HUDM6122 Homework_04

Chenguang Pan

2023-03-01

0.1 Ex 4.1

Consider 51 objects O1, . , O51 assumed to be arranged along a straight line with the jth object being located at a point with coordinate j. Define the similarity sij between object i and object j as... and then apply classical multidimensional scaling to the resulting dissimilarity matrix. Explain the shape of the derived two-dimensional solution.

MY SOLUTION:
First, I define the function of dissimilarities δ_{ij} as follows, where the input paramter X is a n*1 matrix that contains all the coordination of 51 objects.

```
> D_dis <- function(X) {
+   n <- dim(X)[1] # to have the length of any input n*1 matrix
+   S <- matrix(0, n, n) # to make a n*n empty matrix
+   D <- matrix(0, n, n) # to make a n*n empty matrix
+   # to make the similarity matrix by conditions
+   for (i in c(1:n)) {
+     for (j in c(1:n)){
+       if (i == j){S[i,j] <- 9}
+       else if (abs(i-j) >= 1 & abs(i-j) <= 3){S[i,j] <- 8}
+       else if (abs(i-j) >= 4 & abs(i-j) <= 6){S[i,j] <- 7}
+       else if (abs(i-j) >= 7 & abs(i-j) <= 9){S[i,j] <- 6}
+       else if (abs(i-j) >= 10 & abs(i-j) <= 12){S[i,j] <- 5}
+       else if (abs(i-j) >= 13 & abs(i-j) <= 15){S[i,j] <- 4}
+       else if (abs(i-j) >= 16 & abs(i-j) <= 18){S[i,j] <- 3}
+       else if (abs(i-j) >= 19 & abs(i-j) <= 21){S[i,j] <- 2}
+       else if (abs(i-j) >= 22 & abs(i-j) <= 24){S[i,j] <- 1}
+       else if (abs(i-j) >= 25){S[i,j] <- 0}
+     }
+   } # similarity matrix finished!
+   # using the elements in the Similarity matrix to generate Dissimilarities Matrix
+   for (i in c(1:n)) {
+     for (j in c(1:n)) {
+       D[i,j] <- sqrt(S[i,i] + S[j,j] - 2*S[i,j])
+     }
+   } # dissimilarity matrix finished!
+   return(D)
+ }
```

Next, I randomly generate a n*1 matrix with 51 integers by using `sample()` function. And plug this vector to the dissimilarity function above.

```
> obs_ <- function(n) {
+   number_vec <- c(1:n)
+   # change the number array to matrix
+   return(matrix(number_vec,n,1))
+ }
```

The functions above looks good. I try to generate 51 observations and plug them into the dissimilarity matrix function to get the required D matrix.

```
> # generate 51 observations
> observations <- obs_(51)
> # plug the n*1 matrix into the dissimilarity martix function
> D <- D_dis(observations)
> # select a part of the dissimilarity matrix
> D[1:5,1:5]
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000000 1.414214 1.414214 1.414214 2.000000
[2,] 1.414214 0.000000 1.414214 1.414214 1.414214
[3,] 1.414214 1.414214 0.000000 1.414214 1.414214
[4,] 1.414214 1.414214 1.414214 0.000000 1.414214
[5,] 2.000000 1.414214 1.414214 1.414214 0.000000
```

This dissimilarity matrix looks good. Then I run the classical multidimensional scaling to this resulting matrix. Note, this is a non-Euclidean case. Some of the eigenvalue may be negative.

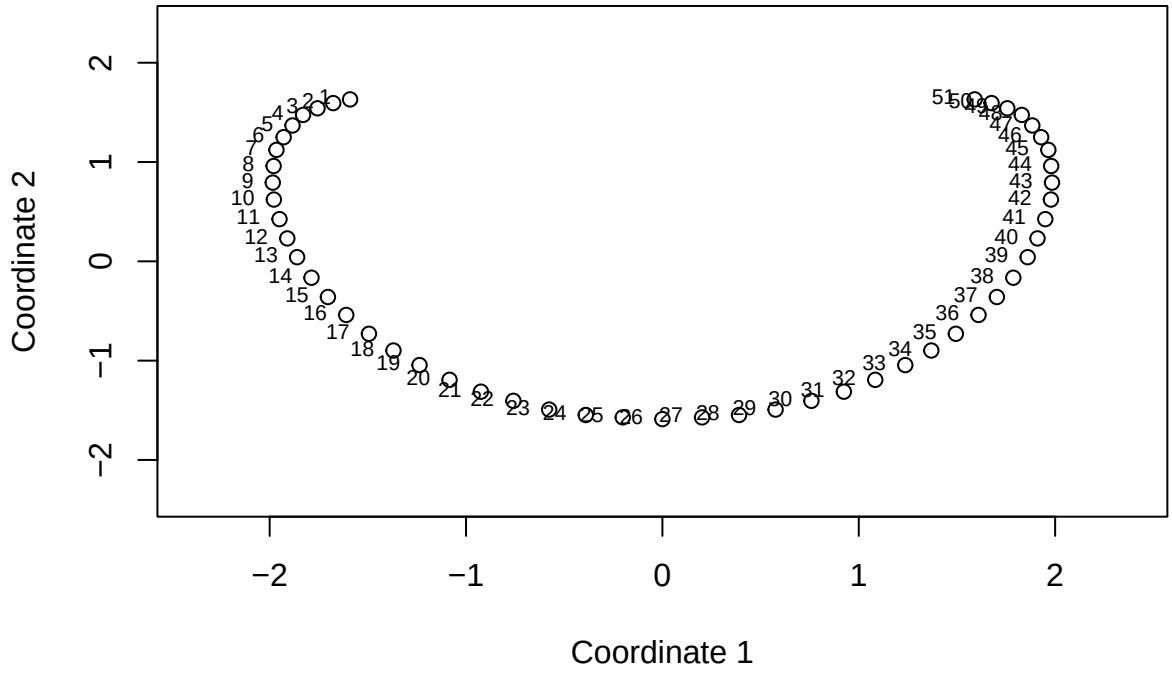
```
> d_mds <- cmdscale(D, k=50, eig = T)
> lam <- d_mds$eig
> # d_mds$points
> cumsum(abs(lam))/sum(abs(lam))
 [1] 0.4428488 0.6744323 0.7382541 0.7657231 0.7924526 0.8183668 0.8430174
 [8] 0.8615740 0.8736449 0.8845189 0.8953701 0.9048445 0.9116561 0.9167756
[15] 0.9217895 0.9267112 0.9314246 0.9355091 0.9389073 0.9422424 0.9455372
[22] 0.9486160 0.9513996 0.9540916 0.9565647 0.9589956 0.9611009 0.9630147
[29] 0.9646832 0.9663473 0.9677584 0.9691622 0.9704291 0.9714991 0.9725150
[36] 0.9734831 0.9744191 0.9753283 0.9762204 0.9771018 0.9775266 0.9778771
[43] 0.9778771 0.9782091 0.9785814 0.9796588 0.9808172 0.9831840 0.9856796
[50] 0.9927144 1.0000000
> cumsum(abs(lam^2))/sum(abs(lam^2))
 [1] 0.7608520 0.9689196 0.9847222 0.9876495 0.9904214 0.9930267 0.9953842
 [8] 0.9967201 0.9972854 0.9977441 0.9982010 0.9985492 0.9987292 0.9988309
[15] 0.9989284 0.9990224 0.9991086 0.9991733 0.9992181 0.9992613 0.9993034
[22] 0.9993402 0.9993702 0.9993983 0.9994221 0.9994450 0.9994622 0.9994764
[29] 0.9994872 0.9994979 0.9995057 0.9995133 0.9995195 0.9995240 0.9995280
[36] 0.9995316 0.9995350 0.9995382 0.9995413 0.9995443 0.9995450 0.9995455
[43] 0.9995455 0.9995459 0.9995465 0.9995510 0.9995562 0.9995779 0.9996021
[50] 0.9997941 1.0000000
```

These values suggest that the first two coordinates will give an adequate representation of the simulated dissimilarity distances. Then, I make the scatter plot using the first two scores.

```
> lim <- range(d_mds$points[,1] * (-1)) * 1.2
>
```

```
> plot(d_mds$points[,1]*(-1), d_mds$points[,2]*(-1),
+      xlab = "Coordinate 1", ylab = "Coordinate 2",
+      xlim = lim, ylim = lim)
```

```
> lim <- range(d_mds$points[,1]) * 1.2
> plot(d_mds$points[,1], d_mds$points[,2],
+      xlab = "Coordinate 1", ylab = "Coordinate 2",
+      xlim = lim, ylim = lim)
> text(d_mds$points[,1], d_mds$points[,2], labels = c(1:51), pos=2,cex = 0.7)
```



This two-dimensional plot has the symmetrical shape, which is reasonable since the mid-point,i.e., the 26th point, has the same dissimilarity with both the points on its left and right. Therefore, if we project all point on y-axis, it actually represent the dissimilarity degree between the each point and the 26th point.

0.2 Ex. 4.2

Write an R function to calculate the chi-squared distance matrices for both rows and columns in a two-dimensional contingency table.

MY SOLUTION

Based on the definition of chi-squared distance matrices, I write the function as follows:

```
> chi_squared_dist_matrices <- function(tbl){
+   # Calculate row sums and column sums
+   row_sums <- apply(tbl, 1, sum)
```

```
+   col_sums <- apply(tbl, 2, sum)
+   # Calculate total count and expected counts
+   total_count <- sum(tbl)
+
+   # to have the dim of the input matrix
+   c <- ncol(tbl)
+   r <- nrow(tbl)
+   # make a empty matrix to load all elements
+   col_d_matrix <- matrix(0,c,c)
+
+   # using loop to write the each ij entries into this matrix
+   for (i in c(1:c)) {
+     for (j in c(1:c)) {
+       d_ij <- 0
+       for (k in c(1:r)) {
+         p_k_dot <- row_sums[k]/total_count
+         p_k_i <- tbl[k,i]/col_sums[i]
+         p_k_j <- tbl[k,j]/col_sums[j]
+         d_ij <- d_ij + (1/p_k_dot)*(p_k_i-p_k_j)^2
+       }
+       col_d_matrix[i,j] <- d_ij
+     }
+   }
+   # make a empty matrix to load all elements
+   row_d_matrix <- matrix(0,r,r)
+
+   # using loop to write the each ij entries into this matrix
+   for (i in c(1:r)) {
+     for (j in c(1:r)) {
+       d_ij <- 0
+       for (k in c(1:r)) {
+         p_dot_k <- col_sums[k]/total_count
+         p_i_k <- tbl[i,k]/row_sums[i]
+         p_j_k <- tbl[j,k]/row_sums[j]
+         d_ij <- d_ij + (1/p_dot_k)*(p_i_k-p_j_k)^2
+       }
+       row_d_matrix[i,j] <- d_ij
+     }
+   }
+   return(list(col_d_matrix, row_d_matrix))
+ }
```

This function return a list that contains both columns distance matrix and rows distance matrix. By using the index, like `[[1]]`or `[[2]]`, to extract the columns or rows distance matrix, respectively. Next, I created a two-dimensional contigency table to test this function.

```
> # Create a contingency table
> tbl <- table(c("A", "A", "B", "B"), c("X", "Y", "X", "Z"))
> tbl

   X Y Z
  A 1 1 0
  B 1 0 1
```

```
> # plug this simulated table into the function above
> dist_m <-chi_squared_dist_matrices(tbl)
> # the columns distance matrix is
> dist_m[[1]]
     [,1] [,2] [,3]
[1,]    0    1    1
[2,]    1    0    4
[3,]    1    4    0
> # the row distance matrix is
> dist_m[[2]]
     [,1] [,2]
[1,]    0    1
[2,]    1    0
```

It looks good!

0.3 Ex. 4.3

In Table 4.7 (from Kaufman and Rousseeuw 1990), the dissimilarity matrix of 18 species of garden flowers is shown. Use some form of multidimensional scaling to investigate which species share common properties.

MY SOLUTION First, import the dataset and using classical multidimensional scaling to investigate the features of flowers.

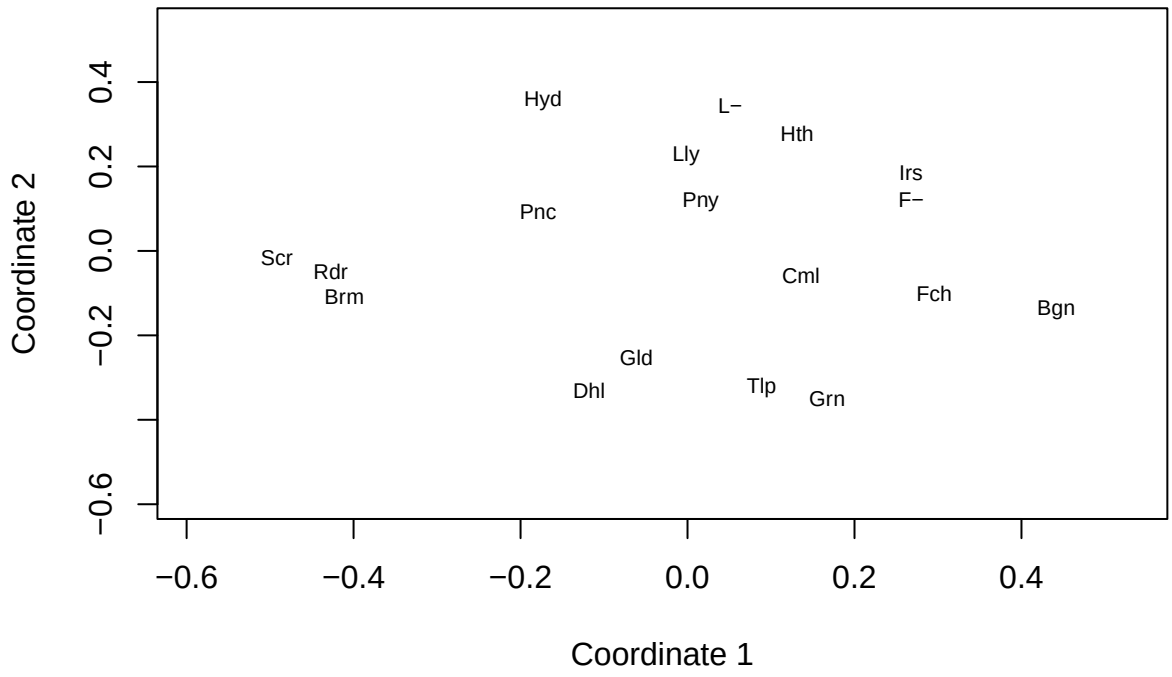
```
> library("MVA")
> data(gardenflower)
> # run the classical multidimensional scaling
> d_mds <- cmdscale(gardenflowers, k = 17, eig=T)
> # get the eigenvalues
> lam <- d_mds$eig
> lam
 [1]  1.173085e+00  8.944353e-01  5.732009e-01  4.843006e-01  2.638516e-01
 [6]  2.298165e-01  8.383417e-02  6.645861e-02  2.984017e-02 -5.551115e-17
[11] -2.147959e-02 -4.094094e-02 -4.366468e-02 -8.867401e-02 -1.045334e-01
[16] -1.275889e-01 -1.714295e-01 -2.045124e-01
```

Some of the eigenvalue are negative, which means the distance is not Euclidean distance.

```
> cumsum(abs(lam))/sum(abs(lam))
 [1] 0.2549273 0.4493002 0.5738645 0.6791096 0.7364481 0.7863903 0.8046086
 [8] 0.8190510 0.8255356 0.8255356 0.8302034 0.8391005 0.8485894 0.8678594
[15] 0.8905760 0.9183028 0.9555567 1.0000000
> cumsum(abs(lam)^2)/sum(abs(lam)^2)
 [1] 0.4611160 0.7291863 0.8392805 0.9178730 0.9412006 0.9588982 0.9612532
 [8] 0.9627331 0.9630315 0.9630315 0.9631861 0.9637478 0.9643866 0.9670214
[15] 0.9706829 0.9761377 0.9859851 1.0000000
```

The result suggests that the first two to three coordinate will give an adequate representation of the observed distance. But I choose the first two coordinates for a better look.

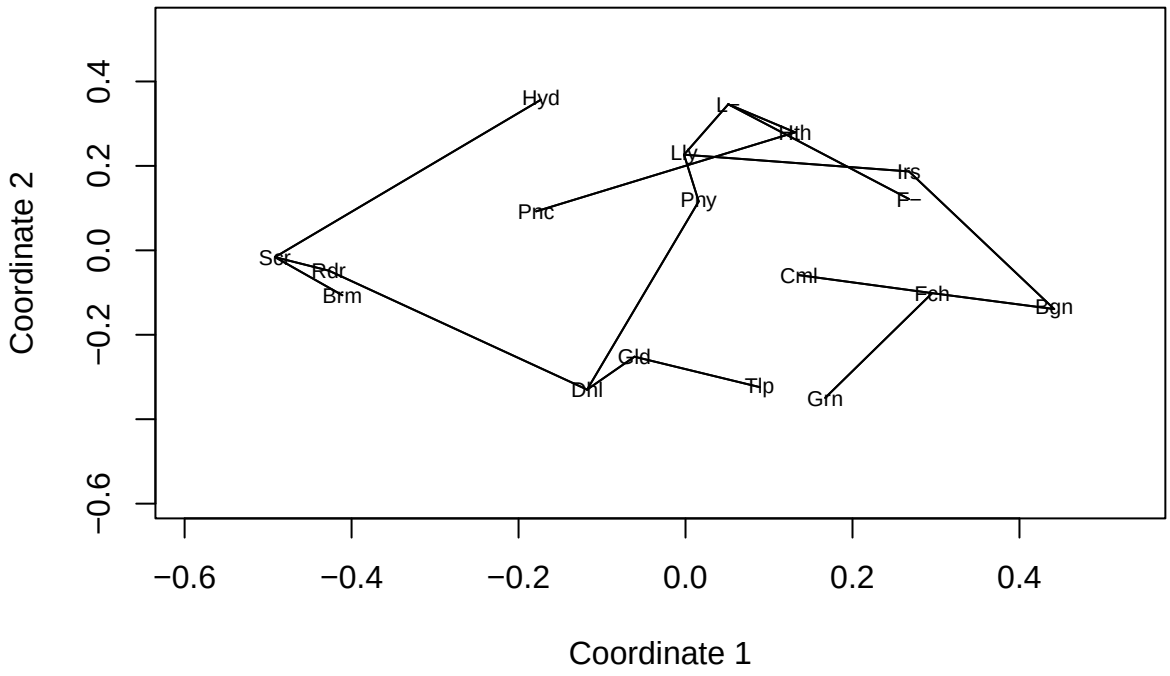
```
> flower_matrix <- as.matrix(gardenflowers)
> colnames(flower_matrix)=c("Bgn", "Brm","Cml","Dhl","F-","Fch","Grn","Gld",
+                           "Hth","Hyd","Irs","Lly","L-","Pny","Pnc",
+                           "Rdr","Scr","Tlp")
> lim <- range(d_mds$points[,1] * 1.2)
> plot(d_mds$points[,1], d_mds$points[,2],
+      xlab = "Coordinate 1", ylab = "Coordinate 2",type="n",
+      xlim = lim, ylim = lim)
>
> text(d_mds$points[,1], d_mds$points[,2],
+      labels = colnames(flower_matrix),
+      cex = 0.7)
```



Although the names is quite long and therefore is a little hard to tell, one can still find that `Scr/Scotch rose`, `Rdr/Red rose`, and `Brm/Broom` have the very similar properties since they clustered around each other comparing to others kinds. However, the two dimensional plot sometimes can not adequately represent the actual relations among the points. Here, I continus to construct a minimum spanning tree to explore the similarities.

```
> library("ape")
> st <- mst(flower_matrix)
> x <- d_mds$points[,1]
> y <- d_mds$points[,2]
> plot(x, y,
+      xlab = "Coordinate 1", ylab = "Coordinate 2",type="n",
+      xlim = lim, ylim = lim)
```

```
> for (i in 1:nrow(flower_matrix)) {
+   w1 <- which(st[i,]==1)
+   segments(x[i],y[i],x[w1],y[w1])
+ }
>
> text(d_mds$points[,1], d_mds$points[,2],
+      labels = colnames(flower_matrix),
+      cex = 0.7)
```



From the minimum spanning tree, we can see that, comparing to `hyd`'s neighbor like `Pnc`,`L-`, and `Lly`, `hyd` is more similar to the cluster of `Scr/Scotch rose`, `Rdr/Red rose`, and `Brm/Broom`. In addition, the `Dhl`, `Gld`, and `Tlp` are more likely to be similar with each other. The `Cml`, `Fch`, `Bgn`, and `Grn` are similar with each other. The rest of the kinds can be clustered to a similar group.