Dimensionality Reduction (contd.)

CS771: Introduction to Machine Learning

Nisheeth

Principal Component Analysis: Recap

- Center the data (subtract the mean $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$ from each data point)
- Compute the $D \times D$ covariance matrix $\mathbf{S}$ using the centered data matrix $\mathbf{X}$ as

$$\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$$ (Assuming $\mathbf{X}$ is arranged as $N \times D$)

- Do an eigendecomposition of the covariance matrix $\mathbf{S}$ (many methods exist)
- Take top $K < D$ leading eigvectors $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K\}$ with eigvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_K\}$
- The K-dimensional projection/embedding of each input is

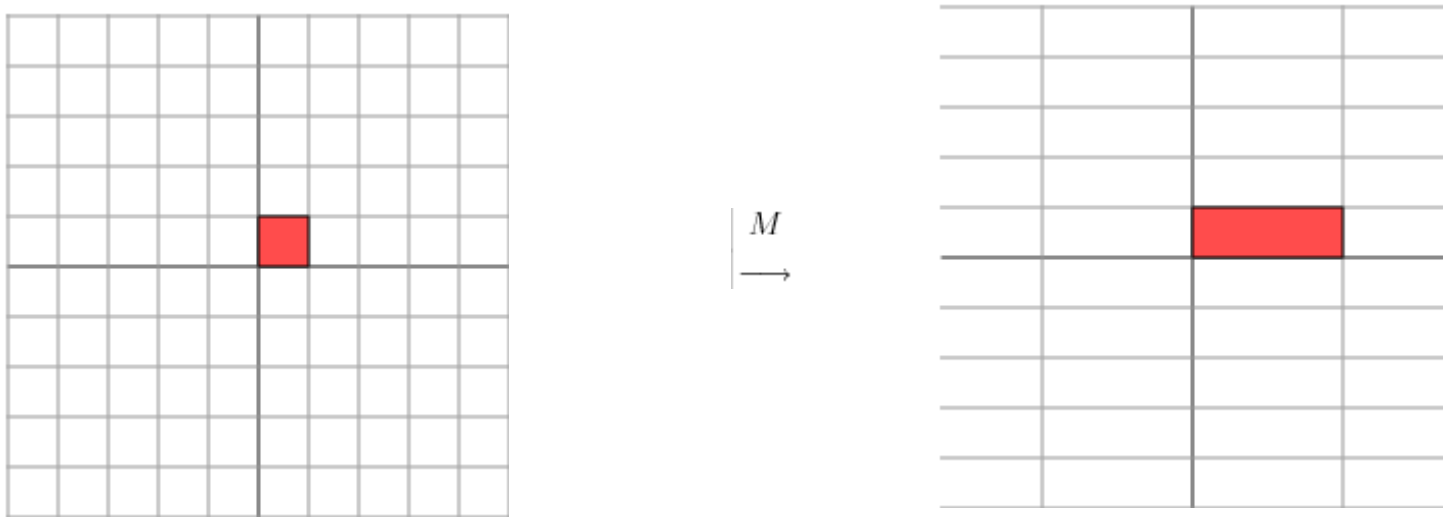
$$\boldsymbol{z}_n \approx \mathbf{W}_K^\top \boldsymbol{x}_n$$

$\mathbf{W}_K = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K]$ is the "projection matrix" of size $D \times K$

Note: Can decide how many eigvecs to use based on how much variance we want to campure (recall that each λ_k gives the variance in the k^{th} direction (and their sum is the total variance)

Eigendecomposition refresher

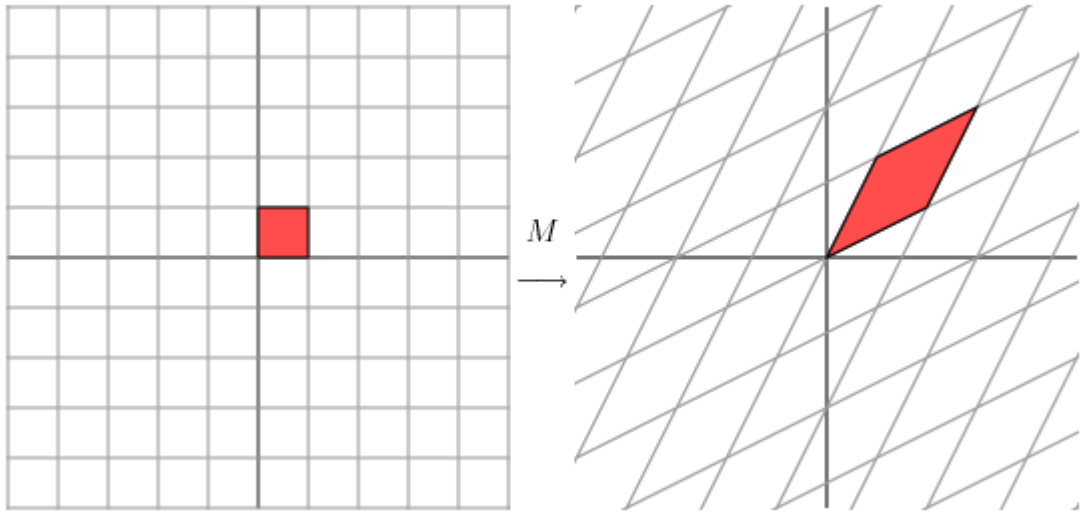
- Recall that matrices are essentially instructions for transforming vectors
- Consider diagonal matrices $M = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$



Visuals for these slides borrowed from this tutorial. Highly recommended.

Symmetric matrices are special

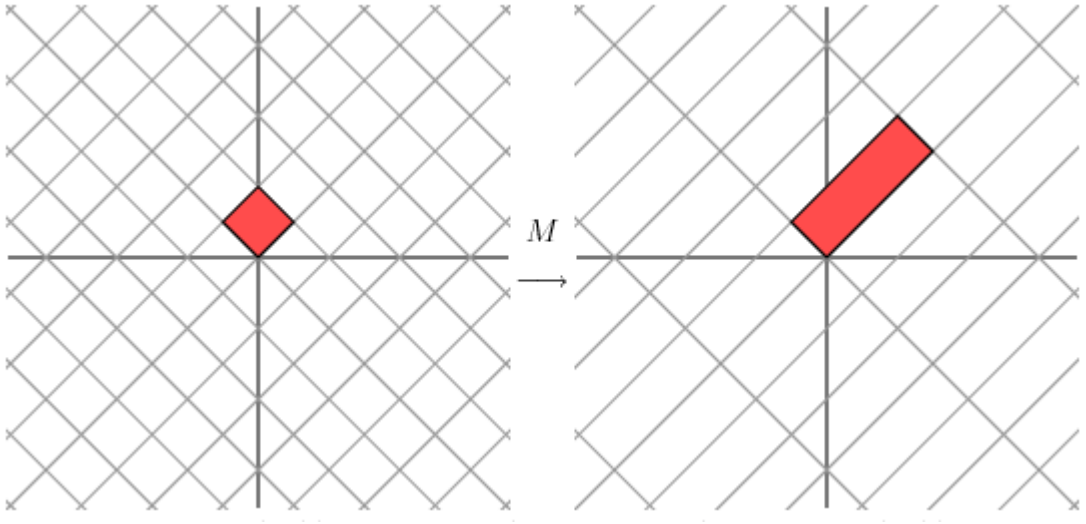
- Consider the symmetric matrix $M = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$
- It has the following effect



- What is it doing?
  - Not clear

Symmetric matrices behave like diagonal matrices

- Consider the same operation on a coordinate system rotated by 45 degrees



- We see that the effect of the symmetric matrix M on this coordinate frame is the same as the effect of a diagonal matrix on the conventional coordinate frame

A simple eigendecomposition

- From 12th class linear algebra

$$det(A - \lambda I) = 0$$
$$(2 - \lambda)(2 - \lambda) - 1 = 0$$
$$\lambda^2 - 4\lambda + 3 = 0$$

$$\lambda = \{3, 1\}$$

- With eigenvectors x1 = {1,1} and x2 = {-1,1}
- What does this mean?

Eigendecomposition to information compression

- The eigendecomposition of M found the vectors we could use to form a coordinate basis
  - In which the matrix operation M on a vector would correspond to a simple scaling operation on the same vector
  - $M\mathbf{x} = \lambda\mathbf{x}$
- Important: each eigenvalue is simply performing a scaling operation in the new coordinate basis. The bigger the eigenvalue, the bigger the transformation
- If we choose to not use some of the eigenvalues, this is equivalent to not using some information in the original matrix
- By selecting to ignore smaller eigenvalues, we compress information about a matrix by looking only at the most important scale transformations that matter
- Pro-tip, for symmetric positive definite matrices, eigendecomposition is the same as singular value decomposition (SVD)

An eigendecomposition application: PCA

- Center the data (subtract the mean $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$ from each data point)
- Compute the $D \times D$ covariance matrix $\mathbf{S}$ using the centered data matrix $\mathbf{X}$ as

$$\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$$ (Assuming $\mathbf{X}$ is arranged as $N \times D$)

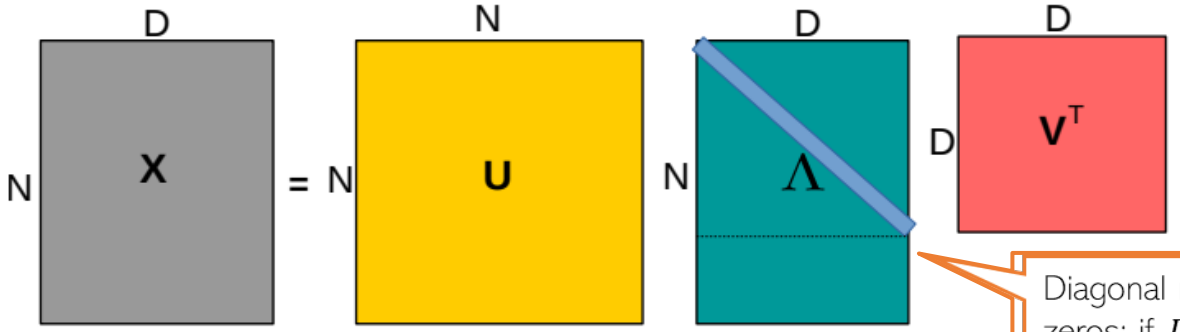
- Do an eigendecomposition of the covariance matrix $\mathbf{S}$ (many methods exist)
- Take top $K < D$ leading eigvectors $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K\}$ with eigvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_K\}$
- The K-dimensional projection/embedding of each input is

$$\boldsymbol{z}_n \approx \mathbf{W}_K^\top \boldsymbol{x}_n$$

$\mathbf{W}_K = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K]$ is the "projection matrix" of size $D \times K$

Singular Value Decomposition (SVD)

- Any matrix $\mathbf{X}$ of size $N \times D$ can be represented as the following decomposition



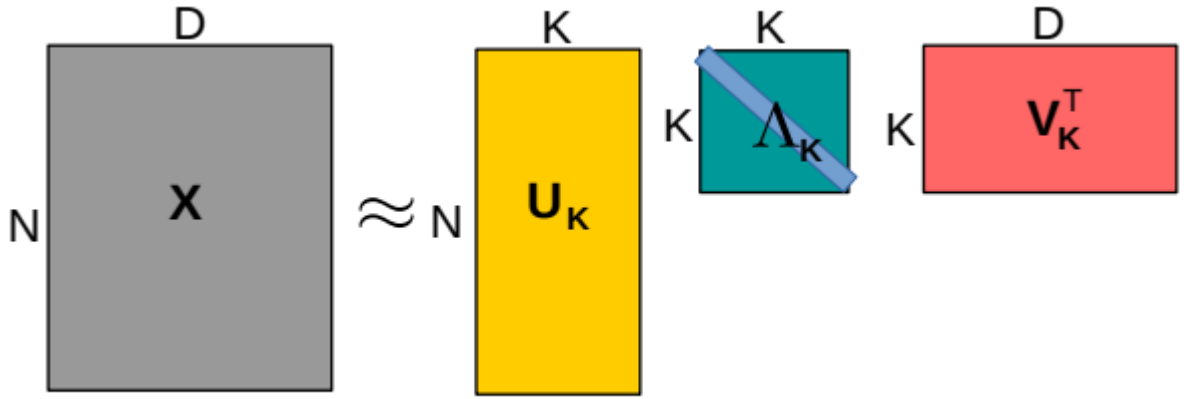
$$\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top = \sum_{k=1}^{\min\{N,D\}} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$$

- $\mathbf{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N]$ is $N \times N$ matrix of left singular vectors, each $\boldsymbol{u}_n \in \mathbb{R}^N$
  - $\mathbf{U}$ is also orthonormal
- $\mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N]$ is $D \times D$ matrix of right singular vectors, each $\boldsymbol{v}_d \in \mathbb{R}^D$
  - $\mathbf{V}$ is also orthonormal
- Λ is $N \times D$ with only $\min(N, D)$ diagonal entries - singular values
- Note: If $\mathbf{X}$ is symmetric then it is known as eigenvalue decomposition ($\mathbf{U} = \mathbf{V}$)

The wikipedia article on this is a star. Highly recommended.

Low-Rank Approximation via SVD

- If we just use the top $K < \min\{N, D\}$ singular values, we get a rank-K SVD



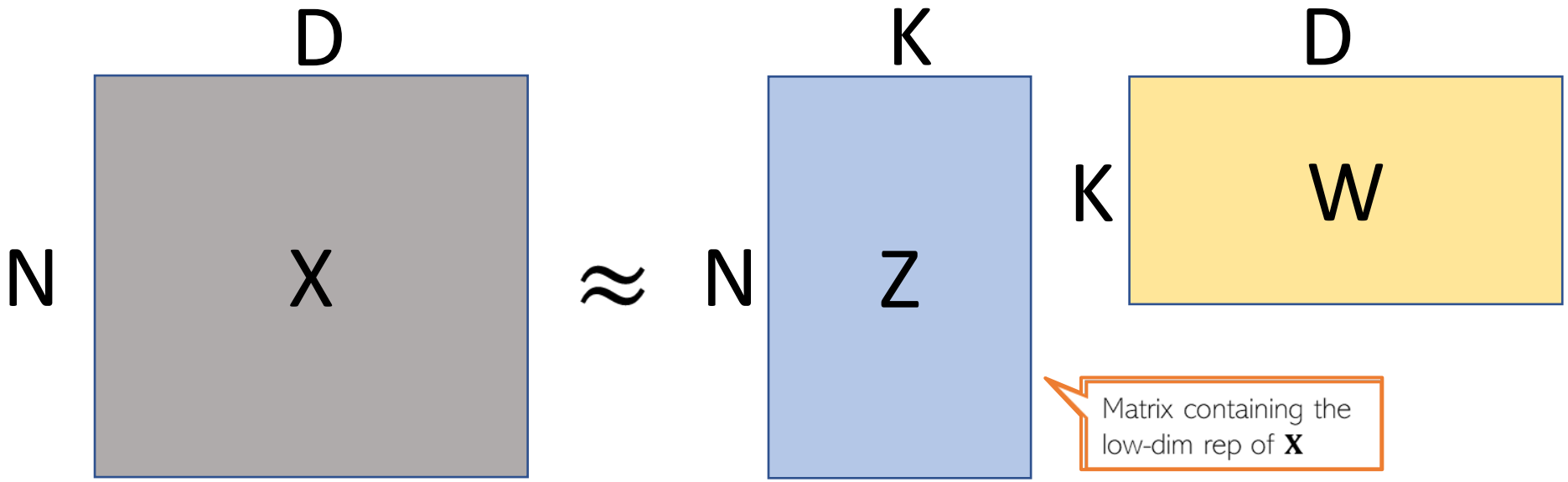
$$\mathbf{X} \approx \hat{\mathbf{X}} = \sum_{k=1}^{K} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top = \mathbf{U}_K \Lambda_K \mathbf{V}_K^\top$$

- Above SVD approx. can be shown to minimize the reconstruction error $\|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|$
  - Fact: SVD gives the best rank-K approximation of a matrix

- PCA is done by doing SVD on the covariance matrix $\boldsymbol{S}$ (left and right singular vectors are the same and become eigenvectors, singular values become eigenvalues)

Dim-Red as Matrix Factorization

- If we don't care about the orthonormality constraints, then dim-red can also be achieved by solving a matrix factorization problem on the data matrix $\mathbf{X}$



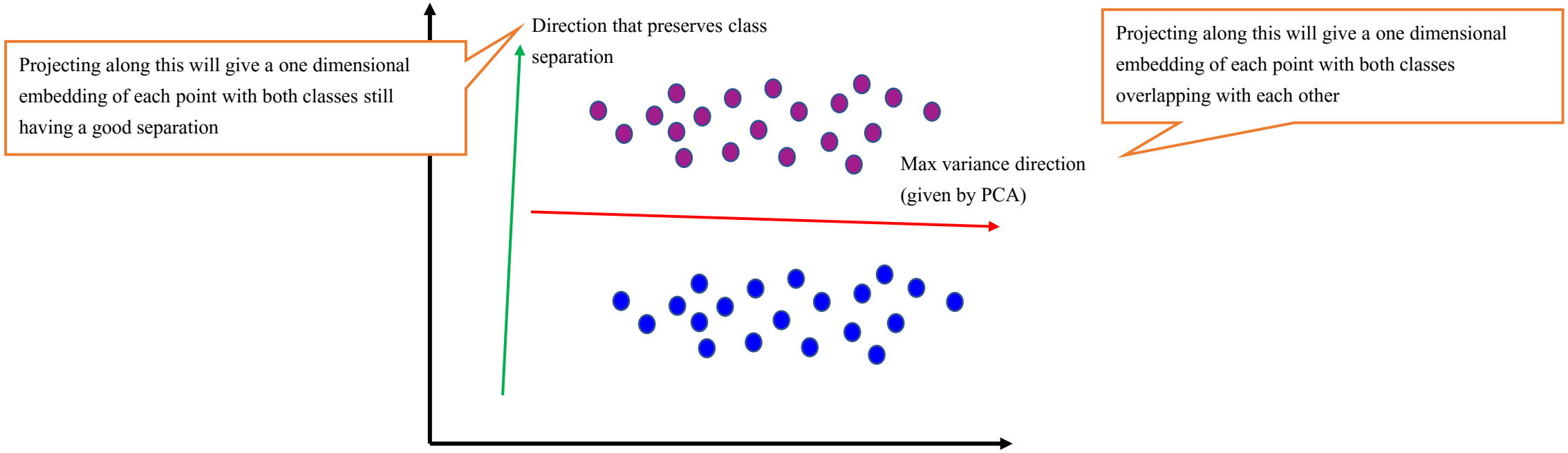
$$\{\hat{\mathbf{Z}}, \hat{\mathbf{W}}\} = \text{argmin}_{\mathbf{Z},\mathbf{W}} \|\mathbf{X} - \mathbf{ZW}\|^2$$

If $K < \min\{D, N\}$, such a factorization gives a low-rank approximation of the data matrix X

- Can solve such problems using ALT-OPT
- Can impose various constraints on $\mathbf{Z}$ and $\mathbf{W}$ (e.g., sparsity, non-negativity, etc)

Supervised Dimensionality Reduction

- Maximum variance directions may not be aligned with class separation directions



- Be careful when using PCA for supervised learning problems
- A better option would be to project such that
  - Points within the same class are close (low intra-class variance)
  - Points from different classes are well separated (the class means are far apart)

Supervised Dimensionality Reduction

- Many techniques. A simple yet popular one is Fisher Discriminant Analysis, also known as Linear Discriminant Analysis (FDA or LDA)

This LDA should not be confused with another very popular ML technique for finding topics in text data (Latent Dirichlet Allocation)

- For simplicity, assume two classes (can be generalized for more than 2 classes too)
- Suppose a projection direction $\boldsymbol{u}$. After projection the means of the two classes are

$$\mu_1 = \frac{1}{N_1} \sum_{n:y_n=1} \boldsymbol{u}^\top \boldsymbol{x}_n, \quad \mu_2 = \frac{1}{N_2} \sum_{n:y_n=2} \boldsymbol{u}^\top \boldsymbol{x}_n$$

Here we considered projection to one dimension but can be generalized to projection to K dim

- Total variance of the points after projection will be $s_1^2 + s_2^2$ where

$$s_1^2 = \frac{1}{N_1} \sum_{n:y_n=1} (\boldsymbol{u}^\top \boldsymbol{x}_n - \mu_1)^2, \quad s_2^2 = \frac{1}{N_2} \sum_{n:y_n=2} (\boldsymbol{u}^\top \boldsymbol{x}_n - \mu_2)^2$$

- Fisher discriminant analysis finds the optimal projection direction by solving

$$\arg\max_{\boldsymbol{u}} \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}$$

The solution to this problem involves solving an eigendecomposition problem that involves within class covariance matrices and between class covariance matrices

Push the means far apart

Make each class tightly packed after projection (small variance)

Dimensionality Reduction given Pairwise Distances between points

Dim. Reduction by Preserving Pairwise Distances

- PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- Often the data is given in form of distances d_{ij} between points $(i, j = 1,2, \ldots, N)$
- Would like to project data such that pairwise distances between points are preserved

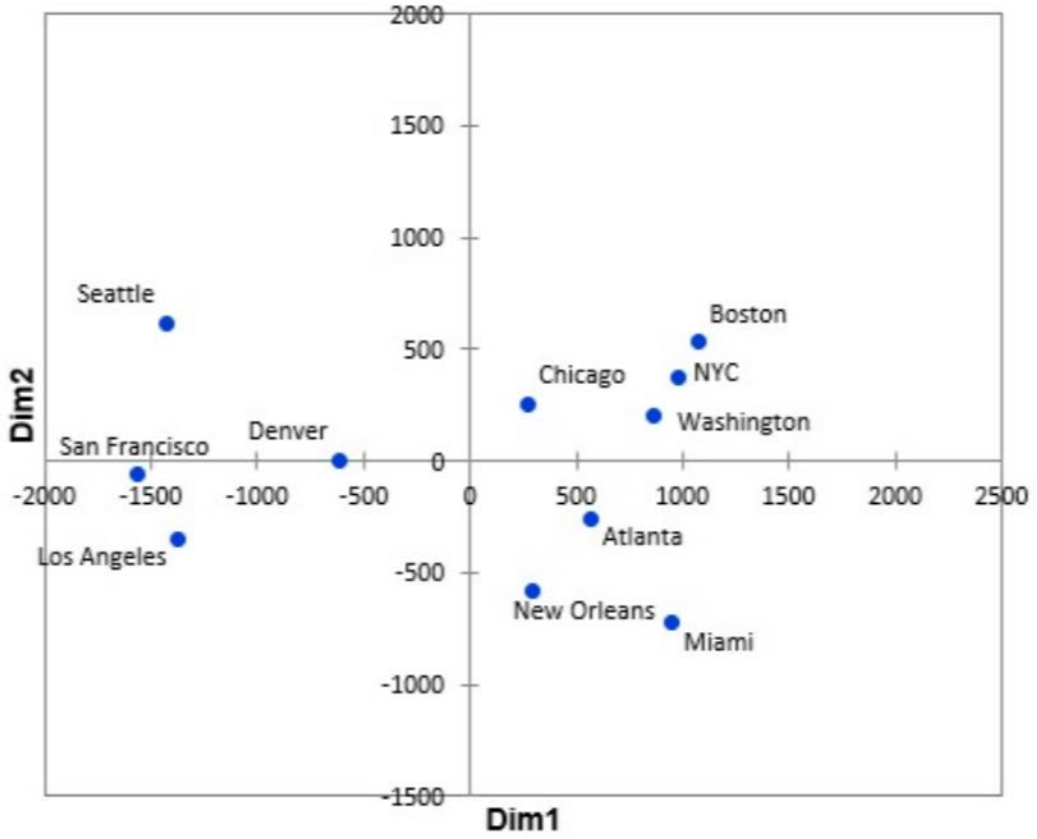
$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{Z}) = \arg\min_{\mathbf{Z}} \sum_{i,j=1}^{N} (d_{ij} - ||\boldsymbol{z}_i - \boldsymbol{z}_j||)^2$$

$\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ denote low-dim embeddings/projections of points i and j

- Basically, if $\boldsymbol{d}_{ij}$ is large (resp. small), would like $\|\boldsymbol{z}_i - \boldsymbol{z}_j\|$ to be large (resp. small)
- Multi-dimensional Scaling (MDS) is one such algorithm
- Note: If $\boldsymbol{d}_{ij}$ is the Euclidean distance, MDS is equivalent to PCA
- The above approach tries to preserve all pairwise distances
  - Can try to preserve pairwise distances only between close-by points (i.e., b/w nearest neighbors). It helps achieve non-linear dim red. Algos like Isomap and locally linear embedding (LLE) do this

MDS: An Example

- Result of applying MDS (with $K = 2$) on pairwise distances between some US cities

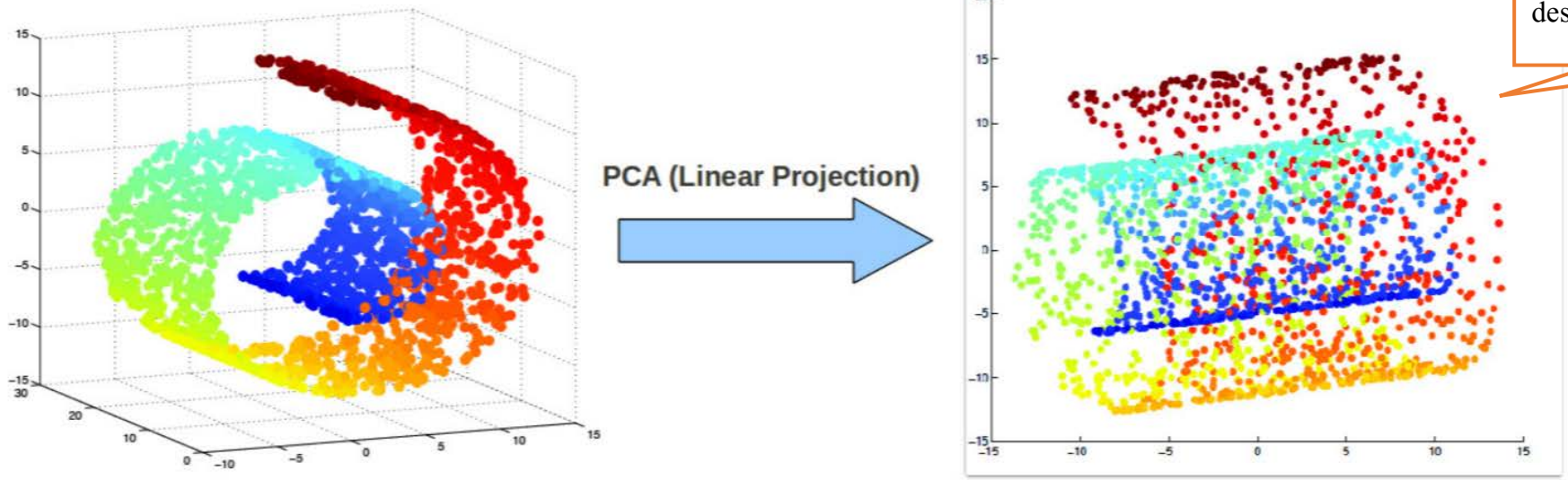


- MDS produces a 2D embedding such that geographically close cities are also close in embedding space

Nonlinear Dimensionality Reduction

Beyond Linear Projections

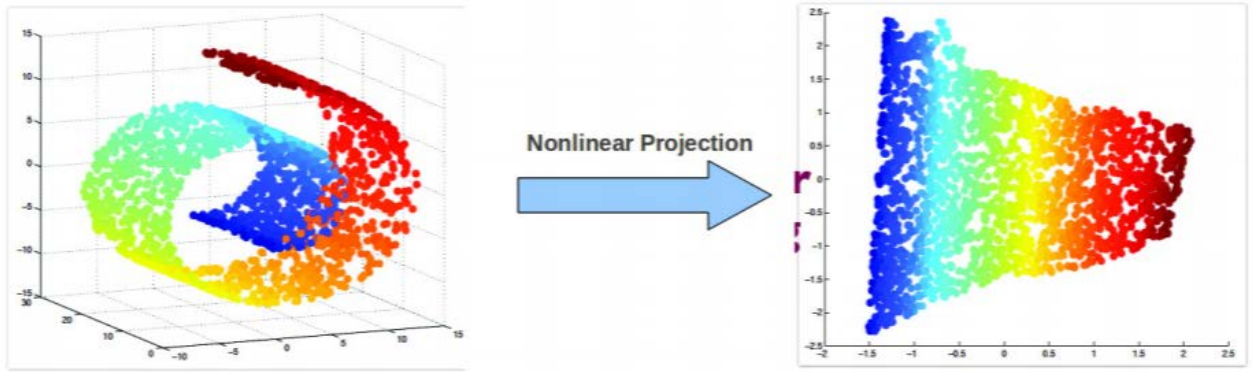
- Consider the swiss-roll dataset (points lying close to a manifold)



- Linear projection methods (e.g., PCA) can't capture intrinsic nonlinearities
  - Maximum variance directions may not be the most interesting ones

Nonlinear Dimensionality Reduction

- We want to a learn nonlinear low-dim projection



Relative positions of points preserved after the projection

- Some ways of doing this
  - Nonlinearize a linear dimensionality reduction method. E.g.:
    - Cluster data and apply linear PCA within each cluster (mixture of PCA)
    - Kernel PCA (nonlinear PCA)
  - Using manifold based methods that intrinsically preserve nonlinear geometry, e.g.,
    - Locally Linear Embedding (LLE), Isomap
    - Maximum Variance Unfolding
    - Laplacian Eigenmap, and others such as SNE/tSNE, etc.

Kernel PCA

- Recall PCA: Given N observations $\boldsymbol{x}_n \in \mathbb{R}^D$, $n = 1,2,\ldots,N$,

$D \times D$ cov matrix assuming centered data

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top$$

D eigenvectors of $\mathbf{S}$

$$\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i \;\; \forall i = 1,\ldots,D$$

- Assume a kernel k with associated M dimensional nonlinear map $\boldsymbol{\phi}$

$M \times M$ cov matrix assuming centered data in the kernel-induced feature space

$$\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$$

M eigenvectors of $\mathbf{C}$

$$\mathbf{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \;\; \forall i = 1,\ldots,M$$

- Would like to do it without computing $\mathbf{C}$ and the mappings $\boldsymbol{\phi}(\boldsymbol{x}_n)'s$ since M can be very large (even infinite, e.g., when using an RBF kernel)

- Boils down to doing eigendecomposition of the $N \times N$ kernel matrix $\mathbf{K}$ (PRML 12.3)
  - Can verify that each $\boldsymbol{v}_i$ above can be written as a lin-comb of the inputs: $\boldsymbol{v}_i = \sum_{n=1}^{N} a_{in}\boldsymbol{\phi}(\boldsymbol{x}_n)$
  - Can show that finding $\boldsymbol{a}_i = [a_{i1}, a_{i2}, \ldots, a_{iN}]$ reduces to solving an eigendecomposition of $\mathbf{K}$
  - Note: Due to req. of centering, we work with a centered kernel matrix $\tilde{\mathbf{K}} = \mathbf{K} - 1_N\mathbf{K} - \mathbf{K}1_N + 1_N\mathbf{K}1_N$

$N \times N$ matrix of all 1s

Locally Linear Embedding

Several non-lin dim-red algos use this idea

Essentially, neighbourhood preservation, but only local

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too
- Given N observations $\boldsymbol{x}_n \in \mathbb{R}^D$, $n = 1,2,\ldots,N$, LLE is formulated as

Solve this to learn weights W_{ij} such that each point $\boldsymbol{x}_i$ can be written as a weighted combination of its local neighbors in the original feature space

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \sum_{i=1}^{N} ||\boldsymbol{x}_i - \sum_{j \in \mathcal{N}(i)} W_{ij}\boldsymbol{x}_j||^2$$

$\mathcal{N}(i)$ denotes the local neighbors (a predefined number, say K, of them) of point $\boldsymbol{x}_i$

- For each point $\boldsymbol{x}_n \in \mathbb{R}^D$, LLE learns $\boldsymbol{z}_n \in \mathbb{R}^K$, $n = 1,2,\ldots,N$ such that the same neighborhood structure exists in low-dim space too

Requires solving an eigenvalue problem

$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \sum_{i=1}^{N} ||\boldsymbol{z}_i - \sum_{j \in \mathcal{N}(i)} W_{ij}\boldsymbol{z}_j||^2$$

- Basically, if point $\boldsymbol{x}_i$ can be reconstructed from its neighbors in the original space, the same weights W_{ij} should be able to reconstruct $\boldsymbol{z}_i$ in the new space too

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. Science 290 (2000)

SNE and t-SNE

Thus very useful if we want to visualize some high-dim data in two or three dims

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point $\boldsymbol{x}_i$, defines the probability of a point $\boldsymbol{x}_j$ being its neighbor as

Neighbor probability in the original space

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

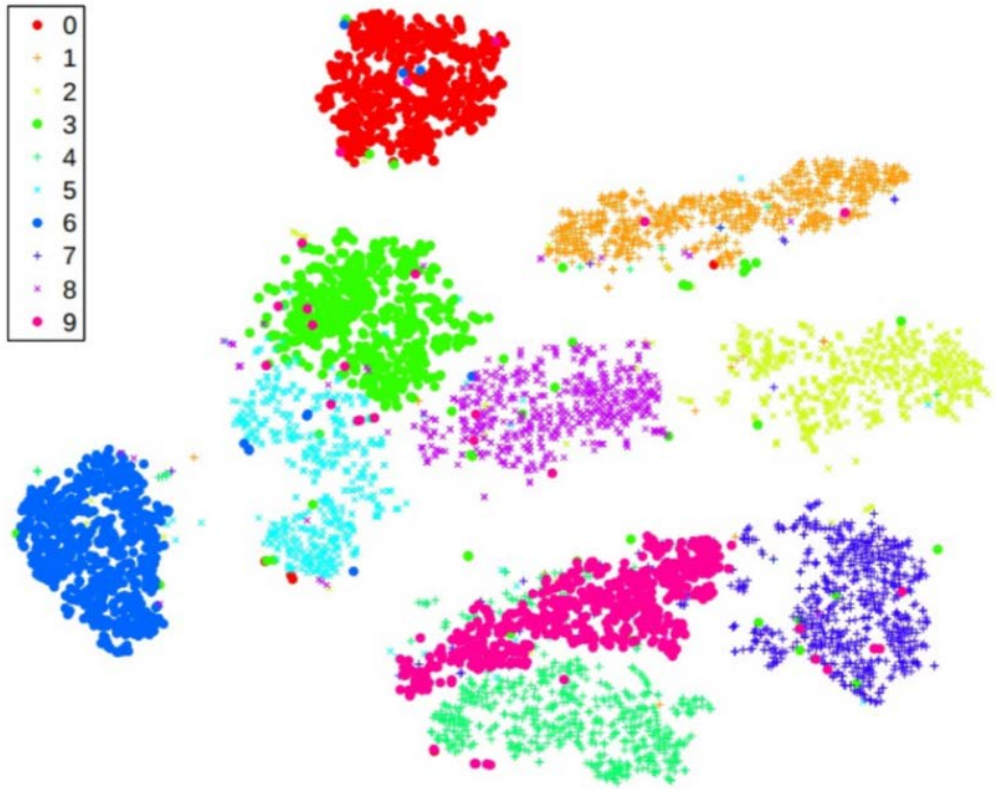
Neighbor probability in the projected/embedding space

$$q_{j|i} = \frac{\exp(-||\boldsymbol{z}_i - \boldsymbol{z}_j||^2/2\sigma^2)}{\sum_{k\neq i}\exp(-||\boldsymbol{z}_i - \boldsymbol{z}_k||^2/2\sigma^2)}$$

- SNE ensures that neighbourhood distributions in both spaces are as close as possible
  - By minimizing their Kullback-Leibler divergence, summed over all points $\sum_{i=1}^{N}\sum_{j=1}^{N} KL(p_{j|i}||q_{j|i})$
- t-SNE (van der Maaten and Hinton, 2008) offers a couple of improvements to SNE
  - Learns $\boldsymbol{z}_i$'s by minimizing symmetric KL divergence
  - Uses Student-t distribution instead of Gaussian for defining $q_{j|i}$

SNE and t-SNE

- Especially useful for visualizing data by projecting into 2D or 3D



Result of visualizing MNIST digits data in 2D (Figure from van der Maaten and Hinton, 2008)