Chapter 4
Feature Extraction

supplementary slides to
*Machine Learning Fundamentals*
©*Hui Jiang* 2020
published by Cambridge University Press

August 2020
Outline

1. Feature Extraction: Concepts
2. Linear Dimension Reduction
4. Nonlinear Dimension Reduction (II): Neural Networks
Feature Extraction

- feature engineering
  - use domain knowledge to manually extract features from raw data
  - e.g. bag-of-words for text, MFC features for speech/audio, SIFT features for image/video

- feature normalization:
  - normalize each dimension towards zero-mean and unit-variance

- feature selection
  - select a subset of the most informative features and discard the rest
  - e.g. filter, wrapper or embedded method

- dimensionality reduction

Figure: represent a text document as a fixed-size bag-of-words feature vector.
Dimensionality Reduction

- utilize a mapping function to convert high-dimensional feature vectors to lower-dimensional ones while retaining the information as much as possible
- the function $f(\cdot)$ maps any point in an $n$-dimensional space to a point in an $m$-dimensional space, where $m \ll n$
- choices for $f(\cdot)$:
  - linear transformation
  - piece-wise linear functions
  - nonlinear functions
  - neural networks
- the learning criterion: what to retain

$y = f(x)$

$x \in \mathbb{R}^n$

$y \in \mathbb{R}^m$
Linear Dimension Reduction

- use a linear mapping function

\[ y = f(x) = A x \]

where \( A \in \mathbb{R}^{m \times n} \) denotes all parameters to be estimated

- depending on the learning criterion:
  - principal component analysis (PCA)
  - linear discriminant analysis (LDA)
Principal Component Analysis (PCA)

- more information \iff larger variance
- PCA aims to search for some orthogonal projection directions to achieve the maximum variances

\[
\{ x_1, x_2, \ldots, x_N \} \xrightarrow{v = w^T x} \{ v_1, v_2, \ldots, v_N \}
\]

\[
\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (v_i - \bar{v})(v_i - \bar{v})
\]

\[
= w^T \left[ \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})^T \right] w
\]

\[
S : \text{sample covariance matrix}
\]

\[
v = x \cdot w = w^T x
\]
Principal Component Analysis (PCA)

- the principal components can be derived:
  \[
  \hat{w} = \arg \max_w w^\top S w
  \]
  subject to
  \[
  w^\top w = 1
  \]
- the method of Lagrange multipliers leads to a closed-form solution:
  \[
  S \hat{w} = \lambda \hat{w}
  \]
  where each principal component \( \hat{w} \) is an eigenvector of \( S \)
- the projection variance equals to the corresponding eigenvalue:
  \[
  \sigma^2 = \hat{w}^\top S \hat{w} = \hat{w}^\top \lambda \hat{w} = \lambda \cdot \| \hat{w} \|^2 = \lambda
  \]
1. Compute the sample covariance matrix \( S \) from training data.

2. Calculate the top \( m \) eigenvectors of \( S \).

3. Form \( A \in \mathbb{R}^{m \times n} \) with an eigenvector in a row:

\[
A = \begin{bmatrix}
- & \hat{w}_1^\top & - \\
- & \hat{w}_2^\top & - \\
& \vdots & \\
- & \hat{w}_m^\top & - 
\end{bmatrix}_{m \times n}
\]

4. For any \( x \in \mathbb{R}^n \), map it to \( y \in \mathbb{R}^m \) as \( y = Ax \).

A few top eigenvalues usually dominate the total variance in PCA.

Figure: the distribution of all eigenvalues in PCA.
Inverse PCA Transformation

- **full PCA without truncation** ($m = n$):
  \[
  \tilde{x} = A^T y = A^T A x = x
  \]

- **truncated PCA** ($m < n$):
  1. method 1: $\tilde{x} = A^T y \neq x$
  2. method 2: $\tilde{x} = A^T y + (I - A^T A) \bar{x} \neq x$

---

**original**  $m = 2$  $m = 50$  $m = 100$  $m = 300$

![Images of digit 8 for different values of m]
Linear Discriminant Analysis (I)

- class labels are known
- how to linearly project data in order to maximize class separation
- Fisher’s linear discriminant analysis (LDA) aims to maximize the ratio:

\[
\hat{w} = \arg \max_w \frac{w^\top S_b w}{w^\top S_w w}
\]

- between-class scatter matrix
  \[
  S_b = \sum_{k=1}^{K} |C_k| (\mu_k - \mu)(\mu_k - \mu)^\top
  \]
- within-class scatter matrix
  \[
  S_w = \sum_{k=1}^{K} S_k
  \]

PCA does not always maximize class separation

Figure: LDA vs. PCA
Fisher’s LDA is equivalent to the following constrained optimization:

\[ w^* = \arg \max_w \ w^T S_b w \]

subject to

\[ w^T S_w w = 1 \]

LDA projections correspond to the eigenvectors of \( S_w^{-1} S_b \)

LDA has at most \( K - 1 \) projection directions

**Figure**: PCA vs. LDA projections in 2-dimensional spaces
Nonlinear Dimension Reduction (I): Manifold Learning

- **manifolds**: nonlinear topological structures in a lower-dimensional space
- **manifold learning**: identify low-dimensional manifolds using some non-parametric approaches
  - locally linear embedding (LLE)
  - multidimensional scaling (MDS)
  - stochastic neighborhood embedding (SNE)
Locally Linear Embedding (LLE)

- **assumption 1**: a locally linear structure in the high-D space $x_i \approx \sum_{j \in N_i} w_{ij} x_j$
- all pair-wise weights can be derived:

$$\{\hat{w}_{ij}\} = \arg \min_{\{w_{ij}\}} \sum_i \|x_i - \sum_{j \in N_i} w_{ij} x_j\|^2$$

subject to $\sum_j w_{ij} = 1 \ (\forall i)$

- **assumption 2**: the same locally linear structure is applied to the low-D space

$$\{\hat{y}_i\} = \arg \min_{\{y_i\}} \sum_i \|y_i - \sum_{j \in N_i} \hat{w}_{ij} y_j\|^2$$

- closed-form solutions exist for LLE
Multidimensional Scaling (MDS)

- preserve all pair-wise distances when projecting from high-D to low-D
- all pair-wise distances in high-D:
  \[ d_{ij} = \| x_i - x_j \| \quad (\forall i, j) \]
- MDS computes all low-D projections:
  \[
  \{ \hat{y}_i \} = \arg \min_{\{ y_i \}} \sum_i \sum_{j > i} \left( \frac{\| y_i - y_j \| - d_{ij}}{d_{ij}} \right)^2
  \]
- isometric feature mapping (Isomap)
  - only compute pairwise distances for nearby vectors in high-D
  - form a sparse graph in the high-D space

Figure: Isomap uses the shortest path in the weighted graph for \( d_{ij} \)
Stochastic Neighborhood Embedding (SNE)

- define a conditional probability distribution in high-D:
  \[ p_{ij} = \frac{\exp\left(-\gamma_i \|x_i - x_j\|^2\right)}{\sum_k \exp\left(-\gamma_i \|x_i - x_k\|^2\right)} \quad (\forall i, j \ i \neq j) \]

- similarly define a conditional probability distribution in low-D:
  - SNE (stochastic neighbor embedding):
    \[ q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_k \exp\left(-\|y_i - y_k\|^2\right)} \quad (\forall i, j) \]
  - t-SNE (t-distributed stochastic neighbor embedding):
    \[ q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \|y_i - y_k\|^2\right)^{-1}} \quad (\forall i, j) \]

- low-D projections are derived by
  \[ \{\hat{y}_i\} = \arg\min_{\{y_i\}} \sum_i \text{KL}(P_i \| Q_i) = \arg\min_{\{y_i\}} \sum_i \sum_j p_{ij} \ln \frac{p_{ij}}{q_{ij}} \]
Nonlinear Dimension Reduction (II): Neural Networks

- Use neural networks as parametric models for the nonlinear mapping function $y = f(x)$ in dimensionality reduction.

- **Autoencoder**
  - Unsupervised: does not require class labels
  - Nonlinear extension of PCA

- **Bottleneck features**
  - Supervised: requires class labels
  - Nonlinear extension of LDA
Autoencoder

neural networks are learned to minimize the difference between inputs and outputs: $\|\hat{x} - x\|^2$
Bottleneck Features

neural networks are learned to project inputs $\mathbf{x}$ to any given class labels

$y = f(\mathbf{x})$