STA141 Assignment 3
Zhen Zhang

```r
library(ggplot2)

digit_image <- read.csv("~/Desktop/digitsTrain.csv")
set.seed(2)
digit_image <- digit_image[sample(1:5000),]
metrics <- c('euclidean', 'manhattan')

create_dist_matrix <- function(metric, data = digit_image, y_identifier = 1) {
  # knowing the y idetifier if possible, and then calculate the distance result
  dist_result <- dist(data[,-1], method = metric)
  # initialize the dist matrix
  dist_matrix <- as.matrix(dist_result)
}

knn_compute <- function(position, k, dist_matrix, label_real, cv_flag = T) {
  # k is the number of knn number
  # This is the function that calculates the labels output for one part of
  # cross-validation.

  # get the row number of distance matrix
  data_N <- dim(dist_matrix)[1]
  # This is a little trick. Since the indices we extract will be the index in this
  # particular distance matrix, but this matrix's columns are lack of a group of
  # column numbers. So it is necessary to transform the indices for the smallest
  # value to the normal indices. For example, if in the fourth group, 3001:4000
  # are missing, then if a index is 2899, do nothing to it, but if a index is 3445,
  # we need to add 1000, which is the size, data_N. the position_expansion is the
  # threshold to justify whether it should be transformed. Any index that is larger
  # than this variable must be tranformed.
  if (cv_flag) position_expansion <- (position - 1) * data_N
  else position_expansion <- position

  # By calling break_tie function, the prediction value will be presented. Note the
  # name of this function is due to it will handle circumstances of tie, and the
  # method of handling this case will be discussed in that function.
  label_pred <- sapply(1:data_N, break_tie, dist_matrix,
                       k, label_real, position_expansion, data_N, cv_flag)
  # return value
  label_pred
}
```

```r
break_tie <- function(i, dist_matrix, k, label_real,
                      position_expansion, data_N, cv_flag = T) {
  # This is the function that calculates the output and check whether this tie. If
  # there is, this function will check all the labels' reciprocal distance with the
  # most frequency, and the one with the most value in reciprocal distance win.

  # order the first k distances
  if (cv_flag == T) indices <- order(dist_matrix[i,])[1:k]
  else indices <- order(dist_matrix[i,])[2:(k+1)]

  # transfer the indices to the standard real ones
  indices_real <- ifelse(indices>position_expansion, indices + data_N, indices)

  # extract the real label, by using the real indices derived above
  reg_labels <- label_real[indices_real]

  # extract all the values with the most frequency
  reg_labels_table <- table(reg_labels)
  reg_labels_max_freq <- max(reg_labels_table)
  result <- as.numeric(names(reg_labels_table[reg_labels_table == reg_labels_max_freq]))

  if (length(result) != 1) {
    # first create the data for further use. This data frame has labels versus its
    # corresponding distances with reciprocals.
    indices_dist <- dist_matrix[i, indices]
    indices_data <- data.frame(labels = reg_labels, dist_reciprocal = 1/indices_dist)

    # first we only need the labels that are in the result, say most frequent ones
    indices_data <- subset(indices_data, labels %in% result)
    # split into groups by label
    indices_data_split <- split(indices_data, indices_data$labels)
    # calculate their sum
    break_tie_vec <- sapply(indices_data_split, function(x) sum(x$dist_reciprocal))
    # select the one with the most sum value
    result <- as.numeric(names(indices_data_split)[which.max(break_tie_vec)])
  }

  result
}

knn_error <- function(label_real, label_pred) {
  # This function computes the mse (error) for subsequent compare, false_indices for
  # subsequent drawing falsely classified pictures, and confusion matrix for insepcting.
  # This function accepts the labels predicted by knn_compute function.
```

```r
  # initialize the label dataframe to compare
  label <- data.frame(real = label_real, pred = label_pred)
  # add error variable. if it is 1, then error exists; 0, no error
  label$error <- ifelse(label$real != label$pred, 1, 0)
  # calculate the mse
  mse <- sum(label$error)/length(label_pred)
  # extract false predict ones
  false_indices <- which(label$error == 1)
  # recognition matrix
  confusion_matrix <- with(label, table(real, pred))
  # return value
  result <- list(label_matrix = label, mse = mse, false_indices = false_indices,
                 confusion_matrix = confusion_matrix)
}

get_cv_dist_matrix <- function(cv, dist_matrix) {
  # This function split the distance matrix into cv groups used for knn prediction

  each_size <- dim(dist_matrix)[1] / cv
  cv_dist_matrix <- lapply(1:cv, function(i) {
    # First create a variable used to identify which rows to use and which column
    # to drop, then subset this index
    position_test <- (each_size*(i-1)+1):(each_size*i)
    dist_matrix[position_test,-position_test]
  })
  # return value
  cv_dist_matrix
}

knn_classfier <- function(dist_matrix, k, cv = 5, ori_data) {
  # This function is the main knn algorithm implementation function. It returns
  # different values by supplying different k value to it. It first spilit the data
  # by the cross-validation folds, and then give the splitted distance matrix to
  # knn_compute to calculate the predicted values. After that, calling the knn_error
  # function to compute the summary statistics for each k value, including mse,
  # false_indices and confusion_matrix.

  # get the cv split matrix and the real label
  cv_dist_matrix <- get_cv_dist_matrix(cv = cv, dist_matrix = dist_matrix)
  label_real <- ori_data$label

  # predict the output
  label_pred <- unlist(lapply(1:cv, function(i) {
    knn_compute(i, k, cv_dist_matrix[[i]], label_real)
  }))
```

```r
  # calculate summary statistics and return it
  knn_result <- knn_error(label_real, label_pred)
}

knn_mse_analysis <- function(knn_result, k_max) {
  # This function combines the summary statistics from many k values, from 1 to k_max,
  # which is normally 10, and then return the model with the smallest mse and its
  # corresponding statistics.

  # first extract the mse from each model, and give them names, print the result
  knn_mse_total <- unlist(sapply(knn_result, `[`, 'mse'))
  names(knn_mse_total) <- 1:k_max
  print(knn_mse_total)
  # select the best model with the least mse value
  best_model <- order(knn_mse_total)[1]
  # extract the best model's confustion_matrix and false_indices
  confusion_matrix <- knn_result[[best_model]]$confusion_matrix
  false_indices <- knn_result[[best_model]]$false_indices
  # return value
  result <- list(mse = knn_mse_total, best_model = best_model,
                 confusion_matrix = confusion_matrix, false_indices = false_indices)
}

ggplot_image <- function(model_best_model_false_indices) {
  # This function draws the image by giving it an index as input using ggplot.
  # It is smart since when you give it multiple inputs, it will draw the number of
  # indices you give it and draw a plot with all the images on one canvas.

  # It first extract the image information by the index from the original dataset.
  # If there are multiple indices, it will classify them by the indices by creating
  # a type variable
  data_for_ggplot_list <- lapply(model_best_model_false_indices, function(i) {
    data.frame(x = rep(1:28),
               y = rep(28:1, each = 28),
               z = as.numeric(digit_image[i, -1]),
               type = i)
  })
  data_for_ggplot <- do.call('rbind', data_for_ggplot_list)

  # ggplot main function
  p <- ggplot(data = data_for_ggplot, aes(x, y, fill = z)) + geom_raster() +
    scale_fill_gradient(low = 'white', high = 'black') + facet_wrap(~type) +
    theme(axis.ticks.x = element_blank(), axis.text.x = element_blank(),
          axis.ticks.y = element_blank(), axis.text.y = element_blank())
  print(p)
}
```

```r
mse_plot <- function(model_mse_total, metric_length, k_max) {
  # This function accepts model_mse_total as input and then plot it by split different
  # metric. This is a plot for each metric by mse versus k value.

  # creates the mse_matrix for plot based on model_mse_total
  mse_matrix <- data.frame(
    mse = unname(unlist(model_mse_total)),
    order = rep(1:k_max, times = metric_length),
    type = as.factor(rep(1:metric_length, each = k_max))
  )
  # draw the plot using ggplot
  mse_plot <- ggplot(mse_matrix, aes(x = order, y = mse, group = type, color = type)) +
    geom_point() +
    geom_line()
  print(mse_plot)
}

false_digit_analysis <- function(i, k = 4, digit_dist_list) {
  # This function does the job of showing misclassified digits by giving the plot of
  # the original digit and the digits that predict it

  # extract the needed distance
  dist_matrix <- digit_dist_list[[1]][i,]
  # predict, get its indices
  predict_digit <- order(dist_matrix)[1:(k + 1)]
  # creating a dataframe for each index
  false_digit_total_list <- lapply(predict_digit, function(i) {
    data.frame(x = rep(1:28),
               y = rep(28:1, each = 28),
               z = as.numeric(digit_image[i, -1]),
               type = i)
  })
  # combine the dataframes into a big one
  false_digit_total_data <- do.call(rbind, false_digit_total_list)

  # ggplot main function
  p <- ggplot(data = false_digit_total_data, aes(x, y, fill = z)) + geom_raster() +
    scale_fill_gradient(low = 'white', high = 'black') + facet_wrap(~type) +
    theme(axis.ticks.x = element_blank(), axis.text.x = element_blank(),
          axis.ticks.y = element_blank(), axis.text.y = element_blank())
  print(p)
}

#----------------------------------------------------------------------------
```

```r
main_one_intermediate <- function(digit_dist_matrix, test_number = 5000,
                                  k_max, print_flag) {
  # This is the intermediate function used to do the knn analysis for each metric.
  # It first calls the knn_classifier function to do the knn analysis for each k,
  # and then calls the knn_mse_analysis function to return the best model and its
  # corresponding statistics.

  # knn analysis for each k
  knn_result <- lapply(1:k_max, function(i) {
    knn_classfier(digit_dist_matrix[1:test_number, 1:test_number],
                  k=i, ori_data = digit_image[1:test_number,])
  })

  # print flag for each metric
  cat('\n')
  if (print_flag == 1) cat('This is for euclidean')
  else cat('This is for manhattan')
  cat('\n')
  cat('\n')

  cat("mse:")
  cat('\n')
  # do the knn_mse_analysis to return the best model and its corresponding statistics
  knn_model_select_result <- knn_mse_analysis(knn_result, k_max)
  cat('\n')

  # show the best model
  best_model <- knn_model_select_result$best_model
  cat("The best model is k = ", best_model)
  cat('\n')
  cat('\n')

  # return value
  knn_model_select_result
}

main_one <- function(digit_dist_list, k_max = 10) {
  # This is the function to accomplish all the tasks for question one. It first
  # calls the main_one_intermediate to find the final statistics for each metric,
  # then extract the mse and plot them on the same plot to compare the k value and
  # metric performance. Finally, plot the false classified images for each metric.

  # calculate the length of metrics
  metric_length <- length(digit_dist_list)

  # return the final statisitcs for each metric by calling main_one_intermediate
```

```r
  knn_model_select_result <- lapply(1:metric_length, function(i) {
    main_one_intermediate(digit_dist_list[[i]], print_flag = i, k_max = k_max)
  })
  knn_model_select_result <- setNames(knn_model_select_result, metrics)

  # extract mse
  model_mse_total <- lapply(1:metric_length,
                            function(i) knn_model_select_result[[i]]$mse)
  # draw mse plot
  mse_plot(model_mse_total, metric_length, k_max)

  # compute the mse with the best model at no cross-validation
  cat("The overall best model is k = 4 at eculidean metric")
  cat("\n")
  cat("\n")
  knn_best_result <- knn_compute(5000, 4,
                                 digit_dist_list[[1]], digit_image$label, cv_flag = F)
  knn_best_mse <- knn_error(label_real = digit_image$label, label_pred = knn_best_result)
  cat("The corresponding confusion matrix is")
  cat('\n')
  cat('\n')
  print(knn_best_mse$confusion_matrix)

  # plot best model false indices' images for each metric. I only select the first
  # 36 indices, since I think it is sufficient to explain the problem.
  ggplot_image(knn_best_mse$false_indices[1:36])

  # This expression shows plots more information of falsely classified digits
  ggplot_image_false <- lapply(c(733, 440), false_digit_analysis, 4, digit_dist_list)
}

#-------------------------------------------------------------------------

get_digit_mean_pixel <- function(ori_data) {
  # This is the function that from train data calculating the mean for each label

  # split the data by its labels
  digit_dist_matrix_split_label <- split(ori_data, ori_data$label)
  # calculate the mean for each label
  digit_mean_pixel <- t(sapply(digit_dist_matrix_split_label, function(x) {
    apply(x, 2, mean)
  }))
  # return value
  digit_mean_pixel
}
```

```r
get_means_dist <- function(ori_data, digit_mean_pixel, metric) {
  # This function gets the distance matrix for the test set and the train set.

  # calculate the length of test data and mean of train data
  ori_data_N <- dim(ori_data)[1]
  digit_mean_pixel_N <- dim(digit_mean_pixel)[1]

  # stack the test data and the mean of train data together
  means_matrix <- rbind(digit_mean_pixel, ori_data)

  # calculate the distance matrix and select the values that are useful to us
  means_dist <- as.matrix(dist(means_matrix, method = metric))[
    11:(ori_data_N+digit_mean_pixel_N), 1:10]

  # return value
  means_dist
}

get_means_pred <- function(cv_flag, ori_data, metric) {
  # This function predicts the values for each cross-validation train and test
  # combination. Since it knows which cv group it is in now, it will select the
  # corresponding train and test dataset and then supplies them to the
  # get_digit_mean_pixel and get_means_dist function to calculate the distance.
  # Then it will return the smallest distance as its predict value.

  # this calculates the test_position for extracting data.
  test_position <- (1000*(cv_flag-1)+1):(1000*cv_flag)

  # Extract the data for test and train and then supplies them to the needed function
  digit_mean_pixel <- get_digit_mean_pixel(ori_data[-test_position,])
  means_dist <- get_means_dist(ori_data[test_position,], digit_mean_pixel, metric)

  # predict the label
  means_label_pred <- apply(means_dist, 1, which.min) - 1
}

means_mse_analysis <- function(means_pred, means_real) {
  # This function analyze the predict value by calculating mse.

  # calculates mse
  means_mse <- lapply(means_pred, function(x) {
    sum(means_real != x) / (length(means_real))
  })
  means_mse
}
```

```r
main_two <- function(ori_data, cv = 5) {
  # This function calculates the mse for different metrics. It first calls the
  # means_pred function by suppling different metrics and then in each metric,
  # calculates predict value in each cv fold. At last, it calculates mse, and makes
  # the output friendly to print.

  # print instruction
  cat('Now come to k-means classification')
  cat('\n')

  # calculates predict value for each metric
  means_pred <- lapply(metrics, function(x) {
    temp <- lapply(1:cv, function(i) {
      get_means_pred(i, ori_data, x)
    })
    unlist(temp)
  })
  # extract real value
  means_real <- ori_data$label

  # calculates mse and give it names
  means_mse <- means_mse_analysis(means_pred, means_real)
  means_mse <- setNames(means_mse, metrics)
  print(means_mse)

  # return value
  means_mse
}

#-----------------------------------------------------------------------

main <- function() {
  # This is the entrace for this assignment. It first reads into data, then calls
  # main_one to do question one, calls main_two to do question two.

  digit_dist_list <- lapply(metrics, create_dist_matrix)
  main_one(digit_dist_list)

  main_two(digit_image)
}

main()
```