```
#Spring STA_141A  Instructor: Groupta
# FINAL_PROJECR
# Members: Hao Luo, Bianca Fung, and Dai Chen
#

input_path = "/Users/haoluo/Desktop/STA141A/final/Data/"
output_train_path = "/Users/haoluo/Desktop/STA141A/final/train.rds"
output_test_path = "/Users/haoluo/Desktop/STA141A/final/test.rds"

# Question 1 ************************************************************************

# function to load a file and convert it to one image per line
load_one_bin = function(filepath){
  con = file(filepath, "rb")
  hex = readBin(con, raw(), 10000*(1024*3+1))
  close(con)
  N = 10000
  hex = as.character(hex)
  hex = matrix(hex, nrow = N, byrow=TRUE)
  #hex = apply(hex, 1, function(x) x) #?????
  hex = t(hex)
  return(hex)
}

# function to save train data
save_training_images = function(input_path, output_path){
  files = list.files(path = input_path, pattern = "data_batch.*.bin", full.names =
TRUE)
  numPixel = 1024
  image_list = lapply(files, function(x) t(load_one_bin(x)))
  image_matrix = do.call("rbind", image_list)
  train = data.frame(image_matrix)
  names(train) = c("label", paste(rep(c("R", "G", "B"), each=numPixel), rep(1:numPixel,
3), sep=''))
  train$label = as.numeric(train$label)
  train[,-1] = apply(train[,-1], 2, as.character)
  save(train, file = output_path)
}

# function to save test data
save_testing_images = function(input_path, output_path){
  files = list.files(path = input_path, pattern = "test_batch.*.bin", full.names =
TRUE)
  numPixel = 1024
  image_list = lapply(files, function(x) t(load_one_bin(x)))
  image_matrix = do.call("rbind", image_list)
  test = data.frame(image_matrix)
  names(test) = c("label", paste(rep(c("R", "G", "B"), each=numPixel), rep(1:numPixel,
3), sep=''))
  test$label = as.numeric(test$label)
  test[,-1] = apply(test[,-1], 2, as.character) #apply(x,2?,as.charac) -> 2 means
columwise
  save(test, file = output_path)
}

save_training_images(input_path, output_train_path)
save_testing_images(input_path, output_test_path)

# Question 2 ************************************************************************
```

```
library(grid)

# load the orginal train and test (the output from question 1)
load("/Users/haoluo/Desktop/STA141A/final/train.rds") #
load("/Users/haoluo/Desktop/STA141A/final/test.rds") #

# subset the train and test sets using the code from piazza
*****************************
data_rescale<-
function(labels,k=5000)sort(as.vector(sapply(unique(labels),function(i)which(labels==i))
[1:k,]))
train2<-train[data_rescale(train[,1],k=500),]
test2<-test[data_rescale(test[,1],k=100),]

save(train2, file = "/Users/haoluo/Desktop/STA141A/final/train2.rds")
save(test2, file = "/Users/haoluo/Desktop/STA141A/final/test2.rds")
load("/Users/haoluo/Desktop/STA141A/final/train2.rds")
load("/Users/haoluo/Desktop/STA141A/final/test2.rds")

tags = read.table("/Users/haoluo/Desktop/STA141A/final/Data/batches.meta.txt")

view_images = function(set, rowNum, tags=tags, plot.new = TRUE, mfrow = c(1,1),
x1=0,x2=0,x3=1,x4=1){
  image_row = set[rowNum,]
  label = image_row$label
  red = image_row[c(1:1024) + 1]
  green = image_row[c(1:1024) + 1 + 1024]
  blue = image_row[c(1:1024) + 1 + 1024 * 2 ]
  images = matrix(paste("#", red, green, blue, sep=''), ncol = 32, byrow = TRUE)
  #grid.raster(images, name = as.character(tags[label+1,]))
  if (plot.new){
  par(mfrow=mfrow, mar=c(0, 0, 3, 0))
  plot.window(xlim=c(0, 1), ylim=c(0, 1), asp=1) # cited
  plot.new()  # cited
  rasterImage(images, 0, 0, 1, 1) # cited
  } else {
    par(mfrow=mfrow, mar=c(0, 0, 3, 0))
    plot.window(xlim=c(0, 1), ylim=c(0, 1), asp=1) # cited
    rasterImage(images, x1,x2,x3,x4)
  }
  title(as.character(tags[label,]), font.main=2) # cited
  # source code: https://stackoverflow.com/questions/27828842/r-add-title-to-images-
with-rasterimage
}

view_images(train2, 500, tags)
#view_images(train2, 500, tags, mfrow=c(2,2))
#view_images(train2, 200, tags, FALSE, mfrow=c(2,2))

# Question 3 ***********************************************************************

# plot images from different classes
selected_obs_for_display = sapply(1:10, function(x) which(train2$label == x)[1])

#images for each class
for (x in c(1:10)){
  view_images(train2, selected_obs_for_display[x], tags)
}

#*******************************
```

```
red_column_ind = c(1:1024) + 1
green_column_ind = c(1:1024) + 1 + 1024
blue_column_ind = c(1:1024) + 1 + 1024 * 2

train2_red = train2
train2_red[,c(green_column_ind, blue_column_ind)] = "00"

train2_green = train2
train2_green[,c(red_column_ind, blue_column_ind)] = "00"

train2_blue = train2
train2_blue[,c(red_column_ind, green_column_ind)] = "00"

view_images(train2_red, 500, tags)
view_images(train2_green, 500, tags)
view_images(train2_blue, 500, tags)

# Q: Which pixels at which color channels seem the most likely to be useful for
classification?
# A: The pixels have the highest variation between classes but the least variation
within the classes are good for classification.

# first, convert the color from hex code to decimal code
train2d = data.frame(apply(train2[,-1], 2, function(x) strtoi(x, 16L))) # 16 hex
train2d_mean = apply(train2d, 2, mean) # overall mean for each pixel
train2d_list = split(train2d, train2$label) # split by label
train2d_list_mean = lapply(train2d_list, function(x) apply(x, 2, mean)) # within group
mean
sse_by_label = lapply(1:length(train2d_list_mean),
             function(x) apply(sweep(train2d_list[[x]], 2, train2d_list_mean[[x]])^2,
2, sum)) # sse for each pixel
sse = apply(do.call("rbind", sse_by_label), 2, sum)
I = length(unique(train2$label)) # number of labels
nT = dim(train2) # total number of observations
mse = sse/(nT-I)
train2d_matrix_mean = do.call("rbind", train2d_list_mean) # a matrix contains pixel
mean for each label
ssr = apply(sweep(train2d_matrix_mean, 2, train2d_mean)^2, 2, sum) # Ybari.-Ybar..
msr = ssr/(I-1)
f_val = msr/mse

sort(f_val, decreasing=TRUE)[1:10] # the top 10 highest F-value = most likely to be
useful for classification
sort(f_val, decreasing=FALSE)[1:10] # the least 10 F-value = least likely to be useful
for classification

# Question 4 ******KNN_predict
FUNCT*****************************************************************

library(devtools)
library(parallelDist)

load("/Users/haoluo/Desktop/STA141A/final/train2.rds")
load("/Users/haoluo/Desktop/STA141A/final/test2.rds")

train_label = train2[,1] #the first column of the training set is the label
train2d = data.frame(apply(train2[,-1], 2, function(x) strtoi(x, 16L)))
test2d = data.frame(apply(test2[,-1], 2, function(x) strtoi(x, 16L))) # exclude label
in first column
test2d.mat = as.matrix(test2d)
```

```
train2d.mat = as.matrix(train2d)

start.time <- Sys.time()
dist_euclidean = parallelDist(rbind(test2d.mat,train2d.mat), method = "euclidean")
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken

start.time <- Sys.time()
dist_manhattan = parallelDist(rbind(test2d.mat,train2d.mat), method = "manhattan")
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken

save(dist_euclidean, file = "/Users/haoluo/Desktop/STA141A/final/dist_euclidean.rds")
save(dist_manhattan, file = "/Users/haoluo/Desktop/STA141A/final/dist_manhattan.rds")

# input:
# train2: a matrix contain hex code, the first column is the label
# test2: a matrix contain hex code for the test cases
# method: distance method

predict_knn = function(train2,
                       test2,
                       method = "euclidean",
                       k=3,
                       dist_matrix = NA){
  library("parallelDist")
  train_label = train2[,1] # the first column of the training set is the label

  if (sum(is.na(dist_matrix))>0){
    train2d = data.frame(apply(train2[,-1], 2, function(x) strtoi(x, 16L)))
    test2d = data.frame(apply(test2[,-1], 2, function(x) strtoi(x, 16L)))
    test2d.mat = as.matrix(test2d)
    train2d.mat = as.matrix(train2d)
    dist_matrix = parallelDist(rbind(test2d.mat,train2d.mat), method = method)
  }

  dist_matrix = as.matrix(dist_matrix)
  D = dist_matrix[-c(1:nrow(test2)), 1:nrow(test2)]
  # D is the pairs of the distance between test set and training set,
  # row represents the training set, column represents the test set

  row.names(D) = 1:nrow(D)
  k_label = apply(D, 2, function(x) train_label[as.numeric(names(sort(x)[1:k]))])
  if (k!=1){
  predict_label = as.numeric(apply(k_label, 2, function(x) names(sort(table(x),
decreasing=TRUE))[1]))
  } else {
    predict_label = as.numeric(apply(k_label, 2, function(x) names(sort(table(x),
decreasing=TRUE))[1]))
  }
  return(mean(train_label!=predict_label))
}

start.time = Sys.time()
select_k = 9
predict_label_euclidean = predict_knn(train2,
                                      test2,
                                      method = "euclidean",
```

```
                                    k=select_k,
                                    dist_matrix = dist_euclidean)
predict_label_euclidean # error rate -> 89.52%
end.time = Sys.time()
time.taken = end.time - start.time
time.taken

# Question 5 ***************************************************************************
load("/Users/haoluo/Desktop/STA141A/final/dist_euclidean.rds")
load("/Users/haoluo/Desktop/STA141A/final/dist_manhattan.rds")
dist_euclidean = as.matrix(dist_euclidean)
dist_manhattan = as.matrix(dist_manhattan)

cv_error_knn = function(train2,
                        test2,
                        method = "euclidean",
                        k=3,
                        foldn = 10,
                        seed = round(runfi(1)*1000),
                        dist_matrix = NA){
  library("parallelDist")
  train_label = train2[,1]

  if (sum(is.na(dist_matrix))>0){
    train2d = data.frame(apply(train2[,-1], 2, function(x) strtoi(x, 16L)))
    test2d = data.frame(apply(test2, 2, function(x) strtoi(x, 16L))) # exclude label in
first column
    test2d.mat = as.matrix(test2d)
    train2d.mat = as.matrix(train2d)
    dist_matrix = parallelDist(rbind(test2d.mat,train2d.mat), method = method) # e.g.
method = "euclidean"
  }

  dist_matrix = as.matrix(dist_matrix)
  D = dist_matrix[-c(1:nrow(test2)), -c(1:nrow(test2))]
  # D contains only the pairwise distance among the training groups, it does not
involve test set anymore
  row.names(D) = 1:nrow(D)
  colnames(D) = 1:nrow(D)

  set.seed(seed)
  folds_labels = sample(1:foldn, size = nrow(train2), replace=TRUE)
  folds_index = split(1:nrow(train2), folds_labels)
  folds_dist = lapply(1:foldn, function(x) D[do.call("c", folds_index[-x]),
folds_index[[x]]])
  # folds distance is a list, each element of the list contains a matrix
  # the row of the matrix represents the (n-1) folds index, column of the matrix
represents the n-th fold index
  foldn_topK_class = lapply(folds_dist, function(x) apply(x, 2, function(y)
train_label[as.numeric(names(sort(y))[1:k])]))
  if (k!=1) {
    foldn_topK_label = lapply(foldn_topK_class, function(x) apply(x, 2, function(y)
names(sort(table(y), decreasing=TRUE))[1]))
  } else {
    foldn_topK_label = lapply(foldn_topK_class, function(x) names(x)[1])
  }
  predict_label = as.numeric(do.call("c", foldn_topK_label)) # predicted label
  matched_true_label = train_label[do.call("c", folds_index)]
  return(mean(matched_true_label != predict_label))
}
```

```
error_euclidean = cv_error_knn(train2,
                                    test2,
                                    method = "euclidean",
                                    k=select_k,
                                    foldn = 10,
                                    seed = 100,
                                    dist_matrix = dist_euclidean)

start.time = Sys.time()
all_error_euclidean = sapply(1:15, function(x) cv_error_knn(train2, test2, k=x, foldn =
10, seed=2018, dist_matrix = dist_euclidean))
end.time = Sys.time()
time.taken = end.time - start.time
time.taken # Time difference of 1.608918 mins

start.time = Sys.time()
all_error_manhattan = sapply(1:15, function(x) cv_error_knn(train2, test2, k=x, foldn =
10, seed=2018, dist_matrix = dist_manhattan))
end.time = Sys.time()
time.taken = end.time - start.time
time.taken # Time difference of 2.260247 mins

distance = data.frame(euclidean = all_error_euclidean, manhattan = all_error_manhattan)
save(distance, file = "/Users/haoluo/Desktop/STA141A/final/distance.rds")

dev.off()
plot(1:15, distance$euclidean, ylab = "Error Rate", xlab = "k", type='p',
     main = "10-fold CV error rates", col='red', ylim=c(0.6,1))
abline(v=c(1:15), col="lightgrey")
lines(1:15, distance$euclidean, col='red')
points(1:15, distance$manhattan, col='blue')
lines(1:15, distance$manhattan, col='blue')
legend("topright", c("Euclidean", "Manhattan"), col = c("red", "blue"), lty=c(1,1))


# Question 6 ***************************************************************************

cv_error_knn2 = function(train2,
                         test2,
                         method = "euclidean",
                         k=3,
                         foldn = 10,
                         seed = round(runfi(1)*1000),
                         dist_matrix = NA){
  library("parallelDist")
  train_label = train2[,1]

  if (sum(is.na(dist_matrix))>0){
    train2d = data.frame(apply(train2[,-1], 2, function(x) strtoi(x, 16L)))
    test2d = data.frame(apply(test2, 2, function(x) strtoi(x, 16L))) # suppose test2
does not include label in first column
    test2d.mat = as.matrix(test2d)
    train2d.mat = as.matrix(train2d)
    dist_matrix = parallelDist(rbind(test2d.mat,train2d.mat), method = method) # e.g.
method = "euclidean"
  }

  dist_matrix = as.matrix(dist_matrix)
  D = dist_matrix[-c(1:nrow(test2)), -c(1:nrow(test2))]
```

```
  # D contains only the pairwise distance among the training groups, it does not
involve test set anymore
  row.names(D) = 1:nrow(D)
  colnames(D) = 1:nrow(D)

  set.seed(seed)
  folds_labels = sample(1:foldn, size = nrow(train2), replace=TRUE)
  folds_index = split(1:nrow(train2), folds_labels)
  folds_dist = lapply(1:foldn, function(x) D[do.call("c", folds_index[-x]),
folds_index[[x]]])
  # folds distance is a list, each element of the list contains a matrix
  # the row of the matrix represents the (n-1) folds index, column of the matrix
represents the n-th fold index
  foldn_topK_class = lapply(folds_dist, function(x) apply(x, 2, function(y)
train_label[as.numeric(names(sort(y))[1:k])]))
  if (k!=1) {
    foldn_topK_label = lapply(foldn_topK_class, function(x) apply(x, 2, function(y)
names(sort(table(y), decreasing=TRUE))[1]))
  } else {
    foldn_topK_label = lapply(foldn_topK_class, function(x) names(x)[1])
  }
  predict_label = as.numeric(do.call("c", foldn_topK_label)) # predicted label
  test_case_index = do.call("c", folds_index)
  matched_true_label = train_label[test_case_index]
  out = list(predict_label = predict_label, matched_true_label = matched_true_label,
index = test_case_index)
  return(out)
}

euclidean_best3k = order(all_error_euclidean)[1:3]
manhattan_best3k = order(all_error_manhattan)[1:3]

euclidean_out = lapply(euclidean_best3k, function(x) cv_error_knn2(train2, test2, k=x,
foldn = 10, seed=2018, dist_matrix = dist_euclidean))
manhattan_out = lapply(manhattan_best3k, function(x) cv_error_knn2(train2, test2, k=x,
foldn = 10, seed=2018, dist_matrix = dist_manhattan))

tags = read.table("/Users/haoluo/Desktop/STA141A/final/Data/batches.meta.txt")


# For euclidean: first best k: 11
mypred = factor(euclidean_out[[1]]$predict_label)
levels(mypred) = as.character(tags[,1])
mytrue = factor(euclidean_out[[1]]$matched_true_label)
levels(mytrue) = as.character(tags[,1])
euc_conf1 = table(data.frame(predict = mypred, true = mytrue))
#write.csv(euc_conf1, "/Users/haoluo/Desktop/STA141A/final/Data/euc_conf1.csv")

# For euclidean: second best k: 8
mypred = factor(euclidean_out[[2]]$predict_label)
levels(mypred) = as.character(tags[,1])
mytrue = factor(euclidean_out[[2]]$matched_true_label)
levels(mytrue) = as.character(tags[,1])
euc_conf2 = table(data.frame(predict = mypred, true = mytrue))

# For euclidean: third best k: 6
mypred = factor(euclidean_out[[3]]$predict_label)
levels(mypred) = as.character(tags[,1])
mytrue = factor(euclidean_out[[3]]$matched_true_label)
levels(mytrue) = as.character(tags[,1])
```

```
euc_conf3 = table(data.frame(predict = mypred, true = mytrue))

# For manhattan: first best k: 12
mypred = factor(manhattan_out[[3]]$predict_label)
levels(mypred) = as.character(tags[,1])
mytrue = factor(manhattan_out[[3]]$matched_true_label)
levels(mytrue) = as.character(tags[,1])
man_conf1 = table(data.frame(predict = mypred, true = mytrue))

# For manhattan: second best k: 14
mypred = factor(manhattan_out[[2]]$predict_label)
levels(mypred) = as.character(tags[,1])
mytrue = factor(manhattan_out[[2]]$matched_true_label)
levels(mytrue) = as.character(tags[,1])
man_conf2 = table(data.frame(predict = mypred, true = mytrue))

# For manhattan: third best k: 8
mypred = factor(manhattan_out[[3]]$predict_label)
levels(mypred) = as.character(tags[,1])
mytrue = factor(manhattan_out[[3]]$matched_true_label)
levels(mytrue) = as.character(tags[,1])
man_conf3 = table(data.frame(predict = mypred, true = mytrue))

#QN6 & QN7 & QN8 ******************************************************
# accuracy rate per class
par(mfrow=c(2,3))
barplot(diag(euc_conf1/500), ylim=c(0, 1), main = "Euclidean, k = 11" , ylab =
"Accuracy rate")
barplot(diag(euc_conf2/500), ylim=c(0, 1), main = "Euclidean, k = 8", ylab = "Accuracy
rate")
barplot(diag(euc_conf3/500), ylim=c(0, 1), main = "Euclidean, k = 6", ylab = "Accuracy
rate")
barplot(diag(man_conf1/500), ylim=c(0, 1), main = "Manhattan, k = 14", ylab = "Accuracy
rate")
barplot(diag(man_conf2/500), ylim=c(0, 1), main = "Manhattan, k = 12", ylab = "Accuracy
rate")
barplot(diag(man_conf3/500), ylim=c(0, 1), main = "Manhattan, k = 8", ylab = "Accuracy
rate")

#***********wrong predicted labels**********************
which(euclidean_out[[1]]$predict_label != euclidean_out[[1]]$matched_true_label)[1:10]
euclidean_out[[1]]$predict_label[2] # deer
euclidean_out[[1]]$matched_true_label[2] # frog
euclidean_out[[1]]$index[1:10]

#**********explore the misclassified images*************
temp_dist = dist(rbind(train2d[24,], train2d), method = "euclidean")
temp_dist = as.matrix(temp_dist)
view_images(train2, 24, tags)
view_images(train2, 4704, tags)

#*********************************************************************
# overall misclassification rate
euc_acc = c(sum(diag(euc_conf1)), sum(diag(euc_conf2)), sum(diag(euc_conf3)))/5000
man_acc = c(sum(diag(man_conf1)), sum(diag(man_conf2)), sum(diag(man_conf3)))/5000
euc_acc # 0.2720 0.2688 0.2682
man_acc # 0.3056 0.3058 0.3056
# best combinations:
# 1: (man, k=14)
# 2: (man, k=12)
```

```
# 3: (man, k=8)

#QN 9 ************************************************************************

test_predict = function(train2,
                        test2,
                        method = "euclidean",
                        k=3,
                        foldn = 10,
                        seed = round(runfi(1)*1000),
                        dist_matrix){
  library("parallelDist")
  train_label = train2[,1]

  dist_matrix = as.matrix(dist_matrix)
  D = dist_matrix[-c(1:nrow(test2)), c(1:nrow(test2))]
  # D contains only the pairwise distance among the training groups, it does not
involve test set anymore
  row.names(D) = 1:nrow(D)
  colnames(D) = 1:nrow(test2)
  tops = apply(D, 2, function(y) train_label[as.numeric(names(sort(y))[1:k])])
  if (k != 1){
  predict_class = as.numeric(apply(tops, 2, function(y) names(sort(table(y),
decreasing=TRUE))[1]))
  } else {
    predict_class = as.numeric(tops)
  }
  true_class = test2[,1]
  return(list(true = true_class, predict = predict_class))
}

# euclidean
result = lapply(1:15, function(x) test_predict(train2, test2, dist_matrix =
dist_euclidean, k = x))
table(result[[1]]) # confusion matrix for k = 1
table(result[[7]]) # confusion matrix for k = 7
test_error_rate_euclidean = sapply(1:15, function(x) mean(result[[x]]$true !=
result[[x]]$predict))

# manhattan
result = lapply(1:15, function(x) test_predict(train2, test2, dist_matrix =
dist_manhattan, k = x))
table(result[[1]]) # confusion matrix for k = 1
table(result[[7]]) # confusion matrix for k = 7
test_error_rate_manhattan = sapply(1:15, function(x) mean(result[[x]]$true !=
result[[x]]$predict))

plot(1:15, test_error_rate_euclidean, col="blue", ylim = c(0.65,0.8), type = "l")
lines(1:15, test_error_rate_manhattan, col="red")
```