Multidimensional Scaling (MDS)

Alireza Sheikh-Zadeh, PhD

Constructing a map from a distance matrix

Multidimensional scaling (MDS) is designed to construct a map showing the relationship between several objects, given only a table of distances between the objects (distance matrix).

Multidimensional scaling is essentially a data reduction technique. It aims to find a set of points in low dimension (e.g., two dimensions) that approximate the possibly high-dimensional configuration represented by the original distance matrix. The classical multidimensional scaling seeks to represent a proximity (distance) matrix by a simple geometrical model or map.

The usefulness of multidimensional scaling comes from the fact that situations often arise where the underlying relationship between objects is unknown, but a distance matrix can be estimated. For example, in psychology, subjects may be able to assess how similar or different individual pairs of objects are without drawing an overall picture of the relationships between the objects. Multidimensional scaling can then provide the picture.

Classical multidimensional scaling

Classical multidimensional scaling starts with a matrix of distance D between n objects that has d_{ij}, the Euclidean distance from object i to object j. We have learned how to calculate Euclidean distances from the original $n \times q$ data matrix; classical multidimensional scaling is essentially concerned with the reverse problem: given the distances, how do we find data matrix in two or three dimensions?

In summary:

- For MDS, we do not use ordinary data; all we need is a distance matrix.
- MDS converts the distance matrix to X and Y coordinates (e.g., for 2D) as a tool for graphical representation.
- MDS tries to develop a 2D representation such that the Euclidean distance in this 2D representation matches the distances in your matrix.
- Classical scaling can be shown to be equivalent to principal components analysis. If distances are Euclidean distance between the points, the coordinates of the map are the principal components. **The goal is that the Euclidean distance between objects in the map is as close as possible to the distance in the given distance matrix.**

- MDS works on any distance matrix, for example, Euclidean, Manhattan, or any other (even subjective) **positive zero-diagonal symmetric matrices**.

Example: Suppose we have a small data of 10 rows and 5 columns.

```
data <- matrix(c(
  3, 5, 6, 1, 4, 2, 0, 0, 7, 2,
  4, 1, 2, 1, 7, 2, 4, 6, 6, 1,
  4, 1, 0, 1, 3, 5, 1, 4, 5, 4,
  6, 7, 2, 0, 6, 1, 1, 3, 1, 3,
  1, 3, 6, 3, 2, 0, 1, 5, 4, 1), ncol = 5)
```

We apply classical multidimensional scaling to the Euclidean distance matrix of the data.

```
cmdscale(dist(data), k= 5, eig = T) # K is the number of coordinates.
```

```
## $points
##         [,1]    [,2]   [,3]   [,4]    [,5]
##  [1,] -1.604  2.3806  2.230 -0.366  0.1154
##  [2,] -2.825 -2.3094  3.952  0.342  0.3317
##  [3,] -1.691 -5.1397 -1.288  0.650 -0.0513
##  [4,]  3.953 -2.4323 -0.383  0.686 -0.0346
##  [5,] -3.598  2.7554  0.255  1.078 -1.2613
##  [6,]  2.952  1.3548  0.190 -2.821  0.1239
##  [7,]  3.469  0.7641 -0.302  1.637 -1.9421
##  [8,]  0.355  2.3141 -2.216  2.924  2.0045
##  [9,] -2.936 -0.0128 -4.312 -2.512 -0.1891
## [10,]  1.926  0.3253  1.873 -1.619  0.9030
##
## $eig
##  [1]  7.52e+01  5.88e+01  4.96e+01  3.04e+01  1.04e+01  3.37e-15  2.55e-15
##  [8]  3.76e-16 -4.08e-15 -6.04e-15
```

```
# You can compare results with the principle component scores using cov matri
x: princomp(data)$scores. The similarity happens only when we use Euclidean (
default) distance for the MDS.
```

'**cmdscale**()' represents principle coordinates. We can use the first two or three coordinates for mapping the data.

It is better to decide the appropriate number of coordinates based on the cumulative proportion of eigenvalues.

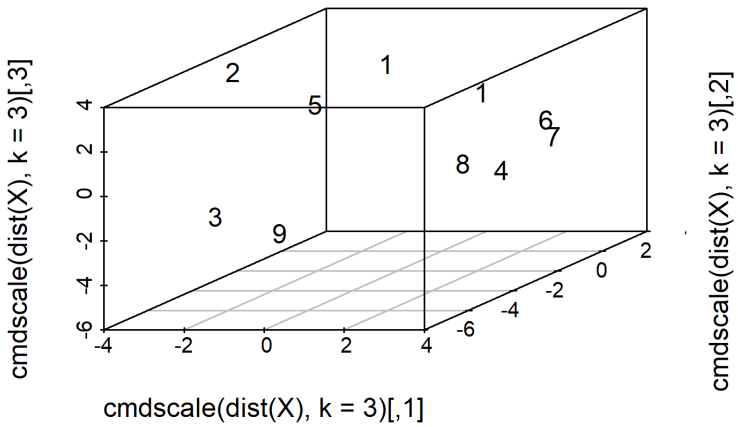
Note: Choose the number of coordinates so that the sum of the positive eigenvalues is approximately equal to the sum of all the eigenvalues (it is possible to have negative eigenvalues).

```
cmd <- cmdscale(dist(data), k= 5, eig = T)
cumsum(cmd$eig)/sum(cmd$eig)
```

```
##  [1] 0.335 0.597 0.818 0.954 1.000 1.000 1.000 1.000 1.000 1.000
```

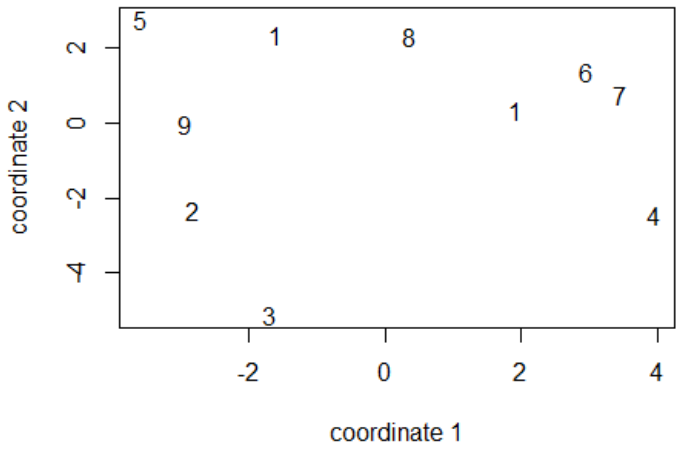
Since the first three coordinates explains more than 80% of eigenvalues, it is reasonable to choose the first three coordinates.

```
library("scatterplot3d")

scatterplot3d(cmdscale(dist(X), k=3), pch = rownames(as.data.frame(X)))
```



However, we use MDS for graphical representation, and the 2D plot is much easier to read.

```
plot(cmd, xlab = "coordinate 1", ylab = "coordinate 2", pch = rownames(as.dat
a.frame(X)))
```



The goal is to develop a 2D representation such that the Euclidean distances in this two-dimensional representation match the original distance matrix.

```
# 2d representation
cmd2 <- cmdscale(dist(data), k=2)
dist(cmd2) # compare this with original dist matrix. However, we see some err
ors; why?
```

```
##        1     2     3     4     5     6     7     8     9
## 2  4.846
## 3  7.521 3.049
## 4  7.351 6.779 6.259
## 5  2.030 5.124 8.122 9.162
## 6  4.670 6.841 7.983 3.917 6.699
## 7  5.324 7.004 7.841 3.233 7.343 0.785
## 8  1.959 5.611 7.729 5.956 3.978 2.769 3.479
## 9  2.739 2.299 5.276 7.302 2.846 6.045 6.452 4.030
## 10 4.084 5.432 6.553 3.422 6.035 1.454 1.604 2.535 4.874
```

```
dist(data)
```

```
##       1    2    3    4    5    6    7    8    9
## 2  5.20
## 3  8.37 6.08
## 4  7.87 8.06 6.32
## 5  3.46 6.56 8.37 9.27
## 6  5.66 8.43 8.83 5.29 7.87
## 7  6.56 8.60 8.19 3.87 7.42 5.00
## 8  6.16 8.89 8.37 6.93 6.00 7.07 5.74
## 9  7.42 9.06 6.86 8.89 6.56 7.55 8.83 7.42
## 10 4.36 6.16 7.68 4.80 7.14 2.65 5.10 6.71 8.00
```

The 3-dimensional representation will make a better match with the original distance matrix.

```
# 3d representation
cmd3 <- cmdscale(dist(data), k=3)
dist(cmd3)
```

```
##        1     2     3     4     5     6     7     8     9
## 2  5.143
## 3  8.303 6.063
## 4  7.802 8.047 6.324
## 5  2.832 6.318 8.268 9.184
## 6  5.096 7.807 8.119 3.959 6.699
## 7  5.895 8.195 7.903 3.234 7.364 0.926
## 8  4.859 8.339 7.785 6.232 4.683 3.668 3.971
## 9  7.092 8.578 6.081 8.291 5.381 7.537 7.597 4.543
## 10 4.100 5.816 7.276 4.100 6.248 2.224 2.703 4.811 7.875
```

```
dist(data) # Better match, but we lose the beauty of having 2D plot
```

```
##       1    2    3    4    5    6    7    8    9
## 2  5.20
## 3  8.37 6.08
## 4  7.87 8.06 6.32
## 5  3.46 6.56 8.37 9.27
## 6  5.66 8.43 8.83 5.29 7.87
## 7  6.56 8.60 8.19 3.87 7.42 5.00
## 8  6.16 8.89 8.37 6.93 6.00 7.07 5.74
## 9  7.42 9.06 6.86 8.89 6.56 7.55 8.83 7.42
## 10 4.36 6.16 7.68 4.80 7.14 2.65 5.10 6.71 8.00
```

We can apply MDS on other types of distance matrices such as Manhattan (taxi driving distance), but you may get negative eigenvalues. This does not make a significant issue; we still can decide the number of coordinates by looking at the absolute (or squared) cumulative proportion of eigenvalues.

```
cmd <- cmdscale(dist(data, method = "manhattan"), k= 5, eig = T)
round(cmd$eig,3)
```

```
##  [1] 280.7 249.4 228.9  92.5  42.5  22.0   0.0 -15.1 -28.0 -56.8
```

```
cumsum(abs(cmd$eig)^2)/sum(abs(cmd$eig)^2) # Again 3 coordinates is >.8, so a
better fit
```

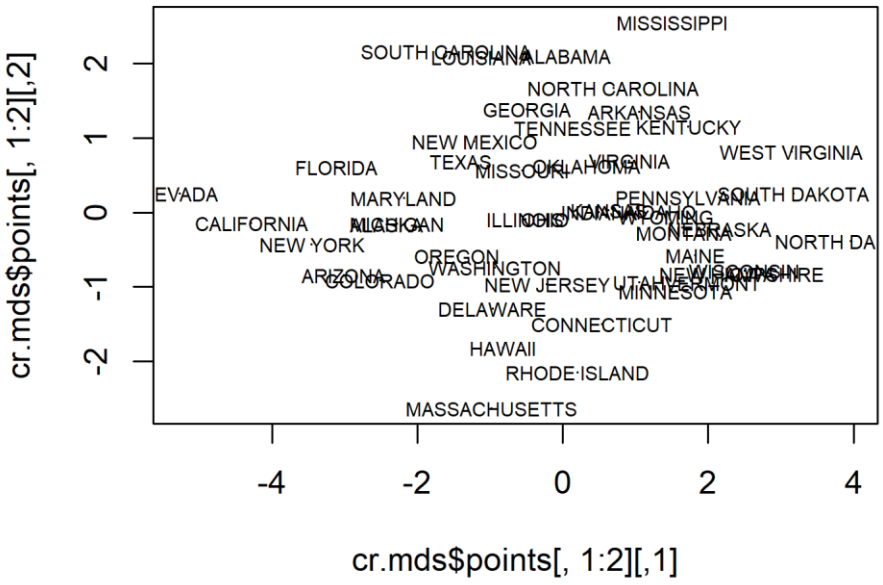
```
##  [1] 0.378 0.676 0.928 0.969 0.977 0.980 0.980 0.981 0.985 1.000
```

Example: Crime data, mapping the observations.

```
cr <- read.csv("https://rb.gy/wu8kvo", row.names = "STATE")

cr.mds = cmdscale(dist(scale(cr)), eig = T)

eignv <- abs(cr.mds$eig)

head(cumsum(eignv)/sum(eignv))
```

```
## [1] 0.588 0.765 0.868 0.914 0.951 0.982
```

```
plot(cr.mds$points[,1:2], pch = ".")
text(cr.mds$points[,1:2], labels = rownames(cr), cex = 0.6)
```



Question 1: What are the coordinates?

Question 2: Can we use MDS for mapping the variables, not the observations?