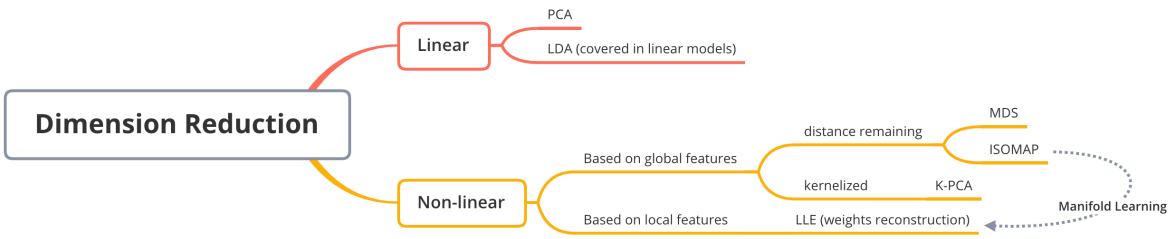
Dimension Reduction

In high dimension, we often have the issue called curse of dimensionality bacause samples are sparse or it's hard to calculate distances. Dimension reduction can not only used to solve this problem, but also extract features.



Note: Linear methods apply linear transformation on the original dataset. That is to say, $Z = W^T X$ where W is the transformation matrix which is orthogonal and X is p x n dimension.

Principal Component Analysis (PCA)

The idea of PCA is to find some principal components to represent original samples. How to find these components? Basically, we expect these principal components that:

- **Pass the closest to a cloud of samples**, in terms of squared Euclidean distance.
- The projection onto them is **the ones with high variances**.

Both goals could derive the consistent solution of PCA.

1. Objective Function & Solution

- Goal 1: Pass the closest to a cloud of samples

  As for this goal, we hope that the restructed sample is close to the original sample as possible. We know that the restructed sample could be represented as $\hat{x}_i = \Sigma_{j=1}^{d'} \mathbf{w_j} z_{ij}$. Therefore this goal would be written as

  $$\Sigma_{i=1}^{m} (\Sigma_{j=1}^{d'} \mathbf{w_j} z_{ij} - x_i)^2 \propto -tr(W^T X X^T W)$$

  Thus, the objective function is

  $$min: -tr(W^T X X^T W)$$
  $$s.t.: W^T W = I$$

- Goal 2: The projection onto them is the ones with high variances

  The projection of sample x_i onto the principal components is $z_i = W^T x_i$ and the variance of samples after projection is $\Sigma_{i=1} W^T x_i x_i^T W$ (centerized samples). Therefore, the objective function is

  $$max: tr(W^T X X^T W)$$
  $$s.t.: W^T W = I$$

The solution of goal 1 and 2 are equivalent. Then we could apply Lagrange multiplier on the objective function:

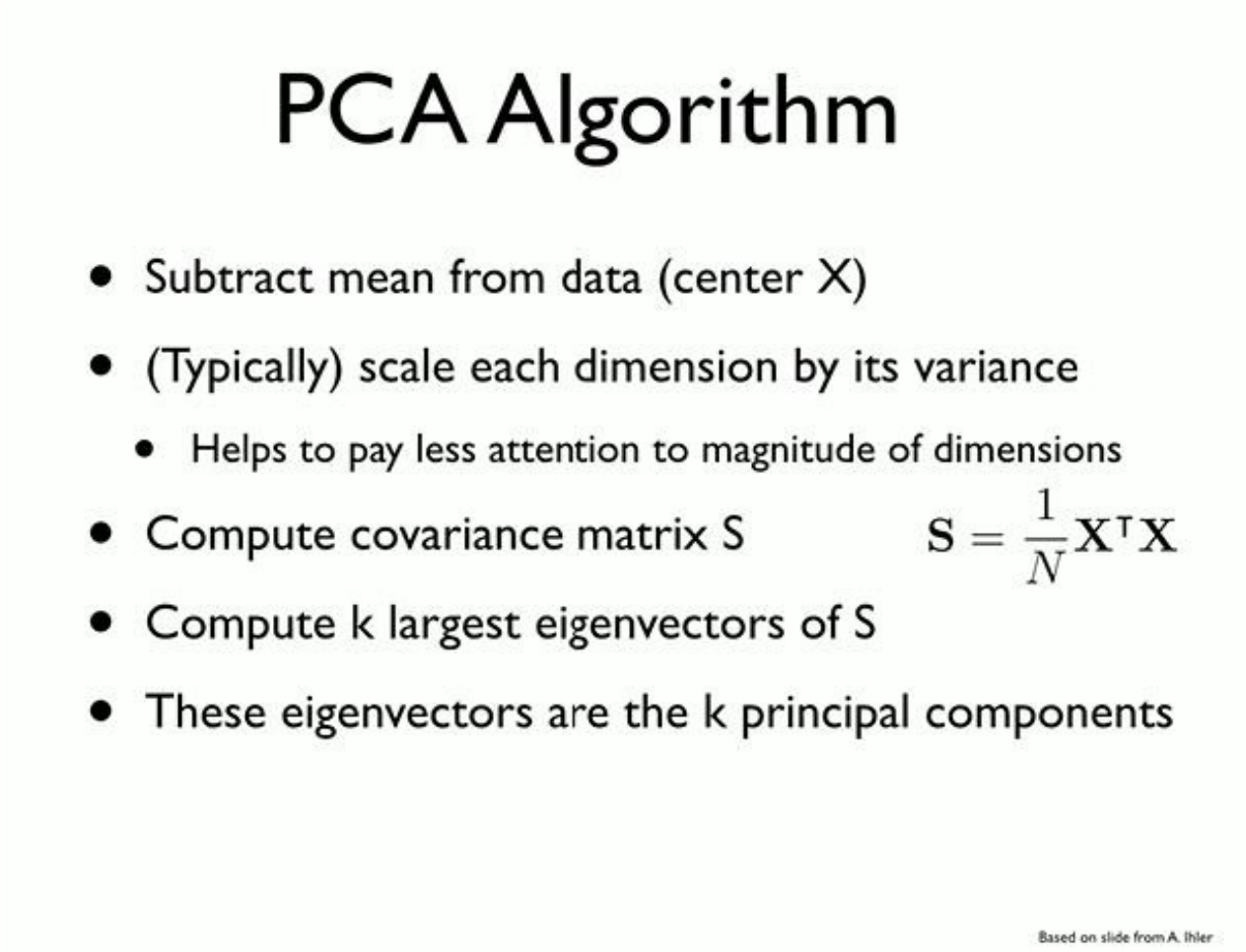
$$L = -tr(W^T X X^T) + \lambda(W^T W - I)$$

Then we could take the derivative of L over w_i and make it to be 0. And we'll get the solution

$$XX^T w_i = \lambda_i w_i$$

This is exactly the form of eigenvectors and eigenvalues.

2. Algorithm



3. Additionals

- Data used in PCA should be centerized first because when we take it as an assumption in derivation.
- How to get eigenvectors and eigenvalues of XX^T ?

  Of course, eigen decomposition is the right answer. However, in practise, we usually use **SVD (singular value decomposition)** on X because it could decompose arbitrary matrix (does not necessarily to be square).
- w_i stands is the principal component vector and λ_i is actually the variance explained by the principal component. This could help us choose the best d'.

Kernelized PCA

Kernel trick has been covered in SVM part. The main idea of KPCA is to project samples onto higher dimension first and then apply PCA to reduce dimension linearly.

KPCA is similar to PCA. The solution for KPCA has a similar form to PCA:

$$Kw_i = \lambda_i w_i$$

In summary, these are the steps for the implementation of the kernel PCA algorithm:

- Choose a kernel function and then obtain K based on original samples.
- Solve eigenvalues and eigenvectors of K.
- For each given data point $\mathbf{x}$, get its j^{th} principal compenents in the feature space $z_j = \Sigma_{i=1}^{m} w_i^j k(x_i, x)$.

Multiple Dimensional Scaling (MDS)

The idea of MDS is to keep the distance between samples unchanged after dimension reduction: $dist(z_i, z_j) = dist(x_i, x_j)$. Below is the steps of MDS.

- Define $B = Z^T Z$ is the inner product matrix after dimension reduction.

  $$b_{ij} = -\frac{1}{2}(dist_{ij}^2 - dist_{i\cdot}^2 - dist_{\cdot j}^2 + dist_{\cdot\cdot}^2)$$

  where

  $$dist_{i\cdot}^2 = \frac{1}{N}\Sigma_{j=1}^{N} dist_{ij}$$
  $$dist_{\cdot j}^2 = \frac{1}{N}\Sigma_{i=1}^{N} dist_{ij}$$
  $$dist_{\cdot\cdot}^2 = \frac{1}{N^2}\Sigma_{i=1}^{N}\Sigma_{j=1}^{N} dist_{ij}$$

- Solve eigenvalues Λ_{d*d} and eigenvectors V_{N*d} of B.
- Get the biggest d' eigenvalues and its corresponding eigenvectors.
- $V\Lambda^{\frac{1}{2}}$ is the new coordinates of samples after dimension reduction.

Isometric Mapping: Isomap

Isomap is actually a variation of MDS. The idea of Isomap is to keep distances between near samples after dimension reduction unchanged. The main difference between Isomap and MDS is the way of computing distances. MDS uses Euclidean distance while Isomap uses geodesic distances.

1. Assumption

The main assumption of Isomap is that our dataset comes from a manifold. In mathematics, a **manifold** is a topological space that locally resembles Euclidean space near each point.

2. Algorithm

- For each data point, find its k nearest neighbours and then set the distances between it and its neighbours to be Euclidean distance and others to be infinity.
- Use the shortest path algorithm, such Dijkstra's algorithm or Floyd algorithm to get the distance matrix between arbitrary two points.

- Take the distance matrix as the input to MDS and then we would get the new coordinates of samples after dimension reduction.

Note: The final result of Isomap is the new coordinates of samples. If we want to get the lower dimension coordinates of new sample, we could fix a regression model of coordinates in high dimension and coordinates in low dimension.

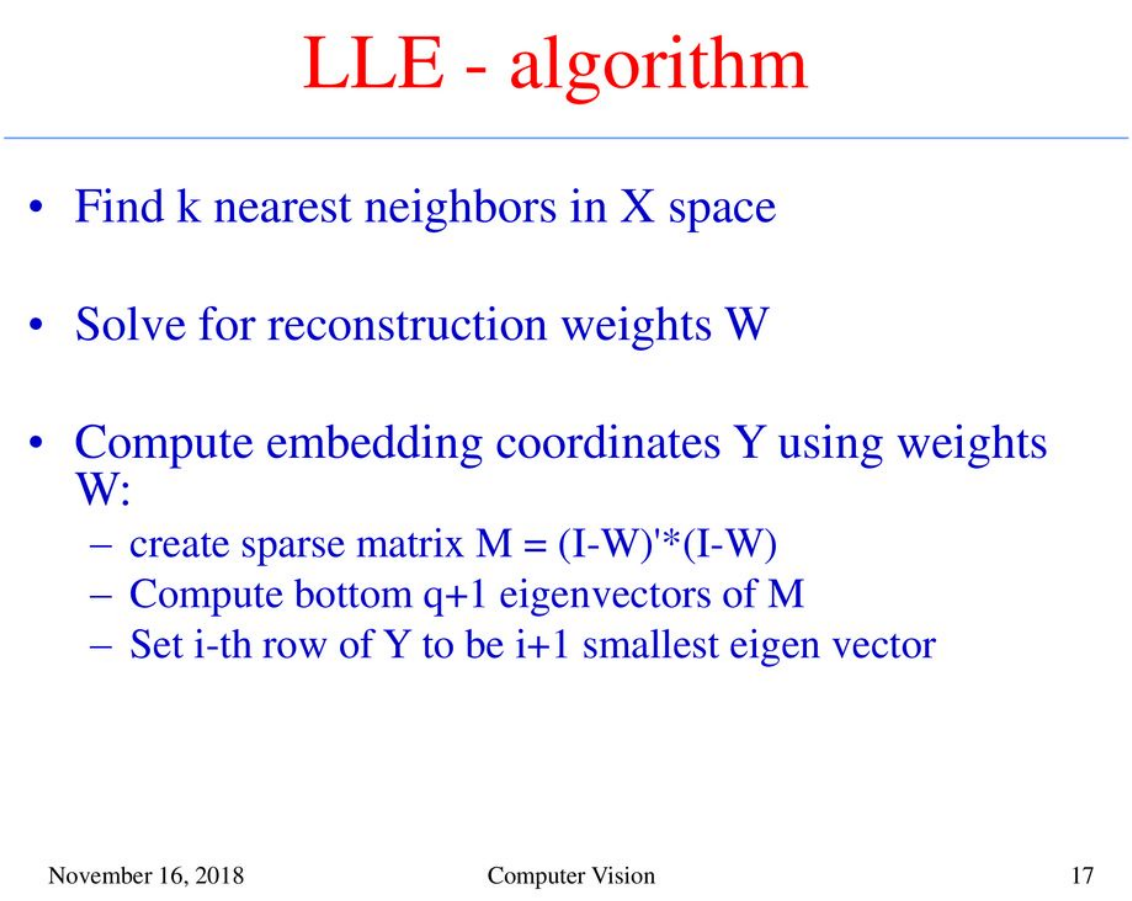
Locally Linear Embedding: LLE

Different from Isomap, LLE tries to keep linear relationship between near samples unchanged after dimension reduction.

1. Assumption

The main assumption of LLE is that near samples do have **linear** relationships.

2. Algorithm



- How to solve for reconstruction weights W ?

  For $i = 1$ to N:

  - create matrix Z consisting of all neighbours of X_i
  - subtract X_i from every column of Z
  - compute the local covariance $C = Z^T Z$
  - solve linear system $CW = 1$ for W
  - set $W_{ij} = 0$ if j is not a neighbor of i
  - set the remaining elements in the i^{th} row of W equal to $W/sum(W)$