Algorithms in Nature

Dimensionality Reduction

Slides adapted from Tom Mitchell and Aarti Singh
High-dimensional data
(i.e. lots of features)

Document classification:
Billions of documents x Thousands/Millions of words/bigrams matrix

Recommendation systems:
480,189 users x 17,770 movies matrix

Clustering gene expression profiles:
10,000 genes x 1,000 conditions
Curse of dimensionality

Why might many features be bad?

• Harder to interpret and visualize
  • provides little intuition of the underlying structure of the data
• Harder to store data and learn complex models
  • statistically and computationally challenging to classify
  • dealing with redundant features and noise
• Possibly worse generalization
Two types of dimensionality reductions

Feature selection: only a few features are relevant to the task

Latent features: a (linear) combination of features provides a more efficient representation than the observed features (e.g. PCA)

For example, topics (sports, politics, economics) instead of individual documents
Facial recognition

Say we wanted to build a human facial recognition system.

**Option 1**: enumerate all 6 billion faces, update as necessary.

**Option 2**: learn a low-dimensional basis that can be used to represent *any* face (PCA: Today)

**Option 3**: learn the basis using insights from how the brain does it (NMF: Wednesday)

(high-dimensionality space of possible human faces)
Principal Component Analysis

A dimensionality reduction technique similar to auto-encoding neural networks:

Learn a *linear* representation of the input data that can best reconstruct it.

Hidden layer: a *compressed* representation of the input data. Think of compression as a form of pattern recognition.
Principal Components Analysis

\[ \text{face}_i = \sum_k c_{ik} \text{eigenface}_k \]
Face reconstruction using PCA

Reconstruction using the first 25 components (eigenfaces), one at a time

1  2  ...  25

Same, but adding 8 PCA components at each step

In general: top k dimensions are the k-dimensional representation that minimizes reconstruction (sum of squared) error.
Principal Component Analysis

Given data points in d-dimensional space, project them onto a lower dimensional space while preserving as much information as possible.

- e.g. find best planar approx to 3D data
- e.g. find best planar approx to $10^4$D data

Principal components are orthogonal directions that capture variance in the data:

1st PC: direction of greatest variability in the data
2nd PC: next orthogonal (uncorrelated) direction of greatest variability: remove variability in the first direction, then find the next direction of greatest variability.

Etc.

Projection of data point $x_i$ (a d-dim vector) onto 1st PC $v$ is $v^T x_i$
PCA: find projections to minimize reconstruction error

Assume data is a set of d-dimensional vectors, where n\textsuperscript{th} vector is:

\[ x^n = < x_1^n, \ldots, x_d^n > \]

We can represent these in terms of any d orthogonal vectors \( u_1, \ldots, u_d \):

\[ x^n = \sum_{i=1}^{d} z_i^n u_i \]

Goal: given M<d, find \( u_1, \ldots, u_M \) that minimizes:

\[ E_M = \sum_{i=1}^{N} ||x^n - \hat{x}^n||^2 \]

where \( \hat{x}^n = \bar{x} + \sum_{i=1}^{M} z_i^n u_i \)

origin is mean-centered coefficient/weight of projection
PCA

Idea: zero reconstruction error if M=d, so all error is due to missing components.

Therefore:  \[ E_M = \sum_{i=M+1}^{d} \sum_{n=1}^{N} [u_i^T (x^n - \bar{x})]^2 \]

\[ = \sum_{i=M+1}^{d} \sum_{n=1}^{N} [u_i^T (x^n - \bar{x})][u_i^T (x^n - \bar{x})] \]

\[ = \sum_{i=M+1}^{d} \sum_{n=1}^{N} [u_i^T (x^n - \bar{x})][(x^n - \bar{x})^T u_i] \]

\[ = \sum_{i=M+1}^{d} u_i^T \Sigma u_i \]

Co-variance matrix  \[ \Sigma_{ij} = \sum_{n=1}^{N} (x_{i}^n - \bar{x}_i)(x_{j}^n - \bar{x}_j)^T \]

Project difference between the original point and the mean onto the basis vector, take the square

Expand and re-arrange

Substitute co-variance matrix

Measures correlation or inter-dependence between two dimensions
PCA contd.

$$E_M = \sum_{i=M+1}^{d} u_i^T \Sigma u_i$$

**Review**: matrix $A$ has eigenvector $u$ with eigenvalue $\lambda$ if: $Au = \lambda u$

$$\rightarrow \Sigma u_i = \lambda_i u_i$$

eigenvalue (scalar)  eigenvector of covariance matrix

$$E_M = \sum_{i=M+1}^{d} \lambda_i$$

The reconstruction error can be exactly computed from the eigenvalues of the covariance matrix.
PCA Algorithm

1. $X \leftarrow \text{Create NXd data matrix with one row vector } x^n \text{ per data point.}$
2. $X \leftarrow \text{subtract mean from each vector } x^n \text{ in } X$
3. $\Sigma \leftarrow \text{compute covariance matrix of } X$
4. Find eigenvectors and eigenvalues of $\Sigma$
5. $\text{PCs} \leftarrow \text{the M eigenvectors with the largest eigenvalues}$

Original representation:
\[ x^n = \langle x^1_n, \ldots, x^d_n \rangle \]

Transformed representation:
\[ \hat{x}^n = \langle u_1^T x^n, \ldots, u_M^T x^n \rangle \]
PCA example

![PCA example](image)
PCA example

Reconstructed data using only first eigenvector (M=1)
PCA weaknesses

- Only allows linear projections
- Co-variance matrix is of size $d \times d$. If $d=10^4$, then $|\Sigma| = 10^8$
  - *Solution*: singular value decomposition (SVD)
- PCA restricts to orthogonal vectors in feature space that minimize reconstruction error
  - *Solution*: independent component analysis (ICA) seeks directions that are statistically independent, often measured using information theory
- Assumes points are multivariate Gaussian
  - *Solution*: Kernel PCA that transforms input data to other spaces
# PCA vs. Neural Networks

<table>
<thead>
<tr>
<th></th>
<th><strong>PCA</strong></th>
<th><strong>Neural Networks</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unsupervised dimensionality reduction</strong></td>
<td>Linear representation that gives best squared error fit</td>
<td>Supervised dimensionality reduction</td>
</tr>
<tr>
<td></td>
<td>No local minima (exact)</td>
<td>Non-linear representation that gives best squared error fit</td>
</tr>
<tr>
<td></td>
<td>Non-iterative</td>
<td>Possible local minima (gradient descent)</td>
</tr>
<tr>
<td></td>
<td>Orthogonal vectors (“eigenfaces”)</td>
<td>Iterative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auto-encoding NN with linear units may not yield orthogonal vectors</td>
</tr>
</tbody>
</table>
Is this really how humans characterize and identify faces?