Dimensionality Reduction (Wrap-up)

Piyush Rai

Introduction to Machine Learning (CS771A)

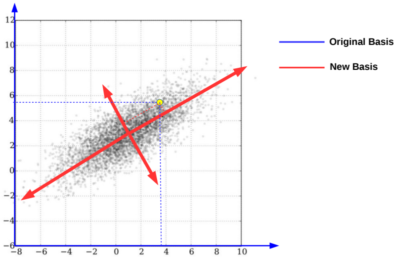
October 11, 2018

Plan for Today

- PCA: The classical view
- Singular Value Decomposition
- A simple technique to compute eigenvectors (Power Iteration)
- Supervised Dimensionality Reduction
- Dimensionality Reduction from Pairwise Distances
- Nonlinear Dimensionality Reduction

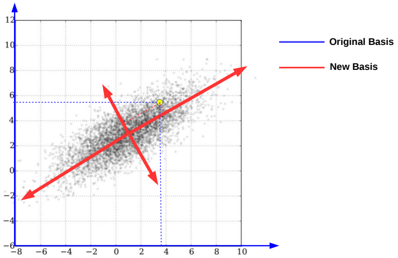
Principal Component Analysis: The Key Idea

- We can change the basis in which we represent the data (and get a new co-ordinate system)



Principal Component Analysis: The Key Idea

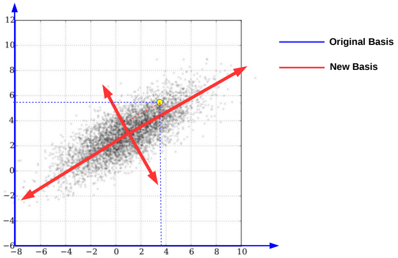
- We can change the basis in which we represent the data (and get a new co-ordinate system)



- If, in the new basis, data has low variance along some dimension, we can ignore those

Principal Component Analysis: The Key Idea

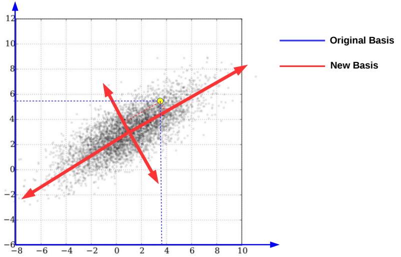
- We can change the basis in which we represent the data (and get a new co-ordinate system)



- If, in the new basis, data has low variance along some dimension, we can ignore those
  - Above picture: Can represent each point using just the first co-ordinate (very little information loss)

Principal Component Analysis: The Key Idea

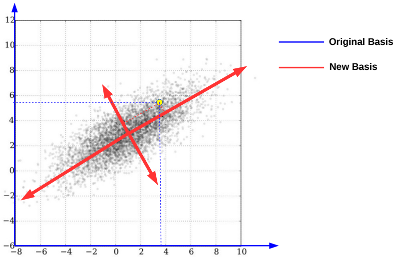
- We can change the basis in which we represent the data (and get a new co-ordinate system)



- If, in the new basis, data has low variance along some dimension, we can ignore those
  - Above picture: Can represent each point using just the first co-ordinate (very little information loss)
  - This helps in reducing dimensionalty: From $\boldsymbol{x} = [x_1, x_2]$ to $\boldsymbol{z} = [z_1, \cancel{z_2}]$ (i.e., 2D to 1D)

Principal Component Analysis: The Key Idea

- We can change the basis in which we represent the data (and get a new co-ordinate system)



- If, in the new basis, data has low variance along some dimension, we can ignore those
  - Above picture: Can represent each point using just the first co-ordinate (very little information loss)
  - This helps in reducing dimensionalty: From $\boldsymbol{x} = [x_1, x_2]$ to $\boldsymbol{z} = [z_1, \cancel{z_2}]$ (i.e., 2D to 1D)
- PCA finds a new basis such that information loss is minimum if we only keep some dimensions

Basis Represention of Data

- Representing a data point $\boldsymbol{x}_n = [x_{n1}, \ldots, x_{nD}]^\top$ in the standard orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} x_{nd} \boldsymbol{e}_d$$

.. where $\boldsymbol{e}_d$ is $D \times 1$ vector with all 0s and 1 at d-th entry

Basis Represention of Data

- Representing a data point $\boldsymbol{x}_n = [x_{n1}, \ldots, x_{nD}]^\top$ in the standard orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} x_{nd} \boldsymbol{e}_d$$

.. where $\boldsymbol{e}_d$ is $D \times 1$ vector with all 0s and 1 at d-th entry (also $\boldsymbol{e}_d^\top \boldsymbol{e}_d = 1$, $\boldsymbol{e}_d^\top \boldsymbol{e}_{d'} = 0$, $d \neq d'$).

Basis Represention of Data

- Representing a data point $\boldsymbol{x}_n = [x_{n1}, \ldots, x_{nD}]^\top$ in the standard orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} x_{nd}\boldsymbol{e}_d$$

.. where $\boldsymbol{e}_d$ is $D \times 1$ vector with all 0s and 1 at d-th entry (also $\boldsymbol{e}_d^\top \boldsymbol{e}_d = 1$, $\boldsymbol{e}_d^\top \boldsymbol{e}_{d'} = 0$, $d \neq d'$).

- Suppose we represent the same data point in a new orthonormal basis $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$$

Basis Represention of Data

- Representing a data point $\boldsymbol{x}_n = [x_{n1}, \ldots, x_{nD}]^\top$ in the standard orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} x_{nd} \boldsymbol{e}_d$$

.. where $\boldsymbol{e}_d$ is $D \times 1$ vector with all 0s and 1 at d-th entry (also $\boldsymbol{e}_d^\top \boldsymbol{e}_d = 1$, $\boldsymbol{e}_d^\top \boldsymbol{e}_{d'} = 0$, $d \neq d'$).

- Suppose we represent the same data point in a new orthonormal basis $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd} \boldsymbol{u}_d$$

where the new co-ordinates for $\boldsymbol{x}_n$ are $\boldsymbol{z}_n = [z_{n1}, \ldots, \boldsymbol{z}_{nD}]$

Basis Represention of Data

- Representing a data point $\boldsymbol{x}_n = [x_{n1}, \ldots, x_{nD}]^\top$ in the standard orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} x_{nd} \boldsymbol{e}_d$$

.. where $\boldsymbol{e}_d$ is $D \times 1$ vector with all 0s and 1 at d-th entry (also $\boldsymbol{e}_d^\top \boldsymbol{e}_d = 1$, $\boldsymbol{e}_d^\top \boldsymbol{e}_{d'} = 0$, $d \neq d'$).

- Suppose we represent the same data point in a new orthonormal basis $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D\}$

$$\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd} \boldsymbol{u}_d$$

where the new co-ordinates for $\boldsymbol{x}_n$ are $\boldsymbol{z}_n = [z_{n1}, \ldots, \boldsymbol{z}_{nD}]$

- Note that each new co-ordinate z_{dn} is a projection of $\boldsymbol{x}_n$ along direction $\boldsymbol{u}_d$

$$z_{nd} = \boldsymbol{x}_n^\top \boldsymbol{u}_d = \boldsymbol{u}_d^\top \boldsymbol{x}_n \quad \text{(verify)}$$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd} \boldsymbol{u}_d$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n$$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d$$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K} (\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d$$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n$$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n$$

- Now we have a $K < D$ dimensional representation $\boldsymbol{z}_n = [z_{n1}, \ldots, z_{nK}]$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n$$

- Now we have a $K < D$ dimensional representation $\boldsymbol{z}_n = [z_{n1}, \ldots, z_{nK}]$

$$\boldsymbol{z}_n = \mathbf{U}_K^\top \boldsymbol{x}_n \quad \text{(verify)}$$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n$$

- Now we have a $K < D$ dimensional representation $\boldsymbol{z}_n = [z_{n1}, \ldots, z_{nK}]$

$$\boldsymbol{z}_n = \mathbf{U}_K^\top \boldsymbol{x}_n \quad \text{(verify)}$$

where $\mathbf{U}_K = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]$ is the $D \times K$ "projection matrix"

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d = \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n$$

- Now we have a $K < D$ dimensional representation $\boldsymbol{z}_n = [z_{n1}, \ldots, z_{nK}]$

$$\boldsymbol{z}_n = \mathbf{U}_K^\top \boldsymbol{x}_n \quad \text{(verify)}$$

  where $\mathbf{U}_K = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]$ is the $D \times K$ "projection matrix"
- The reconstruction error of this approximation is $||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2$

Keeping Only Few Directions..

- So we saw that we can represent data in a new vector space as $\boldsymbol{x}_n = \sum_{d=1}^{D} z_{nd}\boldsymbol{u}_d$
- We can ignore the directions along which the projection z_{nd} is small, and approximate $\boldsymbol{x}_n$ as

$$\boldsymbol{x}_n \approx \hat{\boldsymbol{x}}_n = \sum_{d=1}^{K} z_{nd}\boldsymbol{u}_d = \sum_{d=1}^{K} (\boldsymbol{x}_n^\top \boldsymbol{u}_d)\boldsymbol{u}_d = \sum_{d=1}^{K} (\boldsymbol{u}_d \boldsymbol{u}_d^\top)\boldsymbol{x}_n$$

- Now we have a $K < D$ dimensional representation $\boldsymbol{z}_n = [z_{n1}, \ldots, z_{nK}]$

$$\boldsymbol{z}_n = \mathbf{U}_K^\top \boldsymbol{x}_n \quad \text{(verify)}$$

  where $\mathbf{U}_K = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]$ is the $D \times K$ "projection matrix"

- The reconstruction error of this approximation is $||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2$
- How to choose K directions $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ such that this reconstruction error is minimum?

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2$$

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2$$

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2 = C - \sum_{d=1}^{K}\boldsymbol{u}_d^\top\mathbf{S}\boldsymbol{u}_d \quad \text{(verify)}$$

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2 = C - \sum_{d=1}^{K}\boldsymbol{u}_d^\top\mathbf{S}\boldsymbol{u}_d \quad \text{(verify)}$$

.. where C is a constant that does not depend on $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2 = C - \sum_{d=1}^{K} \boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d \quad \text{(verify)}$$

.. where C is a constant that does not depend on $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$

- Note: $\boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d$ is also the variance the data when projected along direction $\boldsymbol{u}_d$ (exercise)

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2 = C - \sum_{d=1}^{K} \boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d \quad \text{(verify)}$$

.. where C is a constant that does not depend on $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$

- Note: $\boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d$ is also the variance the data when projected along direction $\boldsymbol{u}_d$ (exercise)
- Finding each the optimal direction $\boldsymbol{u}_d$ requires solving

$$\arg\min_{\boldsymbol{u}_d} \mathcal{L}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K) = \arg\max_{\boldsymbol{u}_d} \quad \boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d \qquad \text{s.t. } \boldsymbol{u}_d^\top\boldsymbol{u}_d = 1$$

Directions that Minimize Reconstruction Error..

- Assume $\mathbf{S}$ is the $D \times D$ cov. matrix: $\mathbf{S} = \frac{1}{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming already centered data)
- The reconstruction error for the entire data is given by

$$\mathcal{L}(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_K) = \sum_{n=1}^{N}||\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N}||\boldsymbol{x}_n - \sum_{d=1}^{K}(\boldsymbol{u}_d\boldsymbol{u}_d^\top)\boldsymbol{x}_n||^2 = C - \sum_{d=1}^{K}\boldsymbol{u}_d^\top\mathbf{S}\boldsymbol{u}_d \quad \text{(verify)}$$

  .. where C is a constant that does not depend on $\boldsymbol{u}_1,\ldots,\boldsymbol{u}_K$
- Note: $\boldsymbol{u}_d^\top\mathbf{S}\boldsymbol{u}_d$ is also the variance the data when projected along direction $\boldsymbol{u}_d$ (exercise)
- Finding each the optimal direction $\boldsymbol{u}_d$ requires solving

$$\arg\min_{\boldsymbol{u}_d}\mathcal{L}(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_K) = \arg\max_{\boldsymbol{u}_d}\quad \boldsymbol{u}_d^\top\mathbf{S}\boldsymbol{u}_d \qquad \text{s.t. } \boldsymbol{u}_d^\top\boldsymbol{u}_d = 1$$

- Thus minimizing recon. error w.r.t. $\boldsymbol{u}_d$ equivalent to maximizing the variance of data along $\boldsymbol{u}_d$

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \ \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d(1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \; \boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d + \lambda_d(1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \ \boldsymbol{u}_d^\top \mathbf{S}\boldsymbol{u}_d + \lambda_d(1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \ \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d (1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)
- But which of $\mathbf{S}$'s (D possible) eigenvectors it is?

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \; \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d(1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)
- But which of $\mathbf{S}$'s (D possible) eigenvectors it is?
- Note that since $\boldsymbol{u}_d^\top \boldsymbol{u}_d = 1$, the variance of projected data is

$$\boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d = \lambda_d$$

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \ \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d (1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)
- But which of $\mathbf{S}$'s (D possible) eigenvectors it is?
- Note that since $\boldsymbol{u}_d^\top \boldsymbol{u}_d = 1$, the variance of projected data is

$$\boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d = \lambda_d$$

- Thus the variance is maximized when $\boldsymbol{u}_d$ is the (top) eigenvector with largest eigenvalue λ_1

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \; \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d(1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)
- But which of $\mathbf{S}$'s (D possible) eigenvectors it is?
- Note that since $\boldsymbol{u}_d^\top \boldsymbol{u}_d = 1$, the variance of projected data is

$$\boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d = \lambda_d$$

- Thus the variance is maximized when $\boldsymbol{u}_d$ is the (top) eigenvector with largest eigenvalue λ_1
- We denote the top eigenvector as $\boldsymbol{u}_1$ and it called the first Principal Component (PC)

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \; \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d (1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)
- But which of $\mathbf{S}$'s (D possible) eigenvectors it is?
- Note that since $\boldsymbol{u}_d^\top \boldsymbol{u}_d = 1$, the variance of projected data is

$$\boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d = \lambda_d$$

- Thus the variance is maximized when $\boldsymbol{u}_d$ is the (top) eigenvector with largest eigenvalue λ_1
- We denote the top eigenvector as $\boldsymbol{u}_1$ and it called the first Principal Component (PC)
- Other directions can also be found likewise (with each being orthogonal to all previous ones) using the eigendecomposition of $\mathbf{S}$

Direction of Maximum Variance

- The objective function: $\arg\max_{\boldsymbol{u}_d} \ \boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d + \lambda_d(1 - \boldsymbol{u}_d^\top \boldsymbol{u}_d)$
- Taking the derivative w.r.t. $\boldsymbol{u}_d$ and setting to zero gives

$$\mathbf{S}\boldsymbol{u}_d = \lambda_d \boldsymbol{u}_d$$

- Thus $\boldsymbol{u}_d$ is an eigenvector of $\mathbf{S}$ (with corresponding eigenvalue λ_d)
- But which of $\mathbf{S}$'s (D possible) eigenvectors it is?
- Note that since $\boldsymbol{u}_d^\top \boldsymbol{u}_d = 1$, the variance of projected data is

$$\boldsymbol{u}_d^\top \mathbf{S} \boldsymbol{u}_d = \lambda_d$$

- Thus the variance is maximized when $\boldsymbol{u}_d$ is the (top) eigenvector with largest eigenvalue λ_1
- We denote the top eigenvector as $\boldsymbol{u}_1$ and it called the first Principal Component (PC)
- Other directions can also be found likewise (with each being orthogonal to all previous ones) using the eigendecomposition of $\mathbf{S}$(this is basically the PCA algorithm)

Principal Component Analysis

- Center the data (subtract the mean $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$ from each data point)

Principal Component Analysis

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

$$\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X} \qquad (\text{note: } \mathbf{X} \text{ assumed } D \times N \text{ here})$$

Principal Component Analysis

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

$$\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X} \qquad \text{(note: } \mathbf{X} \text{ assumed } D \times N \text{ here)}$$

- Do an eigendecomposition of the covariance matrix $\mathbf{S}$ (many methods exist)

Principal Component Analysis

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

$$\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X} \qquad \text{(note: } \mathbf{X} \text{ assumed } D \times N \text{ here)}$$

- Do an eigendecomposition of the covariance matrix $\mathbf{S}$ (many methods exist)
- Take top K leading eigenvectors $\{\boldsymbol{u}_k\}_{k=1}^{K}$ with largest eigenvalues $\{\lambda_k\}_{k=1}^{K}$

Principal Component Analysis

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

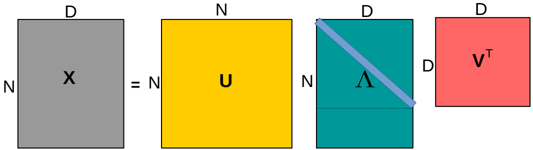
$$\mathbf{S} = \frac{1}{N}\mathbf{X}^\top \mathbf{X} \qquad \text{(note: } \mathbf{X} \text{ assumed } D \times N \text{ here)}$$

- Do an eigendecomposition of the covariance matrix $\mathbf{S}$ (many methods exist)
- Take top K leading eigenvectors $\{\boldsymbol{u}_k\}_{k=1}^{K}$ with largest eigenvalues $\{\lambda_k\}_{k=1}^{K}$
- The K-dimensional projection/embedding of the $N \times D$ data matrix $\mathbf{X}$ is given by

$$\mathbf{Z} = \mathbf{X}\mathbf{U}_K$$

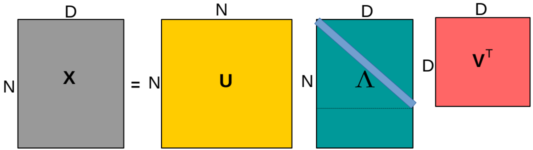
where $\mathbf{U}_K = [\boldsymbol{u}_1 \ \ldots \ \boldsymbol{u}_K]$ is $D \times K$ and embedding matrix $\mathbf{Z}$ is $N \times K$

Singular Value Decomposition (SVD)



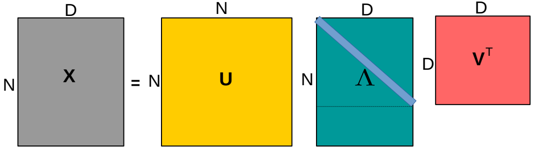
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top$

Singular Value Decomposition (SVD)



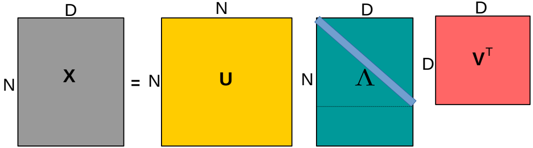
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$

Singular Value Decomposition (SVD)



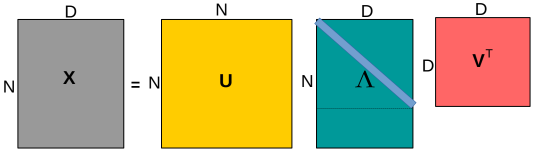
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$
- $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ is $N \times N$, each $\boldsymbol{u}_n \in \mathbb{R}^N$ a left singular vector of $\mathbf{X}$

Singular Value Decomposition (SVD)



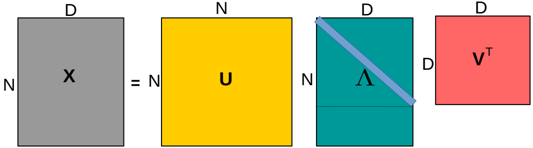
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$
- $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ is $N \times N$, each $\boldsymbol{u}_n \in \mathbb{R}^N$ a left singular vector of $\mathbf{X}$
  - $\mathbf{U}$ is orthonormal: $\boldsymbol{u}_n^\top \boldsymbol{u}_{n'} = 0$ for $n \neq n'$, and $\boldsymbol{u}_n^\top \boldsymbol{u}_n = 1 \Rightarrow \mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$

Singular Value Decomposition (SVD)



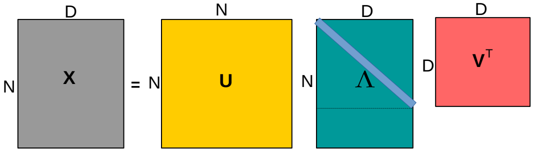
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$
- $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ is $N \times N$, each $\boldsymbol{u}_n \in \mathbb{R}^N$ a left singular vector of $\mathbf{X}$
  - $\mathbf{U}$ is orthonormal: $\boldsymbol{u}_n^\top \boldsymbol{u}_{n'} = 0$ for $n \neq n'$, and $\boldsymbol{u}_n^\top \boldsymbol{u}_n = 1 \Rightarrow \mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$
- Λ is $N \times D$ with only $\min(N, D)$ diagonal entries - singular values

Singular Value Decomposition (SVD)



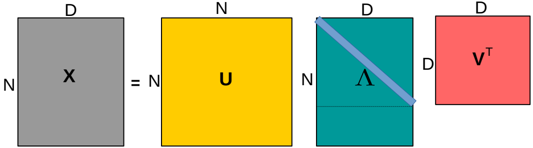
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$
- $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ is $N \times N$, each $\boldsymbol{u}_n \in \mathbb{R}^N$ a left singular vector of $\mathbf{X}$
  - $\mathbf{U}$ is orthonormal: $\boldsymbol{u}_n^\top \boldsymbol{u}_{n'} = 0$ for $n \neq n'$, and $\boldsymbol{u}_n^\top \boldsymbol{u}_n = 1 \Rightarrow \mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$
- Λ is $N \times D$ with only $\min(N, D)$ diagonal entries - singular values
- $\mathbf{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_D]$ is $D \times D$, each $\boldsymbol{v}_d \in \mathbb{R}^D$, a right singular vector of $\mathbf{X}$

Singular Value Decomposition (SVD)



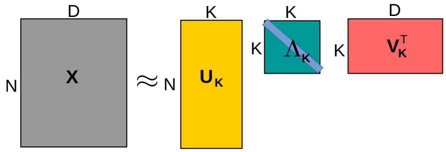
- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$
- $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ is $N \times N$, each $\boldsymbol{u}_n \in \mathbb{R}^N$ a left singular vector of $\mathbf{X}$
  - $\mathbf{U}$ is orthonormal: $\boldsymbol{u}_n^\top \boldsymbol{u}_{n'} = 0$ for $n \neq n'$, and $\boldsymbol{u}_n^\top \boldsymbol{u}_n = 1 \Rightarrow \mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$
- Λ is $N \times D$ with only $\min(N, D)$ diagonal entries - singular values
- $\mathbf{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_D]$ is $D \times D$, each $\boldsymbol{v}_d \in \mathbb{R}^D$, a right singular vector of $\mathbf{X}$
  - $\mathbf{V}$ is orthonormal: $\boldsymbol{v}_d^\top \boldsymbol{v}_{d'} = 0$ for $d \neq d'$, and $\boldsymbol{v}_d^\top \boldsymbol{v}_d = 1 \Rightarrow \mathbf{V}\mathbf{V}^\top = \mathbf{I}_D$

Singular Value Decomposition (SVD)



- We can represent any matrix $\mathbf{X}$ of size $N \times D$ using SVD as $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top = \sum_{k=1}^{\min(N,D)} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\top$
- $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N]$ is $N \times N$, each $\boldsymbol{u}_n \in \mathbb{R}^N$ a left singular vector of $\mathbf{X}$
  - $\mathbf{U}$ is orthonormal: $\boldsymbol{u}_n^\top \boldsymbol{u}_{n'} = 0$ for $n \neq n'$, and $\boldsymbol{u}_n^\top \boldsymbol{u}_n = 1 \Rightarrow \mathbf{U}\mathbf{U}^\top = \mathbf{I}_N$
- Λ is $N \times D$ with only $\min(N, D)$ diagonal entries - singular values
- $\mathbf{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_D]$ is $D \times D$, each $\boldsymbol{v}_d \in \mathbb{R}^D$, a right singular vector of $\mathbf{X}$
  - $\mathbf{V}$ is orthonormal: $\boldsymbol{v}_d^\top \boldsymbol{v}_{d'} = 0$ for $d \neq d'$, and $\boldsymbol{v}_d^\top \boldsymbol{v}_d = 1 \Rightarrow \mathbf{V}\mathbf{V}^\top = \mathbf{I}_D$
- Note: If $\mathbf{X}$ is symmetric then it is known as eigenvalue decomposition (and $\mathbf{U} = \mathbf{V}$ in that case)

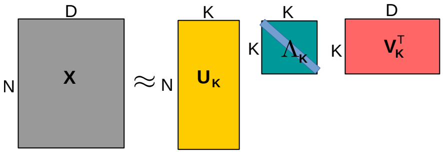
Low-Rank Approximation via SVD



- Rank-K approximation of $\mathbf{X}$ (where $K \ll \min(N, D)$) using K largest in magnitude λ_k's as

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

Low-Rank Approximation via SVD

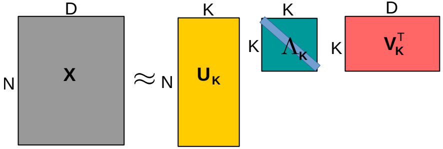


- Rank-K approximation of $\mathbf{X}$ (where $K \ll \min(N, D)$) using K largest in magnitude λ_k's as

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

- The above SVD approximation can be shown to minimize the reconstruction error

Low-Rank Approximation via SVD

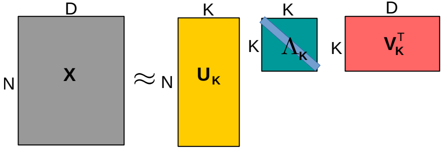


- Rank-K approximation of $\mathbf{X}$ (where $K \ll \min(N, D)$) using K largest in magnitude λ_k's as

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

- The above SVD approximation can be shown to minimize the reconstruction error
  - Fact: SVD gives the best rank-K approximation of a matrix

Low-Rank Approximation via SVD

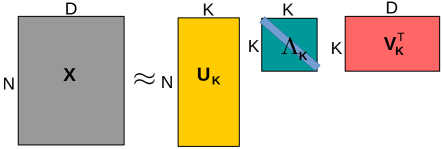


- Rank-K approximation of $\mathbf{X}$ (where $K \ll \min(N, D)$) using K largest in magnitude λ_k's as

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

- The above SVD approximation can be shown to minimize the reconstruction error
  - Fact: SVD gives the best rank-K approximation of a matrix
- PCA basically does SVD on the covariance matrix $\mathbf{S}$ (singular vectors = eigenvectors)

Low-Rank Approximation via SVD



- Rank-K approximation of $\mathbf{X}$ (where $K \ll \min(N, D)$) using K largest in magnitude λ_k's as

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

- The above SVD approximation can be shown to minimize the reconstruction error
  - Fact: SVD gives the best rank-K approximation of a matrix
- PCA basically does SVD on the covariance matrix $\mathbf{S}$ (singular vectors $=$ eigenvectors)
  - Since $\mathbf{S}$ is symmetric, $\mathbf{U} = \mathbf{V}$

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)
- For naïve methods, even to get one eigenvector, we need to perform full eigendecomposition

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)
- For naïve methods, even to get one eigenvector, we need to perform full eigendecomposition
- If we want $K < D$ eigenvectors, there are some more efficient methods

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)
- For naïve methods, even to get one eigenvector, we need to perform full eigendecomposition
- If we want $K < D$ eigenvectors, there are some more efficient methods
- Power Method is one such approach: Sequentially finds the top K eigenvectors of a cov matrix

$$\mathbf{S} = \sum_{k=1}^{D} \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^\top$$

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)
- For naïve methods, even to get one eigenvector, we need to perform full eigendecomposition
- If we want $K < D$ eigenvectors, there are some more efficient methods
- Power Method is one such approach: Sequentially finds the top K eigenvectors of a cov matrix

$$\mathbf{S} = \sum_{k=1}^{D} \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^\top$$

- The overall cost for this method is $O(KD^2)$

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)
- For naïve methods, even to get one eigenvector, we need to perform full eigendecomposition
- If we want $K < D$ eigenvectors, there are some more efficient methods
- Power Method is one such approach: Sequentially finds the top K eigenvectors of a cov matrix

$$\mathbf{S} = \sum_{k=1}^{D} \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^\top$$

- The overall cost for this method is $O(KD^2)$
- Based on the fact that for any vector $\boldsymbol{x} = \sum_{k=1}^{D} z_k \boldsymbol{u}_k$

$$\mathbf{S}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k \boldsymbol{u}_k$$

Computing Eigenvectors: Power Method

- Computing eigenvectors is expensive in general ($O(D^3)$ for finding D eigenvectors)
- For naïve methods, even to get one eigenvector, we need to perform full eigendecomposition
- If we want $K < D$ eigenvectors, there are some more efficient methods
- Power Method is one such approach: Sequentially finds the top K eigenvectors of a cov matrix

$$\mathbf{S} = \sum_{k=1}^{D} \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^\top$$

- The overall cost for this method is $O(KD^2)$
- Based on the fact that for any vector $\boldsymbol{x} = \sum_{k=1}^{D} z_k \boldsymbol{u}_k$

$$\mathbf{S}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k \boldsymbol{u}_k, \quad \text{and} \quad \underbrace{(\mathbf{SS}\ldots,\mathbf{S})}_{M \text{ times}} \boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

Computing Eigenvectors: Power Method

- So we saw that we have

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

Computing Eigenvectors: Power Method

- So we saw that we have

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

- Assuming $\lambda_1 > \lambda_2 \geq \lambda_3 \ldots$ then for large M

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} \approx z_1 \lambda_1^M \boldsymbol{u}_1$$

Computing Eigenvectors: Power Method

- So we saw that we have

$$\underbrace{(\mathbf{SS}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

- Assuming $\lambda_1 > \lambda_2 \geq \lambda_3 \ldots$ then for large M

$$\underbrace{(\mathbf{SS}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} \approx z_1 \lambda_1^M \boldsymbol{u}_1$$

- This gives us a simple algorithm to get the top eigenvector

Computing Eigenvectors: Power Method

- So we saw that we have

$$\underbrace{(\mathbf{SS}\ldots,\mathbf{S})}_{M\text{ times}}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

- Assuming $\lambda_1 > \lambda_2 \geq \lambda_3 \ldots$ then for large M

$$\underbrace{(\mathbf{SS}\ldots,\mathbf{S})}_{M\text{ times}}\boldsymbol{x} \approx z_1 \lambda_1^M \boldsymbol{u}_1$$

- This gives us a simple algorithm to get the top eigenvector
  - Initialize $\boldsymbol{x}_0 \sim \mathcal{N}(0, \mathbf{I}_D)$

Computing Eigenvectors: Power Method

- So we saw that we have

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

- Assuming $\lambda_1 > \lambda_2 \geq \lambda_3 \ldots$ then for large M

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} \approx z_1 \lambda_1^M \boldsymbol{u}_1$$

- This gives us a simple algorithm to get the top eigenvector
  - Initialize $\boldsymbol{x}_0 \sim \mathcal{N}(0, \mathbf{I}_D)$
  - For $m = 1, \ldots, M$, compute $\boldsymbol{x}_m$ as $\boldsymbol{x}_m = \mathbf{S}\boldsymbol{x}_{m-1}$ and normalize it as $\boldsymbol{x}_m = \boldsymbol{x}_m / ||\boldsymbol{x}_m||_2$

Computing Eigenvectors: Power Method

- So we saw that we have

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} = \sum_{k=1}^{D} z_k \lambda_k^M \boldsymbol{u}_k$$

- Assuming $\lambda_1 > \lambda_2 \geq \lambda_3 \ldots$ then for large M

$$\underbrace{(\mathbf{S}\mathbf{S}\ldots,\mathbf{S})}_{M \text{ times}}\boldsymbol{x} \approx z_1 \lambda_1^M \boldsymbol{u}_1$$

- This gives us a simple algorithm to get the top eigenvector
    - Initialize $\boldsymbol{x}_0 \sim \mathcal{N}(0, \mathbf{I}_D)$
    - For $m = 1, \ldots, M$, compute $\boldsymbol{x}_m$ as $\boldsymbol{x}_m = \mathbf{S}\boldsymbol{x}_{m-1}$ and normalize it as $\boldsymbol{x}_m = \boldsymbol{x}_m / ||\boldsymbol{x}_m||_2$
    - After convergence, $\boldsymbol{x}_M$ is the largest eigenvector and $||\mathbf{S}\boldsymbol{x}_M||$ is the largest eigenvalue
- The main dominant cost is computing $\mathbf{S}\boldsymbol{x}_{m-1}$ which is $O(D^2)$.

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a “peeling” techique to get all the top K eigenvectors

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a "peeling" techique to get all the top K eigenvectors
- The basic procedure would be

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a "peeling" techique to get all the top K eigenvectors
- The basic procedure would be
  - Initialize $\mathbf{S}^{(0)} = \mathbf{S}$

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a “peeling” techique to get all the top K eigenvectors
- The basic procedure would be
  - Initialize $\mathbf{S}^{(0)} = \mathbf{S}$
  - For $k = 1, \ldots, K$

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a "peeling" techique to get all the top K eigenvectors
- The basic procedure would be
  - Initialize $\mathbf{S}^{(0)} = \mathbf{S}$
  - For $k = 1, \ldots, K$

$$\{\boldsymbol{u}_k, \lambda_k\} \quad = \quad \text{POWER-METHOD}(\mathbf{S}^{(k-1)})$$

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a "peeling" techique to get all the top K eigenvectors
- The basic procedure would be
  - Initialize $\mathbf{S}^{(0)} = \mathbf{S}$
  - For $k = 1, \ldots, K$

$$\begin{aligned} \{\boldsymbol{u}_k, \lambda_k\} &= \text{POWER-METHOD}(\mathbf{S}^{(k-1)}) \\ \mathbf{S}^{(k)} &= \mathbf{S}^{(k-1)} - \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^\top \quad (\text{“Peeling” the covariance matrix}) \end{aligned}$$

Power Method for All of Top-K Eigenvectors?

- Can use Power Method with a "peeling" techique to get all the top K eigenvectors
- The basic procedure would be
  - Initialize $\mathbf{S}^{(0)} = \mathbf{S}$
  - For $k = 1, \ldots, K$

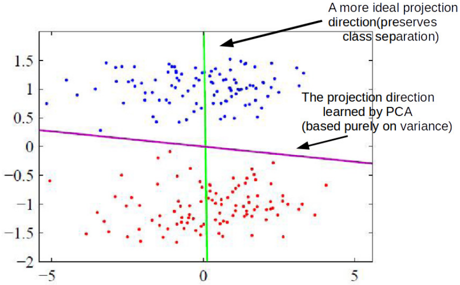
$$
\begin{aligned}
\{\boldsymbol{u}_k, \lambda_k\} &= \text{POWER-METHOD}(\mathbf{S}^{(k-1)}) \\
\mathbf{S}^{(k)} &= \mathbf{S}^{(k-1)} - \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^\top \quad \text{(“Peeling” the covariance matrix)}
\end{aligned}
$$

- Each power iteration is $O(D^2)$, overall cost for getting K eigenvectors is $O(KD^2)$

Supervised Dimensionality Reduction

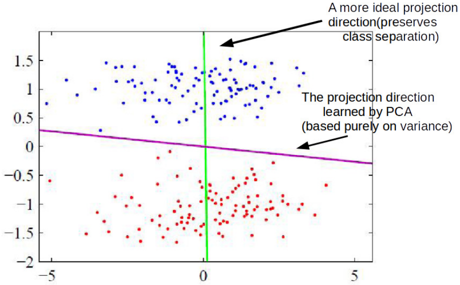
Supervised Dimensionality Reduction

- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)



Supervised Dimensionality Reduction

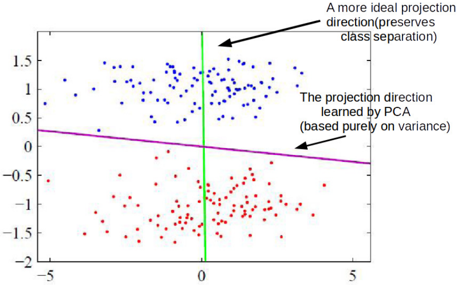
- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)



- A better option would be to project such that

Supervised Dimensionality Reduction

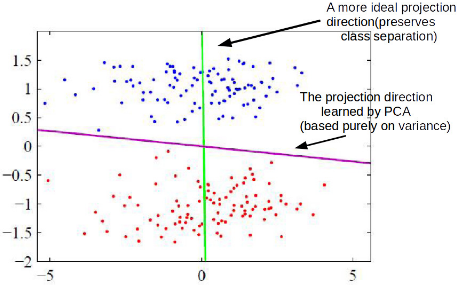
- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)



- A better option would be to project such that
  - Points within the same class are close (low intra-class variance)

Supervised Dimensionality Reduction

- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)



- 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

Supervised Dimensionality Reduction

- Many techniques. A simple yet popular one is Fisher discriminant analysis
- For simplicity, assume two classes (can be generalized for more than 2 classes too)

Supervised Dimensionality Reduction

- Many techniques. A simple yet popular one is Fisher discriminant analysis
- 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

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

Supervised Dimensionality Reduction

- Many techniques. A simple yet popular one is Fisher discriminant analysis
- 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

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

- Total variance will be $s_1^2 + s_2^2$ where

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

Supervised Dimensionality Reduction

- Many techniques. A simple yet popular one is Fisher discriminant analysis
- 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

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

- Total variance 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 as

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

Supervised Dimensionality Reduction

- Many techniques. A simple yet popular one is Fisher discriminant analysis
- 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

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

- Total variance will be $s_1^2 + s_2^2$ where

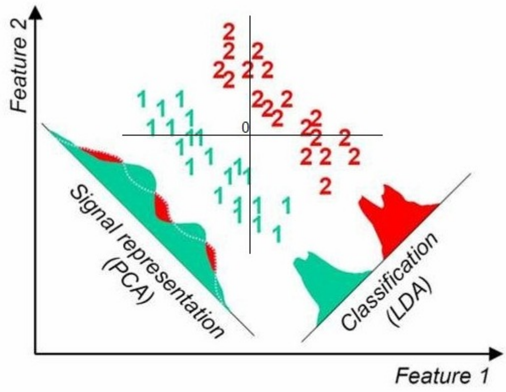
$$s_1^2 = \frac{1}{N_1} \sum_{n:y_n=1} (\boldsymbol{u}^\top \boldsymbol{x}_n - \mu_1)^2, \quad s_2^2 = \frac{1}{N_2} \sum_{n:y_n=2} (\boldsymbol{u}^\top \boldsymbol{x}_n - \mu_2)^2$$

- Fisher discriminant analysis finds the optimal projection direction as

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

- Solution for $\boldsymbol{u}$ depends on eigendecomposition of within class and between class covariance matrices

Supervised Dimensionality Reduction



Can be generalized for projections to more than 1 dimensional space

Picture courtesy: Stackoverflow

Dimensionality Reduction given Pairwise Distances between Points

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$
- Can't apply PPCA/PCA/SVD etc for such data

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$
- Can't apply PPCA/PCA/SVD etc for such data
- In these cases, we want to project the data such that pairwise distances are preserved

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

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$
- Can't apply PPCA/PCA/SVD etc for such data
- In these cases, we want to project the data such that pairwise distances are preserved

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

- If $d_{ij}$is large/small then we want $||\boldsymbol{z}_i - \boldsymbol{z}_j||$ to be large/small

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$
- Can't apply PPCA/PCA/SVD etc for such data
- In these cases, we want to project the data such that pairwise distances are preserved

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

- If $d_{ij}$is large/small then we want $||\boldsymbol{z}_i - \boldsymbol{z}_j||$ to be large/small
- Multi-dimensional Scaling (MDS) is one such algorithm

Dimensionality Reduction by Preserving Pairwise Distances

- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$
- Can't apply PPCA/PCA/SVD etc for such data
- In these cases, we want to project the data such that pairwise distances are preserved

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

- If $d_{ij}$is large/small then we want $||\boldsymbol{z}_i - \boldsymbol{z}_j||$ to be large/small
- Multi-dimensional Scaling (MDS) is one such algorithm
- Can show show that preserving <u>all</u> pairwise <u>Euclidean</u> distances = doing PCA :-)

Dimensionality Reduction by Preserving Pairwise Distances

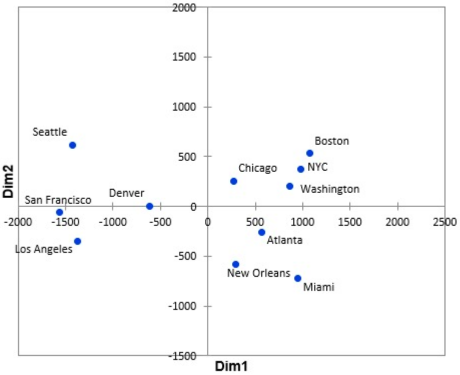
- PPCA/PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- However often the data is given in form of distances d_{ij} for $i = 1, \ldots, N$, $j = 1, \ldots, N$
- Can't apply PPCA/PCA/SVD etc for such data
- In these cases, we want to project the data such that pairwise distances are preserved

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

- If $d_{ij}$is large/small then we want $||\boldsymbol{z}_i - \boldsymbol{z}_j||$ to be large/small
- Multi-dimensional Scaling (MDS) is one such algorithm
- Can show show that preserving <u>all</u> pairwise <u>Euclidean</u> distances = doing PCA :-)
- Important: Often it is better to only preserve distances between nearest neighbors (helps in learning nonlinear projections), methods like locally linear embedding (LLE) and Isomap do this.

Multi-dimensional Scaling: An Illustration

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

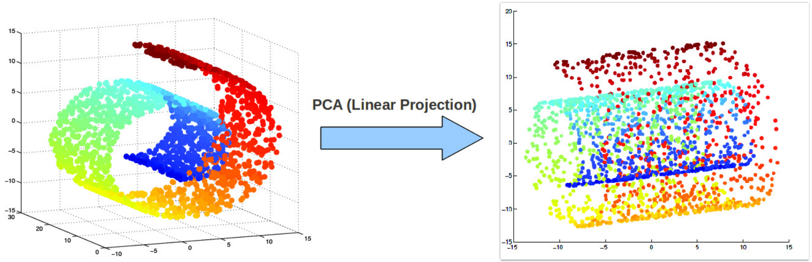


MDS produces a 2D embedding such that geographically close cities are also close in embedding space.

Nonlinear Dimensionality Reduction

Beyond Linear Projections..

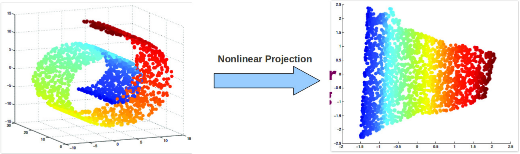
- Consider the swiss-roll dataset (points lying close to a manifold)



- Linear projection methods (e.g., PCA) can't capture intrinsic nonlinearities

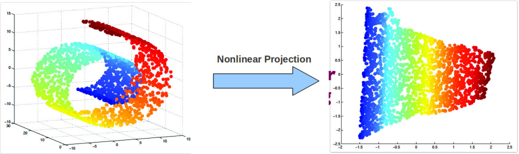
Nonlinear Dimensionality Reduction

- We want to a learn nonlinear low-dim projection



Nonlinear Dimensionality Reduction

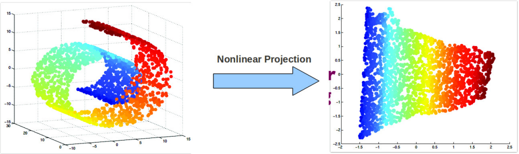
- We want to a learn nonlinear low-dim projection



- Some ways of doing this

Nonlinear Dimensionality Reduction

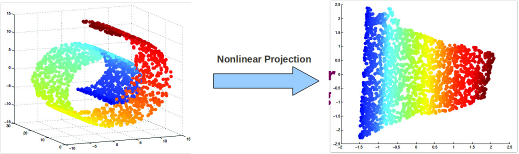
- We want to a learn nonlinear low-dim projection



- Some ways of doing this
  - Nonlinearize a linear dimensionality reduction method. E.g.:
    - Mixture of linear dim-red models like mixture of PPCA (already seen)
    - **Kernel PCA (nonlinear PCA)**

Nonlinear Dimensionality Reduction

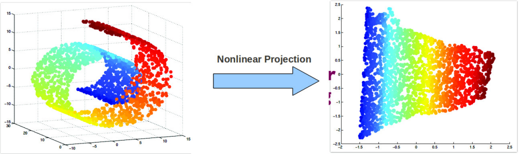
- We want to a learn nonlinear low-dim projection



- Some ways of doing this
  - Nonlinearize a linear dimensionality reduction method. E.g.:
    - Mixture of linear dim-red models like mixture of PPCA (already seen)
    - **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.

Nonlinear Dimensionality Reduction

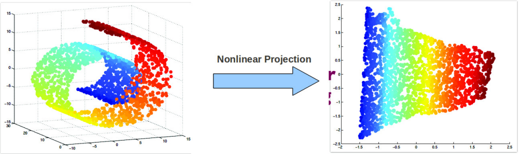
- We want to a learn nonlinear low-dim projection



- Some ways of doing this
  - Nonlinearize a linear dimensionality reduction method. E.g.:
    - Mixture of linear dim-red models like mixture of PPCA (already seen)
    - **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.
  - .. or use unsupervised deep learning techniques (later)

Nonlinear Dimensionality Reduction

- We want to a learn nonlinear low-dim projection



- Some ways of doing this
  - Nonlinearize a linear dimensionality reduction method. E.g.:
    - Mixture of linear dim-red models like mixture of PPCA (already seen)
    - **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.
  - .. or use unsupervised deep learning techniques (later)
- Today, we will briefly look at KPCA, LLE, SNE/tSNE

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

- PCA computes eigenvectors $\boldsymbol{u}_i$ which satisfy $\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ $\forall i = 1, \ldots, D$

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

- PCA computes eigenvectors $\boldsymbol{u}_i$ which satisfy $\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ $\forall i = 1, \ldots, D$
- Let's assume a kernel k with associated M dimensional nonlinear map ϕ

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

- PCA computes eigenvectors $\boldsymbol{u}_i$ which satisfy $\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ $\forall i = 1, \ldots, D$
- Let's assume a kernel k with associated M dimensional nonlinear map ϕ
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\boldsymbol{x}_n) = \mathbf{0}$)

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

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

- PCA computes eigenvectors $\boldsymbol{u}_i$ which satisfy $\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ $\forall i = 1, \ldots, D$
- Let's assume a kernel k with associated M dimensional nonlinear map ϕ
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\boldsymbol{x}_n) = \mathbf{0}$)

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

- Kernel PCA: Compute eigenvectors $\boldsymbol{v}_i$ satisfying: $\mathbf{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ $\forall i = 1, \ldots, M$

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

- PCA computes eigenvectors $\boldsymbol{u}_i$ which satisfy $\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ $\forall i = 1, \ldots, D$
- Let's assume a kernel k with associated M dimensional nonlinear map ϕ
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\boldsymbol{x}_n) = \mathbf{0}$)

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

- Kernel PCA: Compute eigenvectors $\boldsymbol{v}_i$ satisfying: $\mathbf{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ $\forall i = 1, \ldots, M$
- We would like to do this without having to compute $\mathbf{C}$ or $\phi(\boldsymbol{x}_n)$'s (note: M can be very large)

Kernel PCA

- Recall PCA: Given N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\forall \boldsymbol{x}_n \in \mathbb{R}^D$, we define the $D \times D$ covariance matrix (assuming centered data $\sum_n \boldsymbol{x}_n = \mathbf{0}$)

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

- PCA computes eigenvectors $\boldsymbol{u}_i$ which satisfy $\mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$ $\forall i = 1, \ldots, D$
- Let's assume a kernel k with associated M dimensional nonlinear map ϕ
- $M \times M$ covariance matrix **in this space** (assume centered data $\sum_n \phi(\boldsymbol{x}_n) = \mathbf{0}$)

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

- Kernel PCA: Compute eigenvectors $\boldsymbol{v}_i$ satisfying: $\mathbf{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ $\forall i = 1, \ldots, M$
- We would like to do this without having to compute $\mathbf{C}$ or $\phi(\boldsymbol{x}_n)$'s (note: M can be very large)
- This boils down to doing eigendecomposition of the $N \times N$ kernel matrix $\mathbf{K}$ (PRML 12.3)

Locally Linear Embedding (LLE)

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too

Locally Linear Embedding (LLE)

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too
- Given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, LLE is typically formulated as

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

Locally Linear Embedding (LLE)

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too
- Given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, LLE is typically formulated as

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

where $\mathcal{N}(i)$ denotes the set of nearest (say K) neighbors of $\boldsymbol{x}_i$ in the original D-dim space

Locally Linear Embedding (LLE)

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too
- Given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, LLE is typically formulated as

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

where $\mathcal{N}(i)$ denotes the set of nearest (say K) neighbors of $\boldsymbol{x}_i$ in the original D-dim space

- LLE learns $\boldsymbol{z}_n, \ldots, \boldsymbol{z}_N$ such that the same neighborhood structure exists in low-dim space too

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

Locally Linear Embedding (LLE)

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too
- Given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, LLE is typically formulated as

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

where $\mathcal{N}(i)$ denotes the set of nearest (say K) neighbors of $\boldsymbol{x}_i$ in the original D-dim space

- LLE learns $\boldsymbol{z}_n, \ldots, \boldsymbol{z}_N$ such that the same neighborhood structure exists in low-dim space too

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

- Basically, if point i can be reconstructed from its neighbors in the original space, the same weights W_{ij} should be able to reconstruct it in the new space too.

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

- The p's denotes probabilities in original space, the q's denote prob. in embedded space

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

- The p's denotes probabilities in original space, the q's denote prob. in embedded space
- SNE learns $\boldsymbol{z}_i$'s such that distribution P and Q is as close as possible by minimizing $KL(P||Q)$

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

- The p's denotes probabilities in original space, the q's denote prob. in embedded space
- SNE learns $\boldsymbol{z}_i$'s such that distribution P and Q is as close as possible by minimizing $KL(P||Q)$
- t-SNE (van der Maaten and Hinton, 2008) offers a couple of improvements to original SNE

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

- The p's denotes probabilities in original space, the q's denote prob. in embedded space
- SNE learns $\boldsymbol{z}_i$'s such that distribution P and Q is as close as possible by minimizing $KL(P||Q)$
- t-SNE (van der Maaten and Hinton, 2008) offers a couple of improvements to original SNE
  - Learns $\boldsymbol{z}_i$'s by minimizing symmetric KL divergence

SNE and t-SNE

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

- The p's denotes probabilities in original space, the q's denote prob. in embedded space
- SNE learns $\boldsymbol{z}_i$'s such that distribution P and Q is as close as possible by minimizing $KL(P||Q)$
- t-SNE (van der Maaten and Hinton, 2008) offers a couple of improvements to original 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

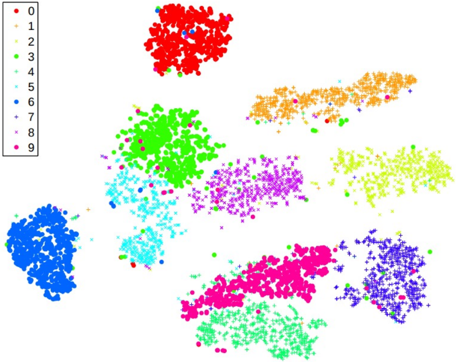
- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D (thus for visualization)
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point i, define the probability of point j being its neighbor as

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

- The p's denotes probabilities in original space, the q's denote prob. in embedded space
- SNE learns $\boldsymbol{z}_i$'s such that distribution P and Q is as close as possible by minimizing $KL(P||Q)$
- t-SNE (van der Maaten and Hinton, 2008) offers a couple of improvements to original 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)