Dimensionality Reduction

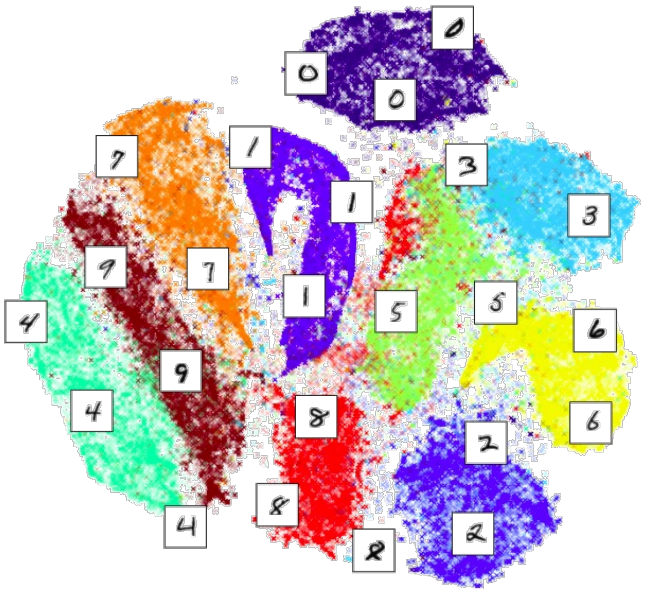
Saerom Park
Department of Industrial Engineering
srompark@unist.ac.kr

Big and High Dimensional Data

- High Dimensions = Lots of Features

- Dimensionality reduction methods aim to extract lower dimensional structure from high dimensional datasets

- Can be used for
  - Visualization
  - Noise removal
  - Removing statistical correlation
  - More efficient uses
  - Better similarity or distance measures



Principal Component Analysis

- Principal Component Analysis
- Kernel PCA

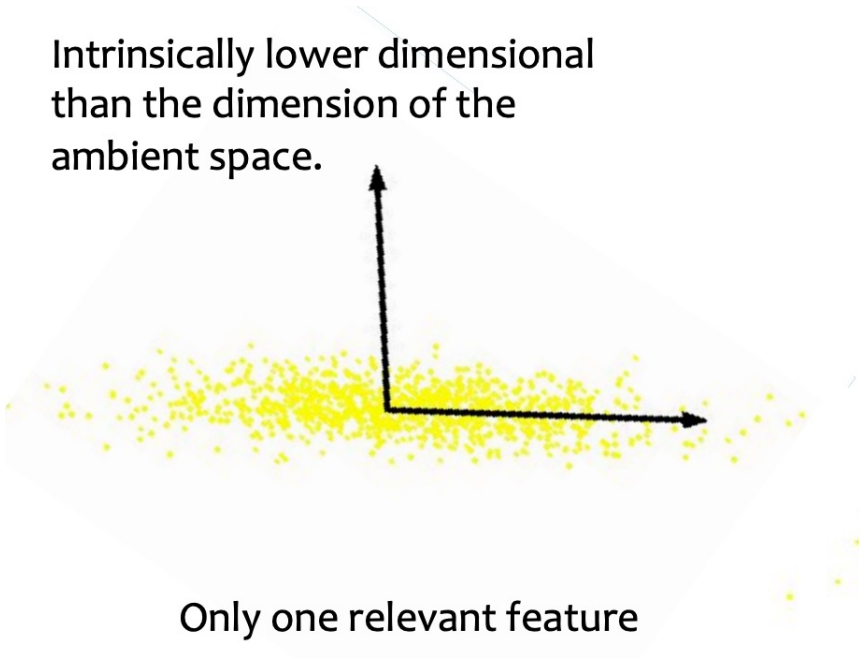
Data Mining
Prof. Saerom Park

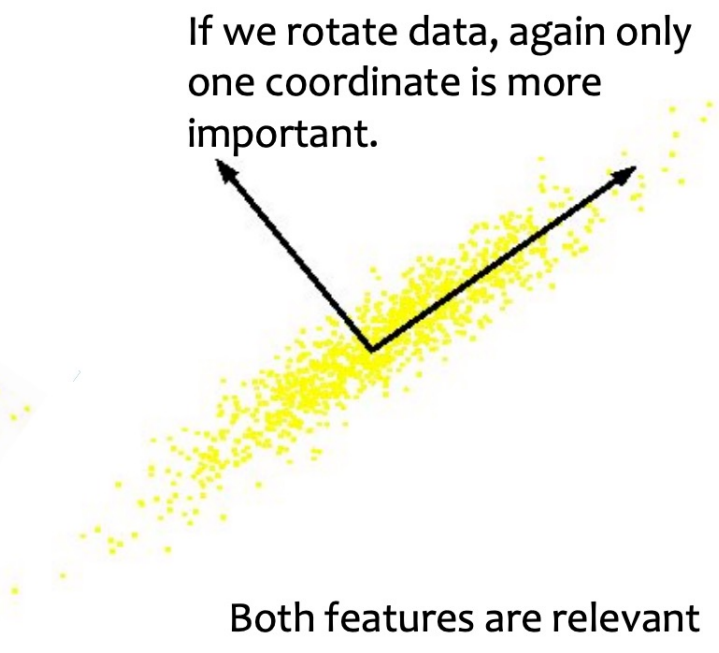
Principal Component Analysis

- PCA reduces a set of features by removing the overlap of information between the original features.
  - Create new features that are linear combinations of the original features.
  - These linear combinations are uncorrelated, and only a few of them contain most of the original information.
- Principal component analysis (PCA) aims to find new features that are statistically uncorrelated.
- PCA extracts variance structure from high dimensional datasets

PCA is an orthogonal projection or transformation of the data into a (possibly lower dimensional) subspace so that the variance of the projected data is maximized.

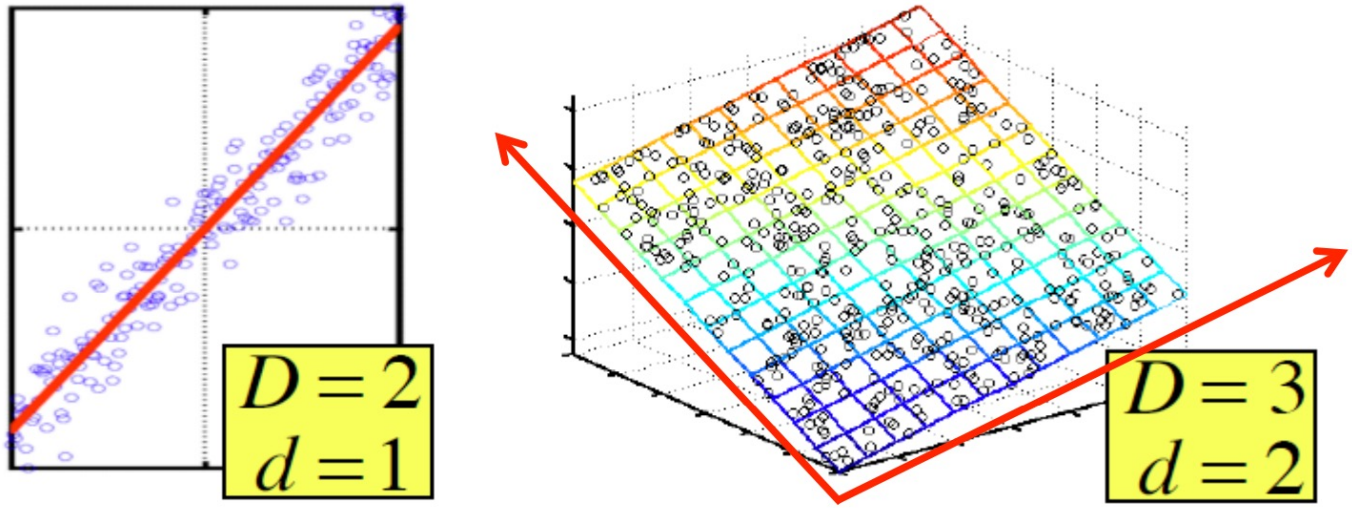
Principal Component Analysis





PCA enables the preservation of most of the variance in a given dataset by identifying the linear subspace

Principal Component Analysis

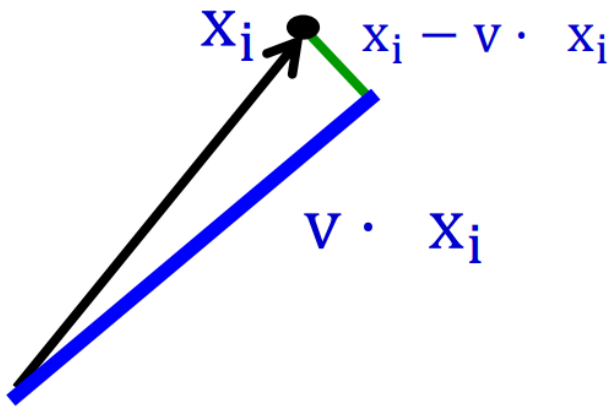


In case where data lies on or **near a low d-dimensional linear subspace**, axes of this subspace are an effective representation of the data.

Identifying the axes is known as ***Principal Components Analysis***, and can be obtained by using classic matrix computation tools (*Eigen or Singular Value Decomposition*).

Principal Components

- ***Principal Components (PC)*** are orthogonal directions that capture most of the variance in the data
  - 1st PC – direction of greatest variability in data
  - 2nd PC – Next orthogonal (uncorrelated) direction of greatest variability
  - ...
  - Remove all variability in the previous directions, then find next direction of greatest variability



PCA takes a *data matrix* $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n]^{\boldsymbol{T}} \in \mathbb{R}^{n \times d}$ and finds the *projection matrix* $\boldsymbol{V} = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_p] \in \mathbb{R}^{d \times p}$ that represents the p-dimensional subspace

Maximum Variance Subspace

- Let $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_p$ denote the p principal components

$$\boldsymbol{v}_i \cdot \boldsymbol{v}_j = 0 \text{ for } i \neq j \text{ and } \boldsymbol{v}_i \cdot \boldsymbol{v}_j = 1 \text{ for } i = j$$

- Assume that data $\boldsymbol{X}$ is centered (mean = 0)
  - if not, we can preprocess it: $\widetilde{\boldsymbol{X}} = \boldsymbol{X} - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\boldsymbol{X}$
- Find vector that maximizes sample variance of projected data

$$\frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{v}^T\boldsymbol{x}_i)^2 = \boldsymbol{v}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v}$$

$$\max_{\boldsymbol{v}} \boldsymbol{v}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v} \quad s.t. \quad \boldsymbol{v}^T\boldsymbol{v} = 1$$

- Lagrangian: $\mathrm{L}(\boldsymbol{v}, \lambda) = \frac{1}{n-1}\boldsymbol{v}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v} - \lambda(\boldsymbol{v}^T\boldsymbol{v} - 1)$

$$\frac{\partial L}{\partial \boldsymbol{v}} = 0 \qquad \boxed{\left(\frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X} - \lambda\boldsymbol{I}\right)\boldsymbol{v} = 0}$$

Eigen Problem

Eigen Decomposition

$$\boldsymbol{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_p] \in \mathbb{R}^{d\times p}, \boldsymbol{Y} = \boldsymbol{XV}$$

$$\mathbf{y_i} = \mathbf{V^T x_i} = \sum_{j=1}^{p} \boldsymbol{v}_j^T \boldsymbol{x}_i$$

- Eigen-decomposition

$$\max_{\boldsymbol{v}_1,\cdots,\boldsymbol{v}_p} trace(\boldsymbol{V}^T\boldsymbol{CV}) . s.t. \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I} \rightarrow max_{\boldsymbol{V}}\, trace\left(\boldsymbol{V}^T\boldsymbol{CV} - \boldsymbol{\Lambda}(\boldsymbol{V}^T\boldsymbol{V} - \boldsymbol{I})\right)$$

The sum of variance after projection

  - $\frac{1}{n-1}\sum_{i=1}^{n} \boldsymbol{y}_i^T\boldsymbol{y}_i = \frac{1}{n-1}\sum_{i=1}^{n}\sum_{j=1}^{p}(\boldsymbol{v}_j^T\boldsymbol{x}_i)^2 = \frac{1}{n-1} trace(\boldsymbol{V}^T\boldsymbol{X}^T\boldsymbol{XV}),\ \boldsymbol{y}_i = (\boldsymbol{v}_1^T\boldsymbol{x}_i, \dots, \boldsymbol{v}_p^T\boldsymbol{x}_i)$
  - $\boldsymbol{cov}(\boldsymbol{Y},\boldsymbol{Y}) = \frac{1}{n-1}\boldsymbol{Y}^T\boldsymbol{Y} = \frac{1}{n-1}\boldsymbol{V}^T\boldsymbol{X}^T\boldsymbol{XV} = \boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Lambda}\mathbf{V^T V} = \boldsymbol{\Lambda}$ Statistically uncorrelated

Eigen Decomposition

- In $\boldsymbol{Cv} = \lambda \boldsymbol{v}$, $\boldsymbol{v}$ (the first PC) is the eigenvector of sample covariance matrix $\mathrm{C} = \frac{1}{\mathrm{n}-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$
  - Sample variance of projection becomes $\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{Cv} = \lambda \boldsymbol{v}^{T}\boldsymbol{v} = \lambda$
- For the eigenvectors and values for the sample variance matrix $\boldsymbol{C}$,
  - Eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots$
  - The 1st PC $\boldsymbol{v_1}$ is the eigenvector of the sample covariance matrix $\boldsymbol{C}$ associated with the largest eigenvalue
  - The 2nd PC $\boldsymbol{v_2}$ is the eigenvector of the sample covariance matrix $\boldsymbol{C}$ associated with the second largest eigenvalue
  - $\sum_{\boldsymbol{j}} var(\boldsymbol{y_j}) = trace\big(\boldsymbol{cov}(\boldsymbol{Y},\boldsymbol{Y})\big) = \lambda_1 + \lambda_2 + \cdots$
    - $\boldsymbol{cov}(\boldsymbol{Y},\boldsymbol{Y}) = \frac{\boldsymbol{1}}{\boldsymbol{n-1}}\boldsymbol{Y}^{\boldsymbol{T}}\boldsymbol{Y} = \frac{\boldsymbol{1}}{\boldsymbol{n-1}}\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{XV} = \boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V} = \boldsymbol{\Lambda}$

Singular Value Decomposition

- For an n × d matrix $\boldsymbol{X}$ of rank r there exists a factorization (Singular Value Decomposition = SVD) as follows:

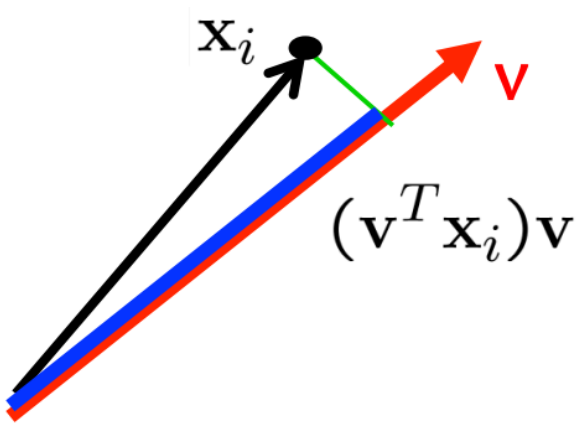
$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}$$

  - The columns of $\boldsymbol{U}$ are orthogonal eigenvectors of $\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}$
  - The columns of $\boldsymbol{V}$ are orthogonal eigenvectors of $\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$
  - Eigenvalues $\mu_1, \dots, \mu_r$ of $\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}$ are the eigenvalues of $\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$ $(rank(\boldsymbol{X}) = r)$
  - Singular values $\sigma_i = \sqrt{\mu_i}$ where $\boldsymbol{\Sigma} = diag(\sigma_1, \dots, \sigma_r)$
  - For eigenvalue λ_i of $\boldsymbol{C} = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X}$, $\lambda_i = \frac{\sigma_i^2}{n-1}$
- Minimum Reconstruction Error
  - Find the subspace that yields minimum MSE reconstruction
    - MSE: $\frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x}_i\|^2$ s.t. $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$
  - Low rank approximation
    - $\min_{\boldsymbol{A}:rank(\boldsymbol{A})=p} \|\boldsymbol{X} - \boldsymbol{A}\|_F = \sigma_{p+1}$

blue² + green² = black²

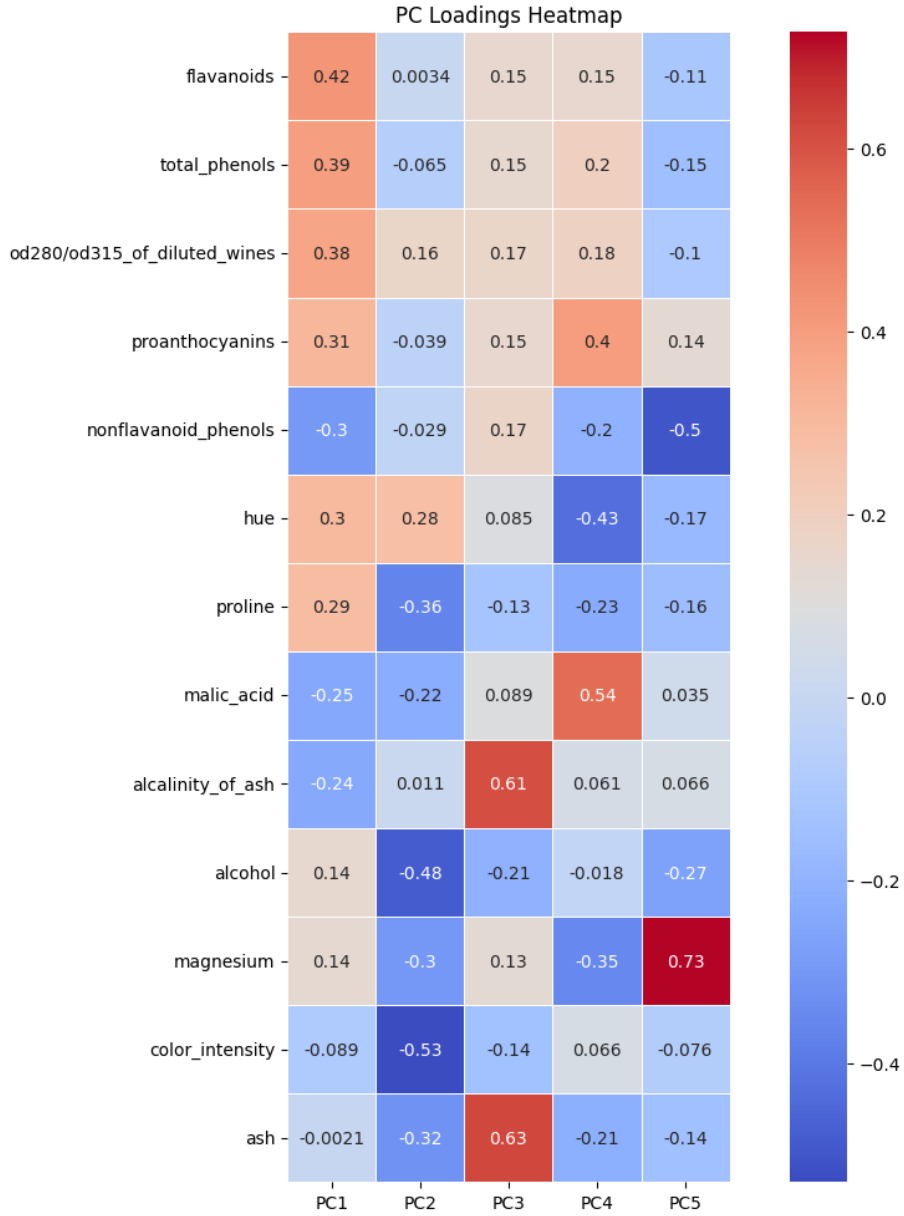
black² is fixed (it's just the data)

So, maximizing blue² is equivalent to minimizing green²



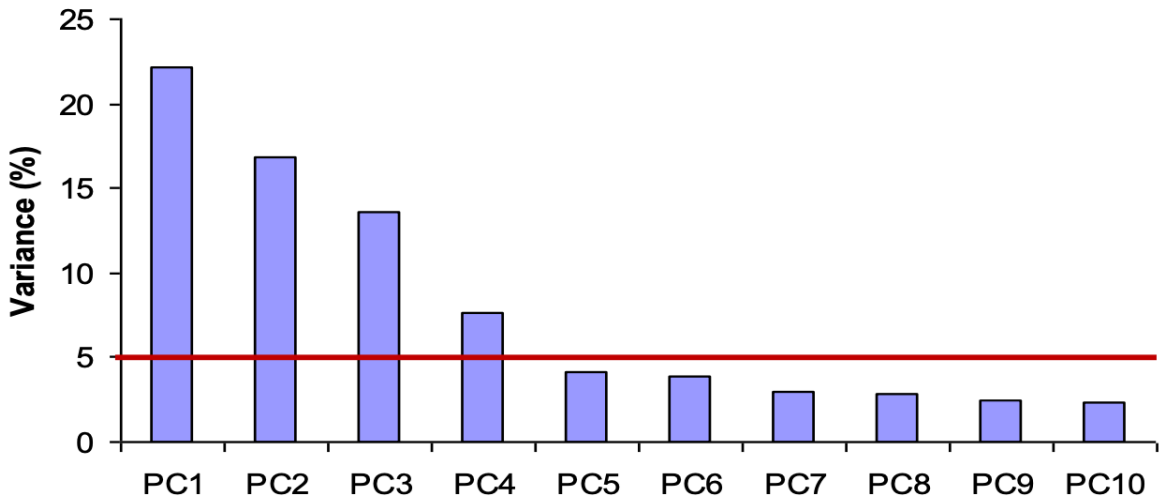
Dimensionality Reduction using PCA

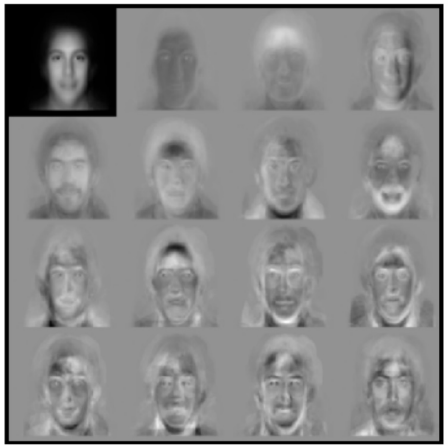
- Original representation
  - Data point: $\boldsymbol{x}_i = \left(x_i^1, \dots, x_i^d\right)$
- Transformed representation
  - Principal components (PCs): a normalized linear combination of the original features $(\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_p})$
  - PC loadings: how much each original variable contributes to that principal component
    - Loading for the i-th PC are given by the column of V $(\boldsymbol{v_i})$
    - The j-th variable on the i-th PC is v_{ij}
  - PC score (Projection): $\boldsymbol{V}^T\boldsymbol{x}_i = \left(\boldsymbol{v}_1^T\boldsymbol{x}_i, \dots, \boldsymbol{v}_p^T\boldsymbol{x}_i\right)$
- Ratio of Explained variance: $\sum_{i=1}^{p} \lambda_i \,/ trace(\boldsymbol{C})$



Dimensionality Reduction using PCA

- In high-dimensional problems, data sometimes lies near a linear subspace, as noise introduces small variability
- Only keep data projections onto principal components with large eigenvalues
  - Can ignore the components of smaller significance
  - Might lose some info, but if eigenvalues are small, do not lose much





Eigenfaces from 7562 images:

top left image is linear combination of rest.

Sirovich & Kirby (1987)
Turk & Pentland (1991)

PCA wrap-up

- Strengths
  - Eigenvector method
  - No tuning of the parameters
  - No local optima

- Weaknesses
  - Limited to second order statistics
  - Limited to linear projections
    - If the structure of data is non-linear, the transformed representation can not capture the original data structure and can lose the important information.

PCA vs LDA

	PCA	LDA
Objective	Maximize variance in the dataset	Maximize separability between known classes
Data Requirements	Feature matrix only	Feature matrix and corresponding class labels
Number of Components	Up to the number of original features (d)`	At most C−1 where C is the number of classes (min(C-1, d))
Usage	Dimensionality reduction, visualization, noise reduction, feature extraction	Classification and dimensionality reduction for classification
Optimization Criteria	Maximizes total variance	Maximizes ratio of between-class variance to within-class variance

Kernel PCA

- Non-linear feature mapping
  - PCA aims to find the subspace that consists of eigenvectors with largest eigenvalues of covariance matrix
- Covariance matrix for non-linear feature mapping
  - Consider a non-linear feature mapping $\boldsymbol{x} \to \Phi(\boldsymbol{x})$

$$\boldsymbol{C}_{\Phi} = \frac{1}{n-1}\sum_{i=1}^{n} \Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T$$

  - Eigenvectors of $\boldsymbol{C}_{\Phi}$ are linear combinations of the feature vectors $\Phi(\boldsymbol{x}_1), \dots, \Phi(\boldsymbol{x}_n)$

$$\lambda \boldsymbol{v} = \boldsymbol{C}_{\Phi}\boldsymbol{v} = \frac{1}{n-1}\sum_{i=1}^{n} \Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T \boldsymbol{v} = \frac{1}{n-1}\sum_{i=1}^{n} \langle \Phi(\boldsymbol{x}_i), \boldsymbol{v} \rangle \, \Phi(\boldsymbol{x}_i) = \lambda \sum_{i=1}^{n} \alpha_i \Phi(\boldsymbol{x}_i)$$

    - $\alpha_i = \frac{1}{\lambda(n-1)} \langle \Phi(\boldsymbol{x}_i), v \rangle$

Kernel PCA

$$\alpha_i = \frac{1}{\lambda(n-1)}\langle \Phi(\boldsymbol{x}_i), \boldsymbol{v}\rangle,\ \boldsymbol{v} = \sum_{\boldsymbol{i}} \alpha_i \Phi(\boldsymbol{x}_{\boldsymbol{i}})$$

- Kernel PCA
  - Suppose that $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)$ and kernel matrix $\boldsymbol{K}$ where $K_{ij} = \left\langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j)\right\rangle$

$$\sum_{i=1}^{n} \langle \Phi(\boldsymbol{x}_k), \Phi(\boldsymbol{x}_i)\rangle \alpha_i = \left\langle \Phi(\boldsymbol{x}_k), \sum_{i=1}^{n} \alpha_i \Phi(\boldsymbol{x}_i)\right\rangle = \langle \Phi(\boldsymbol{x}_k), \boldsymbol{v}\rangle = (n-1)\lambda\alpha_k$$

  - Thus, we need to solve the kernel eigenvalue problem

$$\boldsymbol{K\alpha} = (n-1)\lambda\boldsymbol{\alpha}$$

  - Normalizing Eigenvectors $\langle \boldsymbol{v}, \boldsymbol{v}\rangle = 1$ leads to the condition: $\sum_{i,j} \alpha_i \alpha_j \Phi(x_i)^T \Phi(x_j) = \sum_{i,j} \alpha_i \alpha_j k\left(x_i, x_j\right) = \boldsymbol{\alpha K \alpha} = (n-1)\lambda\langle \boldsymbol{\alpha}, \boldsymbol{\alpha}\rangle = 1$

$$\boldsymbol{\beta} = \begin{bmatrix} --- \boldsymbol{\alpha}_1^T --- \\ \vdots \\ --- \boldsymbol{\alpha}_p^T --- \end{bmatrix} \begin{bmatrix} k(\boldsymbol{x}, \boldsymbol{x}^{(1)}) \\ k(\boldsymbol{x}, \boldsymbol{x}^{(2)}) \\ \vdots \\ k(\boldsymbol{x}, \boldsymbol{x}^{(N)}) \end{bmatrix}$$

- For new test data $\boldsymbol{x}_{\boldsymbol{new}}$

$$\langle v, \Phi(\boldsymbol{x}_{new})\rangle = \left\langle \sum_{i=1}^{n} \alpha_i \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_{new})\right\rangle = \sum_{i=1}^{n} \alpha_i \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_{new})\rangle = \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_{new})$$

- How to obtain the centered feature map?

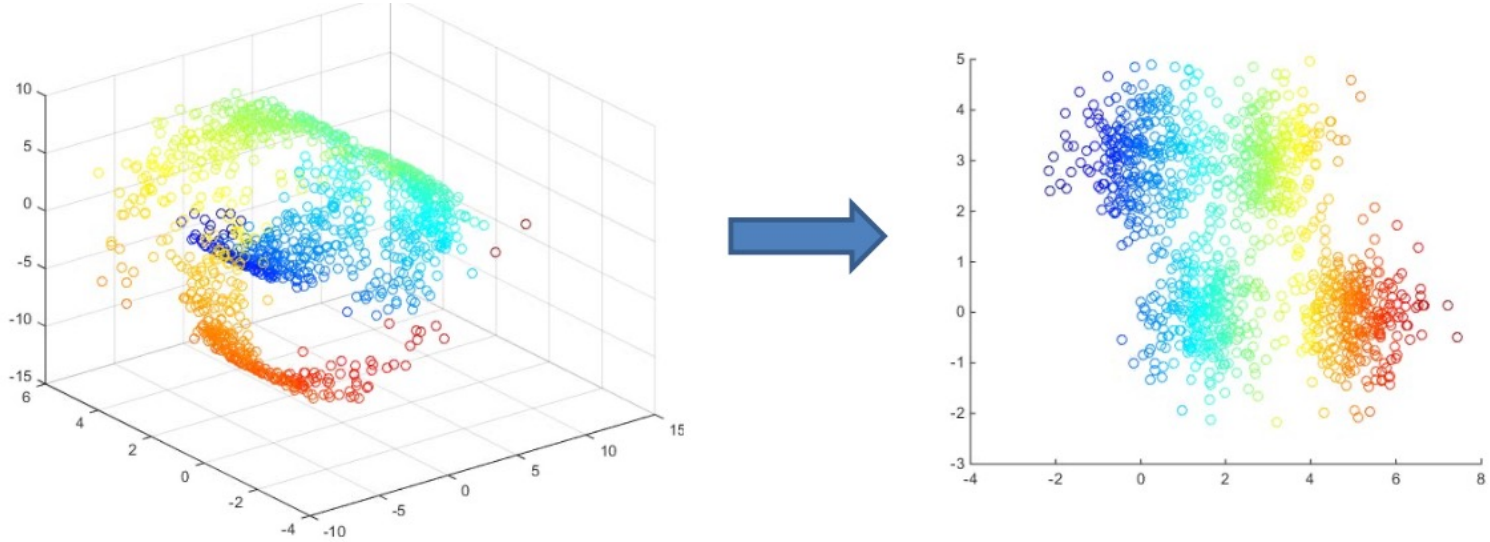
$$\widetilde{\boldsymbol{X}} = \boldsymbol{X} - \frac{1}{n}\boldsymbol{1}_n \boldsymbol{1}_n^T \boldsymbol{X} = \boldsymbol{X} - \widetilde{\boldsymbol{1}}_{\boldsymbol{n}} \boldsymbol{X}$$

Manifold Learning

- Isomap
- Multi-dimensional Scaling
- Laplacian Eigenmap

Data Mining
Prof. Saerom Park

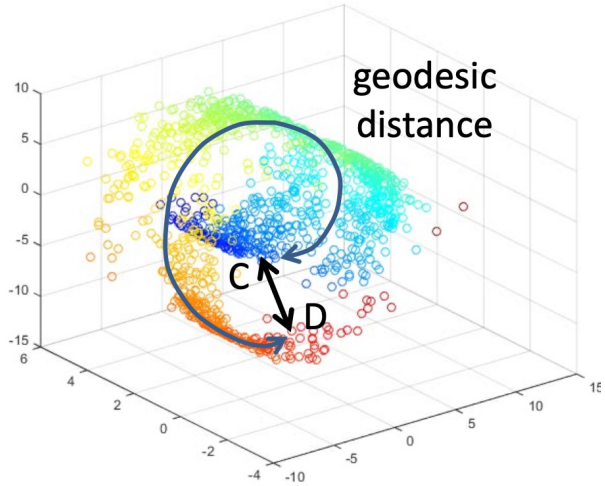
Manifold



- **Key assumption**: High dimensional data actually resides in a *lower dimensional space* that is *locally Euclidean*

- Informally, the manifold is a subset of points in the high-dimensional space that locally looks like a low-dimensional space

Similarity graph models data geometry



- Manifold Structure
  - Global distance ignores the manifold structure of high-dimensional data
  - Geodesic distance: we can consider the distance along the manifold
- Manifold Learning
  - The idea is that (hopefully) once the manifold is “unfolded”, the analysis, such as clustering becomes easy
  - In essence, “unfolding” a manifold is achieved via dimensionality reduction by considering the relationship between local data points
- Graph based on local distances
  - Graph $G = \{V, E\}$, where V is a set of vertices, and $E \subset V \times V$ is a set of edges
  - Graph models pairwise relations between objects (similarity or distance)
  - Weighted graph: each vertex v_{ij} has an associated weight w_{ij} (the strength of the relation between objects)

Similarity graph models data geometry

- We will consider weighted undirected graphs with non-negative weights $w_{ij} \geq 0$. Moreover, we will assume that $\mathrm{w_{ij}} = 0$, if and only if vertices i and j are not connected
- The degree of a vertex $v_i \in \boldsymbol{V}$ is defined as

$$\deg(i) = \sum_{j=1}^{n} w_{ij}$$

- The weighted undirected graph can be represented with a weighted adjacency matrix $\boldsymbol{W}$ that contains weights w_{ij} as its elements

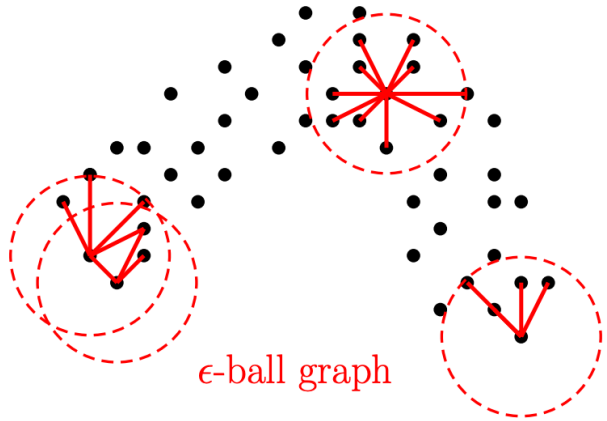
Similarity graph models data geometry

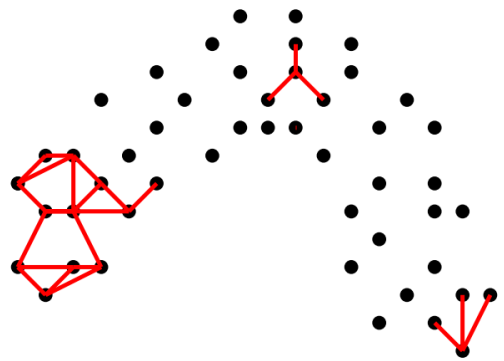
- Geodesic distances can be approximated using a graph in which vertices represent data points
- Let $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ be the Euclidian distance between the points in the original data space

- **Option 1**: define some local radius ϵ

  Connect vertices i and j with an edge if $d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \epsilon$
- **Option 2**: define nearest neighbor threshold k

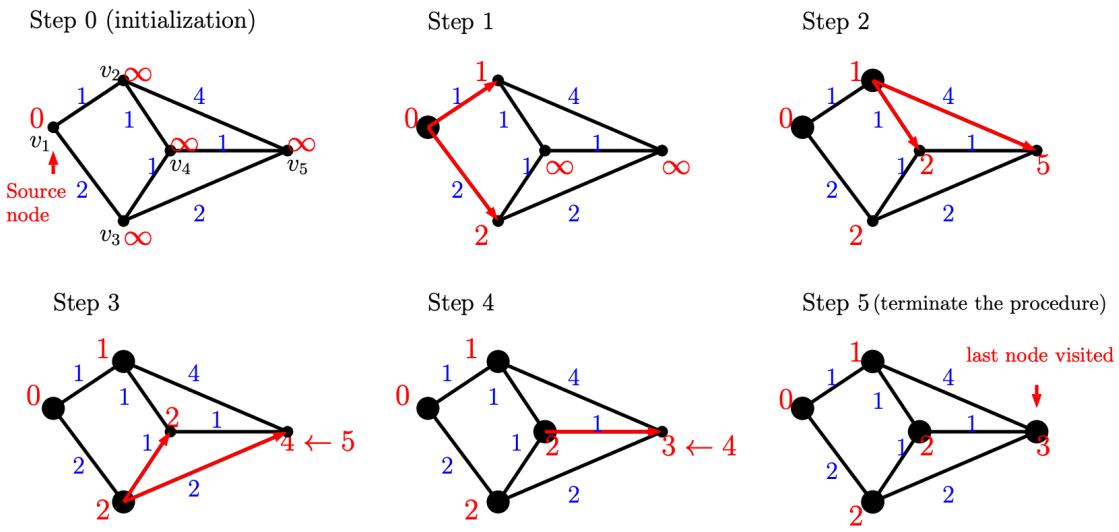
  Connect vertices i and j if i is among the k nearest neighbors of j OR j is among the k nearest neighbors of i

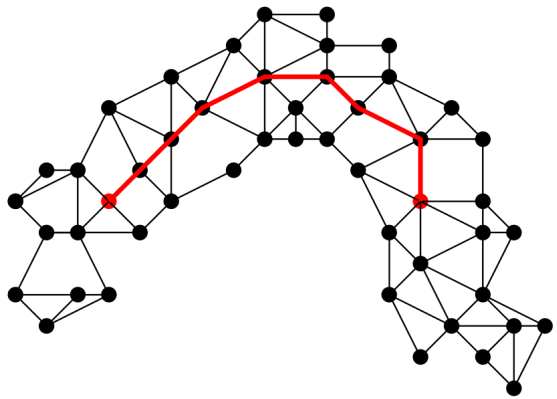




Isomap

- Given n data points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in \mathbb{R}^d$
  1. Compute the k-nearest neighbor graph $\mathrm{G} = (V, E)$
  2. Compute graph distances (geodesic distance) $d_G(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between all points using Dijkstra's algorithm
  3. Embed the points into low dimensions using metric MDS

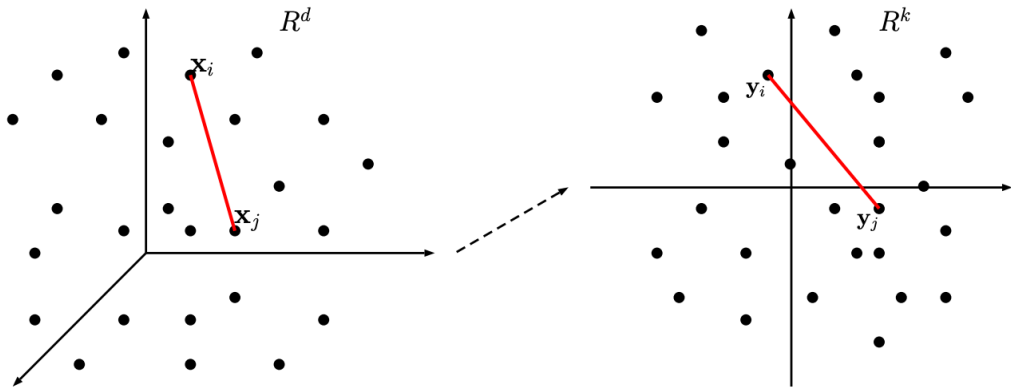




Multi-dimensional Scaling (MDS)

- Multi-dimensional scaling (MDS) aims to find low dimensional representations that preserve the given distances between data points
  - Given n data points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in \mathbb{R}^d$, find low dimensional representations $\boldsymbol{y}_1, \dots, \boldsymbol{y}_n \in \mathbb{R}^p$
- Translational invariance
  - The solutions are not unique because any translation of the new points preserves the pairwise distances
  - Adding a constraint: $\sum \boldsymbol{y}_i = \boldsymbol{0}$
- MDS problem
  - $\left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|_2 \approx d(\boldsymbol{x}_i, \boldsymbol{x}_j) = d_{ij}$ for $\forall i, j$
  - We can use various distance metrics

  i.e., ℓ_p, cosine distance, geodesic distance



Multi-dimensional Scaling (MDS)

Remark: Isomap is based on MDS with geodesic distance defined by Dijkstra algorithm of simiarlity graph

- Relation between distance (proximity) matrix and gram matrix
  - Construct the squared distance matrix: $\boldsymbol{D} = \left[d_{ij}^{\mathbf{2}}\right] \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{n}}$
  - Consider the Gram matrix of embeddings $\boldsymbol{G} = \boldsymbol{Y}\boldsymbol{Y}^T = \left[\boldsymbol{y}_i^T\boldsymbol{y}_j\right]_{i,j=1,\dots,n} \in \mathbb{R}^{n\times n}$ where $\boldsymbol{Y} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_n]^T \in \mathbb{R}^{n\times p}$
  - Then, if we assume $\left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\| = d_{ij}$, we can obtain the following equation:

$$\boldsymbol{y}_i^T\boldsymbol{y}_j = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}d_{i\cdot}^2 - \frac{1}{n}d_{\cdot j}^2 + \frac{1}{n^2}d_{\cdot\cdot}^2\right)$$

$$\boldsymbol{G} = -\frac{1}{2}\left(\boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T\right)\boldsymbol{D}\left(\boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T\right)$$

    - $d_{i\cdot}^2 = \sum_i d_{ij}^2$, $d_{\cdot j}^2 = \sum_j d_{ij}^2$, $d_{\cdot\cdot}^2 = \sum_i\sum_j d_{ij}^2$

How can we get the left relation?

- $d_{ij}^2 = \|\boldsymbol{y}_i\|^2 + \left\|\boldsymbol{y}_j\right\|^2 - 2\boldsymbol{y}_i^T\boldsymbol{y}_j$
- Induce $d_{i\cdot}^2$, $d_{\cdot j}^2$, $d_{\cdot\cdot}^2$

- Finding the embedding $\boldsymbol{Y}$ which has a gram matrix as $\boldsymbol{G}$
  - Construct an eigen problem for $\boldsymbol{G}$ that is a positive semi-definite matrix: $\boldsymbol{G} = \boldsymbol{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathbf{T}} \approx \boldsymbol{V}_{\boldsymbol{p}}\boldsymbol{\Lambda}_{\boldsymbol{p}}\boldsymbol{V}_{\boldsymbol{p}}$
  - Using the top-p eigenvectors, we can construct $\boldsymbol{Y} = \boldsymbol{V}_{\boldsymbol{p}}\boldsymbol{\Lambda}_{\boldsymbol{p}}^{1/2}$

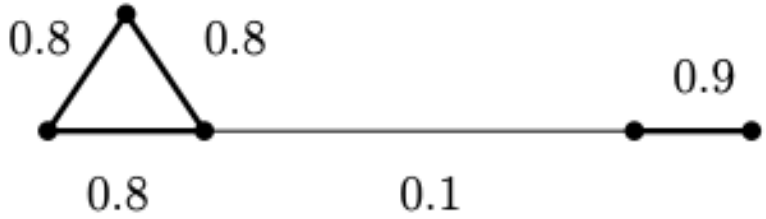
Laplacian Eigenmaps

- The Euclidean distances between nearby points are transformed to similarity scores (to be used as weights) in one of the following ways:
  - **0/1 weights**: $w_{ij} = 1$ if there is an edge between $\boldsymbol{x}_i$, $\boldsymbol{x}_j$
  - **Gaussian weights**: $w_{ij} = \exp\left(-\frac{d(\boldsymbol{x}_i,\boldsymbol{x}_j)^2}{t}\right)$ if there is an edge between x_i , x_j ($t > 0$ is a parameter to be selected by the user)

  If there is **no edge** between two points x_i, x_j , we set $\boldsymbol{w_{ij}} = \mathbf{0}$
- Each of such weighting methods leads to a so-called similarity graph, with weights stored in a weight matrix: $\boldsymbol{W} = \left(w_{ij}\right) \in R^{n\times n}$.
  - Two vertices are *adjacent* if they are connected by an edge (i.e., $w_{ij} > 0$)
  - ***Degree matrix***: $\boldsymbol{D} = diag(d_1, \dots, d_n) \in \mathbb{R}^{n\times n}$ where $d_i = \sum_{j=1}^{n} w_{ij}$

Laplacian Eigenmaps

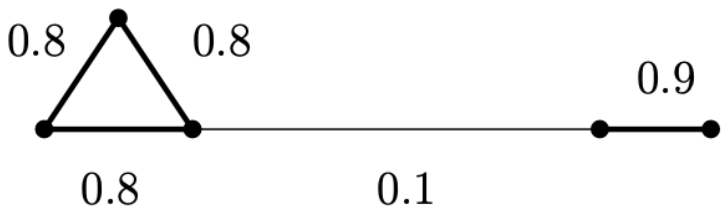
- Example: The following displays a similarity graph on a set of 5 data points (called vertices or nodes), with associated weight matrix W
  - Assuming a weighted similarity graph (constructed on the given data set), we first consider the problem of mapping the graph to a line in a way such that close nodes will still be close on the line. ← Locality-preserving



$$\mathbf{W} = \begin{pmatrix} 0 & .8 & .8 & 0 & 0 \\ .8 & 0 & .8 & 0 & 0 \\ .8 & .8 & 0 & .1 & 0 \\ 0 & 0 & .1 & 0 & .9 \\ 0 & 0 & 0 & .9 & 0 \end{pmatrix}$$

Laplacian Eigenmaps

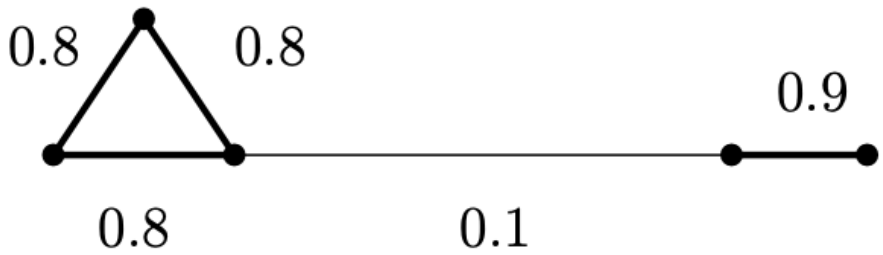
- Laplacian Matrix
  - Given a graph $G = (V, E, W)$ with size $|V| = n$, the graph Laplacian is defined as the following matrix
  - $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \in \mathbb{R}^{n \times n}$, where $\boldsymbol{D} = diag(\boldsymbol{W}\mathbf{1})$
- Graph Laplacian properties
  - $\boldsymbol{L}$ is symmetric
  - All the rows (and columns) sum to 0, i.e., $\boldsymbol{L}\mathbf{1} = \mathbf{0}$. This implies that $\boldsymbol{L}$ has a eigenvalue 0 with eigenvector $\mathbf{1} \in \mathbb{R}^n$.
  - The algebraic (and also geometric) multiplicity of the eigenvalue 0 equals the number of connected components of the graph.



$$\mathbf{L} = \begin{pmatrix} 1.6 & -0.8 & -0.8 & & \\ -0.8 & 1.6 & -0.8 & & \\ -0.8 & -0.8 & 1.7 & -0.1 & \\ & & -0.1 & 1 & -0.9 \\ & & & -0.9 & 0.9 \end{pmatrix}$$

Laplacian Eigenmaps

- Example: For the graph below (which is connected), the eigenvalues of the graph Laplacian are 0 < 0.0788 < 1.8465 < 2.4000 < 2.4747.



$$\mathbf{L} = \begin{pmatrix} 1.6 & -0.8 & -0.8 & & \\ -0.8 & 1.6 & -0.8 & & \\ -0.8 & -0.8 & 1.7 & -0.1 & \\ & & -0.1 & 1 & -0.9 \\ & & & -0.9 & 0.9 \end{pmatrix}$$

Laplacian Eigenmaps

- Let $\boldsymbol{f} = (f_1, \ldots, f_n)^T$ represent the 1D embedding of the nodes. We then formulate the following problem:

$$\min_{f \in \mathbb{R}^n} \frac{1}{2} \sum_{i,j} w_{ij} \left(f_i - f_j\right)^2$$

  - If w_{ij} is large (close to 1, meaning x_i, x_j are originally very close), then f_i, f_j must still be close (otherwise there is a heavy penalty).
  - If w_{ij} is small (close to 0, meaning x_i, x_j are originally very far),then there is much flexibility in putting f_i, f_j on the line.
  - For every vector $f \in \mathbb{R}^n$, we have $\boldsymbol{f^T L f} = \frac{1}{2} \sum_{i=1}^{n} w_{ij} \left(f_i - f_j\right)^2$
    - This implies that $\boldsymbol{L}$ is positive semidefinite and accordingly, its eigenvalues are all nonnegative: $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

Laplacian Eigenmaps

- Proof of the last property

$$\sum_{i,j} w_{ij}\left(f_i - f_j\right)^2 = \sum_{i,j} w_{ij} f_i^2 + \sum_{i,j} w_{ij} f_j^2 - 2\sum_{i,j} w_{ij} f_i f_j = \sum_i d_i f_i^2 + \sum_j d_j f_j^2 - 2\sum_{i,j} w_{ij} f_i f_j$$

$$= 2\boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{D}\boldsymbol{f} - 2\boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{W}\boldsymbol{f} = 2\boldsymbol{f}^{\boldsymbol{T}}(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{f} = 2\boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{L}\boldsymbol{f}$$

Laplacian Eigenmaps

- Original Laplacian Eigenmaps

$$\min_{\boldsymbol{f}\in\mathbb{R}^n} \boldsymbol{f}^T\boldsymbol{L}\boldsymbol{f}\ s.t.\ \boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{D}\boldsymbol{f} = 1$$

  - Lagrangian: $\min_{\mathrm{f}} \boldsymbol{f}^T\boldsymbol{L}\boldsymbol{f} - \boldsymbol{\lambda}\left(\boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{D}\boldsymbol{f} - 1\right)$
- We can obtain the generalized eigen problem: $\boldsymbol{L}\boldsymbol{f} = \lambda\boldsymbol{D}\boldsymbol{f}$
  - The constraint $\boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{D}\boldsymbol{f} = 1$ is for removing the *scaling factor* in $\boldsymbol{f}$
  - We need to remove translation invariance
    - $\mathbf{1}^{\boldsymbol{T}}\ \boldsymbol{f} = \sum_i f_i = 0$ is for removing the *translational invariance*
    - Trivial solution can also be removed (the eigenvector for the zero eigenvalue ($\boldsymbol{L}\mathbf{1} = 0$))
- Let $\boldsymbol{v_1}, \dots, \boldsymbol{v_p} \in \mathbb{R}^n$ denote the first p eigenvectors corresponding to eigen-values $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_p$
  - This determines an embedding for the data $\boldsymbol{x}_i$

$$\boldsymbol{x}_i \to \left(v_{1i}, v_{2i}, \dots, v_{pi}\right) \in \mathbb{R}^p$$

PCA vs. Laplacian Eigenmaps

PCA	Laplacian Eigenmaps
Linear Embedding	Non-linear embedding
Based on the largest eigenvectors of dxd covariance matrix $\mathbf{C} = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X}$	Based on the smallest eigenvectors of nxn Laplacian matrix $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ for the data similarity graph
Eigenvectors give the projection vectors to get embedding of points	Eigenvectors directly give embeddings of data points (the embedding of $\boldsymbol{y}_i$ consist of i-th element of eigenvectors)
$\boldsymbol{y}_i = \left[\boldsymbol{v}_1^T\boldsymbol{x}_i, \boldsymbol{v}_2^T\boldsymbol{x}_i, \dots, \boldsymbol{v}_p^T\boldsymbol{x}_i\right]^T$	$\boldsymbol{y}_i \to \left(v_{1i}, v_{2i}, \dots, v_{pi}\right) \in \mathbb{R}^p$

Appendix

- T-sne*

Data Mining
Prof. Saerom Park

T-Distributed Stochastic Neighbor Embedding*

- Manifold learning
  - *T-Distributed Stochastic Neighbor Embedding* (T-SNE) reduces dimensionality while trying to keep similar data points close and dissimilar data points apart, based solely on how close points are in the original space.
  - The idea behind t-SNE is to find a low-dimensional representation of the data that preserves the distances between points as best as possible.
- T-SNE algorithm
  - T-SNE starts with a random representation for each data point
  - T-SNE aims to
    - points that are close in the original feature space closer (more emphasis on this)
    - points that are far apart in the original feature space farther apart

t-SNE is mostly used for ***visualization***, in particular to visualize clusters of data points in high-dimensional space. It doesn't allow transformations of new data.

T-Distributed Stochastic Neighbor Embedding *

- Stochastic Neighbor Embedding
  - ***Stochastic Neighbor Embedding (SNE)*** starts by converting the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities
  - The conditional probability:

$$p_{j|i} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2/2\sigma_i^2)}$$

  - For nearby datapoints, $p_{j|i}$ is relatively high, whereas for widely separated datapoints, $p_{j|i}$ will be almost infinitesimal
    - Set $p_{i|i} = 0$
  - For embedding $\boldsymbol{y}_i$,

$$q_{j|i} = \frac{\exp\left(-\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2\right)}{\sum_{k\neq i}\exp(-\|\boldsymbol{y}_i - \boldsymbol{y}_k\|^2)}$$

T-Distributed Stochastic Neighbor Embedding *

- Stochastic Neighbor Embedding
  - The cost function C of SNE

$$C = \sum_i KL(P_i || Q_i) = \sum_{i,j} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

  - P_i represents the conditional probability distribution over all other datapoints given datapoint $\boldsymbol{x}_i$, and Q_i represents the conditional probability distribution over all other map points given map point $\boldsymbol{y}_i$
  - The SNE cost function focuses on retaining the local structure of the data in the map for reasonable values of the variance of the Gaussian in the high-dimensional space, σ_i
  - In dense regions, a smaller value of σ_i is usually more appropriate than in sparser regions

$$\frac{\partial C}{\partial \boldsymbol{y}_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(\boldsymbol{y}_i - \boldsymbol{y}_j)$$

It is required to run the optimization several times on a data set to find appropriate values for the parameters σ_i

T-Distributed Stochastic Neighbor Embedding *

- Symmetric SNE
  - Cost function: $C = KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$

$$p_{ij} = \frac{\exp(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2 / 2\sigma^2)}{\sum_{k \neq l} \exp(-\|\boldsymbol{x}_k - \boldsymbol{x}_l\|^2 / 2\sigma^2)}$$

$$q_{ij} = \frac{\exp\left(-\left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|^2\right)}{\sum_{k \neq l} \exp(-\|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)}$$

  - What if there is an outlier?
    - Location of low-dimensional embedding is determined by the relation to outlier!
    - Use $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  - Gradient of symmetric SNE

$$\frac{\partial C}{\partial \boldsymbol{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\boldsymbol{y}_i - \boldsymbol{y}_j)$$

T-Distributed Stochastic Neighbor Embedding *

- Student t-distribution for q_{ij}

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}, \qquad \frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

  - This allows a **moderate distance** in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints.

In the high-dimensional space, we convert distances into probabilities using a **Gaussian distribution**. In the low-dimensional map, we can use a probability distribution that has **much heavier tails** than a Gaussian to convert distances into probabilities.

T-Distributed Stochastic Neighbor Embedding *

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for t=1 **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579-2605

What’s Next?

- Tree-based Models

Data Mining
Prof. Saerom Park