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 load the files to memory and do some pre-processing on the data.

```
# 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
pc_count = 20
```

```
drawImage = function(image, title) {
  image[image > 255] = 255
  image[image < 0] = 0
  r = image[1:1024]
  g = image[1025:2048]
  b = image[2049:3072]
  img_matrix = rgb(r, g, b, maxColorValue=255)

  image(matrix(1:(32*32), 32, 32)[, 32:1], col=img_matrix, axes = FALSE,
        main = title)
}
```

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.

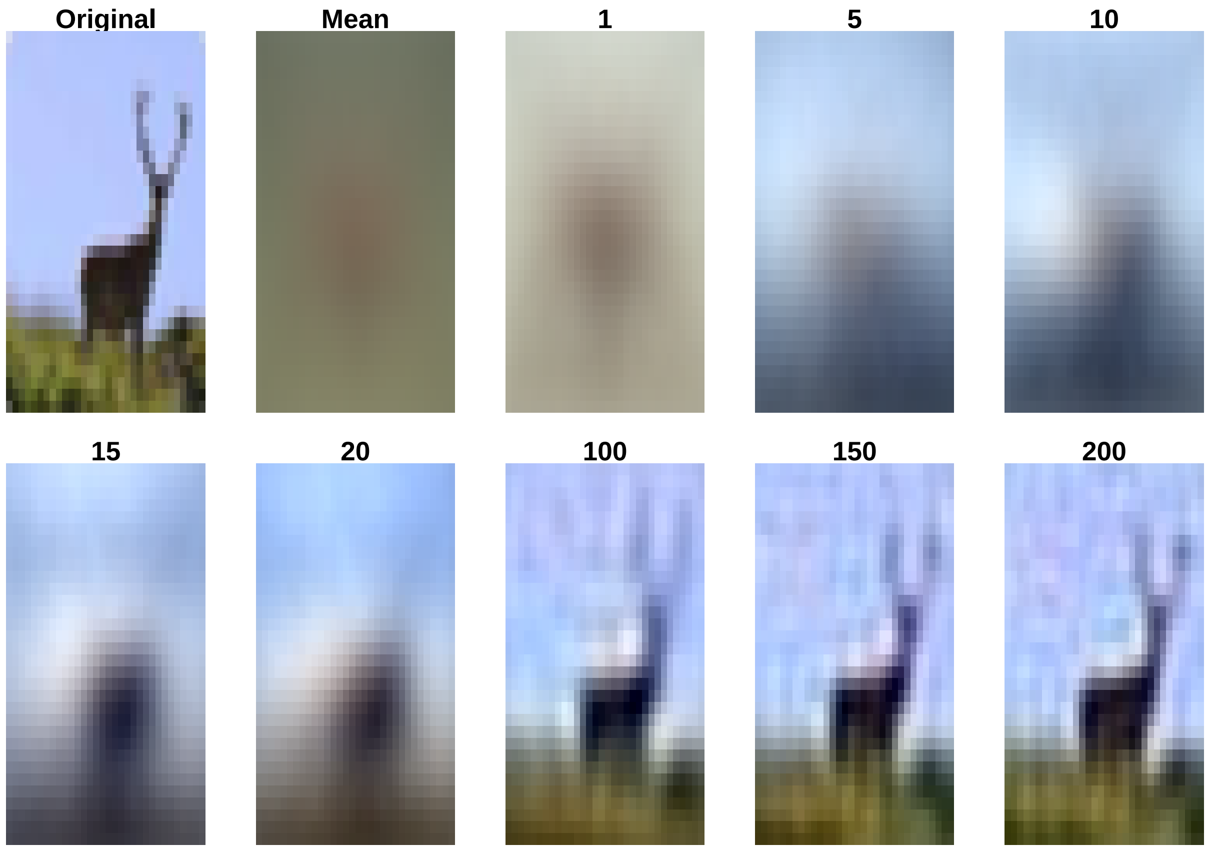
```
#Do PCA using Eigen value and vector
eigen_pca = function (data) {
  mean = colSums(data)
  covmat = cov(data)
  eigen = eigen(covmat, symmetric = TRUE)

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

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

  return (x)
}
```

Let's do PCA for each category.

```r
t_start = Sys.time()
all_pca = list()
pb = txtProgressBar(min = 0, max = label_count, style = 3)

for (i in 1:label_count) {
  all_pca[[i]] = eigen_pca(all_images[[i]][, -3073])
  setTxtProgressBar(pb, i)
}
cat(paste("\nTime Spent:", difftime(Sys.time(), t_start)))
save(all_pca, file="saved/pca.data")
```

To illustrate how well low-dimensional representation works, we approximate a sample image (Dear) by the mean and some principal components. Notice how good the approximation becomes with relatively few components (100 vs. 3072).

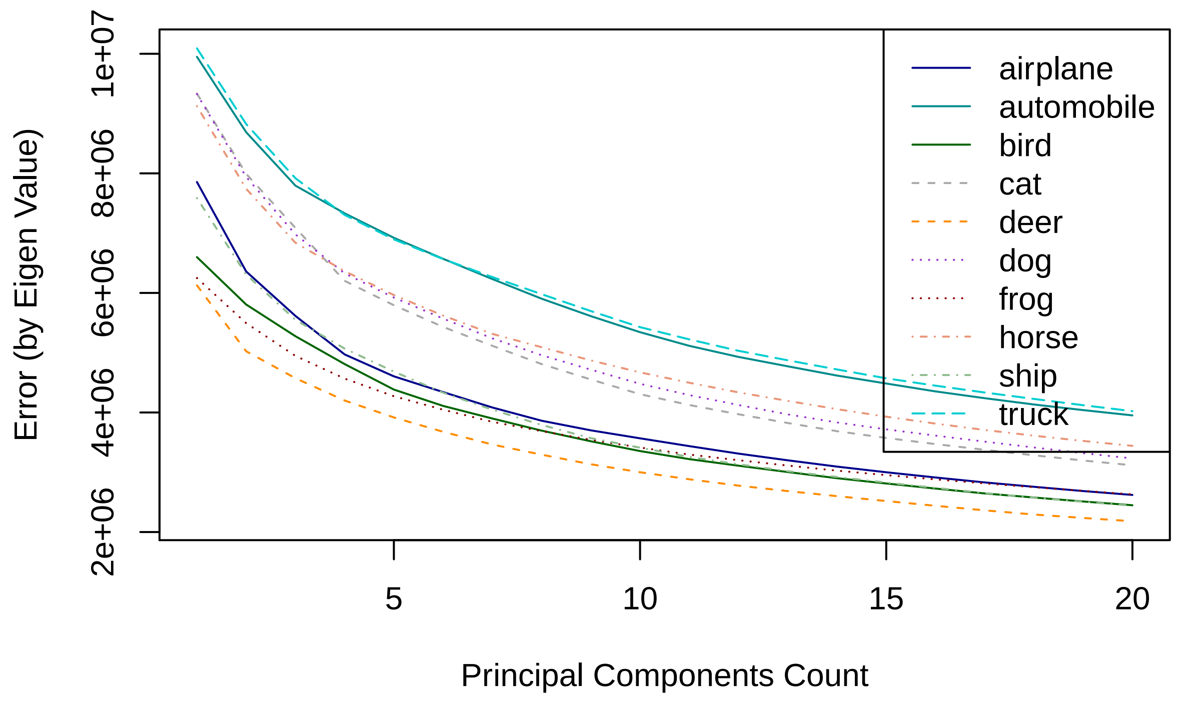
```r
sample_index = 311
sample_label = 5
par(mfrow=c(2,5))
par(mar = c(1,1,1,1))

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



Now, let's calculate and plot the error. We know that the error is the sum of the diagonal elements of the covariance matrix as:

$$\sum_{j>s}^{j=d} var\left(\left\{r^{(j)}\right\}\right)$$

```r
colors = c("darkblue", "darkcyan", "darkgreen", "darkgrey", "darkorange", "darkorchid",
           "darkred", "darksalmon", "darkseagreen", "darkturquoise")
line_types = c(1, 1, 1, 2, 2, 3, 3, 4, 4, 5)

var_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 + 1):image_bytes])
  }
  if(max(e) > max_error) max_error = max(e)
  if(min(e) < min_error) min_error = min(e)
  var_error[[i]] = e
}

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

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

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

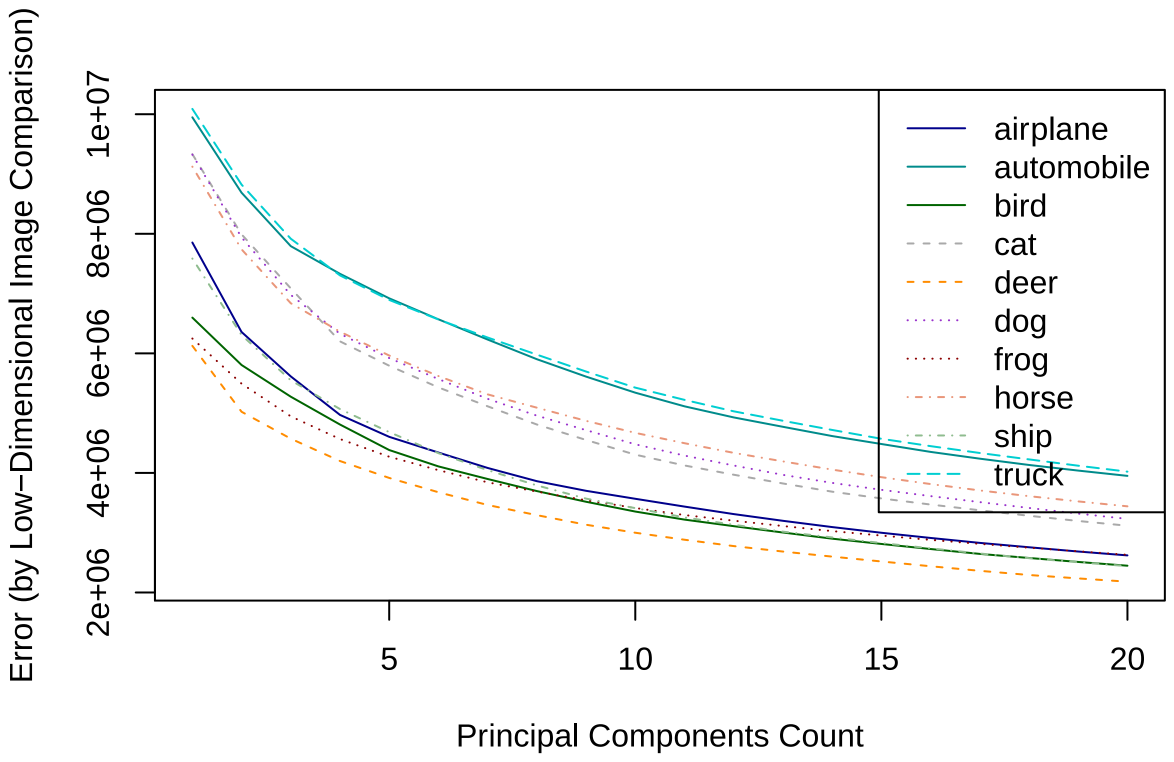


Alternatively, we can also compute the error by making low-dimensional representation for all of the images in a given category. Then, for each image, find the per-pixel differences between the reconstruction and its original image. Square those differences individually and sum them over the whole image. Finally, calculate the mean of this value over the whole category.

```
pca_image_error = function(images, pc, mean, k) {
  diff = apply(images, 1, function(x){
      return (x[-3073] - cld(pc, mean, x[-3073], k))}
    )
  return (mean(apply(diff, 2, function(v){return (crossprod(v, v))})))
}
```

```
t_start = Sys.time()
image_error = list()
e = rep(0, pc_count)
max_error = 0
min_error = 1e+10
pb = txtProgressBar(min = 0, max = label_count * pc_count, style = 3)

for (i in 1:label_count){
  for (k in 1:pc_count){
    e[k] = pca_image_error(all_images[[i]], all_pca[[i]]$pc, all_pca[[i]]$mean, k)
    setTxtProgressBar(pb, (i - 1) * pc_count + k)
  }
  if(max(e) > max_error) max_error = max(e)
  if(min(e) < min_error) min_error = min(e)
  image_error[[i]] = e
}

cat(paste("\nTime Spent:", difftime(Sys.time(), t_start)))
save(image_error, file="saved/image_error.data")
```

```
plot(image_error[[1]], type = "l", xlab="Principal Components Count",
     ylab="Error (by Low-Dimensional Image Comparison)", ylim = c(min_error, max_error),
     col=colors[1], lty = line_types[1])

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

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



We can check and conclude the results are almost identical between 2 approaches.

1.2

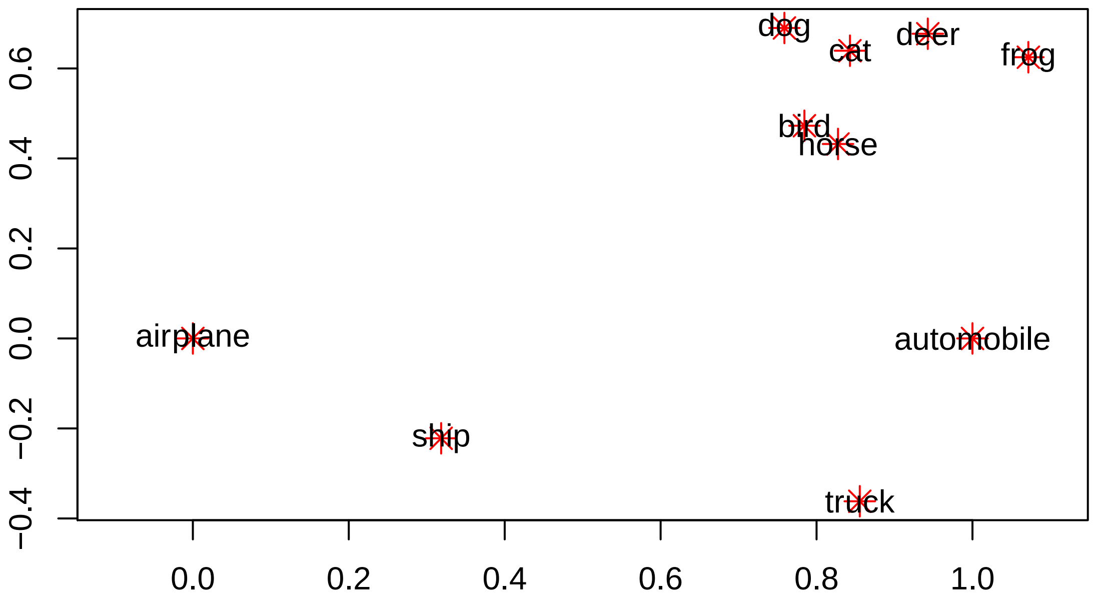
Compute the distances between mean images for each pair of classes. Use principal coordinate analysis to make a 2D map of the means of each categories. For this exercise, compute distances by thinking of the images as vectors.

We follow *Procedure 4.3* on textbook page 82 to calculate 2D coordinates.

```
MDS = function(D, k) {
  size = dim(D)[1]

  A = diag(1, size, size) - matrix(1, size, size) / size
  W = - 1/2 * A %*% D %*% t(A)
  e = eigen(W, symmetric = TRUE)

  return (e$vectors[,1:k] %*% diag(sqrt(e$values[1:k])))
}

#Offset the results so that category 1 is at the origin, rotate and scale
#the results so that category 2 is at (1,0);
transform_2d = function(V) {
  V1 = t(t(V) - V[1,])

  theta = asin(abs(V1[2,1])/sqrt(V1[2,1]^2 + V1[2,2]^2)) + pi/2
  r  = matrix(c(cos(theta),sin(theta),-sin(theta),cos(theta)), 2, 2)  #rotation matrix
  V2 = V1 %*% r
  return (V2 / V2[2,1])
}

#Calculate distance of mean images
D = matrix(0, nrow = label_count, ncol = label_count)

for (i in 1: (label_count - 1)) {
  for (j in (i + 1) : label_count){
    diff = (all_pca[[i]]$mean - all_pca[[j]]$mean)
    D[i, j] = sqrt(crossprod(diff, diff))
    D[j, i] = D[i, j]
  }
}
print(D)
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,]    0.0000 1683.6354 1605.0243 1905.5353 2148.7634 1965.2215
##  [2,] 1683.6354    0.0000  886.2367 1027.6498 1143.0814 1216.0794
##  [3,] 1605.0243  886.2367    0.0000  517.3115  601.2503  701.4682
##  [4,] 1905.5353 1027.6498  517.3115    0.0000  469.7917  412.1817
##  [5,] 2148.7634 1143.0814  601.2503  469.7917    0.0000  617.6971
##  [6,] 1965.2215 1216.0794  701.4682  412.1817  617.6971    0.0000
##  [7,] 2445.6797 1191.1920  913.7475  677.4920  460.5109  828.5811
##  [8,] 1663.6459  950.7861  418.2763  596.3767  684.3469  843.6721
##  [9,]  945.5411 1303.4665 1557.7150 1851.2145 2065.6217 1897.5918
## [10,] 1449.0949  949.9958 1416.6747 1676.4679 1830.7409 1880.2438
##            [,7]      [,8]      [,9]     [,10]
##  [1,] 2445.6797 1663.6459  945.5411 1449.0949
##  [2,] 1191.1920  950.7861 1303.4665  949.9958
##  [3,]  913.7475  418.2763 1557.7150 1416.6747
##  [4,]  677.4920  596.3767 1851.2145 1676.4679
##  [5,]  460.5109  684.3469 2065.6217 1830.7409
##  [6,]  828.5811  843.6721 1897.5918 1880.2438
##  [7,]    0.0000  948.7040 2249.1998 1913.2409
##  [8,]  948.7040    0.0000 1660.2681 1347.3341
##  [9,] 2249.1998 1660.2681    0.0000 1066.9416
## [10,] 1913.2409 1347.3341 1066.9416    0.0000
```

```
V = transform_2d(MDS(D, 2))

plot(V, xlab = "", ylab = "", col="Red", pch = 8, cex = 1.5, xlim = c(-0.1, 1.1))
for(i in 1:label_count){
  text(V[i,1], V[i,2], label_names[i,1], offset = 0)
}
```



It's amazing to see how similar objects close to each other! For example: dog, cat, deer, frog, gird and horse are all animals, they all gathered in top right corner. Automobile and truck are all vehicles and are closely positioned in the plot.

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.

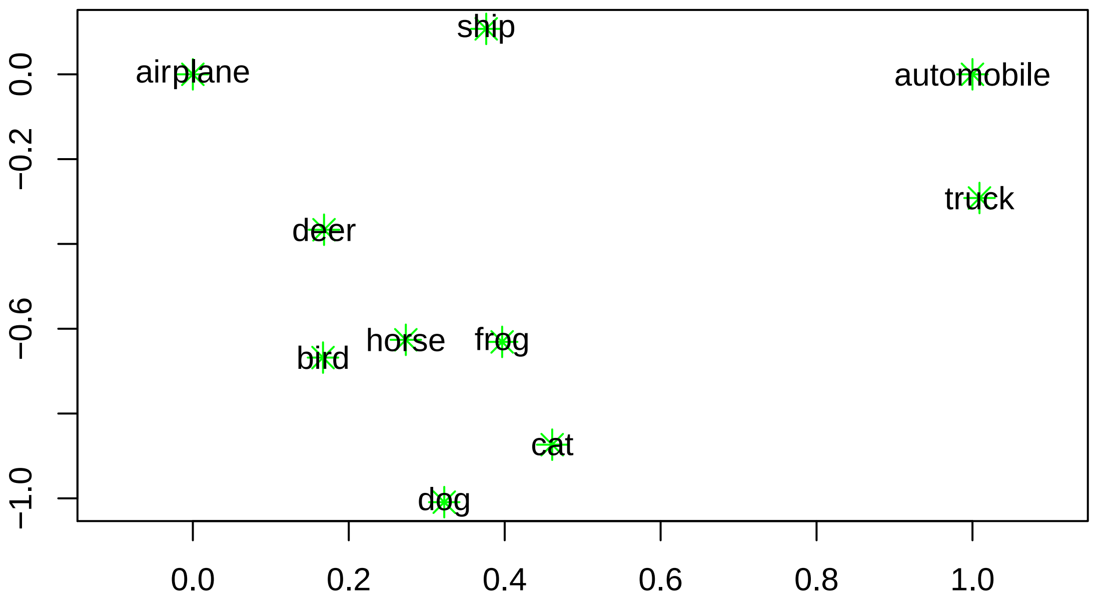
```
t_start = Sys.time()
#Calculate similarity of every classes
S = matrix(0, nrow = label_count, ncol = label_count)
```

```r
pb = txtProgressBar(min = 0, max = sum(1:10), style = 3)
pbc = 1
for (i in 1: label_count) {
  for (j in i : label_count){
    if(i != j) {
      error_1 = pca_image_error(all_images[[i]], all_pca[[j]]$pc,
                               all_pca[[i]]$mean, pc_count)
      error_2 = pca_image_error(all_images[[j]], all_pca[[i]]$pc,
                                 all_pca[[j]]$mean, pc_count)
      S[i, j] = (error_1 + error_2) / 2
      S[j, i] = S[i, j]
    } else{
      S[i, j] = image_error[[i]][pc_count]  #Use the previously computed error
    }

    setTxtProgressBar(pb, pbc)
    pbc = pbc + 1
  }
}
cat(paste("\nTime Spent:", difftime(Sys.time(), t_start)))
save(S, file="saved/similarity.error")
```

```r
print(S)
```

```
##          [,1]    [,2]    [,3]    [,4]    [,5]    [,6]    [,7]    [,8]
##  [1,] 2620501 3723676 2794365 3306932 2554925 3399592 2950223 3394717
##  [2,] 3723676 3950676 3699977 4039273 3450549 4211172 3685206 4230300
##  [3,] 2794365 3699977 2447698 2939377 2423654 2967088 2685785 3199991
##  [4,] 3306932 4039273 2939377 3116479 2889583 3262798 3028700 3546316
##  [5,] 2554925 3450549 2423654 2889583 2180391 2954958 2553877 3041230
##  [6,] 3399592 4211172 2967088 3262798 2954958 3231113 3100441 3610865
##  [7,] 2950223 3685206 2685785 3028700 2553877 3100441 2630244 3341395
##  [8,] 3394717 4230300 3199991 3546316 3041230 3610865 3341395 3441091
##  [9,] 2715857 3489125 2813363 3202341 2543500 3386717 2873799 3386206
## [10,] 3794218 4134930 3634776 3897344 3441359 4075472 3648283 4139677
##          [,9]   [,10]
##  [1,] 2715857 3794218
##  [2,] 3489125 4134930
##  [3,] 2813363 3634776
##  [4,] 3202341 3897344
##  [5,] 2543500 3441359
##  [6,] 3386717 4075472
##  [7,] 2873799 3648283
##  [8,] 3386206 4139677
##  [9,] 2440635 3541624
## [10,] 3541624 4021094
```

```r
V = transform_2d(MDS(S, 2))

plot(V, xlab = "", ylab = "", col="Green", pch = 8, cex = 1.5, xlim = c(-0.1, 1.1))
for(i in 1:label_count){
  text(V[i,1], V[i,2], label_names[i,1], offset = 0)
}
```



Compare this map to the map in the previous exercise, in general, they are similar in terms of similar categories positioned close to each other. For example: dog, cat, deer, frog, gird and horse are grouped together. Automobile and truck and are closely positioned in the plot.

However, if we check what exactly one close to which one, they are quite different. For example: *airplane* is close to *deer* for all animals in this map compared with *automobile* is close to *horse* in the previous map. This is because the similarity values we calculated as: airplane vs. deer is 2.5549247×10^6 and automobile vs. horse is 4.2302999×10^6. The 2D low-dimension reconstruction of this similarity matrix reflect such differences.