Visualization Theory

Multidimensional Scaling

Danushka Bollegala

UNIVERSITY OF LIVERPOOL

Objective

- Assume that we have a dataset that we would like to visualize
- Problem settings
  - Our data are multi-dimensional
    - Apply a dimensionality reduction method such as PCA.
  - Our data points are scalar values (e.g. distances/similarities between some points)
    - Apply **Multi-dimensional scaling** (MDS)

Multidimensional Scaling (MDS)

- Assume that we have distance/similarity matrix (nxn) for *n* data points.
- Examples
  - We asked human subjects to express their preferences
  - We extracted clickthrough data from a search engine
- How can we construct a set of *n* feature vectors for the data points, given their similarity matrix?
  - Problem tackled by MDS

Definitions

- Assume that we are given a distance matrix $\mathbf{D} \in \mathbb{R}^{nxn}$.
- We are interested in finding a set of p dimensional vectors arranged in rows of a matrix $\mathbf{X} \in \mathbb{R}^{nxp}$.
- We first consider computing a matrix $\mathbf{B} = \mathbf{X}\mathbf{X}^\mathbf{T}$ using $\mathbf{D}$.
- If we can find **B** then we can easily compute **X** as we shall see later.

Step 1: Express D in terms of B

- Let us denote the (r,s) element of **B** by b_{rs}
- The squared Euclidean distance, d_{rs}, between the data point *r* and the data point *s* is given by

$$d_{rs}^2 = ||\boldsymbol{x}_r - \boldsymbol{x}_s||_2^2$$

$$d_{rs}^2 = \sum_{i=1}^{p} (x_{ri} - x_{si})^2$$

$$d_{rs}^2 = \sum_{i=1}^{p} x_{ri}^2 - 2\sum_{i=1}^{p} x_{ri}x_{si} + \sum_{i=1}^{p} x_{si}^2$$

$$d_{rs}^2 = b_{rr} - 2b_{rs} + b_{ss} \qquad [1]$$

Standardization of X

- It does not matter how we select the origin of the co-ordinate space.
- Let us assume that all features in **X** are standardized such that their mean value is zero.
- This means the sum of each column of **X** is zero.

$$\sum_{r=1}^{n} x_{ri} = 0 \qquad \forall i = 1, \ldots, p$$ [2]

- By standardization we reduce the freedom of the matrix **X**, thereby constraining it.

Proof: Sum of a row *r* in **B** is zero

- If **X** is standardized, then the sum of any row *r* in **B** becomes zero.

Proof:

$$\begin{aligned}\sum_{s=1}^{n} b_{rs} &= \sum_{s=1}^{n}\sum_{i=1}^{p} x_{ri}x_{si} \\ &= \sum_{i=1}^{p}\sum_{s=1}^{n} x_{ri}x_{si} \\ &= \sum_{i=1}^{p} x_{ri}\sum_{s=1}^{n} x_{si} \\ &= 0 \quad \blacksquare\end{aligned}$$

[3]

Proof: Sum of a column *s* in **B** is zero

- If **X** is standardized, then the sum of any column *s* in **B** becomes zero.

Proof:

$$
\begin{aligned}
\sum_{r=1}^{n} b_{rs} &= \sum_{r=1}^{n} \sum_{i=1}^{p} x_{ri} x_{si} \\
&= \sum_{i=1}^{p} \sum_{r=1}^{n} x_{ri} x_{si} \\
&= \sum_{i=1}^{p} x_{si} \sum_{r=1}^{n} x_{ri} \\
&= 0 \quad \blacksquare
\end{aligned}
$$

[4]

Lets sum-up the squared distances

$$d_{rs}^2 = b_{rr} - 2b_{rs} + b_{ss}$$

$$\sum_{r=1}^{n} d_{rs}^2 = \sum_{r=1}^{n} b_{rr} - 2\sum_{r=1}^{n} b_{rs} + \sum_{r=1}^{n} b_{ss} = \mathrm{tr}(\mathbf{B}) + nb_{ss} \quad [5]$$

sum of diagonal elements of B. This is called the trace of a matrix and is denoted by tr.

This term is zero from [4].

We are adding n times b_{ss}, which has nothing to do with r.

Lets sum-up the squared distances

$$d_{rs}^2 = b_{rr} - 2b_{rs} + b_{ss}$$

$$\sum_{s=1}^{n} d_{rs}^2 = \sum_{s=1}^{n} b_{rr} - 2\sum_{s=1}^{n} b_{rs} + \sum_{s=1}^{n} b_{ss} = \mathrm{tr}(\mathbf{B}) + nb_{rr} \qquad [6]$$

We are adding n times b_{rr}, which has nothing to do with r.

This term is zero from [3].

sum of diagonal elements of B. This is called the trace of a matrix and is denoted by tr.

Lets sum-up the squared distances

$$\sum_{r=1}\sum_{s=1}^{n} d_{rs}^2 = \sum_{r=1} \mathrm{tr}(\mathbf{B}) + n \sum_{r=1} b_{rr}$$

Taking the sum over r in [6].

$$= n\,\mathrm{tr}(\mathbf{B}) + n\,\mathrm{tr}(\mathbf{B})$$

$$= 2n\,\mathrm{tr}(\mathbf{B}) \qquad [7]$$

Quiz: Derive [7] by taking the sum over s in [5].

Express d_{rs} in terms of b_{rs}

- Substitute for b_{rr} and b_{ss} in [1] using [5] and [6]

$$
\begin{aligned}
d_{rs}^2 &= b_{rr} - 2b_{rs} + b_{ss} \\
&= \frac{1}{n}\left(\sum_{r=1}^{n} d_{rs}^2 - \mathrm{tr}(\mathbf{B})\right) - 2b_{rs} + \frac{1}{n}\left(\sum_{s=1}^{n} d_{rs}^2 - \mathrm{tr}(\mathbf{B})\right)
\end{aligned}
$$

- Further substitute for tr(**B**) from [7]

$$
\begin{aligned}
d_{rs}^2 &= b_{rr} - 2b_{rs} + b_{ss} \\
&= \frac{1}{n}\left(\sum_{r=1}^{n} d_{rs}^2 - \frac{1}{2n}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2\right) - 2b_{rs} + \frac{1}{n}\left(\sum_{s=1}^{n} d_{rs}^2 - \frac{1}{2n}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2\right)
\end{aligned}
$$

[8]

Computing **B**

- From [8] we have,

$$b_{rs} = -\frac{1}{2}\left[d_{rs}^2 - \frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 - \frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2\right] \quad [9]$$

Using [9] we can compute all elements in B. The right hand side of [9] contains only the distances d_{rs}, which are given to us.

So at last we have the matrix B that we needed.

Also note that B is a symmetric matrix because the distance matrix D is a symmetric matrix.

Take a deep breath!

Are we done?

- Not quite! We must compute **X** from **B** = $\mathbf{XX}^{\top}$
- Easy... Perform eigenvalue decomposition on **B**
  - **B** = $\mathbf{UGU}^{\top}$
  - where, **U** is an orthogonal matrix $\mathbf{UU}^{\top} = \mathbf{U}^{\top}\mathbf{U} = \mathbf{I}$
  - **G** is a diagonal matrix containing the eigenvalues of **B**
  - Let **X** = $\mathbf{UG}^{1/2}$, then we have

$$(\mathbf{U}\mathbf{G}^{1/2})(\mathbf{U}\mathbf{G}^{1/2})^{\top} = \mathbf{U}\mathbf{G}^{1/2}\mathbf{G}^{1/2}\mathbf{U}^{\top} = \mathbf{U}\mathbf{G}\mathbf{U}^{\top} = \mathbf{B}$$

MDS Algorithm

- INPUT
  - Distance matrix **D**, dimensionality p.
- Output
  - Matrix **X** where each row represents a feature vector for a data point
- Compute **B** using

$$b_{rs} = -\frac{1}{2}\left[d_{rs}^2 - \frac{1}{n}\sum_{r=1}^{n} d_{rs}^2 - \frac{1}{n}\sum_{s=1}^{n} d_{rs}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} d_{rs}^2\right]$$

- Perform eigenvalue decomposition $\mathbf{B} = \mathbf{U}\mathbf{G}\mathbf{U}^T$ on **B** to obtain **U** and **G**. Select the top p eigenvalues/vectors.
- Return $\mathbf{X} = \mathbf{U}\mathbf{G}^{1/2}$

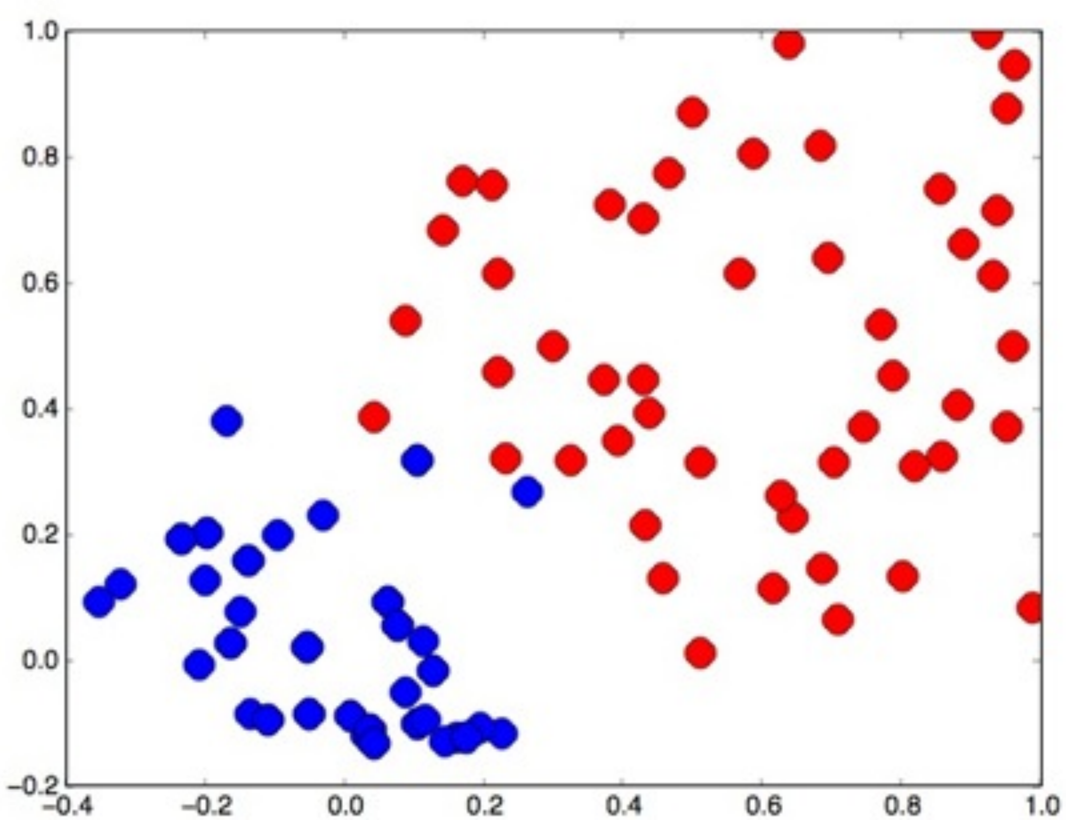
Things to note…

- Your distance matrix **D** might have negative eigenvalues
  - which means it is not Euclidean. You cannot exactly reproduce it using **X**. But this is fine as long as you have some large positive eigenvalues
- Matrix **B** might have negative eigenvalues
  - You get complex values when you compute their square root
  - If you have some high positive values (at least 2), then you can still plot in the 2D space
- The point is to construct vectors **X** that produce the distances in **D** as close as possible
  - We can measure the reproduction error using root mean square error (RMSE) [see classifier evaluations lecture]

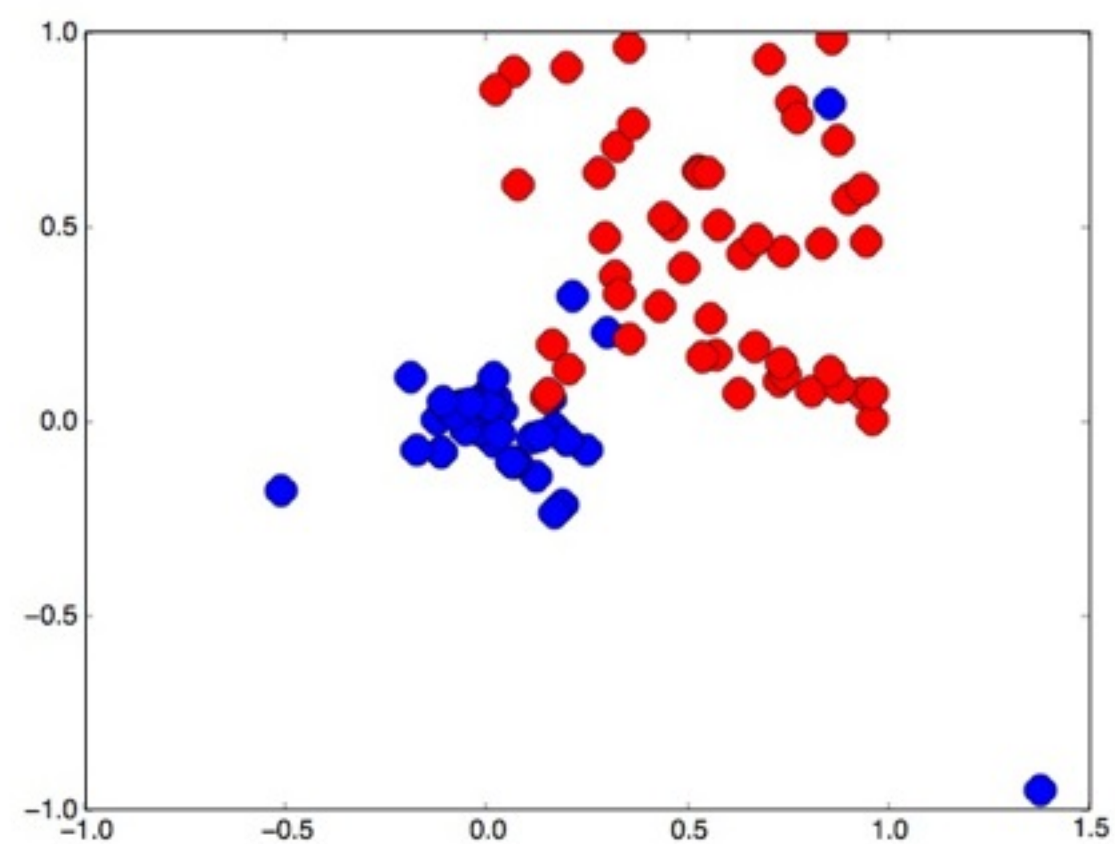
```python
import numpy
import matplotlib.pyplot as plt

def bval(D, r, s):
    n = D.shape[0]
    total_r = numpy.sum(D[:,s] ** 2)
    total_s = numpy.sum(D[r,:] ** 2)
    total = numpy.sum(D ** 2)
    val = (D[r,s] ** 2) - (float(total_r) / float(n)) - (float(total_s) / float(n)) + (float(total) / float(n * n))
    return -0.5 * val

def main():
    n = 50; m = 2
    Y = numpy.random.rand(n, m)
    D = numpy.zeros((n, n), dtype=complex)
    for i in range(0, n):
        for j in range(0, n):
            D[i, j] = numpy.linalg.norm(Y[i] - Y[j])
    B = numpy.zeros((n, n), dtype=complex)
    for i in range(0, n):
        for j in range(0, n):
            B[i,j] = bval(D, i, j)
    print "\nB matix"
    print B
    g, U = numpy.linalg.eig(B)
    idx = g.argsort()[::-1]
    g = g[idx]
    U = U[:,idx]
    print "Eigenvalues =", g
    G = numpy.diag(numpy.sqrt(g))
    X = numpy.dot(U.T, G)
    print "\nMatrix X"
    print X
    error = 0.0
    total = 0
    for i in range(0, n):
        for j in range(i+1, n):
            error += (numpy.linalg.norm(X[i] - X[j]) - D[i, j]) ** 2
            total += 1
    print "RMSE =", numpy.sqrt(error / float(total))
    plt.plot(X[:,0], X[:,1], 'bo', markersize=14)
    plt.plot(Y[:,0], Y[:,1], 'ro', markersize=14)
    plt.show()
    pass

if __name__ == '__main__':
    main()
```

Example 1: 2D data

- Let us consider a set 50 of 2D random feature vectors.
- We compute a distance matrix **D** from these vectors and perform MDS (p=2) on **D**.
- We plot the original vectors (red) and the vectors produced by MDS (blue) in 2D.
- Observe the correlation between RMSE and the MDS results.



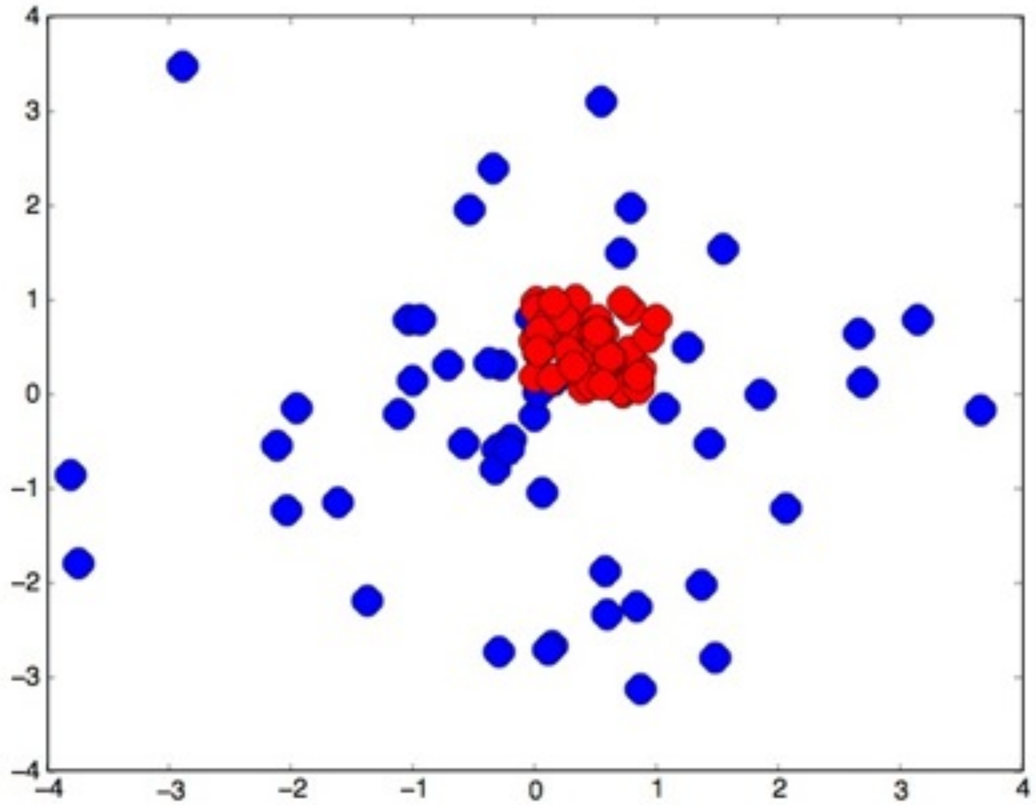
RMSE = 0.299, m = 2



RMSE = 0.472, m = 2

Example 2: High-dimensional Data

- Let us consider a set of 50 random vectors in 1000 dimensional space.
- We compute a distance matrix **D** from these vectors and perform MDS (p=2) on **D**.
- We plot the original vectors and the vectors produced by MDS in 2D space
- It is not possible to see any difference among original data points (shown in red) in the 2D space, if we only consider their first two dimensions.
- However, MDS provides a much better separation among the data points (shown in blue)



m = dimensionality of the actual space
m = 1000
RMSE = 0.3965