Homework 3

CS 498, Spring 2018, Xiaoming Ji

CIFAR-10 is a dataset of 32x32 images in 10 categories, collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It is often used to evaluate machine learning algorithms. You can download this dataset from https:// www.cs.toronto.edu/~kriz/cifar.html.

Firstly, we do some pre-processing on the data.

```r
# Read binary file and convert to integer vectors
# File format is 10000 records following the pattern:
label_names = read.table("cifar-10-batches-bin/batches.meta.txt")
label_count = dim(label_names)[1]
image_bytes = 32 * 32 * 3
```

```r
#Set filenames need to be read
folder_name = "cifar-10-batches-bin"
file_names = c("data_batch_1.bin", "data_batch_2.bin","data_batch_3.bin",
               "data_batch_4.bin","data_batch_5.bin","test_batch.bin")
num.images = 10000 # Set to 10000 to retrieve all images per file to memory
mat_images = matrix(0, nrow = num.images * length(file_names), ncol = 1024 * 3 + 1)

# Cycle through all binary files
index = 1
for (f in 1:length(file_names)) {
  print(paste("Reading ", file_names[f], sep=""))
  to.read = file(paste(folder_name, "/", file_names[f], sep=""), "rb")
  for(i in 1:num.images) {
    l = readBin(to.read, integer(), size=1, n=1, endian="big")
    mat_images[index, 1: image_bytes] = as.integer(readBin(to.read, raw(), size=1,
                                                  n=image_bytes, endian="big"))
    mat_images[index, image_bytes + 1] = l + 1
    #df_images = rbind(df_images, mat_images[index,])
    index = index + 1
  }
  close(to.read)
}

df_images = as.data.frame(mat_images)
save(df_images,file="images.df")
```

```r
# function to run sanity check on photos & labels import
drawImage = function(image) {
  r = image[1:1024]
  g = image[1025:2048]
  b = image[2049:3072]
  img_matrix = rgb(r, g, b, maxColorValue=255)
  #dim(img_matrix) = c(32, 32)
  #img_matrix = t(img_matrix)

  #library(grid)
  image(matrix(1:(32*32), 32, 32)[, 32:1], col=img_matrix)
  #grid.raster(img_matrix, interpolate=FALSE)
  #label_names[[1]][as.integer(images[index, 1024 * 3 + 1])]
}
#drawImage(df_images[3000,-3073])
```

1.1

For each category, compute the mean image and the first 20 principal components. Plot the error resulting from representing the images of each category using the first 20 principal components against the category.

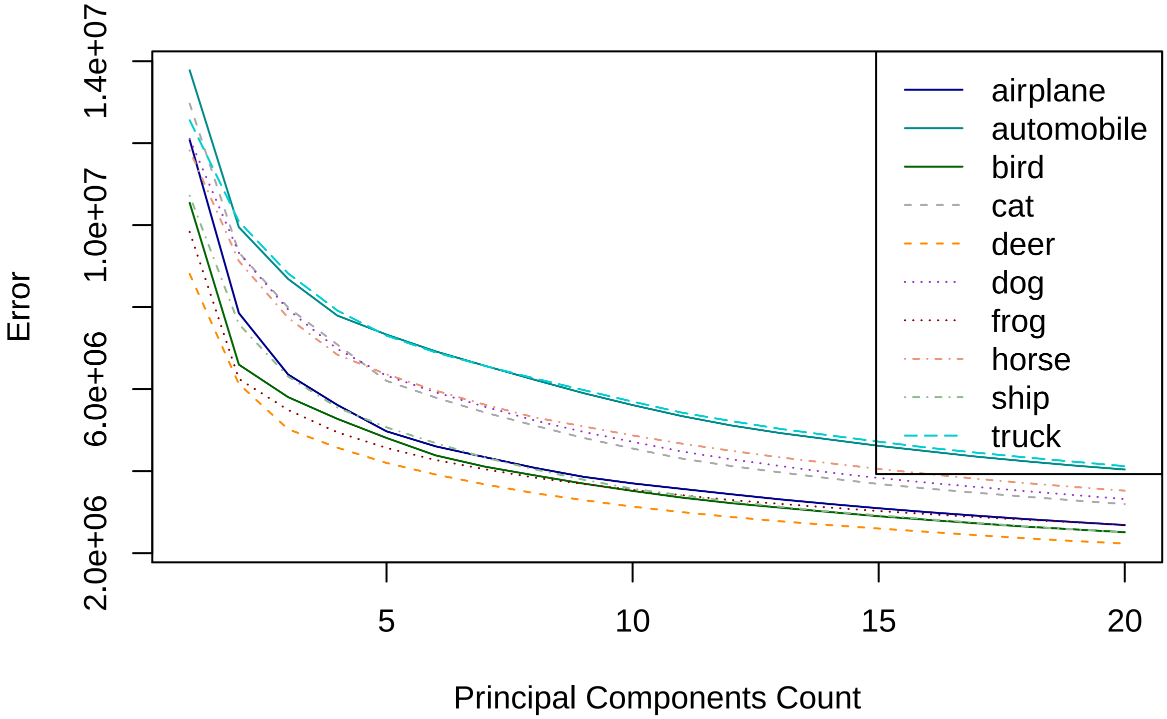
```
load("images.df") #Load file from
```

```
eigen_pca = function (data) {
  mean = colSums(data)
  covmat = cov(data)
  eigen = eigen(covmat)

  return (list(mean = colMeans(data), pc = eigen$vectors, weight = eigen$values))
}

#Constructing a low-dimensional representation:
cld = function(p, x, k) {
  u = p$pc[, 1:k]
  x = u %*% crossprod(u, x - p$mean) + p$mean
  x[x > 255] = 255
  x[x < 0] = 0

  return (x)
}
```

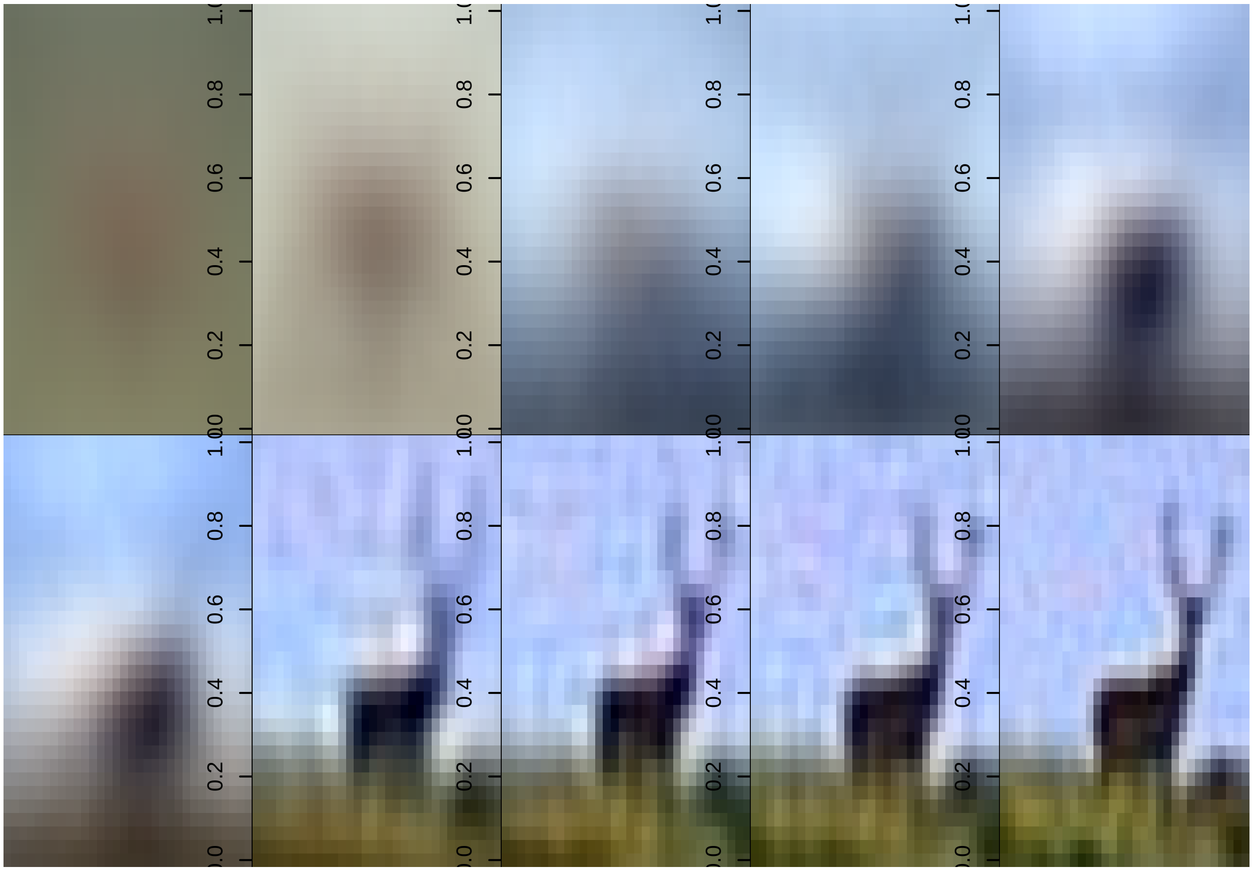
```
timestamp()
all_pca = list()
for (i in 1:label_count) {
  all_pca[[i]] = eigen_pca(df_images[df_images$V3073 == i, -3073])
}
timestamp()
save(all_pca, file="pca.data")
```

```
load("pca.data")
```

```
pc_count = 20

colors = c("darkblue", "darkcyan", "darkgreen", "darkgrey", "darkorange", "darkorchid", "darkred", "darl
line_types = c(1, 1, 1, 2, 2, 3, 3, 4, 4, 5)

error = list()
e = rep(0, pc_count)
max_error = 0
min_error = 1e+10
for (i in 1:label_count){
  for (j in 1:pc_count) {
    e[j] = sum(all_pca[[i]]$weight[j:image_bytes])
  }
  if(max(e) > max_error) max_error = max(e)
  if(min(e) < min_error) min_error = min(e)
  error[[i]] = e
}

plot(error[[1]], type = "l", xlab="Principal Components Count", ylab="Error",
     ylim = c(min_error, max_error), col=colors[1], lty = line_types[1])

for (i in 2:label_count){
  lines(error[[i]], col=colors[i], lty = line_types[i])
}

legend("topright", legend = label_names[,1], lwd = 1, col = colors, lty=line_types)
```



```
sample_index = 3000
sample_label = df_images[sample_index, 3073]
par(mfrow=c(2,5))
par(mar = c(0,0,0,0))

drawImage(all_pca[[sample_label]]$mean)
for (k in c(1, 5, 10, 15, 20, 100, 150, 200, 300)){
  drawImage(cld(all_pca[[sample_label]], t(df_images[sample_index, -3073]), k))
}
```



1.2

Compute the distances between mean images for each pair of classes. Use principal coordinate analysis to make a 2D map of the means of each categories. For this exercise, compute distances by thinking of the images as vectors.

1.3

Here is another measure of the similarity of two classes. For class A and class B, define E(A | B) to be the average error obtained by representing all the images of class A using the mean of class A and the first 20 principal components of class B. Now define the similarity between classes to be (1/2)(E(A | B) + E(B | A)). If A and B are very similar, then this error should be small, because A's principal components should be good at representing B. But if they are very different, then A's principal components should represent B poorly. In turn, the similarity measure should be big. Use principal coordinate analysis to make a 2D map of the classes. Compare this map to the map in the previous exercise? are they different? why?