Spectral Clustering

by HU Pili

June 16, 2013

Outline

- Clustering Problem
- Spectral Clustering Demo
- Preliminaries
    - Clustering: K-means Algorithm
    - Dimensionality Reduction: PCA, KPCA.
- Spectral Clustering Framework
- Spectral Clustering Justification
- Ohter Spectral Embedding Techniques

Main reference: Hu 2012 [4]

Clustering Problem

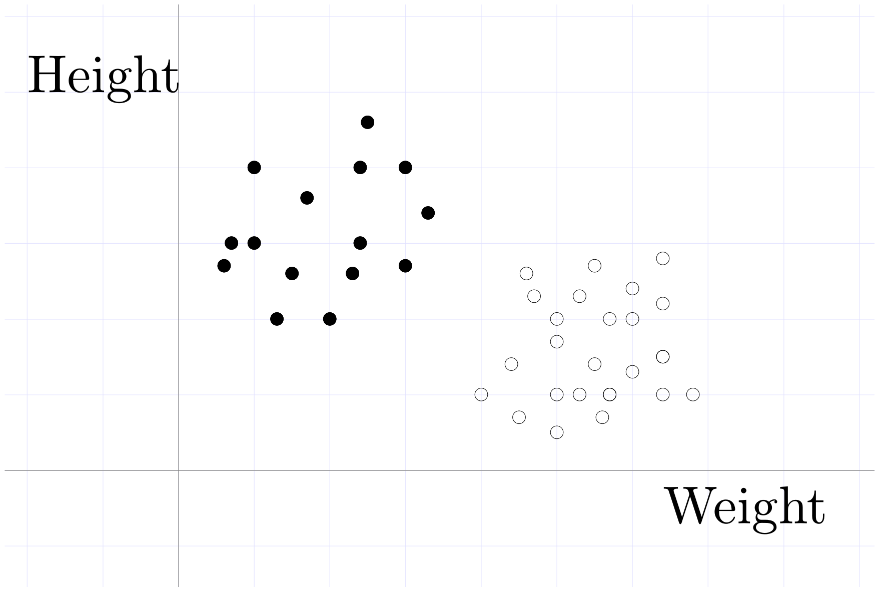

Figure 1. Abstract your target using feature vector

Clustering Problem

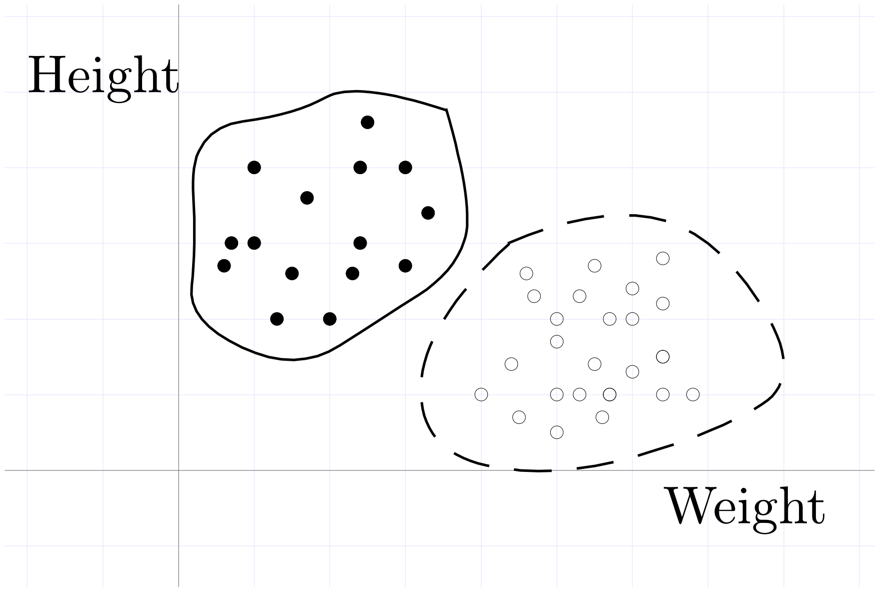

Figure 2. Cluster the data points into K (2 here) groups

Clustering Problem

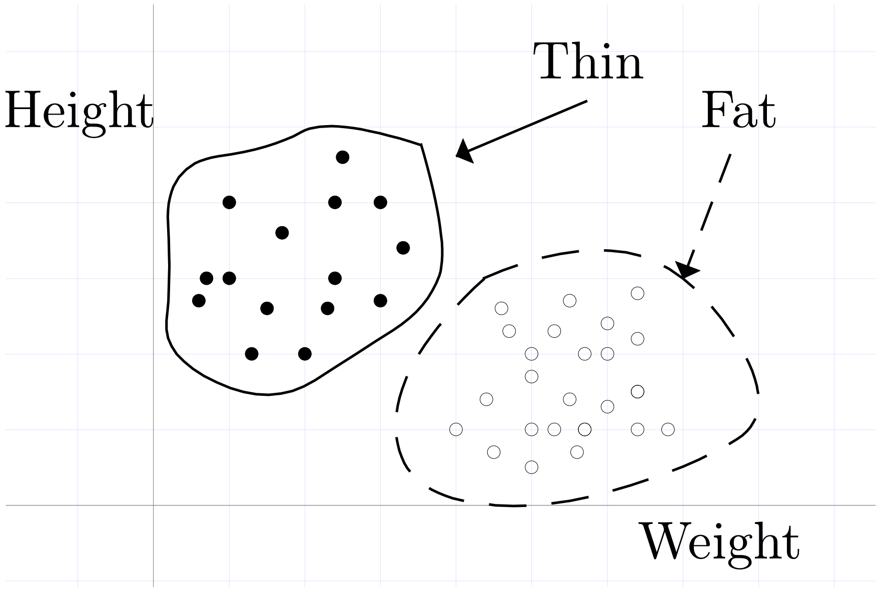

Figure 3. Gain Insights of your data

Clustering Problem

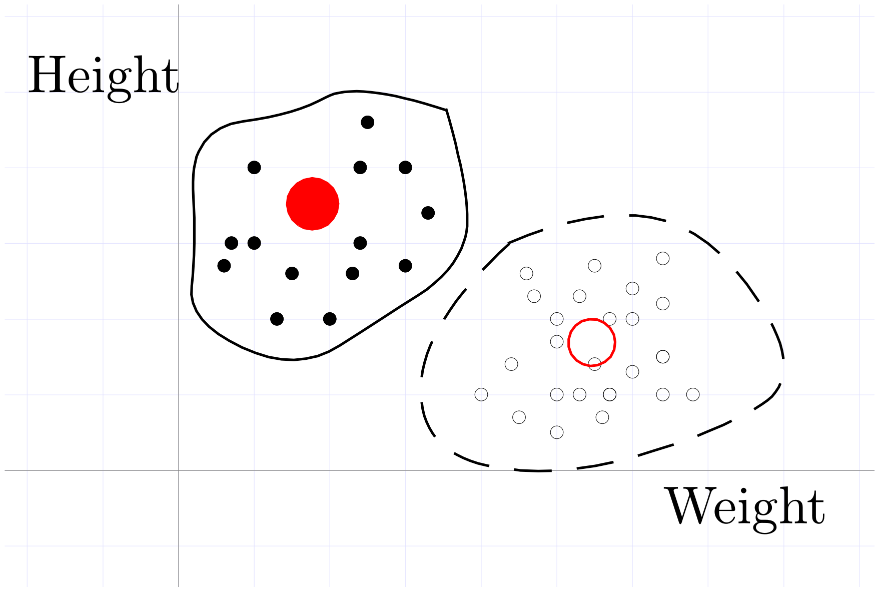

Figure 4. The center is representative (knowledge)

Clustering Problem

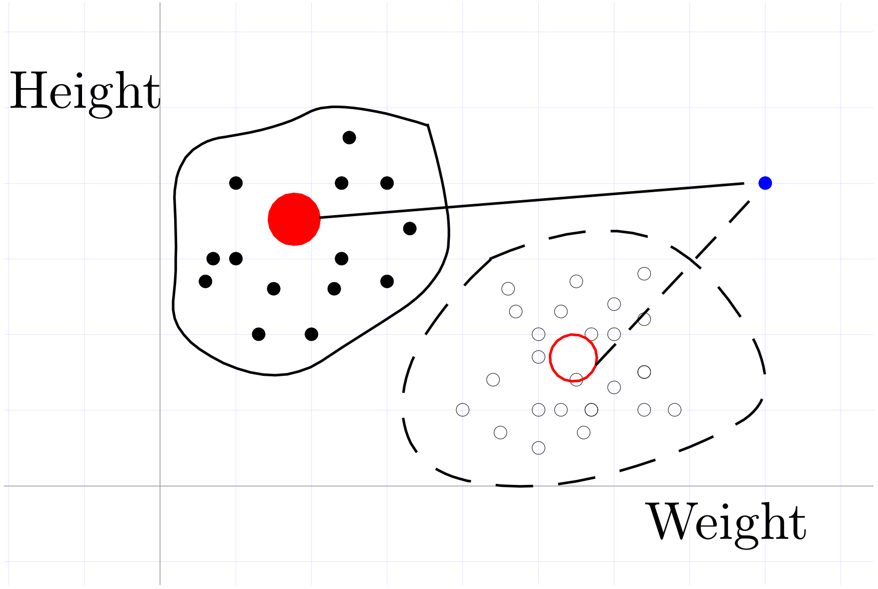

Figure 5. Use the knowledge for prediction

Review: Clustering

We learned the general steps of Data Mining/ Knowledge Discover using a clustering example:

1. Abstract your data in form of vectors.
2. Run learning algorithms
3. Gain insights/ extract knowledge/ make prediction

We focus on 2.

Spectral Clustering Demo

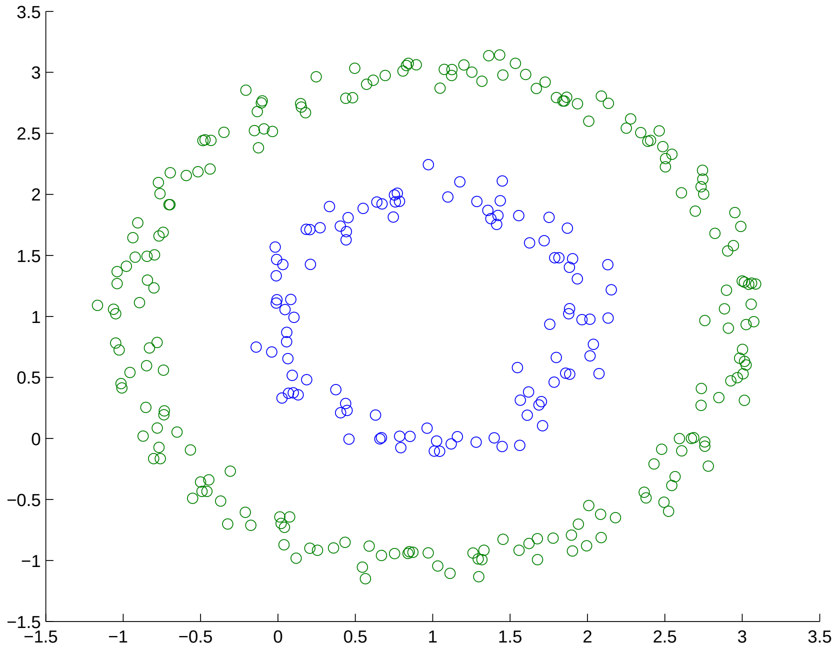

Figure 6. Data Scatter Plot

Spectral Clustering Demo

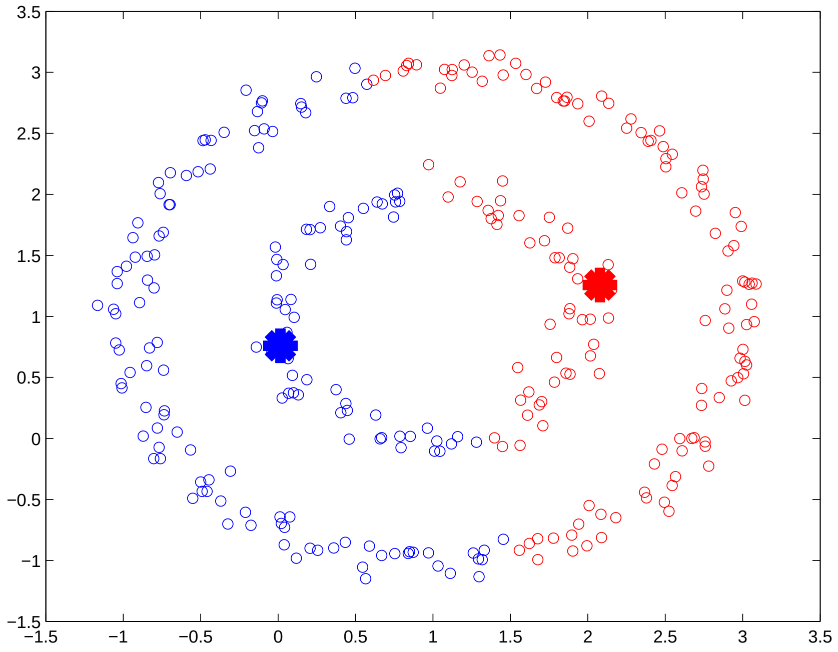

Figure 7. Standard K-Means

Spectral Clustering Demo

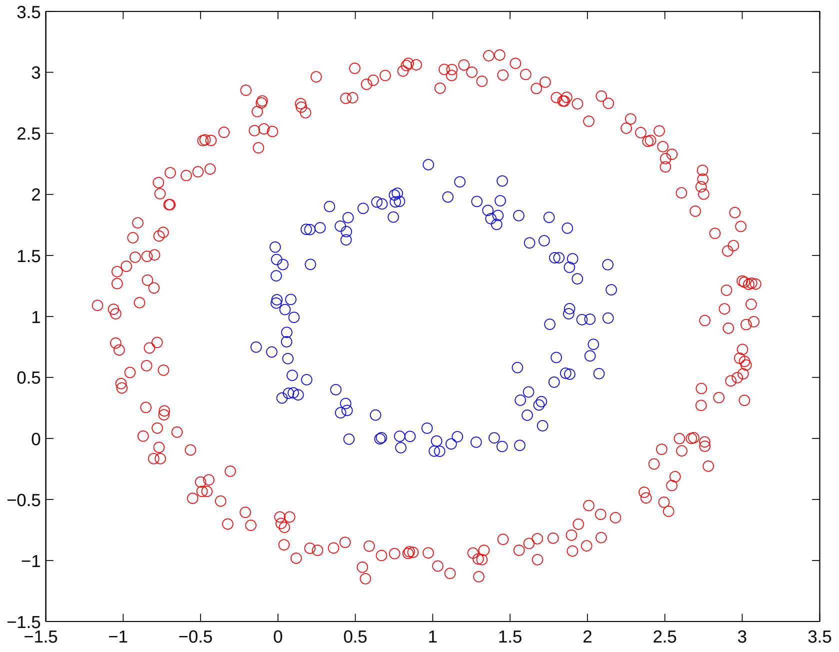

Figure 8. Our Sample Spectral Clustering

Spectral Clustering Demo

The algorithm is simple:

- K-Means:

  ```
  [idx, c] = kmeans(X, K) ;
  ```

- Spectral Clustering:

  ```
  epsilon = 0.7 ;
  D = dist(X') ;
  A = double(D < epsilon) ;
  [V, Lambda] = eigs(A, K) ;
  [idx, c] = kmeans(V, K) ;
  ```

Review: The Demo

The usual case in data mining:

- No weak algorithms
- Preprocessing is as important as algorithms
- The problem looks easier in **another space** (the secret coming soon)

Transformation to another space:

- High to low: dimensionality reduction, low dimension embedding. e.g. Spectral clustering.
- Low to high. e.g. Support Vector Machine (SVM)

Secrets of Preprocessing

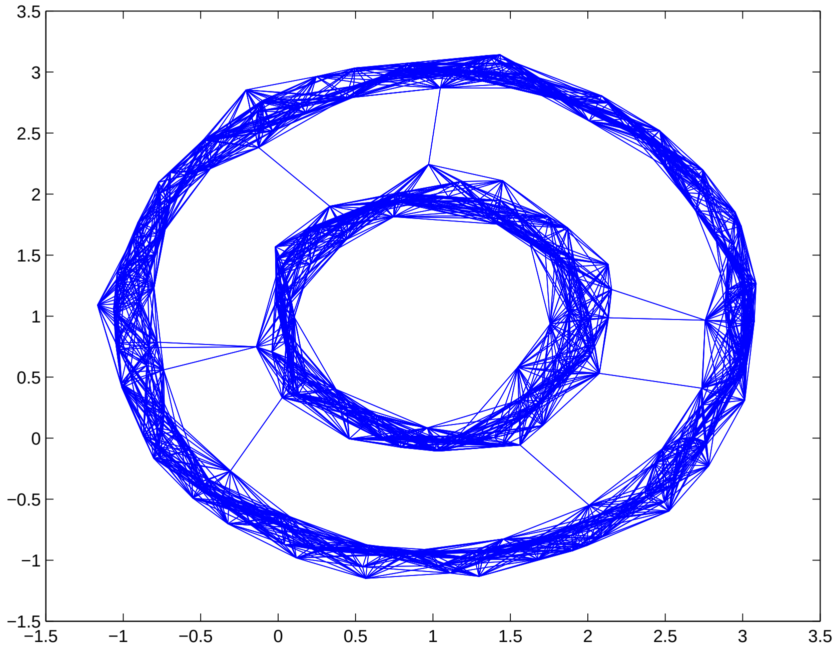

Figure 9. The similarity graph: connect to ε-ball.

```
D = dist(X'); A = double(D < epsilon);
```

Secrets of Preprocessing

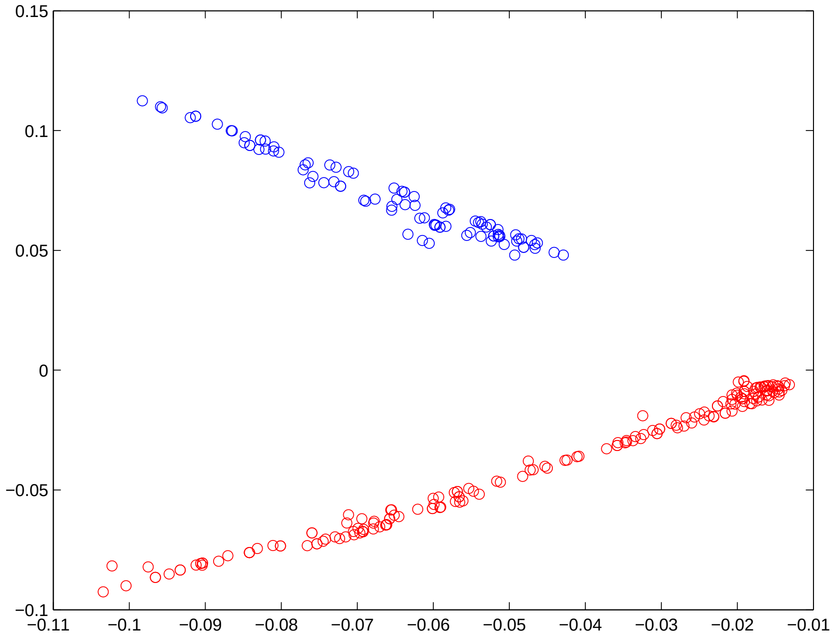

Figure 10. 2-D embedding using largest 2 eigenvectors

```
[V, Lambda] = eigs(A, K);
```

Secrets of Preprocessing

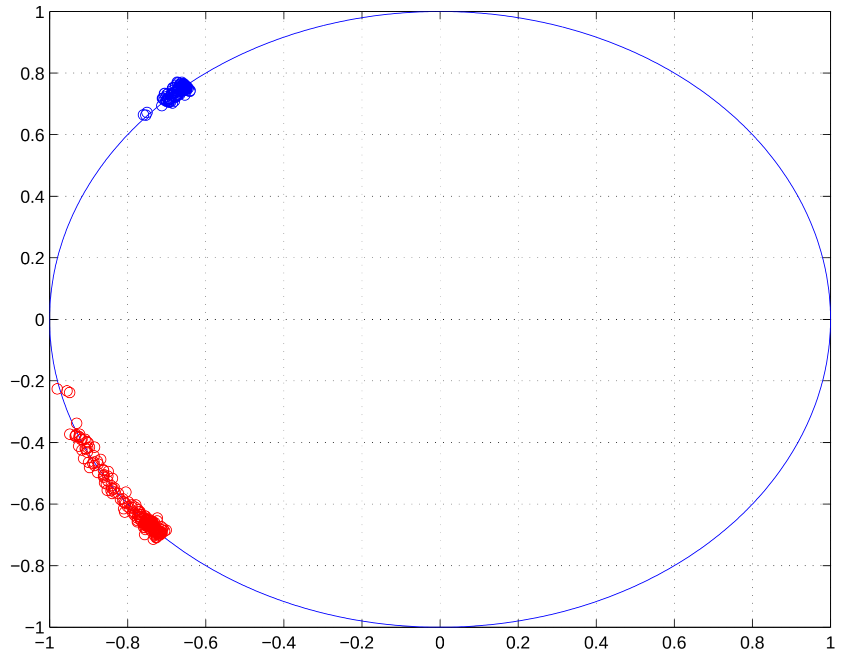

Figure 11. Even better after projecting to unit circle (not used in our sample but more applicable, Brand 2003 [3])

Secrets of Preprocessing

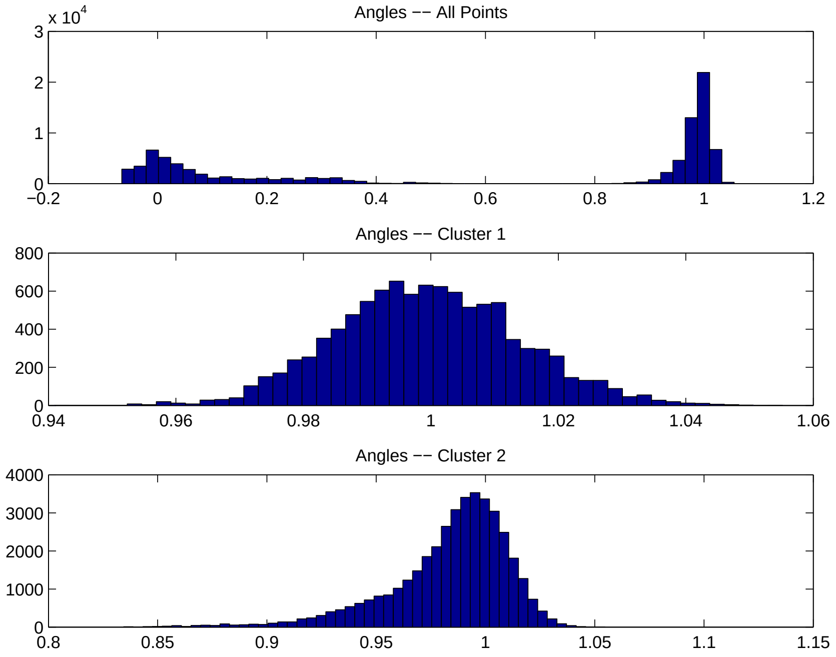

Figure 12. Angle histogram: The two clusters are concentrated and they are nearly perpendicular to each other

Notations

- Data points: $X_{n \times N} = [x_1, x_2, ..., x_N]$. N points, each of n dimensions. Organize in columns.
- Feature extraction: $\phi(x_i)$. d dimensional.
- Eigen value decomposition: $A = U\Lambda U^{\mathrm{T}}$. First d colums of U: U_d.
- Feature matrix: $\Phi_{\hat{n} \times N} = [\phi(x)_1, \phi(x)_2, ..., \phi(x)_N]$
- Low dimension embedding: $Y_{N \times d}$. Embed N points into d-dimensional space.
- Number of clusters: K.

K-Means

- Initialize $m_1, ..., m_K$ centers
- Iterate until convergence:
  - Cluster assignment: $C_i = \arg\min_j \|x_i - m_j\|$ for all data points, $1 \leqslant i \leqslant N$.
  - Update clustering: $S_j = \{i : C_i = j\}, 1 \leqslant j \leqslant K$
  - Update centers: $m_j = \frac{1}{|S_j|} \sum_{i \in S_j} x_i$

Remarks: K-Means

1. A chiken and egg problem.
2. How to initialize centers?
3. Determine superparameter K?
4. Decision boundary is a straight line:
    - $C_i = \arg\min_j \|x_i - m_j\|$
    - $\|x - m_j\| = \|x - m_k\|$
    - $m_i^{\mathrm{T}} m_i - m_j^{\mathrm{T}} m_j = 2x^{\mathrm{T}}(m_i - m_j)$

We address the 4th points by transforming the data into a space where straight line boundary is enough.

Principle Component Analysis

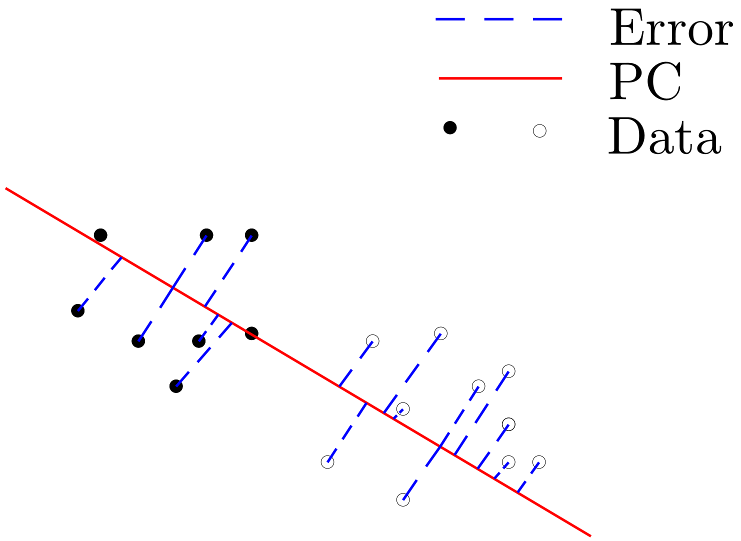

Figure 13. Error minimization formulation

Principle Component Analysis

Assume x_i is already **centered** (easy to preprocess). Project points onto the space spanned by $U_{n\times d}$ with minimum errors:

$$\begin{aligned} \min_{U\in\mathbb{R}^{n\times d}} \quad & J(U)=\sum_{i=1}^{N}\|UU^{\mathrm{T}}x_i-x_i\|^2 \\ s.t. \quad & U^{\mathrm{T}}U=I \end{aligned}$$

Principle Component Analysis

Transform to trace maximization:

$$\begin{aligned} \max_{U \in \mathbb{R}^{n \times d}} \quad & \mathrm{Tr}[U^{\mathrm{T}}(X X^{\mathrm{T}})U] \\ s.t. \quad & U^{\mathrm{T}}U = I \end{aligned}$$

Standard problem in matrix theory:

- Solution of U is given by the largest d eigen vectors of $X X^{\mathrm{T}}$ (those corresponding to largest eigen values). i.e. $X X^{\mathrm{T}}U = U\Lambda$.
- Usually denote $\Sigma_{n \times n} = X X^{\mathrm{T}}$, because $X X^{\mathrm{T}}$ can be interpreted as the covariance of variables (n variables). (Again not that X is centered)

Principle Component Analysis

About $UU^{\mathrm{T}}x_i$:

- x_i: the data points in original n dimensional space.
- $U_{n\times d}$: the d dimensional space (axis) **expressed** using the coordinates of original n-D space. – **Principle Axis**.
- $U^{\mathrm{T}}x_i$: the coordinates of x_i in d-dimensional space; d-dimensional **embedding**; the projection of x_i to d-D space **expressed** in d-D space. – **Principle Component**.
- $UU^{\mathrm{T}}x_i$: the projection of x_i to d-D space expressed in n-D space.

Principle Component Analysis

From principle axis to principle component:

- One data point: $U^{\mathrm{T}} x_i$
- Matrix form: $(Y^{\mathrm{T}})_{d \times N} = U^{\mathrm{T}} X$

Relation between covariance and similarity:

$$\begin{aligned}
(X^{\mathrm{T}} X) Y &= X^{\mathrm{T}} X (U^{\mathrm{T}} X)^{\mathrm{T}} \\
&= X^{\mathrm{T}} X X^{\mathrm{T}} U \\
&= X^{\mathrm{T}} U \Lambda \\
&= Y \Lambda
\end{aligned}$$

Observation: Y is the eigen vectors of $X^{\mathrm{T}} X$.

Principle Component Analysis

Two operational approaches:

- Decompose $(X X^{\mathrm{T}})_{n\times n}$ and Let $(Y^{\mathrm{T}})_{d\times N}=U^{\mathrm{T}}X$.
- Decompose $(X^{\mathrm{T}}X)_{N\times N}$ and directly get Y.

Implications:

- Choose the smaller size one in practice.
- $X^{\mathrm{T}}X$ hint that we can do more with the structure.

Remarks:

- Principle components are what we want in most cases. i.e. Y. i.e. d-dimension embedding. e.g. Can do clustering on coordinates given by Y.

Kernel PCA

Settings:

- A feature extraction function:

$$\phi\colon \mathbb{R}^n \rightarrow \mathbb{R}^{\hat{n}}$$

original n dimension features. Map to $\hat{n}$ dimension.

- Matrix organization:

$$\Phi_{\hat{n}\times N} = [\phi(x)_1, \phi(x)_2, \ldots, \phi(x)_N]$$

- Now Φ is the "data points" in the formulation of PCA. Try to embed them into d-dimensions.

Kernel PCA

According to the analysis of PCA, we can operate on:

- $(\Phi\Phi^{\mathrm{T}})_{\hat{n}\times\hat{n}}$: the covariance matrix of features
- $(\Phi^{\mathrm{T}}\Phi)_{N\times N}$: the similarity/ affinity matrix (in spectral clustering language); the gram/ kernal matrix (in keneral PCA language).

The observation:

- $\hat{n}$ can be very large. e.g. $\phi\colon\mathbb{R}^n\to\mathbb{R}^\infty$
- $(\Phi^{\mathrm{T}}\Phi)_{i,j}=\phi^{\mathrm{T}}(x_i)\phi(x_j)$. Don't need explict ϕ; only need $k(x_i,x_j)=\phi^{\mathrm{T}}(x_i)\phi(x_j)$.

Kernel PCA

$k(x_i, x_j) = \phi^{\mathrm{T}}(x_i)\phi(x_j)$ is the “kernel”. One important property: by definition,

- $k(x_i, x_j)$ is a positive semidefinite function.
- K is a positive semidefinite matrix.

Some example kernals:

- Linear: $k(x_i, x_j) = x_i^{\mathrm{T}} x_j$. Degrade to PCA.
- Polynomial: $k(x_i, x_j) = (1 + x_i^{\mathrm{T}} x_j)^p$
- Gaussion: $k(x_i, x_j) = e^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}$

Remarks: KPCA

- Avoid explicit high dimension (maybe infinite) feature construction.
- Enable one research direction: kernel engineering.
- The above discussion **assume $\boldsymbol{\Phi}$ is centered**! See Bishop 2006 [2] for how to center this matrix (using only kernel function). (or "double centering" technique in [4])
- Out of sample embedding is the real difficulty, see Bengio 2004 [1].

Review: PCA and KPCA

- Minimum error formulation of PCA
- Two equivalent implementation approaches:
  - covariance matrix
  - similarity matrix
- Similarity matrix is more convenient to manipulate and leads to KPCA.
- Kernel is Positive Semi-Definite (PSD) by definition. $K = \Phi^{\mathrm{T}}\Phi$

Spectral Clustering Framework

A bold guess:

- Decomposing $K = \Phi^{\mathrm{T}}\Phi$ gives good low-dimension embedding. Inner product measures similarity, i.e. $k(x_i, x_j) = \phi^{\mathrm{T}}(x_i)\phi(x_j)$. K is similarity matrix.
- In the operation, we acturally do not look at Φ.
- We can specify K directly and perform EVD:

$$X_{n \times N} \to K_{N \times N}$$

- What if we directly give a similarity measure, K, without the constraint of PSD?

That leads to the general spectral clustering.

Spectral Clustering Framework

1. Get similarity matrix $A_{N \times N}$ from data points X. (A: affinity matrix; adjacency matrix of a graph; similarity graph)
2. EVD: $A = U \Lambda U^T$. Use U_d (or post-processed version, see [4]) as the d-dimension embedding.
3. Perform clustering on d-D embedding.

Spectral Clustering Framework

Review our naive spectral clustering demo:

1.
   ```
   epsilon = 0.7 ;
   D = dist(X') ;
   A = double(D < epsilon) ;
   ```

2.
   ```
   [V, Lambda] = eigs(A, K) ;
   ```

3.
   ```
   [idx, c] = kmeans(V, K) ;
   ```

Remarks: SC Framework

- We start by relaxing A (K) in KPCA.
- Lose PSD == Lose KPCA justification? Not exact

$$A' = A + \sigma I$$

- Real tricks: (see [4] section 2 for details)
  - How to form A?
  - Decompose A or other variants of A ($L = D - A$).
  - Use EVD result directly (e.g. U) or use a variant (e.g. $U\Lambda^{1/2}$).

Similarity graph

Input is high dimensional data: (e.g. come in form of X)

- k-nearest neighbour
- ε-ball
- mutual k-NN
- complete graph (with Gaussian kernel weight)

Similarity graph

Input is distance. $\left(D^{(2)}\right)_{i,j}$ is the squred distance between i and j. (may not come from raw x_i, x_j)

$$\begin{aligned}
c &= [x_1^{\mathrm{T}} x_1, \ldots, x_N^T x_N]^{\mathrm{T}} \\
D^{(2)} &= c\,\mathbf{1}^{\mathrm{T}} + \mathbf{1}\,c^{\mathrm{T}} - 2X^{\mathrm{T}}X \\
J &= I - \frac{1}{n}\mathbf{1}\,\mathbf{1}^{\mathrm{T}} \\
X^{\mathrm{T}}X &= -\frac{1}{2} J D^{(2)} J
\end{aligned}$$

Remarks:

- See MDS.

Similarity graph

Input is a graph:

- Just use it.
- Or do some enhancements. e.g. Geodesic distance. See [4] Section 2.1.3 for some possible methods.

After get the graph (or input):

- Adjacency matrix A, Laplacian matrix $L = D - A$.
- Normalized versions:
  - $A_{\text{left}} = D^{-1} A$, $A_{\text{sym}} = D^{-1/2} A\, D^{-1/2}$
  - $L_{\text{left}} = D^{-1} L$, $L_{\text{sym}} = D^{-1/2} L\, D^{-1/2}$

EVD of the Graph

Matrix types:

- Adjacency series: Use the largest EVs.
- Laplacian series: Use the smallest EVs.

Remarks: SC Framework

- There are many possibilities in construction of similarity matrix and the post-processing of EVD.
- Not all of these combinations have justifications.
- Once a combination is shown to working, it may not be very hard to find justifications.
- Existing works actually starts from very different flavoured formulations.
- Only one common property: involve EVD; aka "spectral analysis"; hence the name.

Spectral Clustering Justification

- Cut based argument (main stream; origin)
- Random walk escaping probability
- Commute time: L^{-1} encodes the effective resistance. (where $U\Lambda^{-1/2}$ come from)
- Low-rank approximation.
- Density estimation.
- Matrix perturbation.
- Polarization. (the demo)

See [4] for pointers.

Cut Justification

Normalized Cut (Shi 2000 [5]):

$$\mathrm{NCut} = \sum_{i=1}^{K} \frac{\mathrm{cut}(C_i, V - C_i)}{\mathrm{vol}(C_i)}$$

Characteristic vector for C_i, $\chi_i = \{0, 1\}^N$:

$$\mathrm{NCut} = \sum_{i=1}^{K} \frac{\chi_i^{\mathrm{T}} L \chi_i}{\chi_i^{\mathrm{T}} D \chi_i}$$

Relax χ_i to real value:

$$\begin{aligned} \min_{v_i \in \mathbb{R}^N} \quad & \sum_{i=1}^{K} v_i^{\mathrm{T}} L \, v_i \\ s.t. \quad & v_i^{\mathrm{T}} D \, v_i = 1 \\ & v_i^{\mathrm{T}} \, v_j = 0, \forall i \neq j \end{aligned}$$

This is the generalized eigenvalue problem:

$$L \, v = \lambda D \, v$$

Equivalent to EVD on:

$$L_{\mathrm{left}} = D^{-1} L$$

Matrix Perturbation Justification

- When the graph is ideally separable, i.e. multiple connected components, A and L have characteristic (or piecewise linear) EVs.
- When not ideally separable but sparse cut exists, A can be viewed as ideal separable matrix plus a small perturbation.
- Small perturbation of matrix entries leads to small perturbation of EVs.
- EVs are not too far from piecewise linear: easy to separate by simple algorithms like K-Means.

Low Rank Approximation

The similarity matrix A is generated by inner product in some unknown space we want to recover. We want to minimize the recovering error:

$$\min_{Y \in \mathbb{R}^{N \times d}} \|A - YY^T\|_F^2$$

The standard low-rank approximation problem, which leads to EVD of A:

$$Y = U\Lambda^{1/2}$$

Spectral Embedding Techniques

See [4] for some pointers: MDS, isomap, PCA, KPCA, LLE, LEmap, HEmap, SDE, MVE, SPE. The difference, as said, lies mostly in the construction of A.

Bibliography

[1] Y. Bengio, J. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in neural information processing systems*, 16:177–184, 2004.

[2] C. Bishop. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[3] M. Brand and K. Huang. A unifying theorem for spectral embedding and clustering. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[4] P. Hu. Spectral clustering survey, 5 2012.

[5] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

Thanks

Q/A

Some supplementary slides for details are attached.

SVD and EVD

Definitions of Singular Value Decomposition (SVD):

$$X_{n \times N} = U_{n \times k} \Sigma_{k \times k} V^{\mathrm{T}}_{N \times k}$$

Definitions of Eigen Value Decomposition (EVD):

$$\begin{aligned} A &= X^{\mathrm{T}} X \\ A &= U \Lambda U^{\mathrm{T}} \end{aligned}$$

Relations:

$$\begin{aligned} X^{\mathrm{T}} X &= V \Sigma^2 V^{\mathrm{T}} \\ X X^{\mathrm{T}} &= U \Sigma^2 U^{\mathrm{T}} \end{aligned}$$

Remarks:

- SVD requires $U^{\mathrm{T}}U = I$, $V^{\mathrm{T}}V = I$ and $\sigma_i \geqslant 0$ ($\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_N)$). This is to guarantee the uniqueness of solution.
- EVD does not have constraints, any U and Λ satisfying $A\,U = U\Lambda$ is OK. The requirement of $U^{\mathrm{T}}U = I$ is also to guarantee uniqueness of solution (e.g. PCA). Another benefit is the numerical stability of subspace spanned by U: orthogonal layout is more error resilient.
- The computation of SVD is done via EVD.
- Watch out the terms and the object they refer to.

Out of Sample Embedding

- New data point $x \in \mathbb{R}^n$ that is not in X. How to find the lower dimension embedding, i.e. $y \in \mathbb{R}^d$.
- In PCA, we have principle axis U $(X\,X^{\mathrm{T}} = U \Lambda U^T)$. Out of sample embedding is simple: $y = U^{\mathrm{T}} x$.
- $U_{n \times d}$ is actually a **compact representation of knowledge**.
- In KPCA and different variants of SC, we operate on similarity graph and do not have such compact representation. It is thus hard to explicitly the out of sample embedding result.
- See [1] for some researches on this.

Gaussian Kernel

The gaussian kernel: (Let $\tau = \frac{1}{2\sigma^2}$)

$$k(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} = e^{-\tau \|x_i - x_j\|^2}$$

Use Taylor expansion:

$$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$$

Rewrite the kernel:

$$\begin{aligned} k(x_i, x_j) &= e^{-\tau (x_i - x_j)^{\mathrm{T}} (x_i - x_j)} \\ &= e^{-\tau x_i^{\mathrm{T}} x_i} \cdot e^{-\tau x_j^{\mathrm{T}} x_j} \cdot e^{2\tau x_i^{\mathrm{T}} x_j} \end{aligned}$$

Focus on the last part:

$$e^{2\tau x_i^{\mathrm{T}} x_j} = \sum_{k=0}^{\infty} \frac{1}{k!}(2\tau x_i^{\mathrm{T}} x_j)^k$$

It's hard to write out the form when $x_i \in \mathbb{R}^n, n > 1$. We demo the case when $n = 1$. x_i and x_j are now single variable:

$$\begin{aligned} e^{2\tau x_i^{\mathrm{T}} x_j} &= \sum_{k=0}^{\infty} \frac{1}{k!}(2\tau)^k x_i^k x_j^k \\ &= \sum_{k=0}^{\infty} c(k) \cdot x_i^k x_j^k \end{aligned}$$

The feature vector is: (infinite dimension)

$$\phi(x) = e^{-\tau x^2}\left[\sqrt{c(0)}, \sqrt{c(1)}x, \sqrt{c(2)}x^2, ..., \sqrt{c(k)}x^k, ...\right]$$

Verify that:

$$k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

This shows that Gaussian kernel implicitly map 1-D data to an infinite dimensional feature space.