The Chinese University of Hong Kong, Shenzhen

CSC4008

Techniques for Data Mining

Report for Multi-Dimension Scaling Algorithm

Author:	*Student Number:*
LIU Yuxuan.	118010200
Wang Sixuan	118010305

May 25th, 2022

Q1: Multi-Dimension Scaling Implementation

```python
# Multi-Dimension Scaling Algorithm
def MDS():
  adjacency_matrix, city_coordinate = load_data()
  data_num = city_coordinate.shape[0]
  print('data_num: ', data_num)
  # Proximity Matrix D and D2
  D = adjacency_matrix
  D2= [[D[i][j]**2 for j in range(len(D[i]))] for i in range(len(D))]
  D2 = np.mat(D2)
  # Matrix J
  J = np.eye(data_num) - (1/data_num)*np.ones((data_num, data_num)) * (np.ones((data_num, data_num)) )
  # Matrix B
  B = -0.5 * J * D2 * J
  # Two Eigenvectors with the Largest Eigen Value
  eig_val, eig_vec = np.linalg.eig(B)
  eig_pairs = [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range(data_num)]
  eig_pairs.sort(reverse=True, key=lambda ele:ele[0])
  eigen_vec_1 = eig_pairs[0][1]
  eigen_vec_2 = eig_pairs[1][1]
  # Matrix X
  EM = np.hstack((eigen_vec_1, eigen_vec_2))
  EigenValueMat = np.eye(2)
  for i in range(2):
    EigenValueMat[i, i] = np.sqrt(eig_pairs[i][0])
  X = EM * EigenValueMat
  print('shape of X: ', X.shape)
  print(X)
  # Verify the Adjacency Distance Matrix
  new_adjacency_matrix = np.zeros(shape=(data_num, data_num))
  for i in range(data_num):
    for j in range(data_num):
      dist = 0
      if i != j:
        dist = np.sqrt(
          abs((X[i, 0] ** 2 - X[j, 0] ** 2))
          +
          abs((X[i, 1] ** 2 - X[j, 1] ** 2))
        )
      new_adjacency_matrix[i, j] = dist
  print('New Adjacency Matrix: \n', new_adjacency_matrix)
  # Plot Original Cities
  plt.scatter(city_coordinate[:, 0], city_coordinate[:, 1], c='red')
  # Plot Projected Cities
  plt.scatter(np.asarray(X[:, 0]).flatten(), np.asarray(X[:, 1]).flatten(), c='blue')
  plt.show()
```

Figure 1: Python Implementation of MDS Algorithm

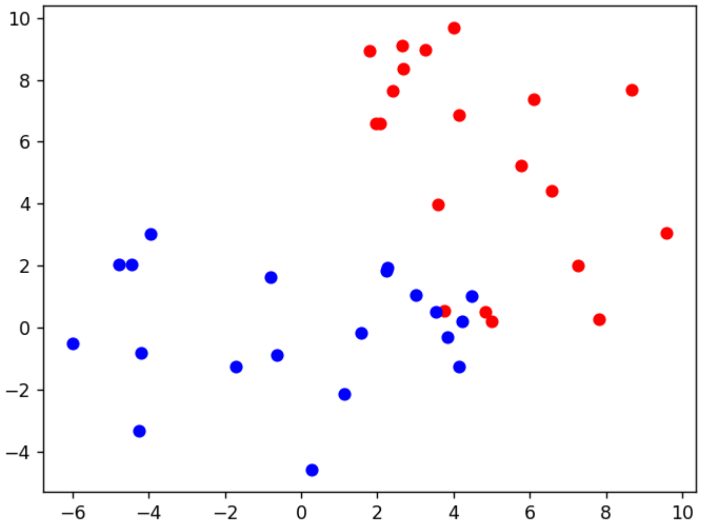

Figure 2: Demo Output (Red as Original and Blue as Projected)

As shown in the figure, after rotating the blue data points by 90 degrees to the left, we can match the original data.

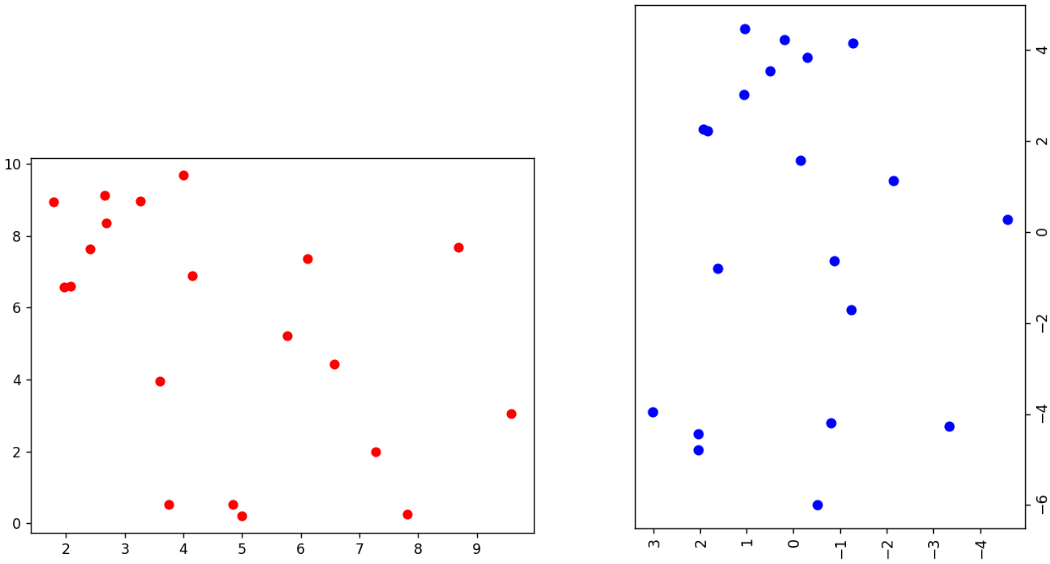

Figure 3: Demo Output Comparison (Blue Points Rotated by 90 Degree)

Q2: Proof of Correctness of MDS Algorithm

Suppose we have a Euclidean adjacency distance matrix $D = \left(\mathrm{d_{ij}}\right)$, we want to find a projected matrix $X = [\mathrm{x_1}, \dots, \mathrm{x_n}]$ so that

$$\left\|\mathrm{x_i} - \mathrm{x_j}\right\| = \mathrm{d_{ij}} \quad \text{and} \quad \sum_{i=1}^{n} = 0 \quad \forall k$$

since X must be centered to avoid multiple results.

Suppose we have already found a n x n Gram matrix $\mathrm{B} = \mathrm{X'X}$, we have

$$\begin{aligned}\mathrm{d_{ij}}^2 &= \left\|\mathrm{x_i} - \mathrm{x_j}\right\|^2 \\ &= \mathrm{x_i}'\mathrm{x_i} + \mathrm{x_j}'\mathrm{x_j} - 2\mathrm{x_i}'\mathrm{x_j} \\ &= \mathrm{b_{ii}} + \mathrm{b_{ij}} - 2\mathrm{b_{ij}}\end{aligned}$$

The constraint of $\sum_{i=1}^{n} \mathrm{x_{ik}} = 0 \quad \forall k$ is now transformed into

$$\sum_{i=1}^{n} b_{ij} = \sum_{i=1}^{n}\sum_{k=1}^{q} x_{ik}x_{jk} = \sum_{k=1}^{q} x_{jk}\sum_{i=1}^{n} x_{ik} = 0$$

Suppose $T = \mathrm{trace}(B) = \sum_{i=1}^{n} b_{ij}$, we have

$$\sum_{i=1}^{n} {d_{ij}}^2 = T + n * b_{jj} \quad and \quad \sum_{j=1}^{n} {d_{ij}}^2 = T + n * b_{ii}$$

$$\downarrow$$

$$\sum_{j=1}^{n}\sum_{i=1}^{n} {d_{ij}}^2 = 2nT$$

By combining the equations above, we can compute B as

$$b_{ij} = -\frac{1}{2}\left({d_{ij}}^2 - {d_{*j}}^2 - {d_{i*}}^2 + {d_{**}}^2\right)$$

$$B = -\frac{1}{2}CD^2C$$

Where $C = I - \frac{1}{n}1'1$

Therefore, we have proved that

$$B = -\frac{1}{2}CD^2C = X'X$$

And X can be computed by the eigen decomposition of B, for $B = V\Lambda V'$:

$$X = \Lambda^{\frac{1}{2}}V'$$

And it is exactly what we do to compute the projected matrix X.

Therefore, we have proved that the MDS algorithm can indeed recover the original distance relationship while reducing the dimension.