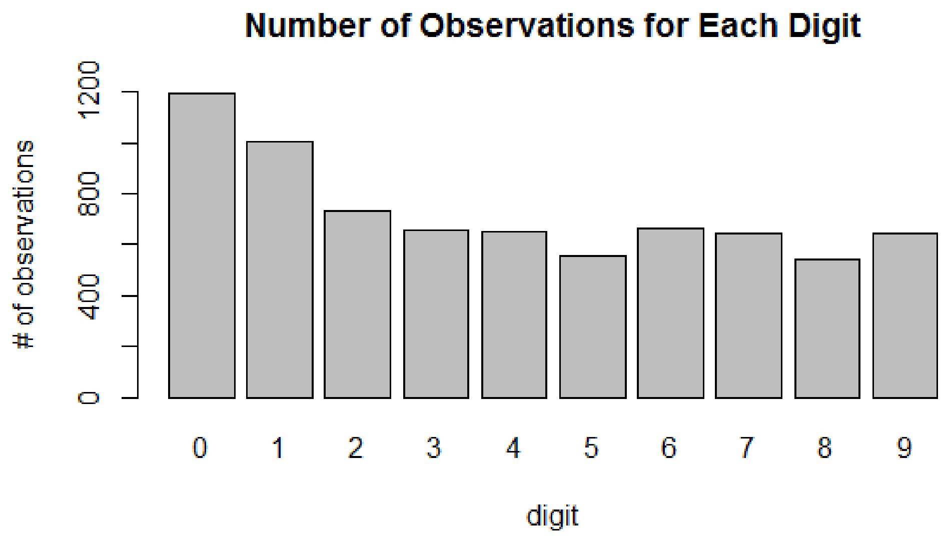
Jennifer Wong
913160121
STA 141A

Final Project

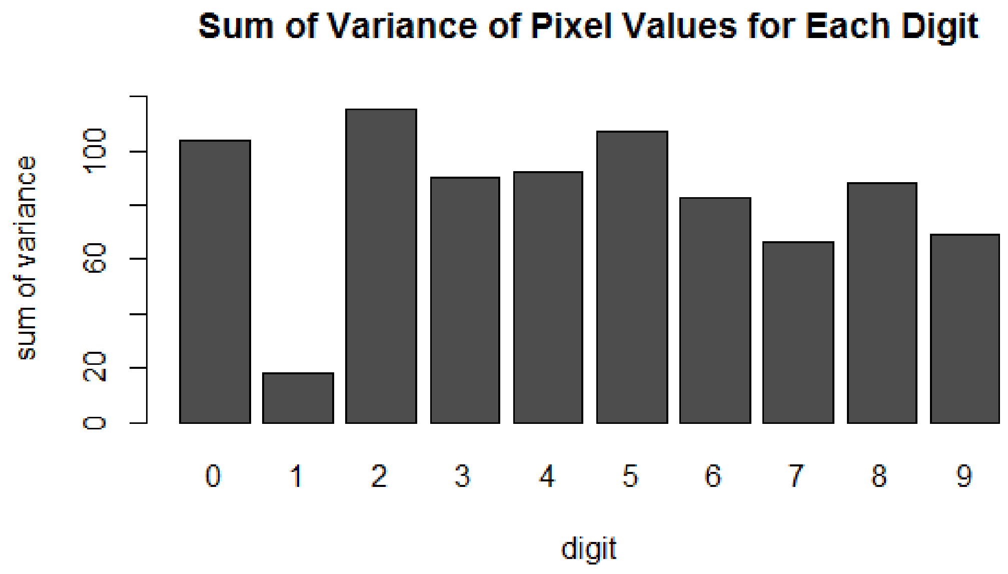
1. The read_digits function can be found in the appendix under "Problem 1".

2. The view_digit function can be found in the appendix under "Problem 2".

3. The barplot below shows the number of observations for each digit. Digit 0 has close to 1,200 observations, which is the most for any digit. This means that 0's occur frequently in zip codes. Out of all the digits, digit 8 has the least number of observations. In general, as the digit increases, there are fewer number of observations.



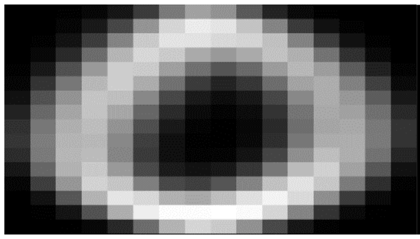
The barplot below shows the sum of variance of pixel values for each digit. Digit 1 has the smallest sum of variance. It implies that there is not a lot of variation between the observations that were classified as digit 1. This is most likely occurs because digit 1 is the simplest digit to write, so most people write 1's the same way. Digit 2 has the largest sum of variance. This means that there is a lot of variance between the observations that were classified as digit 2.



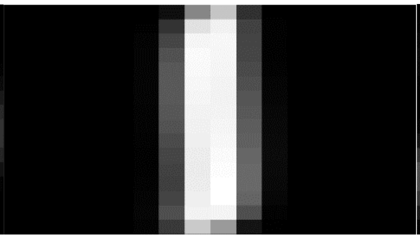
Jennifer Wong
913160121
STA 141A

The images below show what each digit looks like on average. The average image of digit 1 seems the least blurry. This supports the conclusion that there is not much variation between the observations that were classified as digit 1 because it is the simplest digit to write. The average images of digit 2 and digit 5 seem blurrier than the other images. This means that there is high variation between the observations for each class.

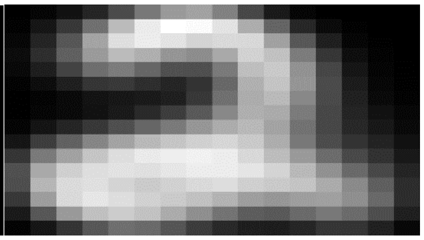
Digit 0



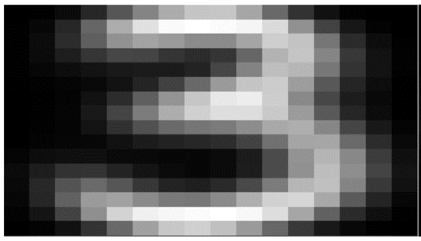
Digit 1



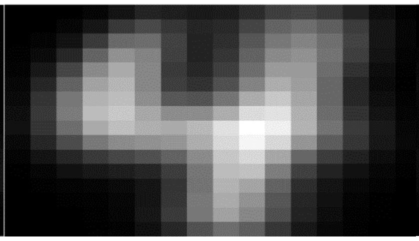
Digit 2



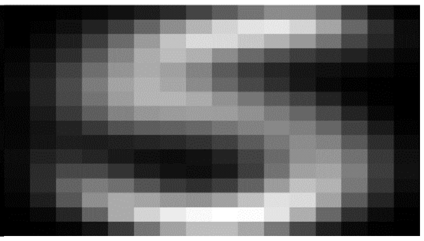
Digit 3



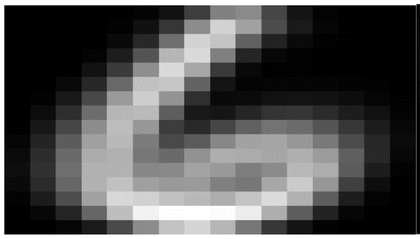
Digit 4



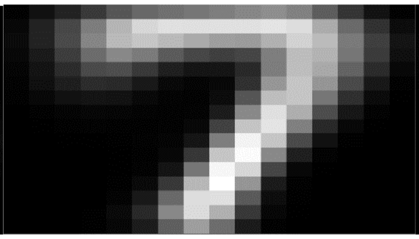
Digit 5



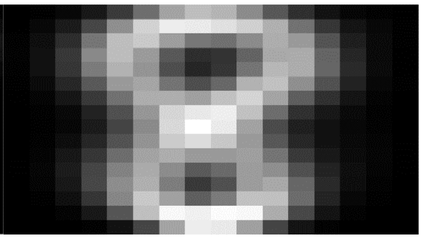
Digit 6



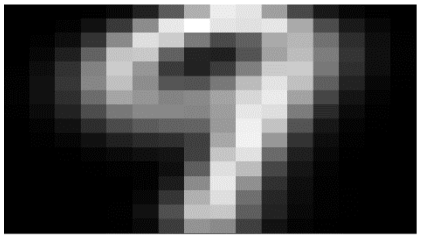
Digit 7



Digit 8



Digit 9



To determine the pixels that are the most useful and the least useful for classification, I first found the average value for each pixel for each digit. Then, I found the variance for each pixel. The pixels with the highest variance are the most useful pixels, since they vary the most from digit to digit. The pixels with the lowest variance are the least useful pixels because they are approximately the same for each class. The most useful pixels, the least useful pixels, and their variance are shown in the tables below.

Most Useful Pixels and Their Variance

Pixel 230	Pixel 213	Pixel 219	Pixel 220	Pixel 120
.4975	.4314	.4215	.4115	.4044

Least Useful Pixels and Their Variance

Pixel 1	Pixel 241	Pixel 16	Pixel 256	Pixel 80
6.0809e-05	7.4424e-05	1.1859e-04	2.5915e-04	3.429e-04

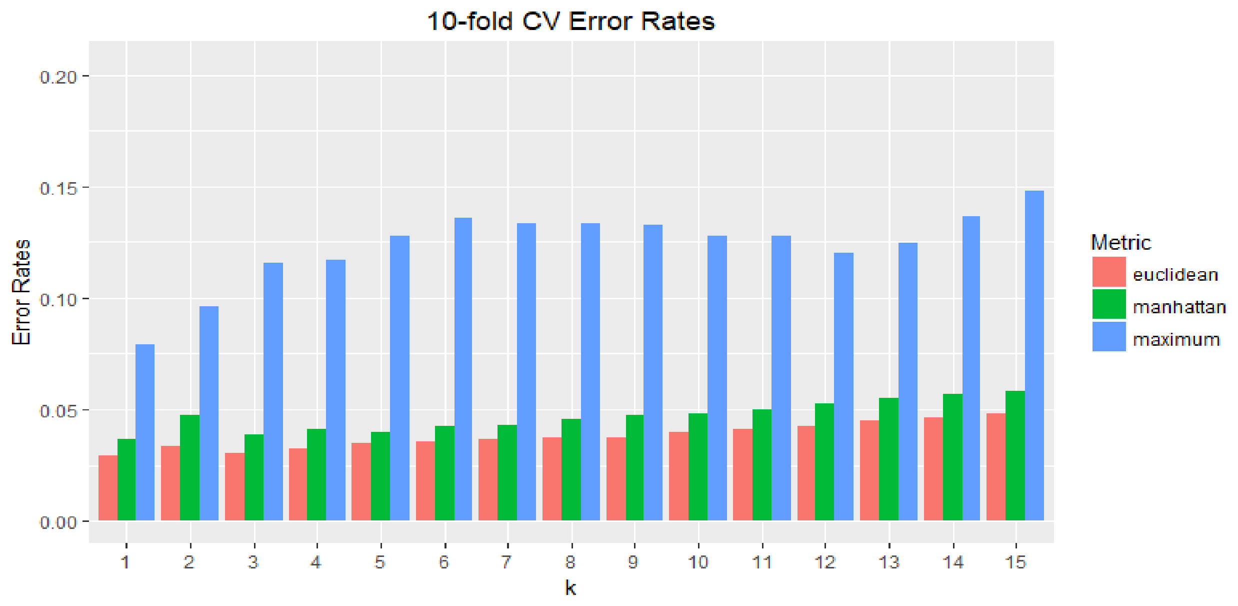
The least useful pixels are located near the left and right-hand side of the image. Since the pixels on the sides of the image are almost always black regardless of the digit, it makes sense that these are the least useful pixels for classification. The most useful pixels are generally located in the center of the image, which where the digit is displayed.

4. The predict_knn function can be found in the appendix under “Problem 4”.

5. The cv_error_knn function can be found in the appendix under “Problem 5”.

One of the strategies that I used to make my function run efficiently was to minimize the number of operations inside a for loop. Initially, I calculated the distances between each point inside a loop. The computation took a long time, so I modified my function so that it only calculated distances once. Another strategy I used was to avoid growing vectors. I stored the error rates of the cross-validation in a vector. I made sure to set the length of the vector before I started putting data into the vector because gradually increasing the length of a vector is very inefficient.

6.



Jennifer Wong
913160121
STA 141A

The barplot on the previous page shows the 10-fold cross-validation error rates for k = 1, …, 15 and the distance metrics “euclidean”, “manhattan”, and “maximum”. The graph clearly shows that the best distance metric to use is “euclidean”, and the worst is “maximum”. The difference in error rates between “euclidean” and “maximum” can be as large as 0.1. The “manhattan” distance metric is only a bit worse than “euclidean”. The error rates for the “maximum” distance metric seem to follow a polynomial curve. For the “euclidean” and “manhattan” distance metrics, the error rates generally increase as k increases. Based on this, I do not think that considering additional values of k would be useful. The “euclidean” and k = 1 combination is the best combination, since it has the lowest error rate.

7. The tables below are the confusion matrices for the 3 best k and distance metric combinations. For all three combinations, the distance metric is “euclidean”. The k values of 1, 3, and 4. The values in the confusion matrices are proportions.

Confusion Matrix for k = 1, distance metric = “euclidean”

True Class

Predicted Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	.1624	0	.0005	.0004	.0004	.0010	.0011	0	.0005	0
1	0	.1376	.0002	0	.0007	0	.0001	.0004	.0007	0
2	.0007	0	.0966	.0007	.0004	.0004	0	.0001	.0001	0
3	.0001	0	.0004	.0868	.0001	.0015	0	0	.0019	.0001
4	0	.0003	0	0	.0851	.0003	0	.0005	.0004	.0008
5	.0003	0	.0001	.0012	0	.0712	.0004	.0001	.0007	0
6	.0003	0	.0001	0	.0001	.0011	.0894	0	.0004	0
7	0	0	.0018	.0001	.0003	.0001	0	.0864	.0003	.0013
8	0	0	.0004	.0008	.0001	.0004	0	0	.0691	0
9	0	0	0	.0001	.0021	.0003	0	.0008	.0001	.0860

Confusion Matrix for k = 3, distance metric = “euclidean”

True Class

Predicted Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	.1627	0	.0008	.0008	.0003	.0014	.0015	0	.0007	.0004
1	0	.1377	.0003	.0001	.0018	.0001	.0003	.0004	.0008	0
2	.0003	.0001	.0964	.0003	.0005	.0004	0	.0001	.0003	0
3	.0003	0	.0005	.0871	0	.0008	0	0	.0015	.0001
4	0	0	.0001	0	.0838	.0004	.0001	.0005	.0004	.0008
5	.0001	0	0	.0005	0	.0720	.0003	0	.0005	0
6	.0004	0	.0001	0	.0007	.0007	.0889	0	.0004	0
7	0	0	.0016	.0001	.0001	0	0	.0867	.0005	.0017
8	0	0	.0001	.0011	0	.0001	0	0	.0689	0
9	0	0	.0001	.0001	.0022	.0027	0	.0007	.0003	.0853

Jennifer Wong
913160121
STA 141A

Confusion Matrix for k = 4, distance metric = "euclidean"

True Class

Predicted Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	.1627	0	.0011	.0005	.0003	.0014	.0023	0	.0005	.0004
1	0	.1377	.0003	.0001	.0022	0	.0003	.0004	.0007	0
2	.0003	.0001	.0964	.0004	.0004	.0004	0	.0001	.0001	0
3	.0003	0	.0007	.0872	0	.0016	0	0	.0023	.0001
4	0	0	.0001	.0001	.0846	.0004	.0001	.0007	.0004	.0012
5	.0001	0	0	.0005	0	.0719	.0001	.0001	.0009	0
6	.0004	0	.0001	0	.0004	.0004	.0882	0	.0004	0
7	0	0	.0012	.0003	.0001	0	0	.0865	.0004	.0023
8	0	0	.0001	.0010	0	0	0	0	.0680	0
9	0	0	.0001	0	.0014	.0001	0	.0005	.0004	.0842

All the combinations are good at predicting digit 1. The confusion matrices of k = 3, and k = 4 are very similar. It appears that when k = 1, digit 5 is not predicted with as much accuracy compared to when k = 3 or when k = 4. However, when k = 1, digit 2 is predicted the best. Since digit 2 has the most variance among its observations, and it is predicted the best when k = 1, I think k = 1, distance metric = "euclidean" is the best combination.

8. The table below is the confusion matrix for the misclassified data when k = 1 and distance metric = "euclidean". Digit 8 is the most misclassified digit. Out of the 237 misclassified points, 46 of them are digit 8. Digit 1 is the least misclassified digit with it being misclassified only once. Many of the misclassified digits were wrongly classified as digit 0. Digit 6 and 5 are often wrongly classified as digit 0.
Based on this confusion matrix, I conclude that the classifier is extremely good at classifying digit 1. However, the classifier is not good at classifying digit 8, and it tends to misclassify digits as digit 0.

Confusion Matrix for Misclassified Data

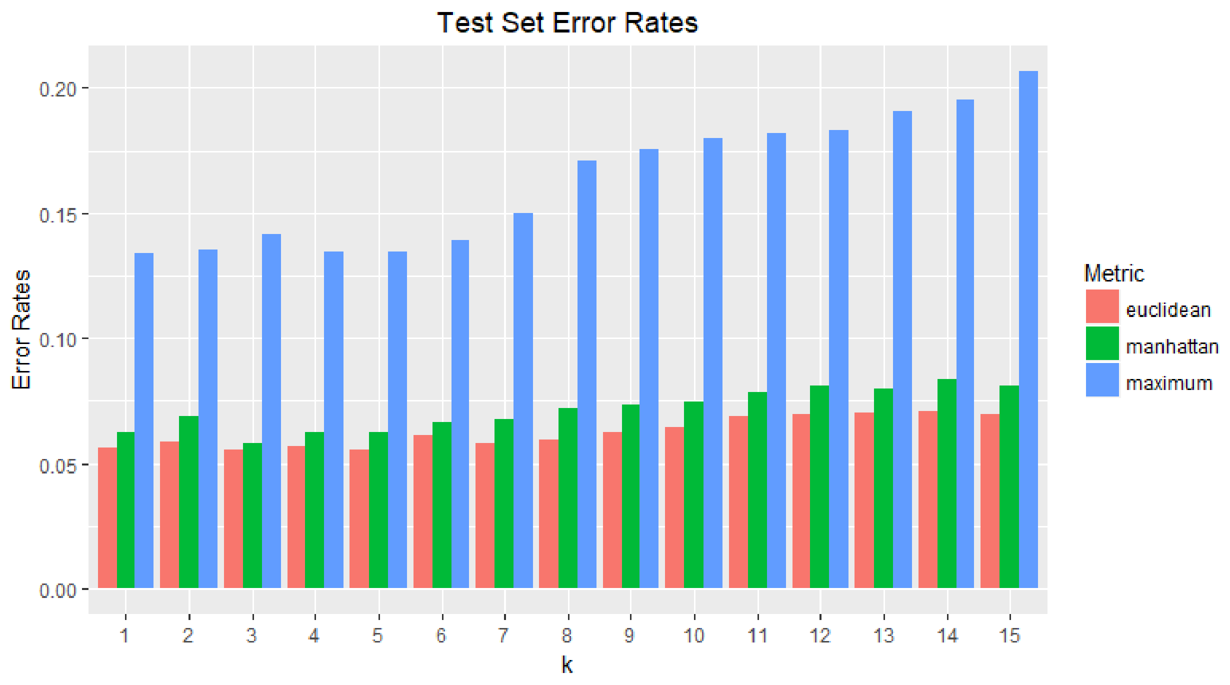
True Class

Predicted Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	8	4	2	10	17	0	4	3
1	0	0	2	1	16	0	2	3	5	0
2	2	1	0	3	3	3	0	1	1	0
3	2	0	5	0	0	12	0	0	17	1
4	0	0	1	1	0	3	1	5	3	9
5	1	0	0	4	0	0	1	1	7	0
6	3	0	1	0	3	3	0	0	3	0
7	0	0	9	2	1	0	0	0	3	17
8	0	0	1	7	0	0	0	0	0	0
9	0	0	1	0	10	1	0	4	3	0

Jennifer Wong
913160121
STA 141A

9. The barplot below shows the test set error rates for k = 1, …, 15 and the distance metrics "euclidean", "manhattan", and "maximum". The "euclidean" distance metric has the lowest error rates and the "maximum" distance metric has the highest error rates. The test set error rates are about .03 higher than the 10-fold cross-validation error rates when the distance metric is "euclidean". This is most likely occurs because the cross-validation error rates were calculated using a lot more data than the test set error rates. More data generally leads to lower error rates. In this case, cross-validation underestimated the true test set error rates. Similar with the cross-validation error rates, as k increases, the error rates tend to increase. Based on this graph, the best combination appears to be k = 5 with distance metric "euclidean".

.



10. With permission from the professor, I worked on this project by myself.

Jennifer Wong
913160121
STA 141A

Citations

1. https://www.datacamp.com/community/tutorials/r-data-import-tutorial#txt

   This citation taught me how to import .txt files in R.

2. https://stackoverflow.com/questions/35441382/change-dimension-of-a-data-frame

   I learned how to change the dimensions of a data frame from this link.

3. https://stackoverflow.com/questions/31882079/r-image-plots-matrix-rotated

   This citation taught me how to reverse the pixels matrix so that it is displayed correctly when using the image() function.

4. https://stackoverflow.com/questions/5638462/r-image-of-a-pixel-matrix

   I learned how to display an image as a grayscale image.

5. https://stackoverflow.com/questions/17721126/simplest-way-to-do-grouped-barplot

   This website taught be how to make a grouped barplot using the library ggplot2.

Jennifer Wong
913160121
STA 141A

Appendix

```
setwd("C:/Users/jenni/Desktop/Final Project")

unzip("digits.zip")

# Problem 1

# This function loads a digits file and converts the columns to appropriate
# datatypes. The argument digits_file is the name of the digits file the
# user wants to load.
read_digits = function(digits_file) {
  data = read.table(digits_file)
  data$V1 = as.factor(data$V1)

  return(data)
}


# Problem 2

# This function displays one observation from the data set as a grayscale
# image. The argument dataset is the data set the user wants to select an
# observation from. The argument observation is the observation number of
# a digit.
view_digit = function(dataset, observation) {
  pixels = dataset[observation, -1]
  pixels = matrix(unlist(t(pixels)), byrow=TRUE, nrow=16, ncol=16)
  pixels = apply(pixels, 2, rev)
  image(t(pixels), col = grey(seq(0, 1, length = 256)), xaxt='n', yaxt='n')
}
```

Jennifer Wong
913160121
STA 141A

```
# Problem 3

# get number of observations for each digit
table(train$V1)
barplot(table(train$V1), ylim=c(0, 1200), xlab = "digit",
        ylab = "# of observations",
        main = "Number of Observations for Each Digit")

# find sum of variance of pixels for each digit
for(digit in levels(train$V1)) {
  data = train[train$V1==digit, -1]
  var = apply(data, 2, var)

  if(digit == "0") {
    variance = sum(var)
  } else {
    variance = cbind(variance, sum(var))
  }
}
colnames(variance) = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9")
barplot(variance, ylim=c(0, 120), xlab = "digit",
        ylab = "sum of variance",
        main = "Sum of Variance of Pixel Values for Each Digit")

# display what each digit looks like on average
for(digit in levels(train$V1)) {
  data = train[train$V1==digit, -1]
  average = cbind(-2, t(colMeans(data)))
  view_digit(average, 1)
}
```

Jennifer Wong
913160121
STA 141A

```
# for each digit, get average of each pixel
for(digit in levels(train$V1)) {
  data = train[train$V1==digit, -1]

  if(digit == "0") {
    average = t(colMeans(data))
  } else {
    average = rbind(average, t(colMeans(data)))
  }
}

# find variance of each pixel to find most and least useful pixels
variance = apply(average, 2, var)
variance = sort(variance, decreasing = TRUE)
head(variance)  # most useful pixels
tail(variance)  # least useful pixels



#Problem 4

# This function uses k-nearest neighbors to predict the label for a point or
# a collection of points. The argument k is the number of nearest neighbors. test is
# a dataframe that contains pixel values of a digit. train is the training dataset.
# metric is the distance metric to use when calculating distance.
predict_knn = function(k, test, train, metric) {
  distances = as.matrix(dist(rbind(test, train[, -1]), method=metric, upper=TRUE,
                            diag=TRUE))
  distances = distances[(nrow(test) + 1):nrow(distances), ]

  predictions = get_predictions(k, nrow(test), train, distances)
  return(predictions)
}

# This function gets the number of counts for each class in neighbors. k is the number
# of nearest neighbors. The argument neighbors contains the points of all the neighbors.
get_counts = function(k, neighbors) {
  counts = numeric(10)

  for(i in 1:k) {
    counts[as.numeric(neighbors[i, 1])] = counts[as.numeric(neighbors[i, 1])] + 1
  }

  return(counts)
}

# This function gets the k-nearest neighbors. k is the number of nearest neighbors.
# train is the training dataset.
get_neighbors = function(k, train) {
  train = train[order(train$distance), ]

  neighbors = train[1:k, ]

  return(neighbors)
}

# This function gets the predictions for k-nearest neighbors. k is the number of nearest
# neighbors. num_test is the number of testing points. train is the training dataset.
# distances is the matrix that contains the distances between every point.
get_predictions = function(k, num_test, train, distances) {
  predictions = numeric(num_test)
  for(i in 1:num_test) {
    train$distance = distances[, i]
    neighbors = get_neighbors(k, train)
    counts = get_counts(k, neighbors)
    predictions[i] = which.max(counts) - 1
  }

  return(predictions)
}



# Problem 5

# This function uses 10-fold cross-validation to estimate the error rate for k-nearest
# neighbors. k is the number of nearest neighbors. data is the dataset to use cross-
# validation for. metric is the distance metric to use when calculating distance.
cv_error_knn = function(k, data, metric) {
  set.seed(23)
  data = data[sample(nrow(data), replace=FALSE), ]
  folds = as.data.frame(cbind(as.numeric(rownames(data)), rep_len(1:10, nrow(data))))
  distances = as.matrix(dist(data[, -1], method=metric, upper=TRUE, diag=TRUE))

  return(do_cv(k, data, folds, distances))
}

# This function does 10-fold cross-validation. k is the number of nearest neighbors.
# data is the dataset to use cross-validation for. folds is the dataframe that contains
# the assignment of indices to folds. distances is the matrix that holds the distances
# between every point.
do_cv = function(k, data, folds, distances) {
  error_rates = numeric(10)

  for(i in 1:10) {
    test = data[as.numeric(rownames(data)) %in% folds[folds$V2==i, 1], ]
    train = data[!(as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]), ]

    dist = distances[, as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]]
    dist = dist[!(as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]), ]

    predictions = get_predictions(k, nrow(test), train, dist)

    error_rates[i] = get_error_rate(test, predictions)
  }

  return(mean(error_rates))
}

# This function gets the error rate. test is the testing dataset. predictions contains
# the predictions of the testing dataset.
get_error_rate = function(test, predictions) {
  wrong_counter = 0

  for(i in 1:length(predictions)) {
    if(predictions[i] != (as.numeric(test[i, 1]) - 1)) {
      wrong_counter = wrong_counter + 1
    }
  }

  error_rate = wrong_counter / nrow(test)

  return(error_rate)
```

```
}



# Problem 6


# make dataframe to store CV error rates
k_metric = as.data.frame(matrix(nrow = 45, ncol = 3))
colnames(k_metric) = (c("Metric", "k", "error_rate"))
k_metric$Metric = rep(c("euclidean", "manhattan", "maximum"), each=15)
k_metric$k = rep(1:15)


# get CV error rates for different distance metrics and k
row_num = 1
set.seed(23)
data = train[sample(nrow(train), replace=FALSE), ]
folds = as.data.frame(cbind(as.numeric(rownames(data)), rep_len(1:10, nrow(data))))
for(metric in c("euclidean", "manhattan", "maximum")) {
  distances = as.matrix(dist(data[, -1], method=metric, upper=TRUE, diag=TRUE))


  for(k in rep(1:15)) {
    k_metric1[row_num, 3] = do_cv(k, data, folds, distances)
    row_num = row_num + 1
  }
}


# plot CV error rates
library(ggplot2)
library(viridis)
ggplot(k_metric1, aes(factor(k), error_rate, fill = Metric)) +
  geom_bar(stat="identity", position = "dodge") +
  scale_color_viridis() +
  labs(x="k", y="Error Rates") +
  ggtitle("10-fold CV Error Rates") +
  theme(plot.title = element_text(hjust = 0.5)) +
  ylim(0, .205)


# Problem 7

# get confusion matrices for each of the 3 best k and distance metric combinations
confusion_matrices = vector("list", 3)
set.seed(23)
data = train[sample(nrow(train), replace=FALSE), ]
folds = as.data.frame(cbind(as.numeric(rownames(data)), rep_len(1:10, nrow(data))))
distances = as.matrix(dist(data[, -1], method="euclidean", upper=TRUE, diag=TRUE))
counter = 1

for(k in c(1, 3, 4)) {
  error_rates = numeric(10)

  for(i in 1:10) {
    test = data[as.numeric(rownames(data)) %in% folds[folds$V2==i, 1], ]
    train = data[!(as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]), ]

    dist = distances[, as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]]
    dist = dist[!(as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]), ]

    predictions = get_predictions(k, nrow(test), train, dist)

    if(i == 1) {
      confusion_matrix = table(as.factor(predictions), test[ , 1]) / nrow(test)
    } else {
      confusion_matrix = confusion_matrix +
        (table(as.factor(predictions), test[ , 1]) / nrow(test))
    }
  }

  confusion_matrices[[counter]] = confusion_matrix / 10
  counter = counter + 1
}



#Problem 8

# get misclassified digits using 10-fold cross-validation
misclassified = as.data.frame(matrix(nrow = 500, ncol = 2))
colnames(misclassified) = (c("actual", "predicted"))
distances = as.matrix(dist(data[, -1], method="euclidean", upper=TRUE, diag=TRUE))
row_num = 1

for(i in 1:10) {
  test = data[as.numeric(rownames(data)) %in% folds[folds$V2==i, 1], ]
  train = data[!(as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]), ]

  dist = distances[, as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]]
  dist = dist[!(as.numeric(rownames(data)) %in% folds[folds$V2==i, 1]), ]

  predictions = get_predictions(k, nrow(test), train, dist)

  for(j in 1:length(predictions)) {
    if(predictions[j] != (as.numeric(test[j, 1]) - 1)) {
      misclassified[row_num, 1] = as.numeric(test[j, 1]) - 1
      misclassified[row_num, 2] = predictions[j]
      row_num = row_num + 1
    }
  }
}

# get confusion matrix for misclassified data
table(misclassified$predicted, misclassified$actual)



# Problem 9

# make dataframe to store CV error rates
test_errors = as.data.frame(matrix(nrow = 45, ncol = 3))
colnames(test_errors) = (c("Metric", "k", "error_rate"))
test_errors$Metric = rep(c("euclidean", "manhattan", "maximum"), each=15)
test_errors$k = rep(1:15)

# get test error rates for different k and distance metrics
row_num = 1
for(metric in c("euclidean", "manhattan", "maximum")) {
  distances = as.matrix(dist(rbind(test[, -1], train[, -1]), method=metric, upper=TRUE,
                             diag=TRUE))
  distances = distances[(nrow(test) + 1):nrow(distances), ]#change to columns?

  for(k in rep(1:15)) {
    predictions = get_predictions(k, nrow(test), train, distances)
    test_errors[row_num, 3] = get_error_rate(test, predictions)
    row_num = row_num + 1
  }
}
```

Jennifer Wong
913160121
STA 141A

```
# plot test error rates
ggplot(test_errors, aes(factor(k), error_rate, fill = Metric)) +
  geom_bar(stat="identity", position = "dodge") +
  scale_color_viridis() +
  labs(x="k", y="Error Rates") +
  ggtitle("Test Set Error Rates") +
  theme(plot.title = element_text(hjust = 0.5))
```