Report

Multidimensional Scaling (MDS)

Multidimensional Scaling is a technique which is used to represent a higher dimensional object in a lower dimension, to observe dissimilarities in the data. MDS uses the dissimilarities between pairs of object as the data to reduce dimension. In 3D geometry, we use the geodesic distance between a pair of object in higher dimension, to map it to a lower dimension space where Euclidean distance between the pair of object is same as the geodesic distance in higher dimension. The mapping is an isometric mapping and thus the problem we are dealing with MDS is one of isometric embedding between two spaces.

Thus, we have a data set of point or shape X and we want to find its embedding in a lower dimensional space say m, so we need a mapping f such that

$$f = \underset{f:\mathcal{X}\to\mathbb{R}^m}{\operatorname{argmin}} \sum_{\substack{i,j \\ i<j}}^{N} \left(\left| d_{\mathbb{R}^m}\left(f(x_i), f(x_j)\right) - d_{\mathcal{X}}(x_i, x_j) \right| \right)^2$$

We denote $f(x_i) = z_i$, which is the m dimensional Euclidean co-ordinate of the image of sample x_i under f , and representing all mapping of point of X to the m dimensional space by the NXm matrix $Z = \{z_i\}$. Thus, the distortion criterion, generally called stress(σ_2), can be written as,

$$\sigma_2(Z, D_x) = \sum_{i<j} \left(\left| d_{ij}(Z) - d_{\mathcal{X}}(x_i, x_j) \right| \right)^2$$

Thus, the MDS problem boils down to find the best Z^* which minimizes the distance between pair of points in the original domain and the reduced domain. Hence,

$$Z^* = \underset{Z\in\mathbb{R}^{N\times m}}{\operatorname{argmin}}\ \sigma_2(Z)$$

This particular $\sigma_2(Z)$ criterion function is called L_2 stress in order to distinguish it from other criterion functions.

The solution of the MDS problem is achieved with minimization of the given stress function. We take a gradient descent approach, which is formalized for this particular problem in SMACOF(scaling by minimization of a convex function) algorithm. We have implemented the steps of the SMACOF algorithm in our program to find the data points in reduced dimensional space. The SMACOF algorithm is given by the following steps. Each implementation of the step is indicated in the program as comments.

SMACOF Algorithm:

1. input: *N X N* matrix of geodesic distances, D_x.
   output: Canonical form Z^* in $\mathbb{R}^m$.
2. initialization: initial $Z^{(0)}$ and $k = 0$.
3. *repeat*
   a. *Calculate* $B(Z^{(k)}, D_x)$.
   b. Multiplicative update: $Z^{(k+1)} = \frac{1}{N} B(Z^{(k)}, D_x) Z^{(k)}$.
   c. $k \longleftarrow k+1$
   until convergence
4. $Z^* = Z^{(k)}$

The *B matrix* is used to calculate the L_2 stress and is defined on input Z and D_x as,

$$b_{ij} = \begin{cases} -d_{ij}\left(x_i, x_j\right) d_{ij}^{-1}(Z) & i \neq j \text{ and } d_{ij}(Z) \neq 0 \\ 0 & i \neq j \text{ and } d_{ij}(Z) = 0 \\ -\sum_{k,i} b_{ik} & i = j \end{cases}$$

All three cases of the B Matrix is handled and indicated in the implementation as comments.

Implementation

We start by creating a set of discrete point on the shape surface. We create a grid of N points, with number of rows and columns equal to sqrt(N) as done in surface 1. This generates an even distribution of points on a surface. Thus, the number of points to be taken as input in our implementation needs to be a perfect square(for surface 1) or divisible by 5(for surface 2), but we can modify it easily to include any number of points N = numRows X numCols points, by specifying numRows and numCols separately as input as shown for surface 2. Surface 2 generates an uneven distribution of points on the surface.

Once we generate the grid of points, which represent a set of N (u, v) parametric coordinates, we map them to the desired shape using the function *cylinder* for Cylinder and generate the list of input points $Zmat = Z^{(0)}$. We calculate distance matrix D_x by calling the *calcLength* function and passing it the list of points as input. The function returns us the N X N geodesic distance matrix of the N points. We use this distance matrix to calculate the B matrix, given the initial set of points $Zmat = Z^{(0)}$. We use the function *calcBMatrix* to get the updated BMatrix and apply the Multiplicative Update to get $Znew = Z^{(k+1)}$.

Now we check for convergence criteria. Ideally our L_2 stress should tend to zero for convergence. Thus we calculate the L_2 stress given by the equation above for $\sigma_2(Z, D_x)$. We observe that this value comes very close to zero, but never becomes zero, as for most practical cases. Hence, we check for the difference in $\sigma_2(Z^{(k)}, D_x)$ and $\sigma_2(Z^{(k+1)}, D_x)$. We take a small threshold (*minError*) value for the difference to signify convergence, and once the difference becomes less than the threshold, convergence is attained. Hence, the $Zstar = Z^{(kstar)}$ which is our output.

Observations

Point Data Set:

As stated in the implementation part, we pre-define the number of point we want for our computation. We tested with small and large datasets all the same. We used N = 25,100, 625, 1600 and 2500 for our test. Of all the data-set size, N=100 gives the ideal output, in terms of time taken until convergence as well as the canonical form generated.
The time taken for 100 iterations with different N along with the number of iteration took to converge for surface 1 is shown in the following table:

N	Time	Converging Iteration
25	~ 2 secs	17
100	~ 28 secs	50
625	~ 1093 secs	> 100
1600	~ 7778 sec (2 hrs, 10 mins,)	> 100
2500	> 2 Hrs	-

The iterations for N = 2500 points could not be completed at the end of 2.5 hrs.

The time taken for 100 iterations with different N along with the number of iteration took to converge for surface 2 is shown in the following table:

N	Time	Converging Iteration
25	~ 2 secs	19
100	~ 28 secs	55
625	~ 1073 secs	> 100

Thus, we obtain a good enough convergence with optimum amount of time with N=100 points. It should be noted that the time reported are for the computation of all 100 iterations. For N= 25 and 100, the convergence is achieved much before 100 iteration. So, actual convergence time is much less than reported, which is not the case with N = 625.

* With lower number of points, although the time is very less, but we do not get a good idea about the surface it conforms to in the lower dimensional space.
* With higher N, it is intuitive to predict the rise in evaluation time because the calculation involves N X N distance matrix and update of the N X N B Matrix in each iteration. Thus, the time taken to converge as well as calculate grows exponentially.
* Also, with higher number of points, the canonical form that we obtain do not give us more information than we get with an optimum number of point such as 100, which will be clear when we observe the shape obtained of various canonical form in the subsequent subsection.

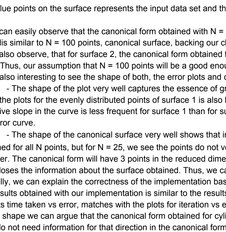
*All approximate times are given for execution done on a single-processor 2.7Ghz Intel Core i5 Processor, with 8GB RAM calculated using Mathematica inbuilt timer.

Convergence and Error Analysis:

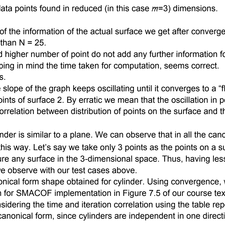
We use the convergence criteria as the difference between the L_2 stress of two consecutive iteration to be less than an threshold value which is the error. The threshold value was calculated experimentally and is valid for all the data set tested. The threshold value comes out to be 0.01 which is small enough to show that the change in L_2 stress or error is minimal in subsequent iterations. Thus if the sum of difference in geodesic distance in higher dimension and euclidean distance in the lower dimension is less than the threshold, error is less than the threshold, and the algorithm converges.

The plots of iteration vs error obtained for various dataset and their convergence iteration is shown below.

Shape: Cylinder
Surface: 1

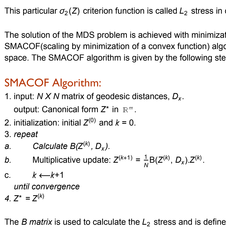


N = 25; Convergence = 17 Iterations

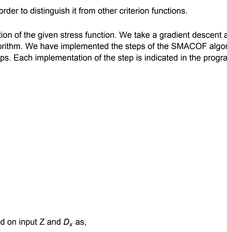


N = 100; Convergence = 50 Iterations

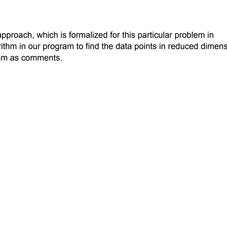
Surface: 2



N = 25; Convergence = 19 Iterations



N = 100; Convergence = 55 Iterations



N = 625; Convergence: >100 Iterations

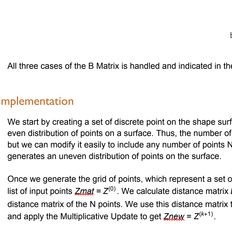
Shape of Canonical form obtained and Correctness of Implementation:

The various Canonical Forms obtained, plotted along with the input surface and the shape are shown below.

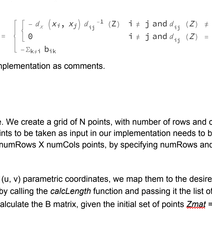
Shape: Cylinder
Surface: 1

N = 25; Convergence = 17 Iterations

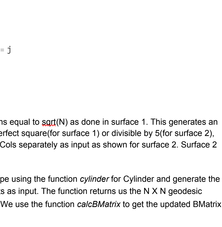
N = 100; Convergence = 50 Iterations



N = 25; Convergence = 19 Iterations



N = 100; Convergence = 55 Iterations



N = 625; Convergence: >100 Iterations

The blue points on the surface represents the input data set and the red points shows the canonical form with data points found in reduced (in this case *m*=3) dimensions.

* We can easily observe that the canonical form obtained with N = 25 for both surfaces does not capture much of the information of the actual surface we get after convergence as they are very dis similar to N = 100 points, canonical surface, backing our claim as N = 100 points a better data set size than N = 25.
* We also observe, that for surface 2, the canonical form obtained for N = 100 and N = 625 is almost similar and higher number of point do not add any further information for the canonical form. Thus, our assumption that N = 100 points will be a good enough input data set size for the algorithm, keeping in mind the time taken for computation, seems correct.
* It is also interesting to see the shape of both, the error plots and canonical surface comparing the two surfaces.
  - The shape of the plot very well captures the essence of gradient descent algorithm, as we see that the slope of the graph keeps oscillating until it converges to a "flat" region. Also, the plots for the evenly distributed points of surface 1 is also less "erratic" than the randomly distributed points of surface 2. By erratic we mean that the oscillation in positive and negative slope in the curve is less frequent for surface 1 than for surface 2. It is interesting to hypothesize the correlation between distribution of points on the surface and the gradient of the error curve.
  - The shape of the canonical surface very well shows that in $\mathbb{R}^3$, the canonical form for a surface on cylinder is similar to a plane. We can observe that in all the canonical form obtained for all N points, but for N = 25, we see the points do not very well show them. It can be understood in this way. Let's say we take only 3 points as the points on a surface of cylinder. The canonical form will have 3 points in the reduced dimensional space. Now these 3 points can capture any surface in the 3-dimensional space. Thus, having lesser number of point loses the information about the surface obtained. Thus, we can give a similar explanation for the results we observe with our test cases above.
* Finally, we can explain the correctness of the implementation based on both convergence, as well as the canonical form shape obtained for cylinder. Using convergence, we can see that the results obtained with our implementation is similar to the results as reported in our textbook. The plots given for SMACOF implementation in Figure 7.5 of our course textbook which reports time taken vs error, matches with the plots for iteration vs error we observe with our implementation considering the time and iteration correlation using the table reported above. Using shape we can argue that the canonical form obtained for cylinder, that resembles a plane is our desired canonical form, since cylinders are independent in one direction and thus we lose do not need information for that direction in the canonical form.