

Lecture 4. Distance-matrix based representations

1. Introduction

PCA identifies a subspace, onto which we project the data to capture most of its variation. PCA does not distort the information it presents, but suppresses information which is not presented. Two points may appear to be close in the 2 or 3 dimensional subspace onto which we project, even though in the full high dimensional space they are not near each other. Multi Dimensional Scaling (MDS) does the opposite: it looks for a low dimensional representation of the data which preserves (as much as possible) the relative distances of the points in the full space. The cost is an overall distortion of the positions (non-linear transformation). The basic quantity one focuses on is the *distance matrix* \mathbf{D} . It's i, j element is the distance between the two points $\bar{\mathbf{x}}^{(i)}$ and $\bar{\mathbf{x}}^{(j)}$;

$$D_{ij} = |\bar{\mathbf{x}}^{(i)} - \bar{\mathbf{x}}^{(j)}| = \sqrt{\sum_{k=1}^d [x_k^{(i)} - x_k^{(j)}]^2} \quad (1)$$

For N points in d dimensions D is an $N \times N$ symmetric matrix.

There is a method, Sorting Points Into Neighborhoods (SPIN), which looks for a permutation (reordering) of the data points, such that the resulting distance matrix is "nice" - it orders the points so that pairs that are near each other in the high dimensional space will be also close in the (linear) reordering. This method does not suffer from either of the problems of PCA and MDS - it does not distort the distance matrix and no information is lost. The price to pay is that one has to acquire ability to recognize shapes and relationships between objects from the distance matrix.

In our example of 4 samples in a $d = 2$ dimensional gene-space, D_{12} is the distance between the first two points,

$$D_{12} = |\bar{\mathbf{e}}^{(5)} - \bar{\mathbf{e}}^{(19)}| = \sqrt{(1-9)^2 + (8-2)^2} = 10 \quad (2)$$

and the full distance matrix is given, [for the points ordered (5),(19),(27),(32)] by

patient No:	(5)	(19)	(27)	(37)
(5)	0	10	10.77	2.82
(19)	10	0	2.82	7.21
(27)	10.77	2.82	0	8.24
(37)	2.82	7.21	8.24	0

(3)

2. Multi Dimensional Scaling

The problem: given N points $\mathbf{x}^{(i)}$ with $i = 1, 2, \dots, N$ in some d dimensional space, find N points $\mathbf{y}^{(i)}$ with $i = 1, 2, \dots, N$ in some $\tilde{d} = 2, 3$ dimensional space, such that their distance matrix $\tilde{\mathbf{D}}$ is as close as possible to the "real" distance matrix \mathbf{D} . The distance matrix $\tilde{\mathbf{D}}$ is defined by

$$\tilde{D}_{ij} = |\tilde{\mathbf{y}}^{(i)} - \tilde{\mathbf{y}}^{(j)}| \quad (4)$$

How does one define the distance between two distance matrices, $\tilde{\mathbf{D}}$ and \mathbf{D} ? There are $N(N-1)/2$ independent matrix elements; usually we cannot satisfy equality of all, i.e. $\tilde{D}_{ij} = D_{ij}$ for all pairs i, j . Hence one defines a cost function that measures the deviation from this perfect solution. The obvious choice is

$$J_0 = \sum_{i < j} (D_{ij} - \tilde{D}_{ij})^2 \quad (5)$$

This choice is however problematic since it measures absolute error. Say for some points \mathbf{x} we found a good solution \mathbf{y} ; if we scale all coordinates \mathbf{x} by a factor $\lambda > 1$, we would like the similarly scaled positions $\lambda\mathbf{y}$ to be a good solution. The new error is, however, λ^2 times larger. Hence one defines scaled cost functions. These choices are

$$J_{ee} = \frac{\sum_{i < j} (D_{ij} - \tilde{D}_{ij})^2}{\sum_{i < j} D_{ij}^2} \quad (6)$$

$$J_{ff} = \sum_{i < j} \left(\frac{D_{ij} - \tilde{D}_{ij}}{D_{ij}} \right)^2 \quad (7)$$

$$J_{ef} = \frac{1}{\sum_{i < j} D_{ij}} \sum_{i < j} \frac{(D_{ij} - \tilde{D}_{ij})^2}{D_{ij}} \quad (8)$$

Remember that the coordinates of the data points $\mathbf{x}^{(i)}$ are given, and we should minimize the cost J over the unknown coordinates $y_k^{(i)}$. There are $\tilde{d}N$ unknowns ($3N$ or $2N$) and we minimize J over them by gradient descent. The solution one obtains is not unique - depends on the initial point.

J_{ee} emphasizes large errors - a difference of 10 between 1000 and 1010 has the same contribution to the cost and the difference between 10 and 20. J_{ff} measures fractional error.

3. Sorting Points Into Neighborhoods (SPIN)

We have N data points in some d dimensional space - say $N = N_s$, the number of samples, and $d = N_g$, the number of genes. Say the points are labelled by α - which could be some code for a patient, or his/her name. Define a function $I[\alpha]$ such that $i = I[\alpha]$ is a one-to-one mapping of the data points to integers $i = 1, 2, \dots, N$. Initially the data points are randomly ordered, and we use their index in this ordering as our $I(\alpha)$.

In the context of our simple example of $N = 4$ points in $d = 2$ dimensional space, the "patient ID" numbers (5), (19), (27), (37) play the role of α , and the random initial ordering we used corresponds to

$$I[(5)] = 1, \quad I[(19)] = 2, \quad I[(27)] = 3, \quad I[(37)] = 4 \quad (9)$$

The distance matrix D given by equation (3) corresponds to this ordering of the points; the entry $D_{1,2} = 10$ corresponds to the distance between the first and second point, i.e. (5) and (19).

If we reorder the points, we get a different distance matrix - one whose rows and columns are in the reordered sequence. The aim of SPIN is to find a reordering (called a permutation) of the points which produces a distance matrix which reveals the structure of the data; for example, the shape of a cloud of points, or the fact that there are two (or more) separate clouds, etc. Note that the diagonal terms are $D_{i,i} = 0$; the terms next to the diagonal represent the distances between two points that are next to each other in our ordering; entries close to the diagonal stand for distances between pairs of points that are ordered near each other.

As mentioned above, in order to take advantage of the power of SPIN, one has to learn to read distance matrices. The following examples demonstrate what is meant by this statement. [slides of shapes]. These shapes appear very artificial - can we find them in real data? It turns out that expression data abounds with one-dimensional extended twisted shapes - reflecting *progression* as function of time, or disease status, etc. [slides shapes in colon cancer, leukemia]

The algorithm has two versions, that achieve different goals. The first is called Side To Side (STS) - it looks for an elongated, basically one-dimensional cloud. In terms of the distance matrix - the desired ordering is such that two points which are very separated in the ordering, will *not* be at a close distance in the real high-dimensional space. This type of ordering forces large values of $D_{i,j}$ to the *corners* of the distance matrix (that are far from the diagonal).

2.1 Side To Side sorting - algorithm

The N data points α are initially ordered in some arbitrary way, $i = I[(\alpha)]$. We first calculate the $N \times N$ distance matrix D . We also initialize the vector \vec{x} , whose elements are given by

$$x_i = i - (N + 1)/2 = [-(N - 1)/2, -(N - 1)/2 + 1, \dots, (N + 1)/2] \quad (10)$$

Now we perform the following steps:

1. $\vec{S} = D \cdot \vec{x}$

The components of this vector are given by

$$S_i = \sum_{j=1}^N D_{ij} x_j \quad (11)$$

We refer to S_i as the *score* of the object $\alpha(i)$

2. Rank the scores in descending order. If the rank of object α is j , set $I[\alpha] = j$, and *order the objects* α accordingly.

3. Check whether $j = i$ for all the objects - i.e. check whether the ordering according to the ranked S_i coincides with the original ordering of the objects. If there is no change - STOP, otherwise - construct the distance matrix D of the reordered objects, and GO TO 1.

We now demonstrate this procedure for the 4 objects of our example and their distance matrix (3).

For $N = 4$ objects the vector \vec{x} is given by

$$\vec{x} = \begin{pmatrix} -3/2 \\ -1/2 \\ +1/2 \\ +3/2 \end{pmatrix} \quad (12)$$

1. Starting with the ordering of eq. (eq:order0) and using the corresponding D from eq. (3), we get (the values were rounded) $\vec{S} =$

$$D \cdot \vec{x} = \begin{pmatrix} +4.6 \\ -2.8 \\ -5.2 \\ -3.7 \end{pmatrix} \quad (13)$$

2. The largest score corresponds to the first object, the second largest to the second, the third largest to the fourth and the fourth largest to the third object. Hence when we reorder our 4 objects according to the scores, the new ordering corresponds to

$$I[(5)] = 1, \quad I[(19)] = 2, \quad I[(27)] = 4, \quad I[(37)] = 3 \quad (14)$$

yielding the ordering (5),(19),(37),(27)

3. This ordering is not the original one ((objects (27) and (37) switched places), hence construct the new D :

patient No:	(5)	(19)	(37)	(27)
(5)	0	10	2.82	10.77
(19)	10	0	7.21	2.82
(37)	2.82	7.21	0	8.24
(27)	10.77	2.82	8.24	0

GO TO 1:

1. multiply the new D by \vec{x} to get a new score vector:

$$\vec{S} = D \cdot \vec{x} = \begin{pmatrix} 12.6 \\ -7.2 \\ 4.5 \\ -13.4 \end{pmatrix} \quad (15)$$

The largest score corresponds to the first object, the second largest to the third, the third largest to the second and the fourth largest to the fourth object. Hence when we reorder our 4 objects according to the scores, the new ordering corresponds to

$$I[(5)] = 1, \quad I[(37)] = 2, \quad I[(19)] = 3, \quad I[(27)] = 4 \quad (16)$$

2. The new ordering, (5),(37),(19),(27) differs from the previous one, hence we reorder the distance matrix:

patient No:	(5)	(37)	(19)	(27)
(5)	0	2.82	10	10.77
(37)	2.82	0	7.21	8.24
(19)	10	7.21	0	2.82
(27)	10.77	8.24	2.82	0

and again GO TO 1:

1. multiply the new D by \vec{x} to get a new score vector:

$$\vec{S} = D \cdot \vec{x} = \begin{pmatrix} 19.7 \\ 11.7 \\ -14.3 \\ -18.8 \end{pmatrix} \quad (17)$$

The largest score corresponds to the first object, the second largest to the second, the third largest to the third and the fourth largest to the fourth. Hence when we reorder our 4 objects according to the scores, we get precisely the previous ordering and hence we STOP. [slide]

What is behind this algorithm? If there is an underlying linear ordering in the data, and we order the points accordingly from side to side, the distances along *first rows* of D will be monotonously increasing, in the *middle rows* they will decrease and then increase and in the *last rows* - monotonously decrease [slide]. The score S_i is in fact the correlation of this set of numbers along the row i with the *increasing vector* \vec{x} . Reordering according to S_i pulls rows with high correlation (i.e. increasing ones) to the beginning of the list, etc, and enhances this type of ordering. Reordering the rows, however, also necessitates reordering the columns and hence the order along the row changes and its score will change - thus one has to iterate the process. [slides] Since large distances affect affect the correlation coeff. more than small ones, STS tries to place them at the corners away from the diagonal (there they get multiplied by the largest components of \vec{x}). Hence STS does not place small distances at those corners.

The method does not yield a clear satisfactory distance matrix when the assumption of an underlying linear ordering of the data is violated. [slide ring].

2.2 Neighborhood sorting - the idea

This algorithm does in effect the following. We have a particular ordering of the objects; lets concentrate on the object at position i in this ordering. We want to move this object to a "better" location - to do this, we scan all possible positions j to which we can move this object and for each j we calculate a score in the following way. Take a (Gaussian) window of width σ and slide it along the ordered points, moving its center from $j = 1$ to $j = N$. For each j we calculate the *average distance* of the points within the window from our point i . This average distance is the *score* S_{ij} of the neighborhood centered on point j , when viewed as a "natural habitat" for point i . Of all these scores we find the *minimal*, and select the corresponding center $j(i)$ as the optimal location of the point i - i.e. the one which has within a window σ of $j(i)$ the points with shortest average distance to i . For each i we find his best $j(i)$ - this is our next permutation i.e. the new ordering of the points! one has to solve some technicalities, e.g. how to break ties, that occur when two points, i and i' , select the same j , etc. [slide] Since we do this in parallel for all i , it could be that i is moved to its "best" j , but simultaneously some of the points in that neighborhood, that were close to i , have moved out of it. Hence i will now find itself with a higher score than the one that brought it to this region. Hence the process is iterated.

Algorithmic implementation:

Construct a symmetric weight matrix \mathbf{W} whose elements are

$$W_{ij} \propto e^{-(i-j)^2/2\sigma^2} \quad (18)$$

Start an iterative process by calculating a *score matrix*

- 1.

$$\mathbf{S} = D \cdot \mathbf{W} \quad (19)$$

The entry S_{ij} is the score for placing point i at position j and σ is the effective width of the window that defines the proposed neighborhood centered on j .

2. Find in each row i of \mathbf{S} the smallest element. The index of this element,

$$j_t(i) = \arg \min_j S_{ij} \quad (20)$$

is the position to which i should move. If there are ties - break at random (say by placing the competing points at $j, j + 1, \dots$).

3. Check if the new order is different than the old - if not, STOP, otherwise GO TO 1.

2.3 Sorting - formal treatment

Questions one has to answer: does these procedures always converge? Does one always reach the same fixed point of the iterative process? What is the complexity? To answer these questions, we showed that one can construct a "cost function" which assigns, for a given distance matrix D a score to every ordering (permutation P) of the data. Next we showed that each iteration of the STS and neighborhood procedures does not increase the score. Hence if when the score is not lowered we do not reorder the points, the process will terminate (i.e. converge to a "local minimum" of the cost). These points, together with the complexity of the algorithms and the underlying computational problems are discussed in detail in The attached preprint of Tsafir et al.