Final Project

Jared Yu

June 14, 2018

```
library(grid); library(broman); library(ggplot2); library(reshape2) # Load libraries
setwd("C:/Users/qizhe/Desktop/STA 141A/final") # Set working directory
```

1

```
list_to_matrix = function(bin_list) {
  bin_list = as.character(bin_list)
  bin_matrix = matrix(bin_list, nrow = 10000, byrow = TRUE)
  return(bin_matrix)
} # Converts a list to a matrix

load_training_images = function(input_dir, output_file) {
  bin_data = list.files(path = input_dir, # Extract binary files
                        pattern = "data_batch_[0-9].bin", full.names = TRUE)
  bin_list = lapply(bin_data, function(x) readBin(con = x, # Read the binary data
                                                  what = "raw", n = 3073*10000))
  bin_matrices = lapply(bin_list, list_to_matrix) # Change vectors to matrices in lis
t
  train = do.call("rbind", bin_matrices) # Combine the list into a single data
  save(train, file = output_file) # Save to file
} # Loads the binary data and converts them to workable data (training)

load_testing_images = function(input_dir, output_file) {
  bin_data = list.files(path = input_dir, # Extract binary file
                        pattern = "test_batch.bin", full.names = TRUE)
  bin_vec = readBin(con = bin_data, what = "raw", n = 3073*10000) # Read the binary da
ta
  bin_vec = as.character(bin_vec) # Set to character
  test = matrix(bin_vec, nrow = 10000, byrow = TRUE) # Change to matrix
  save(test, file = output_file) # Save to file
} # Loads the binary data and converts them to workable data (test)

training_images = load_training_images(input_dir = "C:/Users/qizhe/Desktop/STA 141A/fi
nal",
                                       output_file = "C:/Users/qizhe/Desktop/STA 141A/
final/training_set.rds")

testing_images = load_testing_images(input_dir = "C:/Users/qizhe/Desktop/STA 141A/fina
l",
                                     output_file = "C:/Users/qizhe/Desktop/STA 141A/fi
nal/test_set.rds")

load("C:/Users/qizhe/Desktop/STA 141A/final/training_set.rds") # Load the saved data
load("C:/Users/qizhe/Desktop/STA 141A/final/test_set.rds")

data_rescale <- function(labels, k = 500) {
  sort(as.vector(sapply(unique(labels),
                        function(i) which(labels == i))[1:k, ]))
} # TA code for rescaling

train2 <- train[data_rescale(train[,1], k = 500),] # Rescale the data
test2 <- test[data_rescale(test[,1], k = 100),]
```

2

```
view_images = function(image_data, observation, image_labels) {
  photo_label = as.numeric(image_data[observation,1]) # Get the integer of the observa
tion label
  photo_label = image_labels[photo_label + 1,] # Get the corresponding name of the lab
el

  red = sapply(image_data[observation, 2:1025], hex2dec) # Change colors to decimal
  green = sapply(image_data[observation, 1026:2049], hex2dec)
  blue = sapply(image_data[observation, 2050:3073], hex2dec)

  rgb_mat = matrix(rgb(red, green, blue, # Create matrix of colors
                       maxColorValue = 255), nrow = 32, ncol = 32, byrow = T)
  grid.raster(rgb_mat) # Plot image
  return(photo_label)
} # Display the image of a particular data point

image_labels = read.delim("batches.meta.txt", header = FALSE) # Receive classes of ima
ges
```

3

```
# Find an example of each class within the data
view_images(train2, 105, image_labels) # 1. Frog
```

```
## [1] frog
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 31, image_labels) # 2. Airplane
```

```
## [1] airplane
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 9, image_labels) # 3. Ship
```

```
## [1] ship
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 44, image_labels) # 4. Horse
```

```
## [1] horse
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 10, image_labels) # 5. Cat
```

```
## [1] cat
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 499, image_labels) # 6. Automobile
```

```
## [1] automobile
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 109, image_labels) # 7. Bird
```

```
## [1] bird
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 300, image_labels) # 8. Deer
```

```
## [1] deer
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 358, image_labels) # 9. Truck
```

```
## [1] truck
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
view_images(train2, 785, image_labels) # 10. Dog
```

```
## [1] dog
## Levels: airplane automobile bird cat deer dog frog horse ship truck
```

```
color_range = function(color) {
  if (color == "red") {
    color = 2:1025
  } else if (color == "green") {
    color = 1026:2049
  } else if (color == "blue") {
    color = 2050:3073
  }
  return(color)
} # Determine the range for a specific color

class_index = function(image_labels, class_name) {
  label_index = (which(image_labels == class_name) - 1)
  return(paste0("0",label_index))
} # Return the index of the class

pixel_variance = function(color, image_data) {
  color_subset = color_range(color = color) # Determine the range for given color
  colored_data = image_data[1:nrow(image_data), color_subset] # Subset the color from
the data
  colored_data_mat = matrix(hex2dec(colored_data), # Transform data into matrix
                            nrow = nrow(colored_data), ncol = ncol(colored_data), byro
w = TRUE)
  colored_data_var = apply(colored_data_mat, 2, var)
  special_pixel = which(colored_data_var == max(colored_data_var))
  unspecial_pixel = which(colored_data_var == min(colored_data_var))
  pixels = c(special_pixel, unspecial_pixel)
  print("Special, then unspecial pixels:")
  return(pixels)
} # Find the pixel with greatest variance from a specific color

# Find variance for RGB within test and train data
pixel_variance(color = "red", image_data = test2)
```

```
## [1] "Special, then unspecial pixels:"
```

```
## [1] 719 416
```

```
pixel_variance(color = "green", image_data = test2)
```

```
## [1] "Special, then unspecial pixels:"
```

```
## [1] 450 416
```

```
pixel_variance(color = "blue", image_data = test2)
```

```
## [1] "Special, then unspecial pixels:"
```

```
## [1] 522 416
```

```
pixel_variance(color = "red", image_data = train2)
```

```
## [1] "Special, then unspecial pixels:"
```

```
## [1] 958 111
```

```
pixel_variance(color = "green", image_data = train2)
```

```
## [1] "Special, then unspecial pixels:"
```

```
## [1] 753 111
```

```
pixel_variance(color = "blue", image_data = train2)
```

```
## [1] "Special, then unspecial pixels:"
```

```
## [1] 948  79
```

4

```
library(broman)
top_k = function(dist_mat, k, test_data, training_data) {
  top_k_mat = label_indices = matrix(NA, nrow = nrow(dist_mat),
                                     ncol = k) # Create dummy matrix for ordered label
s

  for (i in 1:nrow(dist_mat)) {
    top_5 = names(head(sort(dist_mat[i,], # Retrieve the top k training observations t
hat match the test
                            decreasing = FALSE), k))
    top_5_index = as.integer(top_5) - nrow(test_data) # Convert to format of training
indices

    top_k_mat[i,] = top_5_index # Fill the rows of the dummy matrix
  } # Fill a dummy matrix with the top k training observations

  for (i in 1:nrow(label_indices)) {
    label_indices[i,] = training_data[top_k_mat[i,], 1]
  } # Determine the label of each of the top k training observations

  label_indices2 = as.integer(label_indices) + 1 # Convert to the image_labels indices
  label_indices2 = matrix(label_indices2, # Convert back to matrix
                          nrow = nrow(label_indices), ncol = ncol(label_indices))

  return(label_indices2)
} # Find the top k labels per observation from the distance matrix

vote_k = function(test_data, training_data, k_mat) {
  vote_label = rep(NA, nrow(k_mat)) # Dummy vector for vote labels

  for (i in 1:nrow(k_mat)) {
    vote_label[i] = sample(names(which(table(k_mat[i,]) == # Vote for the test label
                                         sort(table(k_mat[i,]), decreasing = TRUE)
[1])), 1)
  }
  vote_label = as.integer(vote_label) # Convert back to integer

  return(vote_label)
} # Vote for the test label, and determine the accuracy

predict_knn = function(test_data, training_data, distance_metric, k) {
  test_data2 = test_data[,-1] # Remove the label column from both test and training da
ta
  training_data2 = training_data[,-1]

  test_train_mat = rbind(test_data2, training_data2) # Combine both into a single matr
ix
  test_train_mat2 = matrix(hex2dec(test_train_mat), # Convert to integer pixels, and k
```

```
eep as matrix
                            nrow = (nrow(test_data2) + nrow(training_data2)), ncol = nc
ol(test_data2))

  dist_mat = as.matrix(dist(test_train_mat2, method = distance_metric)) # Take the dis
tance matrix of the combined matrix
  dist_mat2 = dist_mat[1:nrow(test_data2), # Subset the informative parts of the dista
nce matrix
                       (nrow(test_data2) + 1):(nrow(test_data2) + nrow(training_data
2))]

  k_mat = top_k(dist_mat = dist_mat2, k = k, # Discern the k nearest labels
                test_data = test_data, training_data = training_data)

  k_votes = vote_k(test_data = test_data, # Vote for the test label
                   training_data = training_data, k_mat = k_mat)

  return(k_votes)
}

predict_knn(test_data = test2, training_data = train2, distance_metric = "euclidean",
k = 3)
```

```
##   [1]  5  9  9  1  5  5  7  3  9  9  3  9  5  8  9  3  3  5  2  5  3  1  1
##  [24]  5  5  5  7  8  8  5  7  3  5  7  9  5  5  9  3  6  9  7  8  7  1  1
##  [47]  6  1  5  3  9  9  9  4  9  9  6  7  1  3  5  6  7  6  5  3  9  1  4
##  [70] 10  3  7  9  9  1  3  5  4  5  9  9  4  7  9  3  5  4  9  9  9  1  3
##  [93]  9  2  5  5  5  1  1  5  5  5  4  5  2  2  7  7  6  6  3  1  7  5  5
## [116]  5  1  5  5  5  9  5  3  7  9  5  1  4  8  3  3  2  1  6  6  3 10  1
## [139]  3  9  3  3  5  6  9  3  3  6  7  9  1  8  4  1  5  5  1  9  6  5  1
## [162]  5  4  3 10  5  9  3  6  1  9  8  3  9  5 10  1  5  7  1  4  5  3  5
## [185]  7  8  7  3  1  9  9  5  3  7  7  5  9  1  4  6  7  1  9  3  4  3  1
## [208]  3  3  1  7  9  3  3  9  3  1  9  1  5  7  5  8  1  4  5  5  3  7  5
## [231]  4  2  5  9  1  1  5 10  1  7  1  1  3  1  3  7  7  1  5  3  7  7  6
## [254]  8  5  9  5  1  8  5  9  9  4  1  5  9  3  5  8  3  3  3  7  4  9  4
## [277]  9  3  6  1  8  5  4  2  8  5  6  1  9  5  9  3  3  9  7  5  3  1  9
## [300]  5  5  5  5  3  5 10  1  5  6  3  9  6  1  1  4  6  3  3  4  5  6  4
## [323]  7  5  7  3  5  6  5  7  2  8  5  5  5  3  7  9  1  5  5  7  1  1  9
## [346]  6  5  3  9  5 10  5  9  3  5  6  3  5  9 10  3  6  9  1  1  9  5  5
## [369]  1  2  7  3  5  9  5  8  9  3  3  3  5  9  3  1  3  1  9  5  3  7  1
## [392]  3  3  7  4  5  3  7  1  9 10  5  5  9 10  7  3  5  1  3  5  9  3  5
## [415] 10  9  1  8  1  5  7  5  9  7  3  9  6  5  3  7  3  1  6  5  5  7  9
## [438]  5  3  4  9  5  2  9  3  3  1  1  3  1  7  9  4  3 10  5  3  3  4  1
## [461]  5  5  3  6  3  3  8  3  6  1  3  5  1  9  3  5  5  2  7  5  9  3  1
## [484]  5  7  9  2  3  1  1  9  3  9  2  9  9  5  1  3  1  3  5  3  3 10  1
## [507]  7  5  6  4  7  5  7  9  5  1  9  9  9  9  5  1  1  9  1  3  3  1  1
## [530]  7 10  6  3  9  5  3  5  5  1  6  9 10  7  7  6  6  2  3  3  4  5  3
## [553]  8  7  3  9  3  4  3  3  9  3  7  5  5  6  9  4  1  7  7  8  5  7  7
## [576]  5  9  9  3  9  3  8  7  5  3  6  5  6  1  9  4  9  3  3  9  5  3  5
## [599]  7  6  6  3  2  5  5  7  9  5  7 10  5  9  9  9  7  5  8  6  8  4  5
## [622]  5  7  3  8  5  6  5  3  7  5  8  6  9  5  3  1  4  9  3  3  7  1  5
## [645]  3  9  4  3  5  3  6 10  6  5  1  3  9  2  5  3 10  5  8  7  5  9  6
## [668]  1  1  3  5  4  5  7  5  5  5  7  1  6  3  4  2  6  1  5  5  6  5  5
## [691]  5  3  9  5  9  9  9  5  8  5  8  1  1  7  5  5  9  3  9  9  6  9  9
## [714]  8  3  5  3  5  5  9  2  9  1  9  9  1  4  5  5  7  5  1  5  7  6  3
## [737]  4  1  6  6  5  6  8  8  5  5  3  6  1  4  9  3  2  3  3  5  3  9  1
## [760]  7  6  3  3  9  3  3  5  9  5  1  7  3  3  5  3  5  5  5  5  7  5  7
## [783]  5  5  5  1  4  5  5  1  1  3  7  6  3  7  5  7  4  5  8  9  3  4  5
## [806]  7  9  5  3  3  6  7  5  5  2  2  7  7  5  3 10  3  5  9  5  7  3  6
## [829]  7  1  1  7  9  5  5  5  9  6  4  3  8  7  5  5  7  2  5  9  5  5  4
## [852]  9  9  9  1  5  5  1  4  1  5  5  4  1  1  9  5  4  5  1  6  5  5  3
## [875]  4  1  3  3  9  5  9  7  3  1  1  3  9  3  9  5  5  3  7  1  9  7  9
## [898]  6  8  5  3  7  1  1  9  9  5  5  4  5  3  4  9  3  9  3  5  2  7  3
## [921]  6  7  9  3  1  3  6  3 10  8  9  3  8  4  3  3  5  3  4  9  5  5  5
## [944]  3  1  5  5  3  5  5  5  7  3  7  5  7  5  7  1  1  1  3  7  8 10  4
## [967]  4  1  7  1  3  3  3 10  5  5  1  5  4  9  1  7  7  5  5  5  1  5  5
## [990]  3  5  9  5  3  5  9  9  5  7  3
```

5

```
### Create the entire distance matrix once for Euclidean and Manhattan distances
test2c = test2[,-1]; test2d = apply(test2c, 2, function(x) strtoi(x, 16L)) # Remove la
bels, set to integer
train2c = train2[,-1]; train2d = apply(train2c, 2, function(x) strtoi(x, 16L)) # Same
for train data
dist_euclidean = dist(rbind(test2d, train2d)); dist_euc = as.matrix(dist_euclidean) #
Create distance matrix
save(dist_euc, file = "dist_euclidean") # Save file
dist_manhattan = dist(rbind(test2d, train2d), method = "manhattan"); dist_man = as.mat
rix(dist_manhattan)
save(dist_man, file = "dist_manhattan") # Same for Manhattan distances

# read in dist_euc
# load("dist_euclidean.rda"); load("dist_manhattan.rda"); load("test_set.rds"); load
("training_set.rds")

cv_error_knn = function(train2, test2, k = 3, numOfFolds = 10, all_distance){
  n = nrow(train2)
  m = nrow(test2)
  real_labels = train2[,1] # Retrieve labels
  all_distance = as.matrix(all_distance) # Change to distance matrix
  all_distance = all_distance[-c(1:m), -c(1:m)] # Subset the correct data
  colnames(all_distance) = 1:nrow(all_distance)
  row.names(all_distance) = 1:nrow(all_distance)
  classes = sample(rep(1:10,500)) # Generate list of classes
  indexes = split(1:n, classes) # indexes[[1]] show the index of the images from train
ing set goes to first fold
  fold_distance = lapply(1:numOfFolds, function(x) all_distance[do.call("c", indexes
[-x]), indexes[[x]]]) # calculate distance of one fold vs the other folds (9 folds)
  top_ks = lapply(fold_distance, function(x) apply(x, 2, function(y) real_labels[as.nu
meric(names(sort(y))[1:k])])) # select the top k
  if (k == 1) {
    top_classes = lapply(top_ks, function(x) names(x)[1])
  } else {
    top_classes = lapply(top_ks, function(x) apply(x, 2, function(y) names(sort(table
(y), decreasing=TRUE))[1]))
  }
  return(list(true = as.numeric(real_labels[do.call("c", indexes)]), predict = as.nume
ric(do.call("c", top_classes))))
}
```

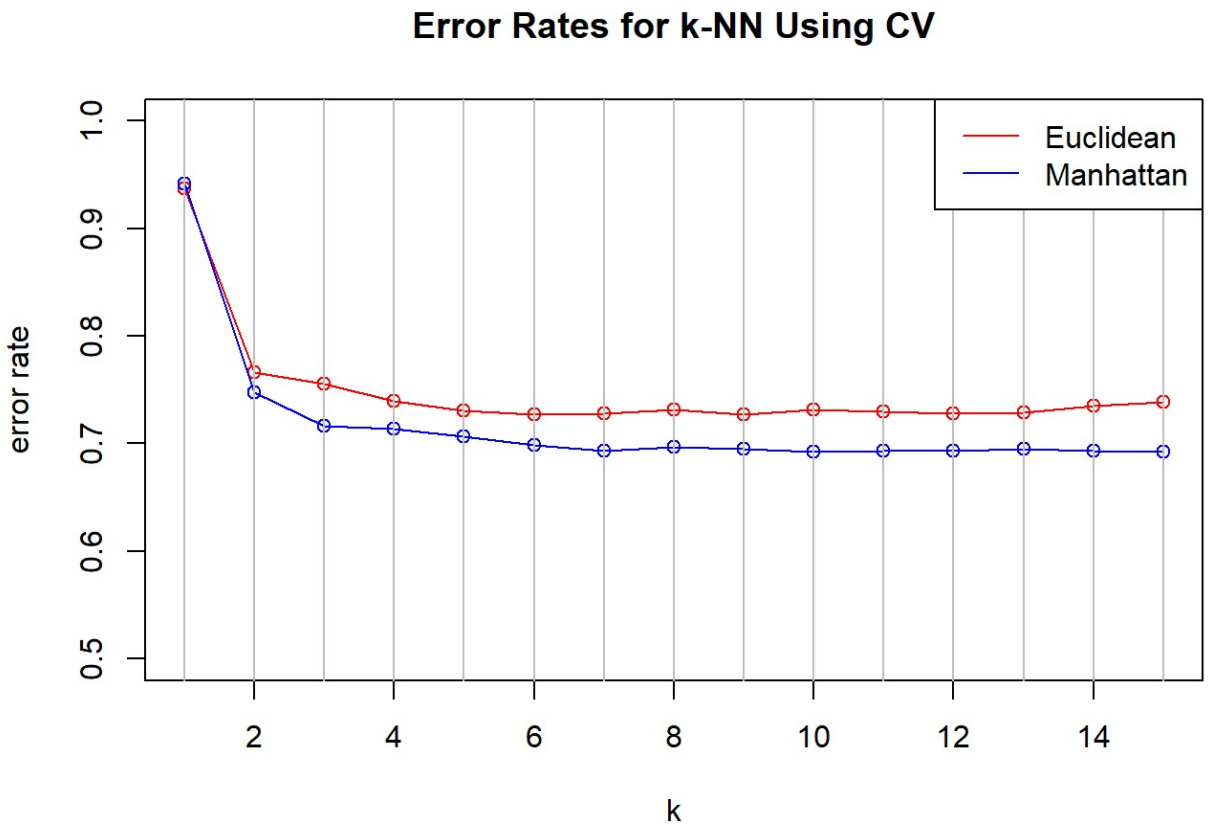
6

```
err.euc = err.man = rep(0, 15) # Create empty vector for errors
err.euc.true = err.man.true = err.euc.predict = err.man.predict = vector("list", 15)
# Empty list for pred/true

set.seed(456) # Set seed
for (i in c(1:15)) {
  out_euc = cv_error_knn(train2, test2, k = i, numOfFolds = 10, all_distance = dist_eu
c)
  err.euc[i] = mean(out_euc$true != out_euc$predict)
  err.euc.true[[i]] = out_euc$true
  err.euc.predict[[i]] = out_euc$predict

  out_man = cv_error_knn(train2, test2, k = i, numOfFolds = 10, all_distance = dist_ma
n)
  err.man[i] = mean(out_man$true != out_man$predict)
  err.man.true[[i]] = out_man$true
  err.man.predict[[i]] = out_man$predict
} # Produce output for k = 1,...,15 and 10 folds for Euclidean and Manhattan distances

plot(1:15, err.euc, main = "Error Rates for k-NN Using CV", # Plot the error rates
     xlab= "k", ylab= "error rate", ylim=c(0.5, 1), col="red", type = "l")
points(1:15, err.euc, col = "red"); lines(1:15, err.man, col="blue")
points(1:15, err.man, col = "blue"); abline(v=c(1:15), col="grey")
legend("topright", c("Euclidean", "Manhattan"), col = c('red', 'blue'), lty=c(1,1))
```



7

```
# Determine the top 3 for Euclidean and Manhattan distances
order(err.euc)[1:3] # 9, 6, 7
```

```
## [1] 9 6 7
```

```
order(err.man)[1:3] # 10, 15, 7
```

```
## [1] 10 15  7
```

```
# Euclidean Confusion Matrix
table(data.frame(true = err.euc.true[[9]], predict = err.euc.predict[[9]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 244   0  70   8  29   2  23   5 118   1
##    1  74  40  67  18 112  13  44  10 107  15
##    2  70   1 211  17 143   4  24   4  24   2
##    3  43   4 140  63 124  43  46   7  28   2
##    4  30   0 142  12 246   4  36  10  19   1
##    5  44   2 117  53 136  74  48   7  18   1
##    6  24   1 154  20 154  15 126   2   4   0
##    7  57   4 121  16 173  16  26  55  29   3
##    8 121   3  34   7  42   9   6   4 267   7
##    9  62  19  73  20  77  11  31  13 155  39
```

```
table(data.frame(true = err.euc.true[[6]], predict = err.euc.predict[[6]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 260   0  71   7  31   3  22   5 101   0
##    1  73  44  66  22 123  12  35   6 102  17
##    2  76   2 216  24 128   4  22   3  25   0
##    3  53   2 129  81 125  41  40   7  21   1
##    4  35   1 138  20 235   7  38   9  15   2
##    5  41   2 123  68 125  73  43   6  17   2
##    6  26   3 166  23 152   8 116   3   2   1
##    7  65   1 124  22 153  19  31  54  27   4
##    8 145   3  30  13  42  10   7   9 238   3
##    9  87  21  68  18  70  11  26  15 138  46
```

```
table(data.frame(true = err.euc.true[[7]], predict = err.euc.predict[[7]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 248   0  73   5  32   2  21   5 114   0
##    1  79  40  67  24 121  10  36   4  99  20
##    2  63   2 217  24 133   6  28   4  22   1
##    3  48   1 139  62 125  40  51   8  26   0
##    4  38   1 134  14 243   3  37  10  19   1
##    5  32   2 124  58 127  81  47   7  21   1
##    6  18   2 153  30 159   6 126   3   2   1
##    7  60   2 123  23 154  19  31  56  26   6
##    8 138   4  31  14  38  10   6   6 245   8
##    9  87  25  64  21  74   9  29  11 139  41
```

```
# Manhattan Confusion Matrix
table(data.frame(true = err.man.true[[10]], predict = err.man.predict[[10]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 253   3  63   3  28   1  18   6 123   2
##    1  56  78  60  22 100  11  46  10  95  22
##    2  63   1 229  25 114   8  26   9  24   1
##    3  45   5 113  72 119  40  65  13  24   4
##    4  46   3 134  13 235   3  32  14  17   3
##    5  42   2 110  65 116  83  48   5  27   2
##    6  25   2 151  26 140   7 142   2   4   1
##    7  54   2  99  20 159  14  30  86  26  10
##    8 118   6  29   6  38   8   8   3 275   9
##    9  72  38  49  29  45   7  22  32 119  87
```

```
table(data.frame(true = err.man.true[[15]], predict = err.man.predict[[15]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 259   1  54   7  28   2  19   6 121   3
##    1  43  59  66  21 115   8  44  10 100  34
##    2  64   0 213  20 122   7  30   9  33   2
##    3  43   4 121  68 112  34  75  13  25   5
##    4  37   3 140  11 238   5  34  10  19   3
##    5  34   5 113  51 125  83  51   7  28   3
##    6  23   2 154  23 138  11 138   4   6   1
##    7  54   0 112  19 163  12  18  89  23  10
##    8 103   5  24   6  39   8   5   5 298   7
##    9  53  31  57  16  44   5  30  31 142  91
```

```
table(data.frame(true = err.man.true[[7]], predict = err.man.predict[[7]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 259   2  73   8  26   2   8   7 113   2
##    1  65  86  52  30  98  11  38   9  86  25
##    2  62   2 222  23 126   7  23   7  23   5
##    3  54   9 126  80 111  43  49  10  15   3
##    4  48   3 141   8 236   3  34  14  12   1
##    5  47   1 106  65 112  92  39  12  24   2
##    6  29   5 167  35 120   7 131   2   4   0
##    7  64   6 102  25 139  12  28  88  28   8
##    8 125   9  37   4  31  11   4   8 263   8
##    9  75  40  53  28  48   7  27  25 119  78
```

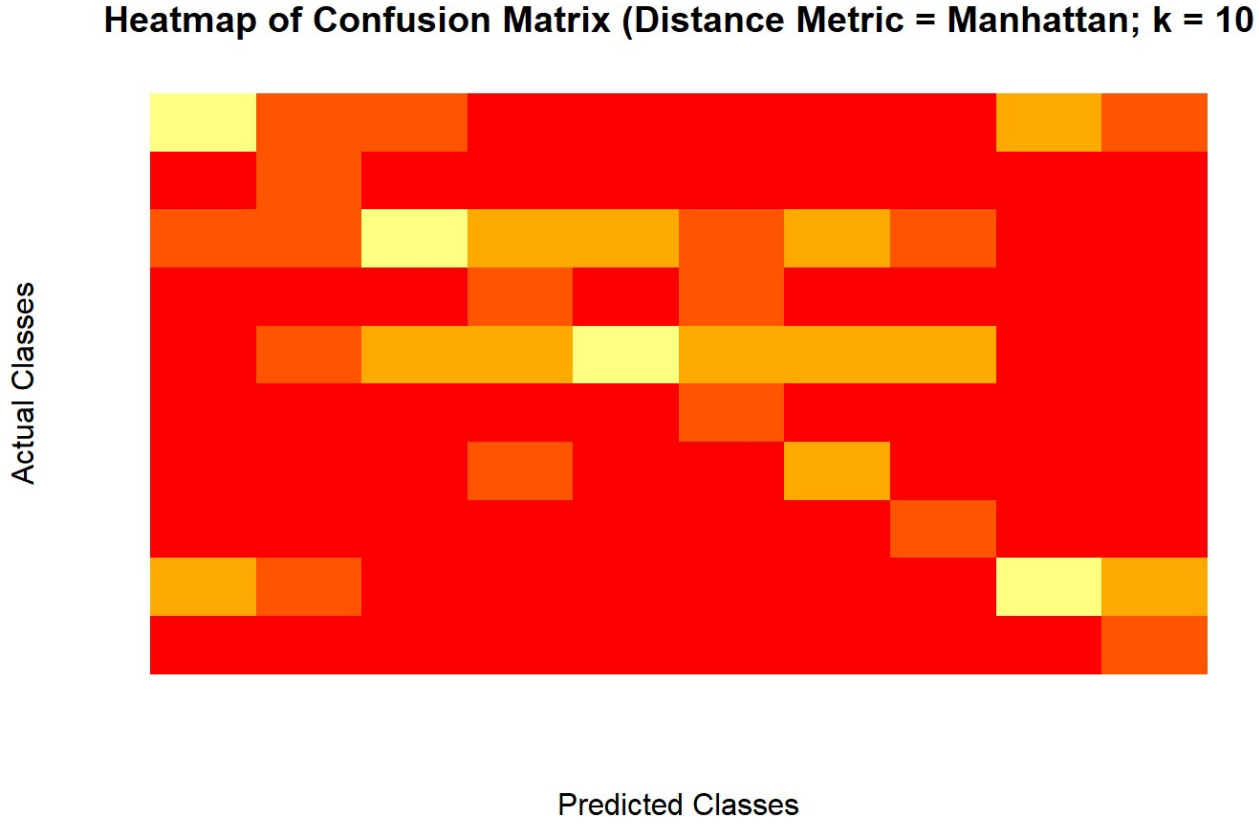
8

```
table(data.frame(true = err.man.true[[10]], predict = err.man.predict[[10]]))
```

```
##     predict
## true   0   1   2   3   4   5   6   7   8   9
##    0 253   3  63   3  28   1  18   6 123   2
##    1  56  78  60  22 100  11  46  10  95  22
##    2  63   1 229  25 114   8  26   9  24   1
##    3  45   5 113  72 119  40  65  13  24   4
##    4  46   3 134  13 235   3  32  14  17   3
##    5  42   2 110  65 116  83  48   5  27   2
##    6  25   2 151  26 140   7 142   2   4   1
##    7  54   2  99  20 159  14  30  86  26  10
##    8 118   6  29   6  38   8   8   3 275   9
##    9  72  38  49  29  45   7  22  32 119  87
```

```
# Heatmap
bestcombo.tab = table(data.frame(true = err.man.true[[10]], predict = err.man.predict
[[10]]))
bestcombo.df = matrix(bestcombo.tab, ncol = 10)

image(bestcombo.df[, ncol(bestcombo.df):1],
      xlab = "Predicted Classes", ylab = "Actual Classes",
      main = "Heatmap of Confusion Matrix (Distance Metric = Manhattan; k = 10",
      axes = F, col = heat.colors(5))
```


Heatmap of Confusion Matrix (Distance Metric = Manhattan; k = 10
Actual Classes
Predicted Classes

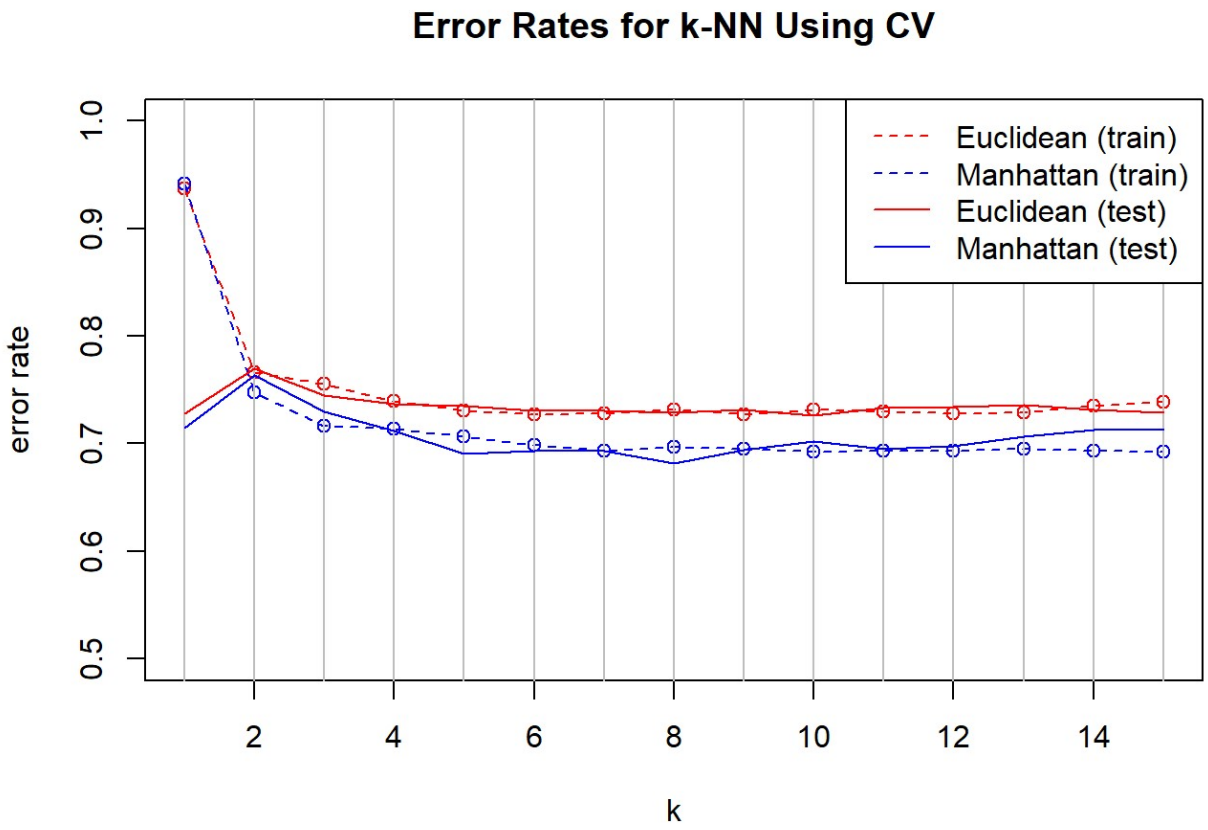
9

```
test_error_knn = function(train2, test2, k = 10, all_distance){
  n = nrow(train2) # Get rows for test and train data
  m = nrow(test2)
  real_labels = train2[,1] # Subset true labels from train
  all_distance = as.matrix(all_distance) # Convert to distance matrix
  all_distance = all_distance[-c(1:m), c(1:m)] # Subset the test x train data
  colnames(all_distance) = 1:m # Correct column and row names
  row.names(all_distance) = 1:nrow(all_distance)
  top_k = apply(all_distance, 2, function(y) real_labels[as.numeric(names(sort(y))[1:
k])]) # Find top k
  if (k == 1) {
    predict_labels = as.numeric(top_k)
  } else {
    predict_labels = apply(top_k, 2, function(x) as.numeric(names(sort(table(x), decre
asing=TRUE))[1]))
  }
  return(list(true = as.numeric(test2[,1]), predict = predict_labels))
} # Find the knn error rate for the test data

euc_test_err = rep(0, 15); man_test_err = rep(0, 15) # Create empty vectors
set.seed(141)
for (i in c(1:15)) {
  predict1 = test_error_knn(train2, test2, k = i, dist_euc)
  euc_test_err[i] = mean(predict1$true != predict1$predict)

  predict1 = test_error_knn(train2, test2, k = i, dist_man)
  man_test_err[i] = mean(predict1$true != predict1$predict)
} # Retrieve the error rates for the test data

plot(1:15, err.euc, main = "Error Rates for k-NN Using CV", # Plot the error rates
     xlab= "k", ylab= "error rate", ylim=c(0.5, 1), col="red", type = "l", lty = 2)
points(1:15, err.euc, col = "red"); lines(1:15, err.man, col="blue", lty = 2)
points(1:15, err.man, col = "blue"); abline(v=c(1:15), col="grey")
lines(1:15, euc_test_err, col = "red"); lines(1:15, man_test_err, col = "blue")
legend("topright", c("Euclidean (train)", "Manhattan (train)", "Euclidean (test)", "Ma
nhattan (test)"),
       col = c('red', 'blue', 'red', 'blue'), lty=c(2,2,1,1))
```




```
order(euc_test_err)[1:3] # 10, 1, 8
```

```
## [1] 10  1  8
```

```
order(man_test_err)[1:3] # 8, 5, 6
```

```
## [1] 8 5 6
```