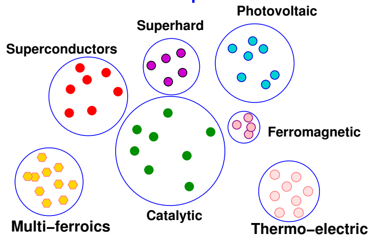
APPLICATIONS OF GRAPH LAPLACEANS: CLUSTERING, EMBEDDING

- *The clustering problem; Basic method: K-means*
- *Similarity graphs; kNN graphs*
- *Measures of separation: edge cuts, normalized cuts, etc*
- *Application: Segmentation*
- *Graph embeddings; Laplacean Eigenmaps*
- *Locally Linear Embeddings (LLE)*
- *Explicit mappings; PCA, LPP, ONPP,..*
- *Building a knn graph*

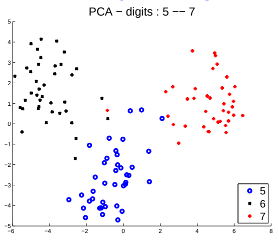
Clustering

➤ Problem: we are given n data items: $x_1, x_2, \cdots, x_n$. Would like to *'cluster'* them, i.e., group them so that each group or cluster contains items that are similar in some sense.

➤ Example: materials



➤ Example: Digits



➤ Each group is a 'cluster' or a 'class'

➤ 'Unsupervised learning'

What is 'Unsupervised Learning'?

Ans: Class of methods that do not exploit labeled data

➤ Example of digits: perform a 2-D projection

➤ Images of same digit tend to cluster (more or less)

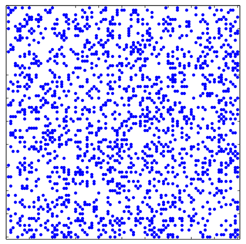
➤ Such 2-D representations are popular for visualization

➤ Can also try to find natural clusters in data, e.g., in materials

➤ Basic clusterning technique: K-means

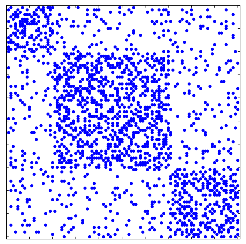
Example: Community Detection

➤ Communities modeled by an 'affinity' graph [e.g., 'user A sends frequent e-mails to user B'] . [data: www-personal.umich.edu/~mejn/netdata/]



← Original Adj. matrix

Goal: Find ordering so blocks are as dense as possible →



➤ Use 'blocking' techniques for sparse matrices

➤ Advantage of this viewpoint: need not know # of clusters.

Example of application Data set from :

http://www-personal.umich.edu/~mejn/netdata/

➤ Network connecting bloggers of different political orientations [2004 US presidentual election]

➤ 'Communities': liberal vs. conservative

➤ Graph: $1,490$ vertices (blogs) : first 758: liberal, rest: conservative.

➤ Edge: $i \to j$: a citation between blogs i and j

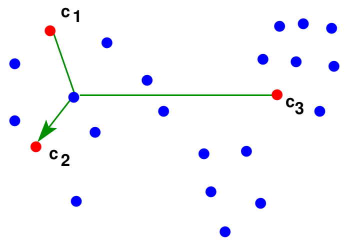
➤ Blocking algorithm (Density theshold=0.4): subgraphs [note: density = $|E|/|V|^2$.]

➤ Smaller subgraph: conservative blogs, larger one: liberals

A basic method: K-means

➤ A basic algorithm that uses Euclidean distance

1. Select p initial centers: $c_1, c_2, ..., c_p$ for classes $1, 2, \cdots, p$
2. For each x_i do: determine *class* of x_i as $\text{argmin}_k \|x_i - c_k\|$
3. Redefine each c_k to be the centroid of class k
4. Repeat until convergence



➤ Simple algorithm

➤ Works well (gives good results) but can be slow

➤ Performance depends on initialization

Methods based on similarity graphs

➤ Class of Methods that perform clustering by exploiting a graph that describes the similarities between any two items in the data.

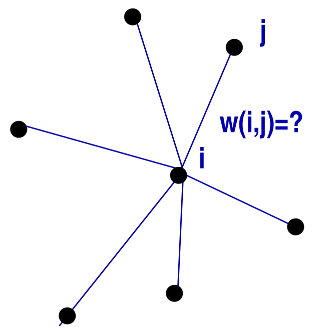
➤ Need to:

1. decide what nodes are in the neighborhood of a given node
2. quantify their similarities - by assigning a weight to any pair of nodes.

Example: For text data: Can decide that any columns i and j with a cosine greater than 0.95 are 'similar' and assign that cosine value to w_{ij}

First task: build a 'similarity' graph

➤ Goal: to build a similarity graph, i.e., a graph that captures similarity between any two items



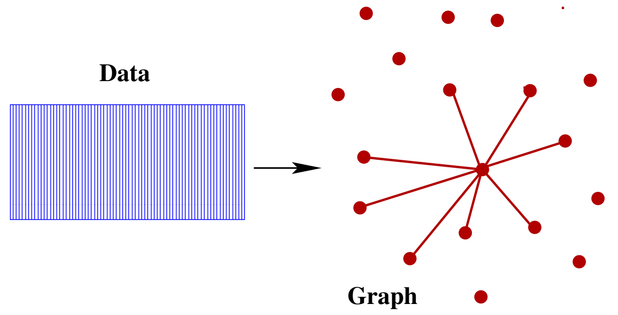
➤ Two methods: K-nearest Neighbor graphs or use Gaussian ('heat') kernel

K-nearest neighbor graphs

- Given: a set of n data points $X = \{x_1, \ldots, x_n\} \to$ vertices
- Given: a proximity measure between two data points x_i and x_j – as measured by a quantity $dist(x_i, x_j)$
- Want: For each point x_i a list of the 'nearest neighbors' of x_i (edges between x_i and these nodes).
- Note: graph will usually be directed $\to$ need to symmetrize

Nearest neighbor graphs

- For each node, get a few of the nearest neighbors $\to$ Graph



- Problem: How to build a nearest-neighbor graph from given data
- We will revisit this later.

Two types of nearest neighbor graph often used:

ϵ-graph:	Edges consist of pairs (x_i, x_j) such that $\rho(x_i, x_j) \le \epsilon$
kNN graph:	Nodes adjacent to x_i are those nodes x_ℓ with the k with smallest distances $\rho(x_i, x_\ell)$.

- ϵ-graph is undirected and is geometrically motivated. Issues: 1) may result in disconnected components 2) what ϵ?
- kNN graphs are directed in general (can be trivially fixed).
- kNN graphs especially useful in practice.

Similarity graphs: Using 'heat-kernels'

Define weight between i and j as:

$$w_{ij} = f_{ij} \times \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{\sigma_X^2}} & \text{if } \|x_i - x_j\| < r \\ 0 & \text{if not} \end{cases}$$

- Note $\|x_i - x_j\|$ could be any measure of distance...
- f_{ij} = optional = some measure of similarity - other than distance
- Only nearby points kept.
- Sparsity depends on parameters

Edge cuts, ratio cuts, normalized cuts, ...

➤ Assume now that we have built a 'similarity graph'

➤ Setting is identical with that of graph partitioning.

➤ Need a Graph Laplacean: $L = D - W$ with $w_{ii} = 0, w_{ij} \geq 0$ and $D = diag(W * ones(n, 1))$ [in matlab notation]

➤ Partition vertex set V in two sets A and B with

$$A \cup B = V, \quad A \cap B = \emptyset$$

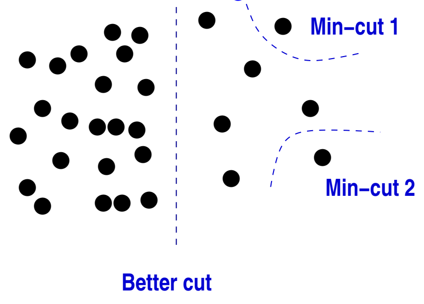
➤ Define

$$\boxed{cut(A, B) = \sum_{u \in A, v \in B} w(u, v)}$$

➤ First (naive) approach: use this measure to partition graph, i.e.,

... Find A and B that minimize $cut(A, B)$.

➤ Issue: Small sets, isolated nodes, big imbalances,



Ratio-cuts

➤ Standard Graph Partitioning approach: Find A, B by solving

$$\boxed{\text{Minimize} \quad cut(A, B), \text{ subject to } |A| = |B|}$$

➤ Condition $|A| = |B|$ not too meaningful in some applications - too restrictive in others.

➤ Minimum Ratio Cut approach. Find A, B by solving:

$$\boxed{\text{Minimize} \quad \frac{cut(A,B)}{|A|.|B|}}$$

➤ Difficult to find solution (original paper [Wei-Cheng '91] proposes several heuristics)

➤ Approximate solution : spectral .

Theorem [Hagen-Kahng, 91] If λ_2 is the 2nd smallest eigenvalue of L, then a lower bound for the cost c of the optimal ratio cut partition, is:

$$c \geq \frac{\lambda_2}{n}.$$

Proof: Consider an optimal partition A, B and let $p = |A|/n, q = |B|/n$. Note that $p + q = 1$. Let x be the vector with coordinates

$$x_i = \begin{cases} q & \text{if } i \in A \\ -p & \text{if } i \in B \end{cases}$$

Note that $x \perp \mathbb{1}$. Also if (i, j) == an edge-cut then $|x_i - x_j| = |q - (-p)| = |q + p| = 1$, otherwise $x_i - x_j = 0$. Therefore:
$x^T L x = \sum_{(i,j) \in E} w_{ij}(x_i - x_j)^2 = w(A, B)$.
In addition: $\|x\|^2 = pq^2 n + qp^2 n = pq(p + q)n = pqn = \frac{|A|.|B|}{n}$.

Therefore, by the Courant-Fischer theorem:

$$\lambda_2 \le \frac{(Lx, x)}{(x, x)} = n \times \frac{w(A, B)}{|A|.|B|} = n \times c.$$

Hence result. ■

➤ Idea is to use eigenvector associated with λ_2 to determine partition, e.g., based on sign of entries. Use the ratio-cut measure to actually determine where to split.

Normalized cuts [Shi-Malik,2000]

➤ Recall notation $w(X, Y) = \sum_{x\in X, y\in Y} w(x, y)$ - then define:

$$\boxed{\text{ncut}(A, B) = \frac{cut(A,B)}{w(A,V)} + \frac{cut(A,B)}{w(B,V)}}$$

➤ Goal is to avoid small sets A, B

✍1 What is $w(A, V)$ in the case when $w_{ij} == 1$?

➤ Let x be an indicator vector: $x_i = \begin{cases} 1 \ if \ i \in A \\ 0 \ if \ i \in B \end{cases}$

➤ Recall that: $x^T L x = \sum_{(i,j)\in E} w_{ij}|x_i - x_j|^2$ (note: each edge counted once)

➤ Therefore:

$$cut(A, B) = \sum_{x_i=1, x_j=0} w_{ij} = x^T L x$$
$$w(A, V) = \sum_{x_i=1} d_i = x^T W \mathbb{1} = x^T D \mathbb{1}$$
$$w(B, V) = \sum_{x_j=0} d_j = (\mathbb{1} - x)^T W \mathbb{1} = (\mathbb{1} - x)^T D \mathbb{1}$$

➤ Goal now: to minimize ncut

$$\min_{A,B} \text{ncut}(A, B) = \min_{x_i \in\{0,1\}} \frac{x^T L x}{x^T D x} + \frac{x^T L x}{(\mathbb{1} - x)^T D x}$$

➤ Let $\boxed{k = \frac{\sum_{x_i>0} d_i}{\sum_i d_i}; \quad b = \frac{k}{1-k}; \quad \text{and:} \quad y = (\mathbb{1} + x) - b(\mathbb{1} - x)}$

➤ Then it can be shown that we need to solve:

$$\boxed{\begin{array}{ll} \min\limits_{y_i \ \{1,-b\}} & \frac{y^T L y}{y^T D y} \\ \text{Subject to} & y^T D \mathbb{1} = 0 \end{array}}$$

➤ + Relax $\to$ need to solve Generalized eigenvalue problem

$$Ly = \lambda D y$$

➤ $y_1 = \mathbb{1}$ is eigenvector associated with eigenvalue $\lambda_1 = 0$

➤ y_2 associated with second eigenvalue solves problem.

A few properties

✍02 Show that

$$ncut(A,B) = \sigma \times \frac{cut(A,B)}{w(A,V) \times w(B,V)}$$

where σ is a constant

✍03 How do ratio-cuts and normalized cuts compare when the graph is d-regular (same degree for each node).

Extension to more than 2 clusters

➤ Just like graph partitioning we can:

1. Apply the method recursively [Repeat clustering on the resulted parts]
2. or compute a few eigenvectors and run K-means clustering on these eigenvectors to get the clustering.

Application: Image segmentation

➤ First task: obtain a graph from pixels.

➤ Common idea: use "Heat kernels"

➤ Let F_j = feature value (e.g., brightness), and Let X_j = spatial position.

Then define

$$w_{ij} = e^{\frac{-\|F_i - F_j\|^2}{\sigma_I^2}} \times \begin{cases} e^{\frac{-\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if} \|X_i - X_j\| < r \\ 0 & \text{else} \end{cases}$$

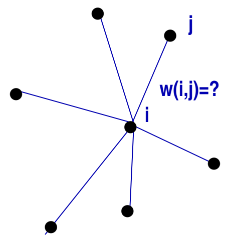
➤ Sparsity depends on parameters

✍04 Run *test_mir* and *test_jump*

Spectral clustering: General approach

1 Given: Collection of data samples $\{x_1, x_2, \cdots, x_n\}$

2 Build a similarity graph between items

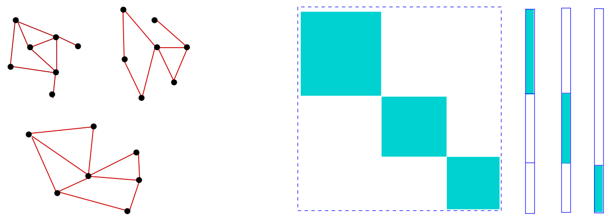


3 Compute (smallest) eigenvector (s) of resulting graph Laplacean

4 Use k-means on eigenvector (s) of Laplacean

➤ For Normalized cuts solve generalized eigen problem.

➤ Recall observation made earlier:

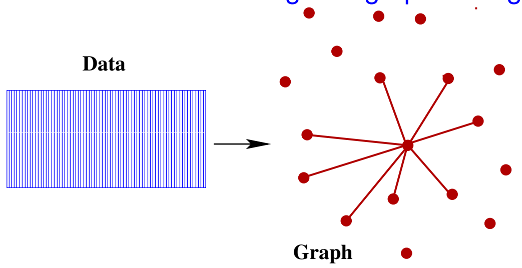


➤ Alg. Multiplicity of eigenvalue zero = # connected components.

KNN GRAPHS

Building a nearest neighbor graph

➤ Question: How to build a nearest-neighbor graph from given data?



➤ Will demonstrate the power of a divide a conquer approach combined with the Lanczos algorithm.

Recall: Two common types of nearest neighbor graphs

ϵ-graph:	Edges consist of pairs (x_i, x_j) such that $\rho(x_i, x_j) \le \epsilon$
kNN graph:	Nodes adjacent to x_i are those nodes x_ℓ with the k with smallest distances $\rho(x_i, x_\ell)$.

➤ ϵ-graph is undirected and is geometrically motivated. Issues: 1) may result in disconnected components 2) what ϵ?

➤ kNN graphs are directed in general (can be trivially fixed).

➤ kNN graphs especially useful in practice.

Divide and conquer KNN: key ingredient

- Key ingredient is *Spectral bisection*
- Let the data matrix $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$
- Each column == a data point.
- Center the data: $\hat{X} = [\hat{x}_1, \ldots, \hat{x}_n] = X - ce^T$
  where c == centroid; $e = ones(d, 1)$ (matlab)

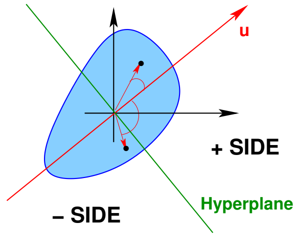
Goal: Split $\hat{X}$ into halves using a hyperplane.

Method: Principal Direction Divisive Partitioning D. Boley, '98.

Idea: Use the (σ, u, v) = largest singular triplet of $\hat{X}$ with: $u^T\hat{X} = \sigma v^T$.

- Hyperplane is defined as $\langle u, x\rangle = 0$, i.e., it splits the set of data points into two subsets:

$$X_+ = \{x_i \mid u^T\hat{x}_i \geq 0\} \quad \text{and} \quad X_- = \{x_i \mid u^T\hat{x}_i < 0\}.$$



- Note that $u^T\hat{x}_i = u^T\hat{X}e_i = \sigma v^T e_i \rightarrow$

$$X_+ = \{x_i \mid v_i \geq 0\} \quad \text{and} \quad X_- = \{x_i \mid v_i < 0\},$$

where v_i is the i-th entry of v.

- In practice: replace above criterion by

$$\boxed{X_+ = \{x_i \mid v_i \geq \text{med}(v)\} \ \& \ X_- = \{x_i \mid v_i < \text{med}(v)\}}$$

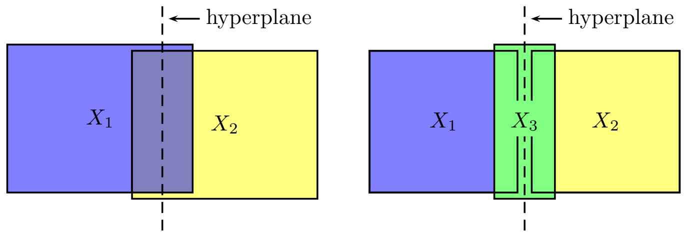
where *med*(v) == *median of the entries of* v.

- For largest singular triplet (σ, u, v) of $\hat{X}$: use Golub-Kahan-Lanczos algorithm or Lanczos applied to $\hat{X}\hat{X}^T$ or $\hat{X}^T\hat{X}$
- Cost (assuming s Lanczos steps) : $O(n \times d \times s)$; Usually: d very small

Two divide and conquer algorithms

Overlap method: divide current set into two overlapping subsets X_1, X_2

Glue method: divide current set into two disjoint subsets X_1, X_2 plus a third set X_3 called gluing set.



- Exploit recursivity

The Overlap Method

➤ Divide current set X into two overlapping subsets:

$$X_1 = \{x_i \mid v_i \geq -h_\alpha(S_v)\} \quad \text{and} \quad X_2 = \{x_i \mid v_i < h_\alpha(S_v)\},$$

- where $S_v = \{|v_i| \mid i = 1, 2, \ldots, n\}$.
- and $h_\alpha(\cdot)$ is a function that returns an element larger than $(100\alpha)\%$ of those in S_v.

➤ Rationale: to ensure that the two subsets overlap $(100\alpha)\%$ of the data, i.e.,

$$|X_1 \cap X_2| = \lceil \alpha |X| \rceil .$$

The Glue Method

Divide the set X into two disjoint subsets X_1 and X_2 with a gluing subset X_3:

$$X_1 \cup X_2 = X, \quad X_1 \cap X_2 = \emptyset, \quad X_1 \cap X_3 \neq \emptyset, \quad X_2 \cap X_3 \neq \emptyset.$$

Criterion used for splitting:

$$X_1 = \{x_i \mid v_i \geq 0\}, \quad X_2 = \{x_i \mid v_i < 0\},$$
$$X_3 = \{x_i \mid -h_\alpha(S_v) \leq v_i < h_\alpha(S_v)\}.$$

Note: gluing subset X_3 here is just the intersection of the sets X_1, X_2 of the overlap method.

Theorem The time complexity for the overlap method is

$$T_{\rm o}(n) = \Theta(dn^{t_{\rm o}}), \qquad \text{where:} \qquad t_{\rm o} = \log_{2/(1+\alpha)} 2 = \frac{1}{1 - \log_2(1+\alpha)}.$$

Theorem The time complexity for the glue method is

$$T_{\rm g}(n) = \Theta(dn^{t_{\rm g}}/\alpha), \quad \text{where} \quad t_{\rm g} \equiv \text{sol. to the equ.:} \quad \frac{2}{2^t} + \alpha^t = 1.$$

Example: When $\alpha = 0.1$, then $t_{\rm o} = 1.16$ while $t_{\rm g} = 1.12$.

Reference:

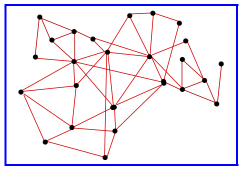
Jie Chen, Haw-Ren Fang and YS, "Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection" JMLR, vol. 10, pp. 1989-2012 (2009).

GRAPH EMBEDDINGS

Graph embeddings

- We have seen how to build a graph to represent data
- *Graph embedding* does the opposite: maps a graph to data

Given: a graph that models some data (e.g., a kNN graph)

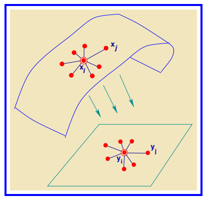


$\longrightarrow$ Data: $Y = [y_1, y_2, \cdots, y_n]$ in $\mathbb{R}^d$

- Trivial use: visualize a graph ($d = 2$)
- Wish: mapping should preserve *similarities* in graph.

Vertex embedding: map every vertex x_i to a vector $y_i \in \mathbb{R}^d$

- Many applications [clustering, finding missing link, semi-supervised learning, community detection, ...]



- Graph captures similarities, closeness, ..., in data

Objective: Build a mapping of each vertex i to a data point $y_i \in \mathbb{R}^d$

- Many methods do this
- Eigenmaps and LLE are two of the best known

- Eigenmaps uses the *graph Laplacean*
- Recall: Graph Laplacean is a matrix defined by :

$$\boxed{L = D - W}$$

$$\begin{cases} w_{ij} \geq 0 & \text{if } j \in Adj(i) \\ w_{ij} = 0 & \text{else} \end{cases} \qquad D = \text{diag}\left[d_{ii} = \sum_{j \neq i} w_{ij} \right]$$

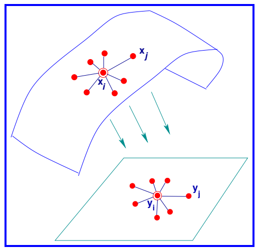
with $Adj(i)$ = neighborhood of i (excludes i)

- Remember that vertex i represents data item x_i. We will use i or x_i to refer to the vertex.
- We will find the y_i's by solving an optimization problem.

The Laplacean eigenmaps approach

Laplacean Eigenmaps [Belkin-Niyogi '01] *minimizes*

$$\mathcal{F}(Y) = \sum_{i,j=1}^{n} w_{ij} \|y_i - y_j\|^2 \quad \text{subject to} \quad Y D Y^\top = I$$



Motivation: if $\|x_i - x_j\|$ is small (orig. data), we want $\|y_i - y_j\|$ to be also small (low-Dim. data)

- Original data used indirectly through its graph
- Objective function can be translated to a trace (see Property 3 in Lecture notes 9) and will yield a sparse eigenvalue problem

➤ Problem translates to:

$$\min_{\begin{cases} Y \in \mathbb{R}^{d\times n} \\ Y D Y^\top = I \end{cases}} \text{Tr}\left[Y(D-W)Y^\top\right] .$$

➤ Solution (sort eigenvalues increasingly):

$$(D-W)u_i = \lambda_i D u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d$$

➤ An $n \times n$ sparse eigenvalue problem [In 'sample' space]

➤ Note: can assume $D = I$. Amounts to rescaling data. Problem becomes

$$\boxed{(I-W)u_i = \lambda_i u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d}$$

Locally Linear Embedding (Roweis-Saul-00)

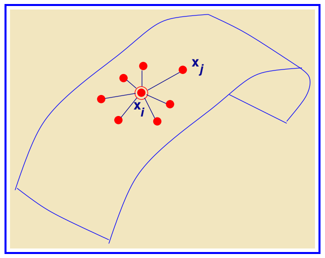
➤ LLE is very similar to Eigenmaps. Main differences:

1) Graph Laplacean matrix is replaced by an 'affinity' graph

2) Objective function is changed: want to preserve graph

1. Graph: Each x_i is written as a convex combination of its k nearest neighbors:
$x_i \approx \Sigma w_{ij} x_j, \quad \sum_{j \in N_i} w_{ij} = 1$
➤ Optimal weights computed ('local calculation') by minimizing

$$\|x_i - \Sigma w_{ij} x_j\| \quad \text{for} \quad i = 1, \cdots, n$$



2. Mapping:

The y_i's should obey the same 'affinity' as x_i's $\rightsquigarrow$

Minimize:

$$\sum_i \left\| y_i - \sum_j w_{ij} y_j \right\|^2 \quad \text{subject to:} \quad Y \mathbb{1} = 0, \quad Y Y^\top = I$$

Solution:

$$\boxed{(I - W^\top)(I-W)u_i = \lambda_i u_i; \qquad y_i = u_i^\top \; .}$$

➤ $(I - W^\top)(I - W)$ replaces the graph Laplacean of eigenmaps

Implicit vs explicit mappings

➤ In Eigenmaps and LLE we only determine a set of $y_i's$ in $\mathbb{R}^d$ from the data points $\{x_i\}$.

➤ The mapping $y_i = \phi(x_i), i = 1, \cdots, n$ is implicit

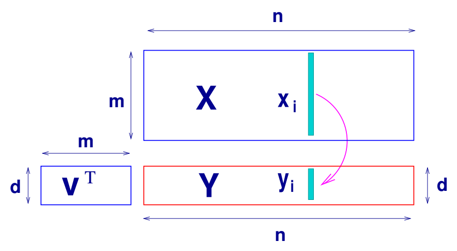
➤ Difficult to compute a y for an x that is not one of the x_i's

➤ Inconvenient for classification. Thus is known as the "The out-of-sample extension" problem

➤ In Explicit (also known as linear) methods: mapping ϕ is known explicitly (and it is linear.)

Locally Preserving Projections (He-Niyogi-03)

➤ LPP is a linear dimensionality reduction technique



➤ Recall the setting:
Want $V \in \mathbb{R}^{m \times d}$; $Y = V^\top X$

➤ Starts with the same neighborhood graph as Eigenmaps: $L \equiv D - W$ = graph 'Laplacean'; with $D \equiv diag(\{\Sigma_i w_{ij}\})$.

➤ Optimization problem is to solve

$$\min_{Y \in \mathbb{R}^{d \times n},\ YDY^\top = I} \quad \Sigma_{i,j} w_{ij} \left\| y_i - y_j \right\|^2, \quad Y = V^\top X.$$

➤ Difference with eigenmaps: Y is an explicit projection of X

➤ Solution (sort eigenvalues increasingly)

$$XLX^\top v_i = \lambda_i XDX^\top v_i \quad y_{i,:} = v_i^\top X$$

➤ Note: essentially same method in [Koren-Carmel'04] called 'weighted PCA' [viewed from the angle of improving PCA]

ONPP (Kokiopoulou and YS '05)

➤ Orthogonal Neighborhood Preserving Projections

➤ A linear (orthogonoal) version of LLE obtained by writing Y in the form $Y = V^\top X$

➤ Same graph as LLE. Objective: preserve the affinity graph (as in LLE) *but* with the constraint $Y = V^\top X$

➤ Problem solved to obtain mapping:

$$\min_V \text{Tr}\left[V^\top X(I - W^\top)(I - W)X^\top V\right]$$
$$\text{s.t. } V^T V = I$$

➤ In LLE replace $V^\top X$ by Y

More recent methods

➤ Quite a bit of recent work - e.g., methods: node2vec, DeepWalk, GraRep, ... See the following papers ... among many others :

[1] *William L. Hamilton, Rex Ying, and Jure Leskovec* *Representation Learning on Graphs: Methods and Applications* arXiv:1709.05584v3

[2] *Shaosheng Cao, Wei Lu, and Qiongkai Xu* *GraRep: Learning Graph Representations with Global Structural Information*, CIKM, ACM Conference on Information and Knowledge Management, 24

[3] *Amr Ahmed, Nino Shervashidze, and Shravan Narayanamurthy*, *Distributed Large-scale Natural Graph Factorization* [Proc. WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil]

Example: Graph factorization

- Line of work in Papers [1] and [3] above + others
- Instead of minimizing $\sum w_{ij} \|y_i - y_j\|_2^2$ as before

... try to minimize $$\boxed{\sum_{ij} |w_{ij} - y_i^T y_j|^2}$$

- In other words solve: $\min_Y \|W - Y^T Y\|_F^2$
- Referred to as *Graph factorization*
- Common in $\boxed{\text{knowledge graphs}}$