Statistics 5401
34. Multidimensional Scaling

Gary W. Oehlert
School of Statistics
313B Ford Hall
612-625-1557
gary@stat.umn.edu

Multidimensional scaling tries to find low dimensional representations of points, based originally on a matrix of distances between the points.

It is a cousin of principal components, but the original distances do not have to be Euclidean.

We have a distance matrix $\mathbf{D}$ $n \times n$. We want to choose a dimension k (typically $k = 2$) and construct an $(n \times k)$ matrix $\mathbf{X}$ so that the distances between the rows of $\mathbf{X}$ match the corresponding elements of $\mathbf{D}$.

Then distances we see when plotting points in $\mathbf{X}$ reflect the more complex, presumably high-dimensional distances coming from $\mathbf{D}$.

What we will actually try to do is match the squares of the distances between the rows with the squares of the distances in $\mathbf{D}$. The squared distance between $\vec{\mathbf{X}}_i$ and $\vec{\mathbf{X}}_j$ (d_{ij}^2) is

$$\begin{aligned} d_{ij}^2 &= \sum_{\ell=1}^{k}(X_{i\ell} - X_{j\ell})^2 \\ &= \sum_{\ell=1}^{k} X_{i\ell}^2 + \sum_{\ell=1}^{k} X_{j\ell}^2 - 2\sum_{\ell=1}^{k} X_{i\ell}X_{j\ell} \end{aligned}$$

If we make a matrix of the d_{ij}^2s, the i row contains an additive term of $\sum_{\ell=1}^{k} X_{i\ell}^2$, the j column contains an additive term of $\sum_{\ell=1}^{k} X_{j\ell}^2$, and the i, jth element contains an additive term of $-2\sum_{\ell=1}^{k} X_{i\ell}X_{j\ell}$.

If we take the matrix of squared distances d_{ij}^2, subtract the row means, then subtract the column means, and then take $-.5$ times the difference, we are left with a matrix with elements

$$\tilde{d}_{ij} = \sum_{\ell=1}^{k} X_{i\ell}X_{j\ell}$$

Look at this again, we get

$$\tilde{d}_{ij} = \sum_{\ell=1}^{k} X_{i\ell}X_{j\ell}$$

This expresses our (centered and rescaled) matrix of squared distances as a sum of outer products of the columns of $\mathbf{X}$.

Thus we can “recover” $\mathbf{X}$ from the (centered and rescaled) matrix of squared distances $\tilde{d}$ via SVD or an eigenvalue decomposition.

$$\tilde{d} = \mathbf{H}\Lambda\mathbf{H}'$$

so

$$\mathbf{X} = \mathbf{H}\Lambda^{1/2}$$

and

$$\tilde{d} = \mathbf{X}\mathbf{X}'$$

where $\Lambda^{1/2}$ is a diagonal matrix of the square roots of the eigenvalues of $\tilde{d}$.
Hold on! We haven't *really* recovered $\mathbf{X}$. Let $\mathbf{H}_k$ be any $k \times k$ orthogonal matrix. Note that

$$\mathbf{Y} = \mathbf{X}\mathbf{H}_k = \mathbf{H}\Lambda^{1/2}\mathbf{H}_k$$

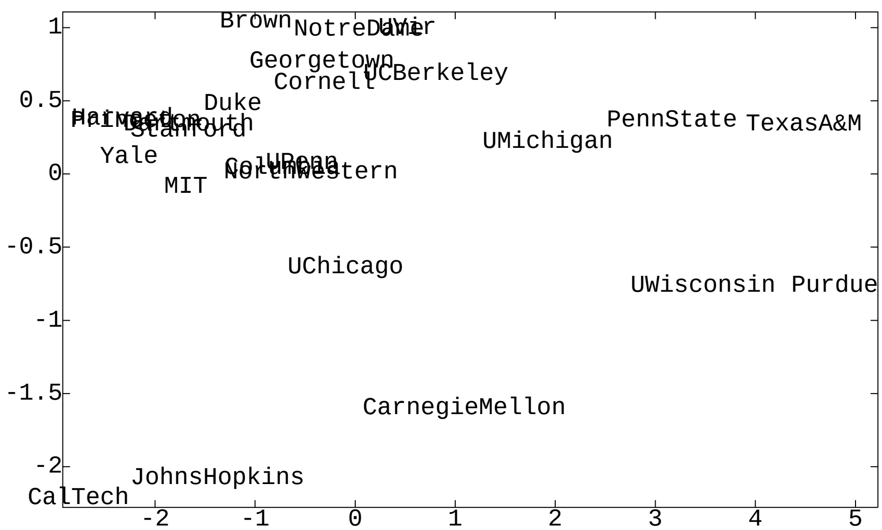
also satisfies

$$\tilde{d} = \mathbf{Y}\mathbf{Y}'$$

Thus we only recover $\mathbf{X}$ up to some rotation.
So how do we do multidimensional scaling. We're trying to find an $\mathbf{X}$ with distances between rows that match $\mathbf{D}$. So pretend that $\mathbf{D}$ really came from $\mathbf{X}$ and "recover" $\mathbf{X}$.
Square the elements in $\mathbf{D}$, subtract row means, subtract column means, multiply the difference by $-.5$, and then do an eigenvector/eigenvalue decomposition of the result. Rescale the first k eigenvectors by the square roots of the corresponding eigenvalues, and voila, we have
Classic (metric) Multidimensional Scaling.

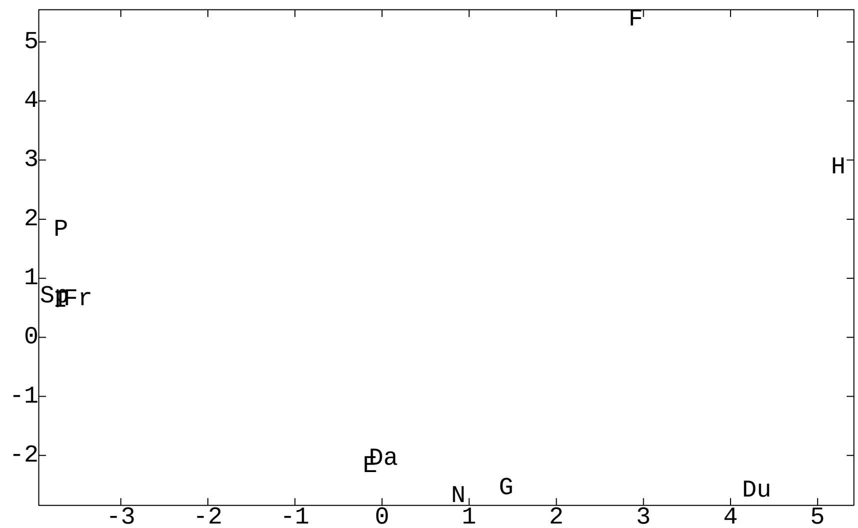
```
Cmd> readdata("",school,x1,x2,x3,x4,x5,x6)
Read from file "~/JW5data/T12-9.DAT"
Column 1 saved as factor school
Column 2 saved as REAL vector x1
Column 3 saved as REAL vector x2
Column 4 saved as REAL vector x3
Column 5 saved as REAL vector x4
Column 6 saved as REAL vector x5
Column 7 saved as REAL vector x6

Cmd> X <- hconcat(x1,x2,x3,x4,x5,x6)

Cmd> X <- X/describe(X,stddev:T)'

Cmd> dim(X)
(1)           25           6

Cmd> D2 <- matrix(rep(0,25*25),25)

Cmd> for(i,run(6)) {
D2 <- D2 + (X[,i]-X[,i]')^2
;;
}

Cmd> D2s <- D2

Cmd> D2s <- D2s - sum(D2s)/25

Cmd> D2s <- D2s - sum(D2s')'/25
```

```
Cmd> D2s <- -.5 * D2s

Cmd> eigenvals(D2s)
 (1)   110.69   18.884   6.8775   3.9307
 (5)   2.9833  0.63482   lots of 0s

Cmd> (110.69+18.884)/sum(eigenvals(D2s))
(1)      0.89982

Cmd> Y <- eigen(D2s)$vectors[,run(2)]*\
sqrt(eigenvals(D2s)[run(2)]')

Cmd> chplot(Y[,1],Y[,2]," ",xaxis:F,yaxis:F)

Cmd> addstrings(Y[,1],Y[,2],getlabels(school))
```




```
Cmd> s <- vecread("")
Read from file "~/JW5data/T12-4.DAT"

Cmd> S <- triunpack(s)

Cmd> D <- 10-S

Cmd> print(D,format:"f3.0",labels:F)
D:
  0   2   2   7   6   6   6   6   7   9   9
  2   0   1   5   4   6   6   6   7   8   9
  2   1   0   6   5   6   5   5   6   8   9
  7   5   6   0   5   9   9   9  10   8   9
```

```
  6   4   5   5   0   7   7   7   8   9   9
  6   6   6   9   7   0   2   1   5  10   9
  6   6   5   9   7   2   0   1   3  10   9
  6   6   5   9   7   1   1   0   4  10   9
  7   7   6  10   8   5   3   4   0  10   9
  9   8   8   8   9  10  10  10  10   0   8
  9   9   9   9   9   9   9   9   9   8   0

Cmd> D2 <- D^2

Cmd> D2s <- D2

Cmd> D2s <- D2s - sum(D2s)/11

Cmd> D2s <- D2s - sum(D2s')'/11

Cmd> D2s <- -.5 * D2s

Cmd> eigenvals(D2s)
 (1)    110.8   71.209   31.683   21.895
 (5)   13.598   8.5499   2.3585        0
 (9) -0.06506  -1.0985  -3.1124

Cmd> (110.8+71.2)/sum(abseigenvals(D2s)))
(1)      0.68843

Cmd> Y <- eigen(D2s)$vectors[,run(2)]*\
sqrt(eigenvals(D2s)[run(2)]')

Cmd> lang <- vector("E","N","Da","Du",\
"G","Fr","Sp","I","P","H","F")

Cmd> chplot(Y[,1],Y[,2],lang,xaxis:F,yaxis:F)
```



In some instances, we have dissimilarities, but not really distances. In particular, the difference of 1 between dissimilarities of 0 and 1 may not have any relation to the difference of 1 between dissimilarities of 9 and 10.

In such a case, we want points $\mathbf{X}$ such that the distances between the rows of $\mathbf{X}$ have the same order as the dissimilarities, but the actual distances don't matter.

This is *Nonmetric Multidimensional Scaling*.

For any set of points $\mathbf{X}$, compute the distances d_{ij}.

Let $\hat{d}_{ij}$ be an isotonic fit of these distances to the ordering from $\mathbf{D}$. This means that the $\hat{d}_{ij}$s are the closest numbers to the d_{ij}s that obey the correct ordering from $\mathbf{D}$. (Use the pool adjacent violators algorithm to get the isotonic fit.)

Define the *stress* to be

$$\text{Stress} = \left[\frac{\sum_{i<k} (d_{ik} - \hat{d}_{ik})^2}{\sum_{i<k} d_{ik}^2} \right]^{1/2}$$

Nonmetric MDS finds a matrix of points $\mathbf{X}$ to minimize the stress. $\mathbf{X}$ is not unique; rotations don't change the stress, and rescaling all the variables by the same factor doesn't change the stress.

Some people prefer to minimize the SStress

$$\text{SStress} = \left[\frac{\sum_{i<k} (d_{ik}^2 - \hat{d}_{ik}^2)^2}{\sum_{i<k} d_{ik}^4} \right]^{1/2}$$