabs[1])
                b = which(labels == highLabs[2])
                asum = sum(a) # Summing the values of nearest neighbor for 1st tie
                bsum = sum(b)
                if (asum < bsum) {# THe two distances of label 1 are closer
                        yhat = as.numeric(highLabs[1])
                }
        else {
                yhat = as.numeric(highLabs[2])
        }
}

else { # There is a tie, more than a two way tie
        highLabs = as.numeric(highLabs)
        b = sapply(1:length(highLabs), function(i){ # For each tie
                a = which(labels == highLabs[i])
                asum = sum(a) # Sum how close each one is
        })
        almostyhat = highLabs[which(b == min(b))] # Want lowest sum
        if (length(almostyhat == 1)){
                yhat = almostyhat
        }
        else { # Tie of tie sums
                # Just going to pick the first one
                yhat = almostyhat[1]
        }
```

```
        }
  }
        if(length(yhat) > 1){
                yhat = yhat[1]
        }

        return(yhat)
}

confMat = KNNConf(digits, bestk, euclidean)

# Question 6:
# -----------

# Modifying KKN again:
KNNPlot = function(digits, k, distance){
  # Don't need to shuffle data

  # Loop through each row of distance matrix and get prediction:
  predictions = sapply(1:5000, function(i){
    # Need to find that section in the distance matrix and remove the unwanted part:
    wanted = distance[i, -(i)]
    # Send to eachGroupConf which returns the prediction:
    yhat = eachGroupConf(wanted, k, digits)
  })

  # Now want to return the predictions for plotting, already have digts:
  return(predictions)
}

pred = KNNPlot(digits, bestk, euclidean)
truth = digits[1:5000, 1]

# Plotting 4s that were misclassified as 9s:(28 of them)
par(mfrow = c(7,4), mar = rep(0,4))
locs = sapply(1:5000, function(i){
  if ((truth[i] == 4) & (pred[i] == 9)){
    draw(digits[i, -1])
  }
})
```

```
# Plotting 2s misclassified as ones: (15 of them)
par(mfrow = c(4,4), mar = rep(0,4))
locs2 = sapply(1:5000, function(i){
  if ((truth[i] == 2) & (pred[i] == 1)){
    draw(digits[i, -1])
  }
})

# Eights misclassified as threes: (14 of them)
par(mfrow = c(4,4), mar = rep(0,4))
locs3 = sapply(1:5000, function(i){
  if ((truth[i] == 8) & (pred[i] == 3)){
    draw(digits[i, -1])
  }
})
```

 # Distance to Average and Cross Validation

 # ==========

```
# Function that splits the data and sends each fold to DistAvg2(), and then returns the
# mean of the misclassification rates for each fold:
DistAvg = function(data, typeDist){
        # Shuffle data:
        digits = data[sample(nrow(data), replace = FALSE),] # has row.names

        # Group 1:
         test1 = digits[1:1000, ]
        train1 = digits[-(1:1000), ]
         result1 = DistAvg2(test1, train1, digits, typeDist)

        # Group 2:
        test2 = digits[1001:2000, ]
        train2 = digits[-(1001:2000), ]
        result2 = DistAvg2(test2, train2, digits, typeDist)

        # Group 3:
        test3 = digits[2001:3000, ]
        train3 = digits[-(2001:3000), ]
        result3 = DistAvg2(test3, train3, digits, typeDist)
```

```
	# Group 4:
	test4 = digits[3001:4000, ]
	train4 = digits[-(3001:4000), ]
	result4 = DistAvg2(test4, train4, digits, typeDist)

	# Group 5:
	test5 = digits[4001:5000, ]
	train5 = digits[-(4000:5000), ]
	result5 = DistAvg2(test5, train5, digits, typeDist)

	return(mean(c(result1, result2, result3, result4, result5)))
}

# Function that returns the missclassification rate:
DistAvg2 = function(test, train, digits, typeDist){
	# List of 10, each contains a subset of digits per label:
	labSplit = lapply(0:9, function(lab){
		oneLab = subset(train, train[, 1] == lab)
		colMeans(oneLab)
	})

	# Need to turn labSplit into a matrix:
	avgs = do.call(rbind, labSplit)
	avgs = avgs[, -1] # Getting rid of label col
	stack = rbind(test[,-1], avgs)
	distance = as.matrix(dist(stack, method = typeDist))
	distance = distance[1:1000,-(1:1000)] # Removing what we don't want
	# Now need to order like KNN:
	predictions = t(apply(distance, 1, function(row){
		row = order(row) # 1 = 0, 2 = 1, and so on...
		yhat = row[1] - 1
	}))
	mc = mean(test[,1] != predictions) #misclassification

	return(mc)
}

# Calling the functions:
mcEuc = DistAvg(digits, "euclidean")
```

mcMan = DistAvg(digits, "manhattan")

Resources

- Piazza
- Code and hints from lecture and discussion
- Links:
    - https://stat.ethz.ch/pipermail/r-help/2004-December/062499.html
    - https://stat.ethz.ch/R-manual/R-devel/library/base/html/colnames.html
    - https://stat.ethz.ch/R-manual/R-devel/library/base/html/colSums.html