# Multidimensional Scaling

Junjie ( JJ ) Wang

Introduction:

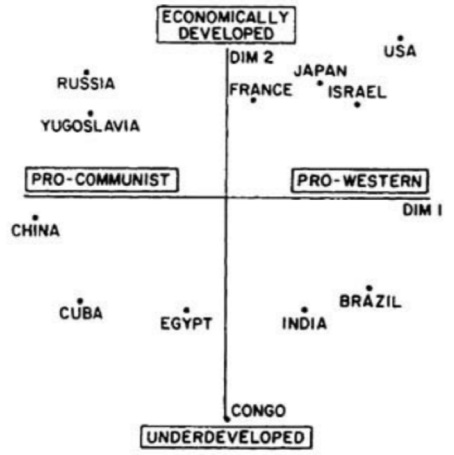
Multidimensional scaling refers to a set of algorithms that are used for information visualization. To be specific, it's used to visualize the similarity (or dissimilarity) of individuals in a dataset by placing them on a low dimensional space (most commonly, 2-D or 3-D) so that the distance between different individuals could be saw easily through their position on the plot.

| | BRZ | CON | CUB | EGY | FRA | IND | ISR | JPN | CHI | RUS | USA | YUG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BRAZIL | | | | | | | | | | | | |
| CONGO | 7 | | | | | | | | | | | |
| CUBA | 8 | 5 | | | | | | | | | | |
| EGYPT | 2 | 7 | 6 | | | | | | | | | |
| FRANCE | 3 | 3 | 2 | 6 | | | | | | | | |
| INDIA | 2 | 7 | 2 | 6 | 2 | | | | | | | |
| ISRAEL | 3 | 2 | 3 | 8 | 5 | 7 | | | | | | |
| JAPAN | 2 | 1 | 1 | 4 | 3 | 7 | 2 | | | | | |
| CHINA (MAINLAND) | 2 | 6 | 8 | 7 | 4 | 5 | 3 | 7 | | | | |
| RUSSIA | 2 | 3 | 7 | 7 | 3 | 6 | 3 | 6 | 8 | | | |
| U.S.A. | 5 | 3 | 3 | 4 | 8 | 5 | 7 | 8 | 1 | 2 | | |
| YUGOSLAVIA | 2 | 2 | 8 | 3 | 4 | 5 | 4 | 6 | 8 | 8 | 2 | |

(A) SUBJECT 17

| STIMULI | DIM. 1 | DIM. 2 |
|---|---|---|
| BRAZIL | .338 | -.218 |
| CONGO | .005 | -.536 |
| CUBA | -.341 | -.253 |
| EGYPT | -.089 | -.272 |
| FRANCE | .006 | .235 |
| INDIA | .189 | -.271 |
| ISRAEL | .306 | .222 |
| JAPAN | .219 | .274 |
| CHINA (MAINLAND) | -.449 | -.055 |
| RUSSIA | -.331 | .297 |
| U. S. A. | .407 | .379 |
| YUGOSLAVIA | -.321 | .199 |

(a)




(b)

distance

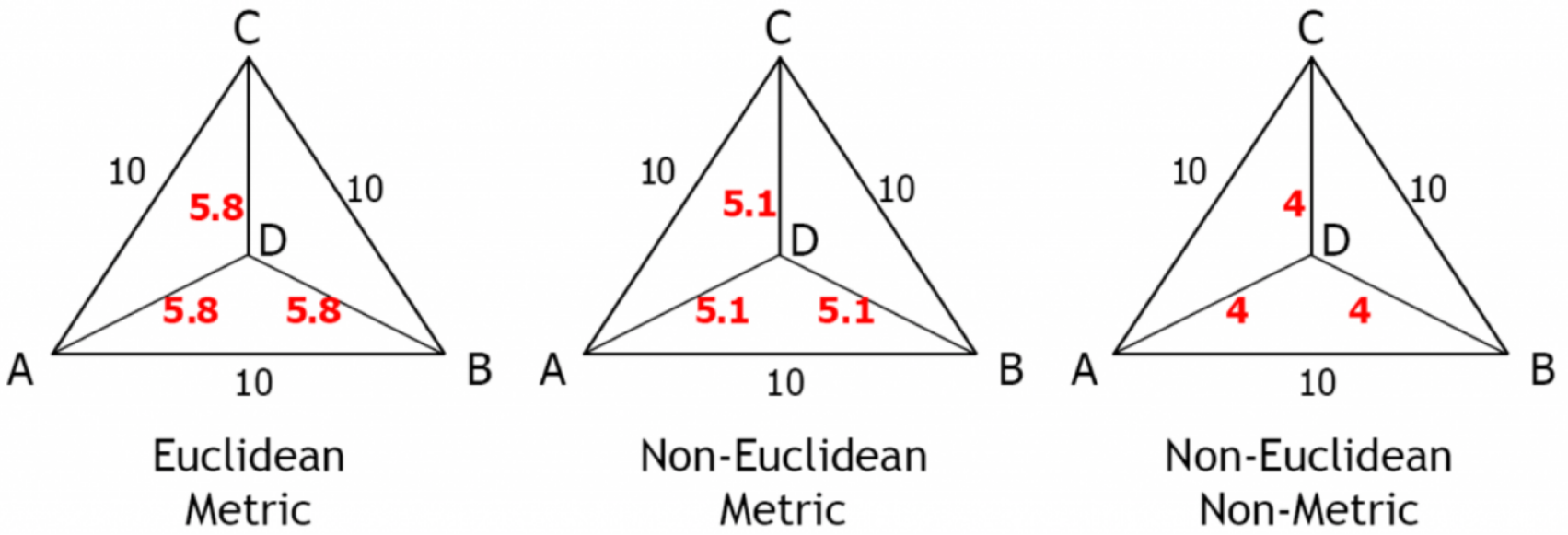
Metrics distance, it must satisfy:

1. $d(x, y) \geq 0$,
2. $d(x, y) = 0 \; if \; and \; only \; if \; x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$

Hence, for a metrics distance matrix, it's a symmetry matrix, with diagonal elements to be zero and off-diagonal elements to be positive (must be).

Of course, there exists non-metrics distance. In the graph theory, the distance between two nodes can be defined as the sum of weight of the shortest path. HOWEVER, the weight could be set to be negative.

Euclidean distance: $d(x, y) = \|x - y\|_2$



Goal of MDS: Given distance matrix of the dataset, seek to find optimal configuration $x_1, x_2, \dots x_n \epsilon R^q$ (n: number of individuals and p: dimension) so that :

$$d_{ij} \approx \|x_i - x_j\|_2 \text{ as close as possible}$$

Even in the case that the original distance matrix is non Euclidean distance!
Note, $x_1, x_2, \dots x_n$ is not unique, if X = $[x_1, x_2, \dots x_n]$ is the solution, then $X^* = X + C$ is also the solution.

Classical Multidimensional scaling:

Here, I try to explain the relationship between the distance matrix we already have and the configuration we need to find in cMDS.

1) let's assume that we already have a set of centered configuration $x_1, x_2, \dots x_n \epsilon R^q$ for some $q \geq n - 1$ so that their distances are the same as those corresponding distances in the matrix distance. Here, by saying centered configuration, it means:

$$\sum_{i=1}^{n} x_{ik} = 0, for\ k \epsilon q$$

2) Since $d_{ij} = \|x_i - x_j\|_2 = x_i'x_i + x_j'x_j - 2x_i'x_j$. Now, we build a gram matrix $B = X'X$ , so that

$$d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$$

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

3) With a notation T = trace(B) = $\sum_{i=1}^{n} b_{ii}$, we have:

$$\sum_{i=1}^{n} d_{ij}^2 = \sum_{i=1}^{n} b_{ii} + \sum_{i=1}^{n} b_{jj} - \sum_{i=1}^{n} 2b_{ij} = \mathrm{T} + \mathrm{n}\, b_{jj}$$

Same for

$$\sum_{j=1}^{n} d_{ij}^2 = \mathrm{T} + \mathrm{n}\, b_{ii}$$

And then,

$$\sum_{j=1}^{n}\sum_{i=1}^{n} d_{ij}^2 = \sum_{j=1}^{n}\left(T - nb_{jj}\right) = nT + n\sum_{j=1}^{n} b_{jj} = 2nT$$

3) Eventually, we can use $d_{ij}^2$ to express $b_{ij}$:

$$b_{ij} = -\frac{1}{2}(d_{ij}^2 - b_{jj} - b_{ii})$$

$$= -\frac{1}{2}\left(d_{ij}^2 - \frac{\sum_{j=1}^{n} d_{ij}^2 - T}{n} - \frac{\sum_{i=1}^{n} d_{ij}^2 - T}{n}\right)$$

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

4) now, we have gram matrix B, a solution of X is given by the eigen-decomposition of B. That's is , for $B = V\Sigma V'$,

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

5) X lies in the eigenspace where the first coordinate contains the largest variation. To reduce the dimension to p, we can just pick up the first p columns of X, these columns preserves the distance of $d_{ij}$ better than the rest. Then,

$$X_p = \Sigma_p^{\frac{1}{2}} V_p'$$

Where $\Sigma_p^{\frac{1}{2}}$ is the first p by p sub matrix of $\Sigma^{\frac{1}{2}}$, and $V_p$ is the first p columns of $V$.

Other multidimensional Scaling Algorithm

Metric MDS ( used for quantitative distance):
Given a low dimension p and a monotone function $f$, metric MDS seeks to find an optimal configuration X that gives

$$f(d_{ij}) \approx d_{ij}^{*} = \|x_i - x_j\|$$

monotone function $f$ could look like that : $f(d_{ij}) = \alpha + \beta d_{ij}$, but for the most common choice, we just let $f(d_{ij}) = d_{ij}$. However, it's different form cMDS, cause it minimizes the difference between $d_{ij}^{*}$ and $f(d_{ij})$ through minimizing the cost function:

$$Stress = (\sum_{i<j} (d_{ij}^{*} - f(d_{ij}))^2 / \sum d_{ij}^2)^{\frac{1}{2}}$$

Or

$$\text{Sammon Stress} = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij}^{*} - f(d_{ij}))^2}{d_{ij}}$$

Optimal solution is given by numerical computation.

In other application of MDS, distance is know by their rank order. The other information  is not available.

Non-Metric MDS is introduced here to visualize this kind of distance matrix.
The only information we need to keep from the distance matrix is the order, e.g: $d_{ij} < d_{fl} <\ldots< d_{mn}$
The monotone function used here is more general, it only need to preserve the order, like:

$$d_{ij} < d_{fl} \rightarrow f\left(d_{ij}\right) < f(d_{fl})$$

Its goal is to minimize a cost function:

$$stress = \left(\frac{\sum_{i<j}(f\left(d_{ij}\right) - d_{ij}^{*})^2}{\sum_{i<j} f\left(d_{ij}\right)^2}\right)^{\frac{1}{2}}$$

Application:

**Output 33.1.1: City Mileage Data Set**

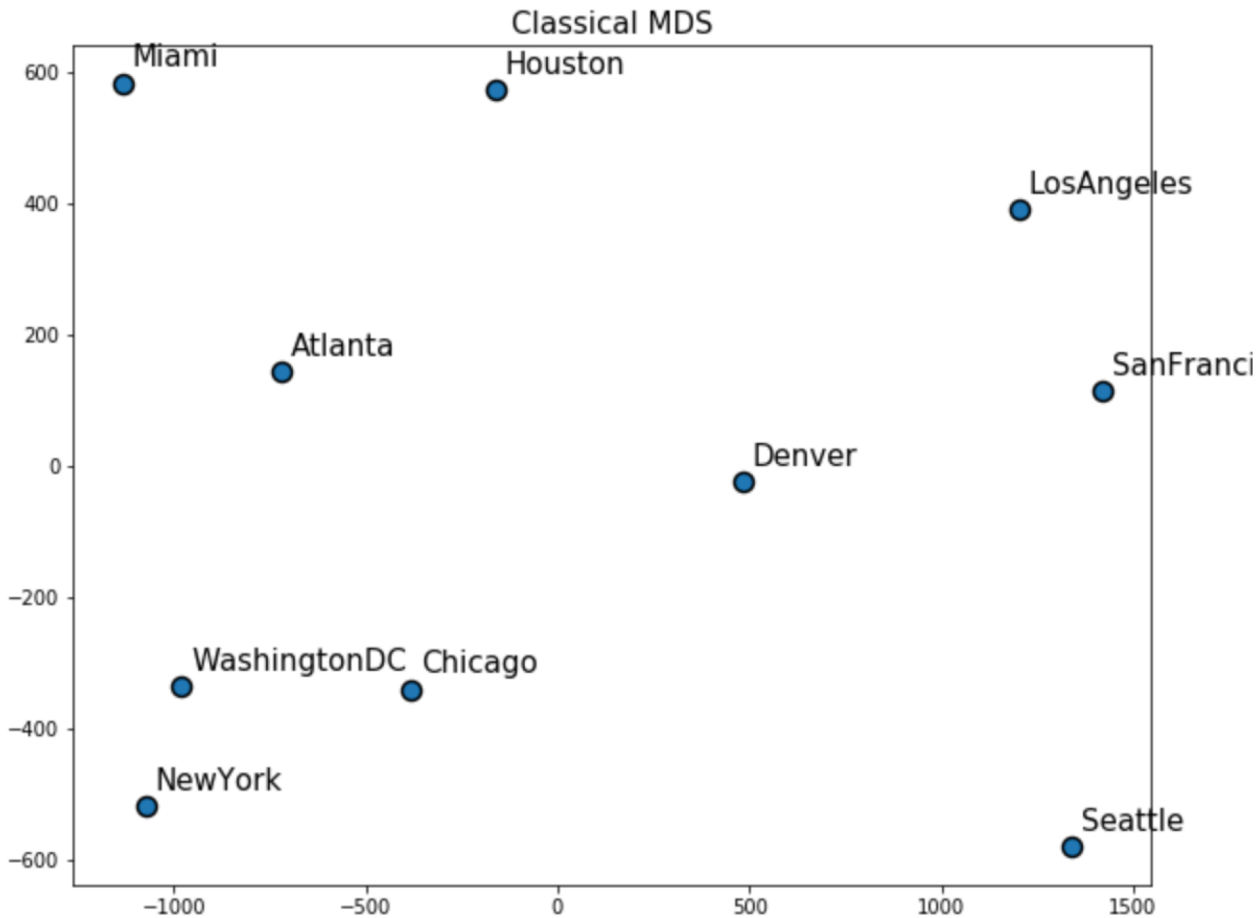
| Atlanta | Chicago | Denver | Houston | LosAngeles | Miami | NewYork | SanFrancisco | Seattle | WashingtonDC | City |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | . | . | . | . | . | . | . | . | . | Atlanta |
| 587 | 0 | . | . | . | . | . | . | . | . | Chicago |
| 1212 | 920 | 0 | . | . | . | . | . | . | . | Denver |
| 701 | 940 | 879 | 0 | . | . | . | . | . | . | Houston |
| 1936 | 1745 | 831 | 1374 | 0 | . | . | . | . | . | Los Angeles |
| 604 | 1188 | 1726 | 968 | 2339 | 0 | . | . | . | . | Miami |
| 748 | 713 | 1631 | 1420 | 2451 | 1092 | 0 | . | . | . | New York |
| 2139 | 1858 | 949 | 1645 | 347 | 2594 | 2571 | 0 | . | . | San Francisco |
| 2182 | 1737 | 1021 | 1891 | 959 | 2734 | 2408 | 678 | 0 | . | Seattle |
| 543 | 597 | 1494 | 1220 | 2300 | 923 | 205 | 2442 | 2329 | 0 | Washington D.C. |

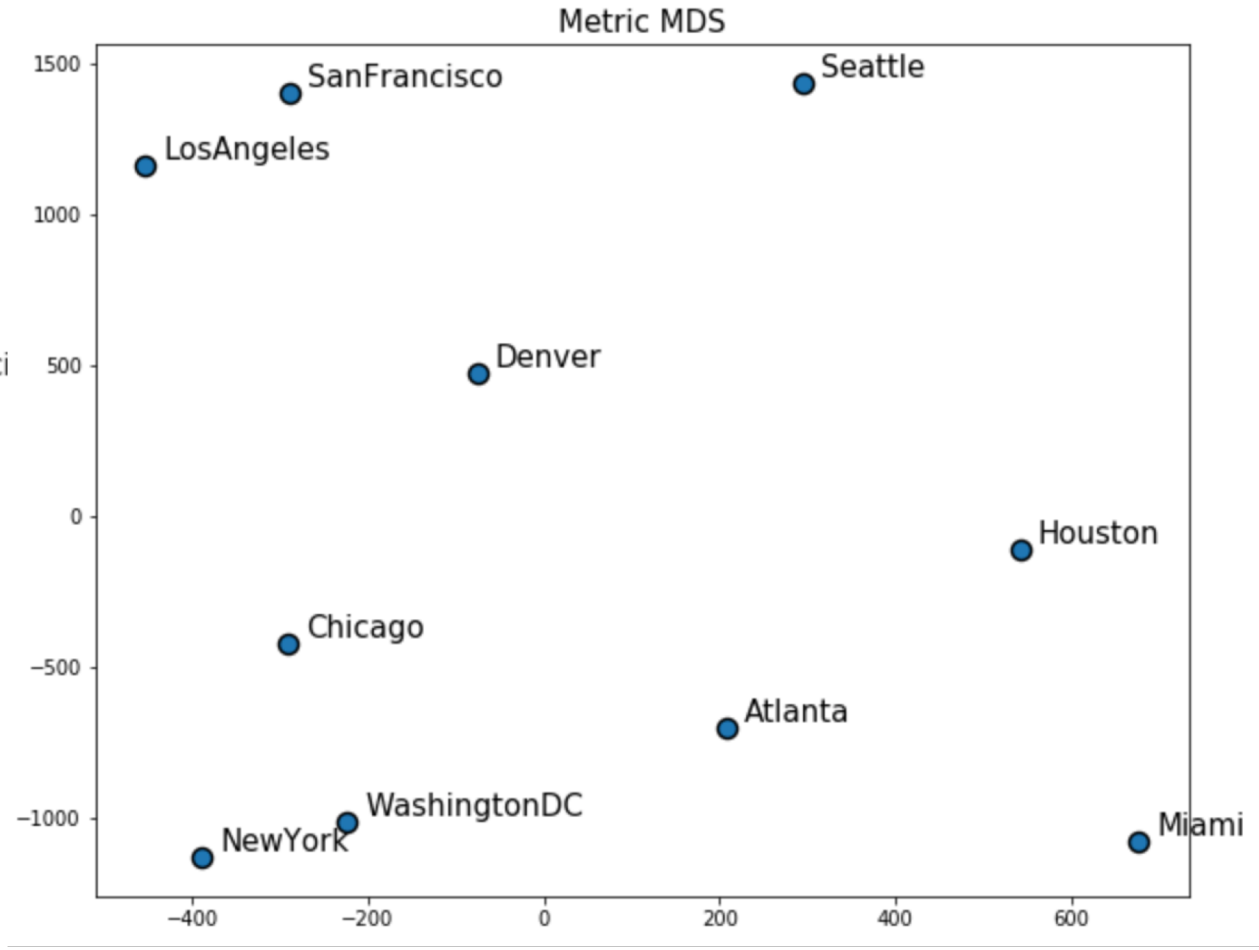
Python code:

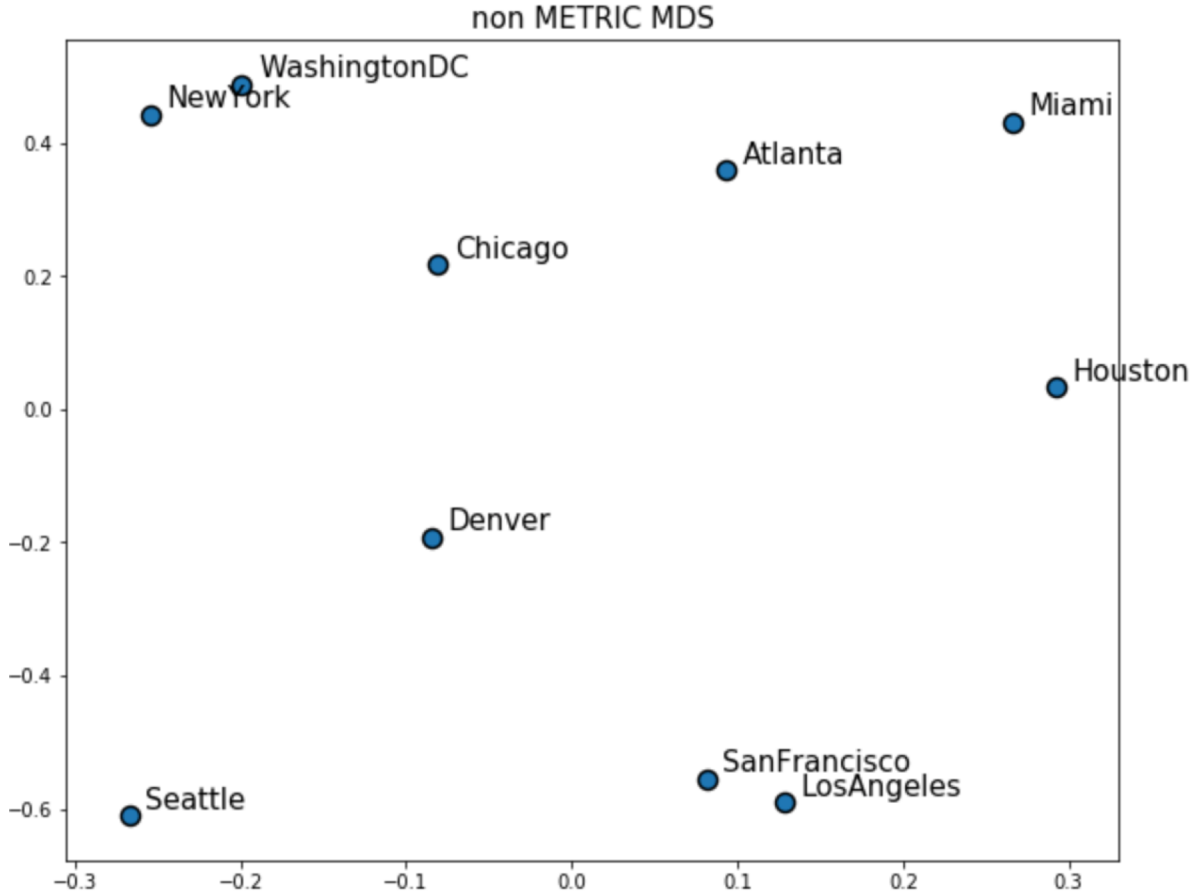
cMDS: sorry, scikit-learn doesn't offer this algorithm.

Metric MDS & non Metric MDS:

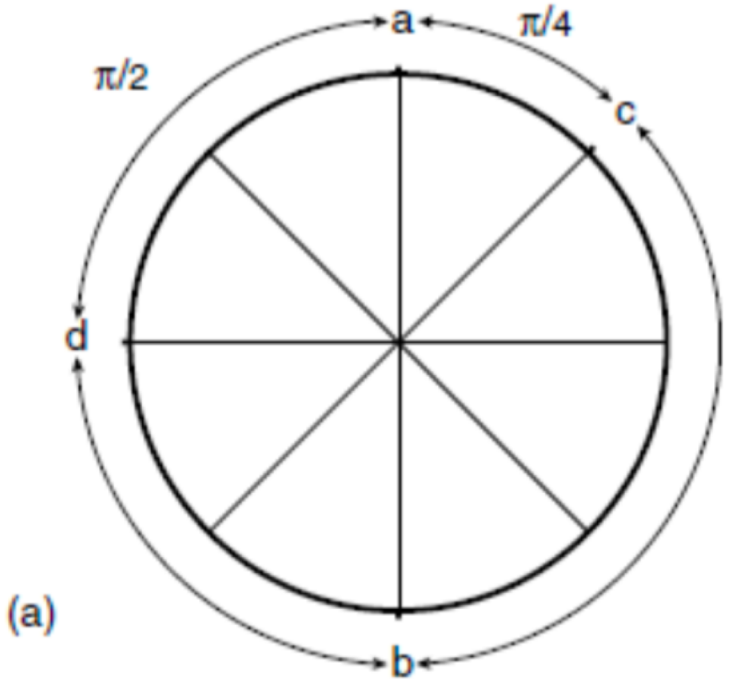
*class* sklearn.manifold.MDS(*n_components=2, metric=True, n_init=4, max_iter=300, verbose=0, eps=0.001, n_jobs=1, random_state=None, dissimilarity='euclidean'* )



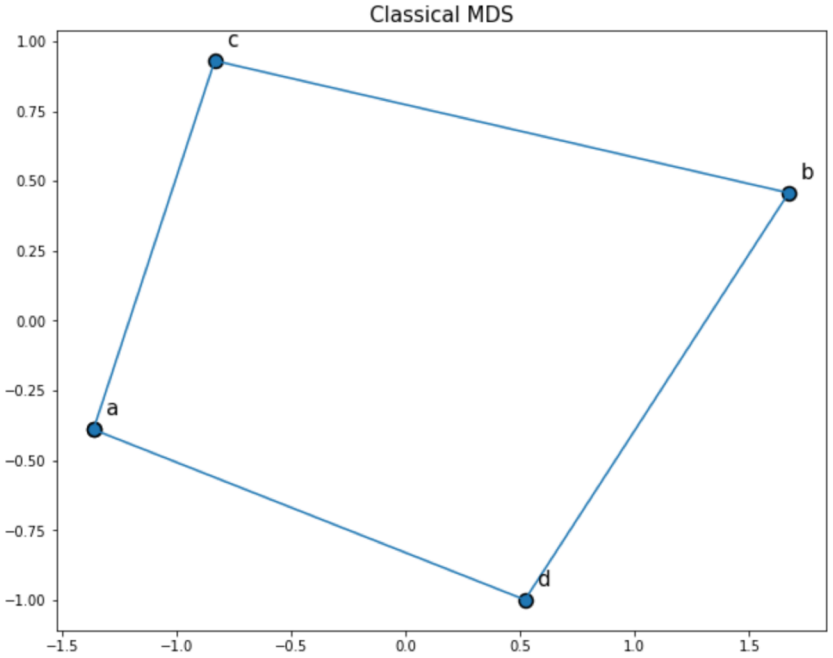

non METRIC MDS
WashingtonDC
NewYork
Miami
Atlanta
Chicago
Houston
Denver
SanFrancisco
LosAngeles
Seattle
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

Application on non-Euclidean distance




```
array([[ 0.        ,  1.57079633,  3.14159265,  0.78539816],
       [ 1.57079633,  0.        ,  1.57079633,  2.35619449],
       [ 3.14159265,  1.57079633,  0.        ,  2.35619449],
       [ 0.78539816,  2.35619449,  2.35619449,  0.        ]])
```

```
array([[ 0.        ,  0.94434314,  3.54311028,  0.13147861],
       [ 0.94434314,  0.        ,  0.82908664,  0.37109223],
       [ 3.54311028,  0.82908664,  0.        ,  2.30953474],
       [ 0.13147861,  0.37109223,  2.30953474,  0.        ]])
```



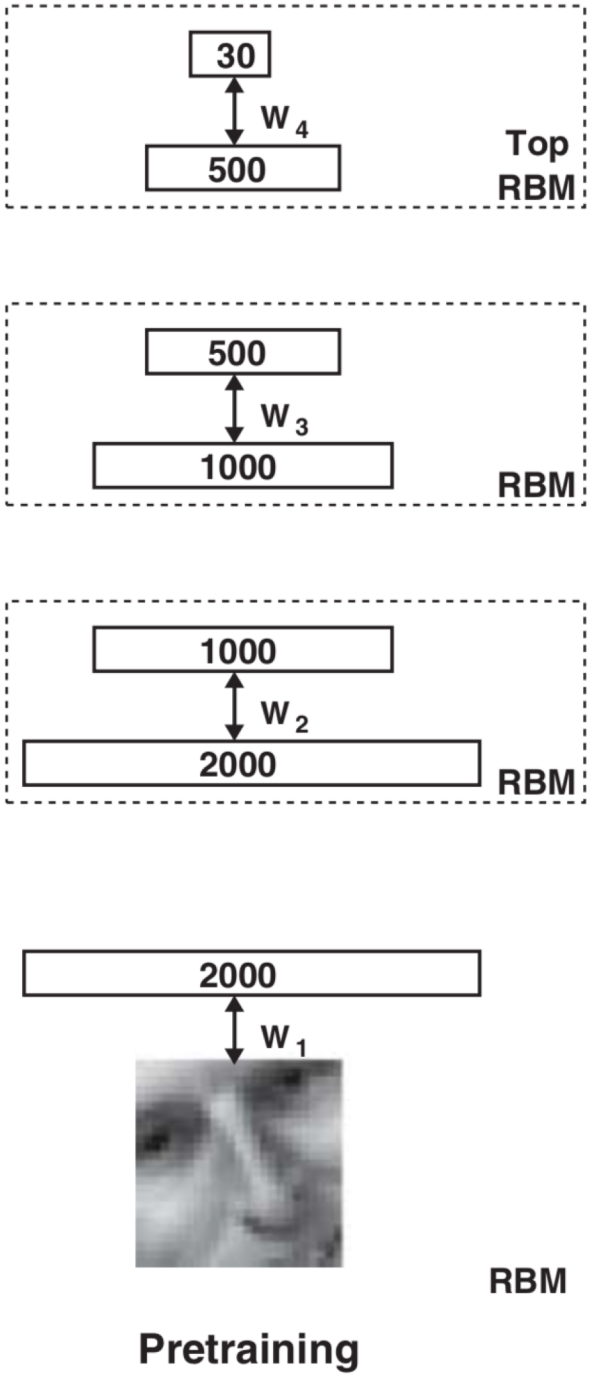
Additional material

Besides the basic math logic of MDS I talked about in the class.
Here I could like to do a small project related to another dimensional reduction technique, that is autoencoder.
Diff from the one in textbook, the autoencoder I introduced here includes the pretrained process of hidden layer, and the goal of my project is to reproduce the result that with the pretrained hidden layer, the gradient could be closer to a good solution than without pretrained. Also, the result of the simple PCA and incremental PCA could also be involved for comparison.

Introduce of restricted Boltmann machine (RBM)
RBM is a two layer network, in which stochastic, binary pixels are connected to stochastic, binary feature detectors using symmetrically weighted connections. The pixels correspond to visible units of the RBM because their states are observed; the feature detectors correspond to hidden units. A joint configuration (v, h) of the visible and hidden units has an energy given by:

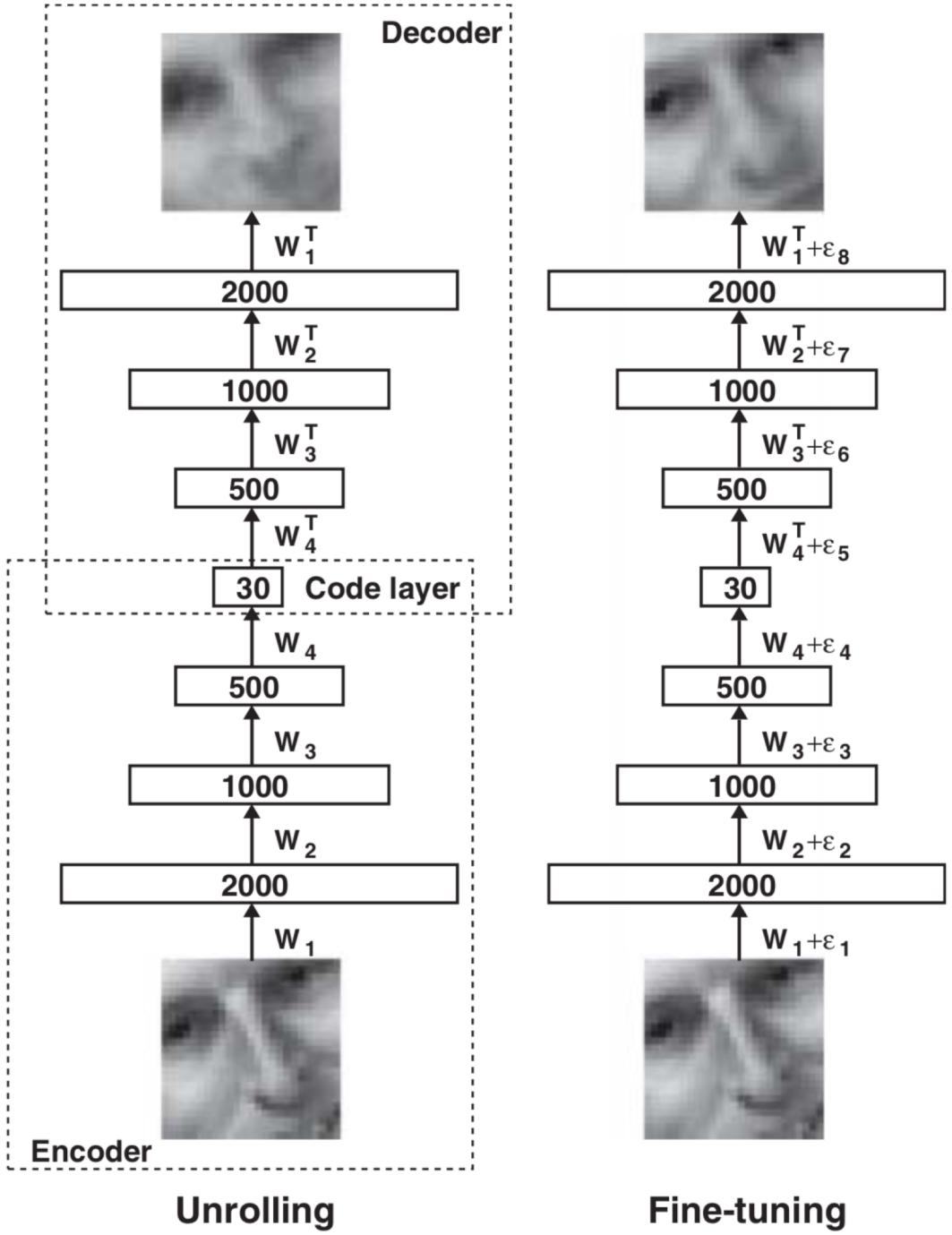
$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{pixels}} b_i v_i - \sum_{j \in \text{features}} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

where $v_i$ and $h_j$ are the binary states of pixel i and feature j, $b_i$ and $b_j$ are their biases, and $w_{ij}$ is the weight between them.




For the pretrain part:

1. For a autoencoder, only consider the encoder part, and break them into two layer network. Train the weight of each two layer network using RBM separately.
2. In the reference, the train set is broke into mini-batch contains 100 samples, and each two layer network would be trained for 50 epochs.
3. The weights were updated after each mini-batch using the averages in the energy function with a learning rate of 0.1. In addition, 0.9 times the previous update was added to each weight and 0.00002 times the value of the weight was subtracted to penalize large weights. (for my project, i didn't make it that complicated)
4. Weights were initialized with small random values sampled from a normal distribution with zero mean and standard deviation of 0.1.




After pretraining the weight between each layer, we stack the encoder layer part and invert the weight of encoder part to make it as the initial weight of decoder part.
Fine tunning the autoencoder in the usual way with early stopper.

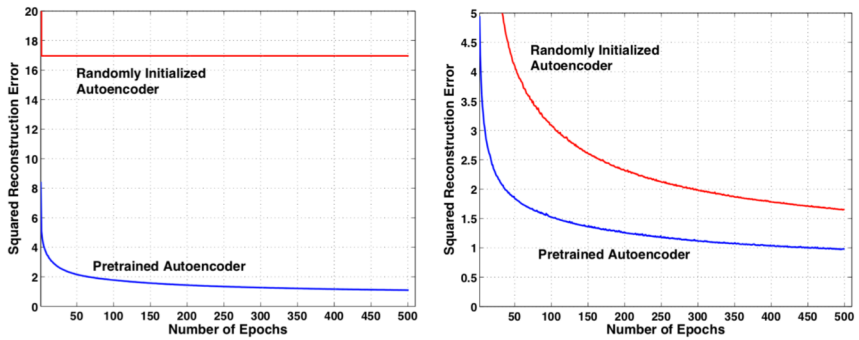
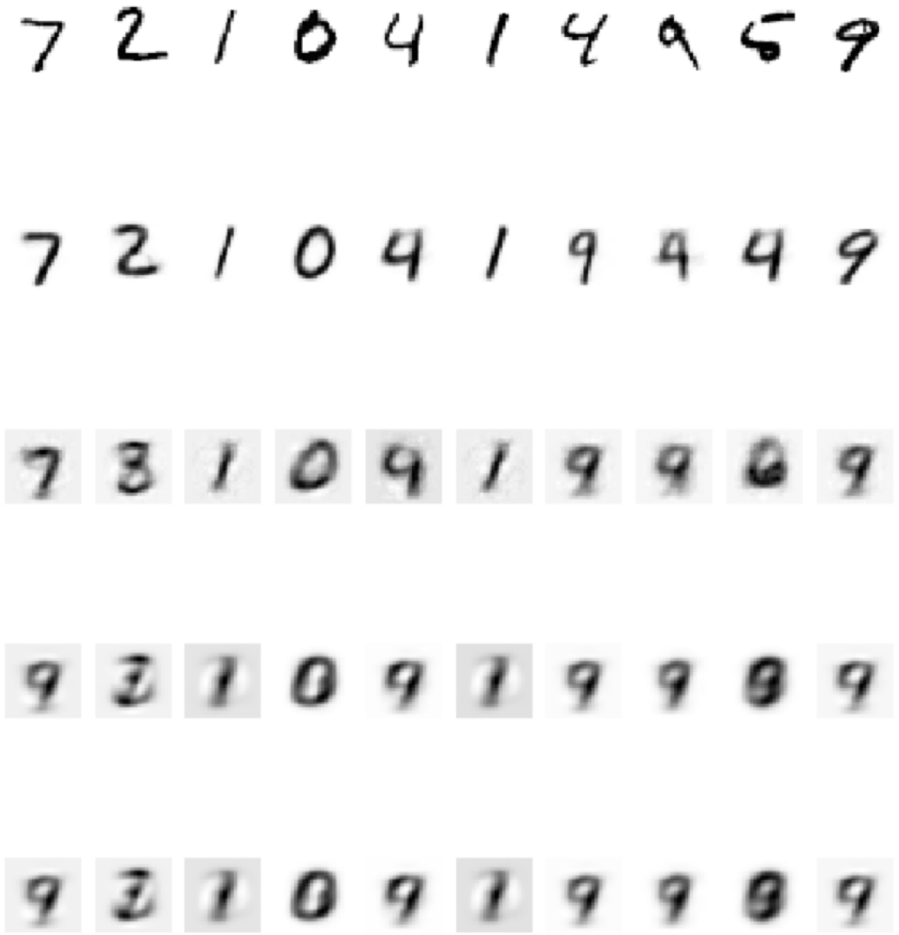



Fig. S1: The average squared reconstruction error per test image during fine-tuning on the curves training data. Left panel: The deep 784-400-200-100-50-25-6 autoencoder makes rapid progress after pretraining but no progress without pretraining. Right panel: A shallow 784-532-6 autoencoder can learn without pretraining but pretraining makes the fine-tuning much faster, and the pretraining takes less time than 10 iterations of fine-tuning.

As the graph provide by the reference shows, the pretrained autoencoder can generate a better solution with a quick speed. The effect is more obvious for deep neural network.



The result of my project:
I use the MINST dataset here.
The 1st row is the original number pic
The 2nd row is the number pic coming out from pretrained autoencoder that has the encoder structure of 784-1000-500-250-2
The 3rd row is the number pic coming out from the same structure autoencoder without pretrained.
The 4th row is the number pic coming from the simple PCA with 2-components.
The 5th row is the number pic coming from the incremental PCA with 2-components.

It's obvious the pic in 2nd row are more similar to the pic in the 1st row. It looks clear, while the pic of other rows are not that clear and easy to identified.

The MSE of pretrained autoencoder is much more smaller than the other 3 techniques. Interesting thing is that the MSE of autoencoder is even higher than simple PCA, it's the worst !
Maybe without pretrained, the stochastic initial value could easily guide the gradients to a local minimum, and the result could be terrible.

Reference


1. G. E. Hinton* and R. R. Salakhutdinov, "*Reducing the Dimensionality of Data with Neural Networks*", *SCIENCE* 313 (2016)
2. G. E. Hinton* and R. R. Salakhutdinov, "*Support Material for Reducing the Dimensionality of Data with Neural Networks",* www.sciencemag.org/cgi/content/full/313/5786/504/DC1
3. Liyanaarachchi Lekamalage Chamara Kasun, Yan Yang, Guang-Bin Huang, and Zhengyou Zhang, "*Dimension Reduction With Extreme Learning Machine", IEEE TRANSACTIONS ON IMAGE PROCESSING*, VOL. 25, NO. 8, AUGUST 2016
4. Shufeng Tan and Michael L. Mavrovouniotis, "*Reducing Data Dimensionality through optimizing Neural Network Inputs", AIChE Journal June 1995 Vol. 41, No. 6*
5. Coding of pretrained autoencoder, I refer to the work here: https://github.com/Cospel/rbm-ae-tf
6. Coding of autoencoder without pretrained, I refer to the work of textbook: https://github.com/jjkindergarten/ageron_handson-ml
7. Introduction of Euclidean dissimilarities: http://37steps.com/1617/non-euclidean-and-non-metric-dissimilarities/
8. Example I used for MDS refers to: http://support.sas.com/documentation/cdl/en/statug/67523/HTML/default/viewer.htm#statug_mds_gettingstarted.html
9. The math part of MDS refers to: http://www.stat.pitt.edu/sungkyu/course/2221Fall13/lec8_mds_combined.pdf