Principal Component Analysis

Given \( x^{(1)} \ldots x^{(m)} \in \mathbb{R}^n \)

Goal: reduce to lower dim. data \((k \ll n)\)

what you care about is this axis \( \rightarrow \) principal axis along which the data varies.

Reduce from 2D to 1D \( \rightarrow \) remove noise

Pre-processing:

Set \( \mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \)

Replace \( x^{(i)} \) with \( x^{(i)} - \mu \)

Set \( s_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2 \)

Replace \( x^{(i)} \) with \( \frac{x^{(i)}}{s_j} \)

Find direction \( u \), s.t. when I project the data to it, the projection varies widely.

If \( ||u|| = 1 \), \( x^{(i)} \) projected on \( u \) has length \( x^{(i)T} u \)

Choose \( u \) to maximize:

\[
\max_{u: ||u|| = 1} \sum_{i=1}^{m} (x^{(i)T} u)^2
\]

\[
= \frac{1}{m} \sum_{i=1}^{m} (u^T x^{(i)})(x^{(i)T} u) = u^T \left[ \frac{1}{m} \sum_{i=1}^{m} x^{(i)} x^{(i)T} \right] u
\]
$u$ is the principal eigenvector of $\Sigma = \text{cov. matrix}$

$$\frac{1}{m} \sum_{i=1}^{m} x(i)x(i)^T$$

Eigenvalue

$$Au = \lambda u \quad \text{for eigenvector}$$

largest eigenvalue

$$\max_{u \in \mathbb{R}^n} u^T \Sigma u \text{ s.t. } u^T u = 1$$

Lagrange

$$L(u, \lambda) = u^T \Sigma u - \lambda (u^T u - 1)$$

$$\frac{\partial L(u, \lambda)}{\partial u} = \Sigma u - \lambda u = 0$$

$$\Sigma u = \lambda u.$$ 

$u$ = principal eigenvector of $\Sigma$

If we want $k$-dim subspace,

choose $u_1, \ldots, u_k$ to be $k$ top eigenvector of $\Sigma$

$\lambda_k$ correspond to $k$ highest eigenvalues.

$U_1, \ldots, U_k$: new basis of representing data.

at beginning we have $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}^n$

New representation:

$$y^{(i)} = (u_1^T x^{(i)}, u_2^T x^{(i)}, \ldots, u_k^T x^{(i)}) \quad y^{(i)} \in \mathbb{R}^k$$

another view of PCA.

minimize sum of squared distance from the pt of its projection.
Use of PCA
- visualization of high-dim data
- compression: work with lower dim data instead of high-dim data
- learning: more features → more complex → overfitting
  Use PCA to reduce dimensionality of features
- Anomaly detection: pts far away from your subspace

Matching
- Distance calculation.
  Data points often lie in a subspace — the rest of the space might be noise.

In original space
Similarity = \( \cos \Theta \)

Latent Semantic Indexing (LSI)
- does not have the preprocessing step

Problem of PCA
- Covariance matrix computation.
  When your data
  \( 100 \times 100 \) pixel image \( \times 10,000 \in \mathbb{R}^{10,000} \)
  \( \Sigma \)

Summary PCA
- Preprocess \( \Sigma \)
- Cov. matrix compute
- Find top \( k \) eigenvectors of \( \Sigma \)
Implement PCA using SVD—singular value decomposition

Say we have any $A \in \mathbb{R}^{m \times n}$

decompose $A = UDV^T$

$m \times n$ $m \times m$ $n \times n$ $n \times m$

$D = \text{diagonal} = \begin{bmatrix} \xi_1 & \xi_2 & \cdots & 0 \\ 0 & \xi_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \xi_n \end{bmatrix}$  $\xi_i = \text{singular values of matrix } A$

$A = [U]_{m \times n} \begin{bmatrix} D \\ \end{bmatrix}_{n \times m} [V^T]_{n \times n}$

(Using svd command in matlab).

svd = $O(n^3)$ not sure?

$U$'s columns: eigenvectors of $AA^T$

$V$'s columns: eigenvectors of $A^TA$

Can be used to compute eigenvector of PCA efficiently.

$\Sigma = \sum_{i=1}^{n} x(i) \cdot x(i)^T$

given Design matrix

$X = \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(m) \end{bmatrix}$

$\Sigma = X^TX = \begin{bmatrix} x(1)^T \\ x(2)^T \\ \vdots \\ x(m)^T \end{bmatrix}$

To get top $k$ eigenvector of $\Sigma$,

$X = UDV^T$, top $k$ columns of $V$ are the top $k$ eigenvectors of $X^TX = \Sigma$.
Spectral clustering.

data

affinity matrix

\[ W_{nm} = e^{-\frac{\|s_n - s_m\|^2}{\sigma^2}} \]

\[ \Rightarrow A = \text{func}(W) \]

disconnected graph

\[
\begin{align*}
\delta_i &= \sum_j w_{ij} = \text{degree of vertex } i, \\
D &= \text{diag}(d_1, d_2, \ldots, d_n) \text{ degree matrix} \\
|A| &= \# \text{ of vertices in } A \\
\text{vol}(A) &= \sum_{i \in A} d_i
\end{align*}
\]

\[ \text{Cut}(A, B) := \sum_{i \in A, j \in B} W_{ij} \]

Balanced min-cut

\[ \min \text{ cut}(A, B) \text{ s.t. } |A| = |B| \]

Ratio cut

\[ \frac{\text{cut}(A, B)}{|A| + |B|} \]

Normalized cut

\[ \frac{\text{cut}(A, B)}{|\text{vol}(A)| + |\text{vol}(B)|} \]

\[ \min \text{ cut}(A, B) = \min \frac{1}{2} f^T (D-W) f \quad \text{s.t. } f^T 1 = 0 \quad f \in \mathbb{R}^n \]

L = D - W \quad \text{un-normalized graph Laplacian}

\[ \min \frac{f^T L f}{f^T f} \quad \text{s.t. } f^T 1 = 0 \]

\[ = \min \frac{f^T L f}{f^T f} \Rightarrow \lambda = \text{smallest eigenvalue of } L \]
Another paper has used $P = D^{−1}W$ ("normalized" affinity matrix) instead of $L$; and take the eigenvector corresponding to the largest eigenvalue instead of the smallest. The two algorithms, the one using $L$ matrix and the one using $P$ matrix can be shown to be similar:

\[ (D - W)r = \mu Dr \]
\[ D^{-1}(D - W)r = \mu r \]
\[ D^{-1}(W)r = (1-\mu)r \]
\[ L = 1-\mu \]
\[ \lambda = 1-\mu \]
\[ v = r \]

Equality between using $L$ and $P$ matrix: i.e.

$Lv = \lambda v \rightarrow$ take smallest eigenvalue

$Pv = (1-\mu)v \rightarrow$ take largest eigenvalue