Feature extraction and selection
Dick de Ridder and Cor Veenman
ASCI A1 - Advanced Pattern Recognition 2008

Feature space
- A \( p \)-dimensional space,
  - in which each dimension is a feature
  - containing \( N \) (labelled) samples (objects)

Questions for today:
- What can we do in this feature space?
- Why and how should we lower the number of features?

Density estimation
- Density estimation is hardest underlying problem
  - Probability density function, \( p(x) \); per class, \( p(x | \omega_i) \)

What is the probability of an object of class A? And an object of class B?

Density estimation (2)
- Approaches:
  - Histograms: need a lot of data
  - Simple models: e.g. a Gaussian
  - Mixture models: e.g. mixtures of Gaussians
  - Non-parametric: e.g. Parzen estimation

Simplest model: Gaussian
\[
p(x) = \frac{1}{\sqrt{2\pi^d \det(G)}} \exp \left( -\frac{1}{2} (x - \mu)^T G^{-1} (x - \mu) \right)
\]

\[G = \begin{bmatrix}
\text{cov}(x_1,x_1) & \text{cov}(x_1,x_2) & \cdots & \text{cov}(x_1,x_d) \\
\text{cov}(x_2,x_1) & \text{cov}(x_2,x_2) & \cdots & \text{cov}(x_2,x_d) \\
\vdots & \vdots & \ddots & \vdots \\
\text{cov}(x_d,x_1) & \text{cov}(x_d,x_2) & \cdots & \text{cov}(x_d,x_d)
\end{bmatrix}
\]
- Rule-of-thumb: need 10 times as many samples as features to estimate parameters
- For example, for 40 \( \times \) 40 pixel face recognition: \( 10 \times (1600 + \frac{1}{2} \times 1600 \times 1599) = 12.8 \times 10^6 \) samples
Curse of dimensionality

- Problem: too few samples in too many dimensions (the curse of dimensionality)

- For starters: in high-dimensional spaces, our 2D/3D intuition does not work anymore...

High-dimensional spaces

- Example: neighbourhood capturing 10% of uniformly distributed data in hypercube

\[ \mathbb{R}^1 \] sides of 0.1\( r \), e.g. in \( \mathbb{R}^2 \): sides of 0.89... not a “small block” anymore!

High-dimensional spaces (2)

- Example: all points are boundary points

  - 1000 \( N(0,1) \) samples in \( \mathbb{R}^2 \): 1% on convex hull
  - 1000 \( N(0,1) \) samples in \( \mathbb{R}^{20} \): 95% on convex hull

High-dimensional spaces (3)

- Example: points tend to have equal distances

  - Distance to mean:
    \[
    \frac{\text{std}(d^2)}{\text{mean}(d^2)}
    \]

  - Distance between samples: squared Euclidean distances of points in \( \mathbb{R}^{1024} \sim N(1024,32\sqrt{2}) \), so distances are all equal within 10%

High-dimensional spaces (4)

- For classification purposes, this means that for increasing dimensionality \( p \):
  - local, distance-based methods suffer most
    e.g. error for NN-methods goes up exponentially with \( p \)
  - global, more restricted models suffer less
    e.g. error for linear models goes up linearly with \( p \)

  - So...
    - controlling classifier complexity is important (later...)
    - \( p \) should be kept as low as possible: dimensionality reduction

Dimensionality reduction

- Problem: too few samples in too many dimensions (the curse of dimensionality)

- Solution: drop dimensions (features)

  - Feature selection
  - Feature extraction

- Question:
  - Which dimensions can safely be dropped?
  - What is the best subset of features to keep?
Dimensionality reduction (2)

- Use of dimensionality reduction:
  1. **Fewer parameters** give faster algorithms and parameters are easier to estimate
     
     Note: the curse of dimensionality means that discarding information may actually improve results!
  2. **Explaining** which measurements are useful and which are not (reducing redundancy)
  3. **Visualisation** of data is a powerful tool when designing pattern recognition systems

Feature selection v extraction

- **Feature selection:** select \( d \) out of \( p \) measurements
- **Feature extraction:** map \( p \) measurements to \( u \) measurements

Feature selection v extraction (2)

<table>
<thead>
<tr>
<th>Selection</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cut in measurements</td>
<td>expensive</td>
</tr>
<tr>
<td></td>
<td>easy interpretation</td>
<td>often approximative</td>
</tr>
<tr>
<td>Extraction</td>
<td>cheap</td>
<td>need all measurements</td>
</tr>
<tr>
<td></td>
<td>can be nonlinear</td>
<td>criterion sub-optimal</td>
</tr>
</tbody>
</table>

Feature selection v extraction (3)

- Think of selection and extraction as finding a mapping \( f \) for every \( x \); for both, we need:
  - a **criterion function**, a model of what makes a good mapping e.g. error, class overlap, information loss
  - a **search algorithm**, a method of finding the mapping given the criterion e.g. pick the best single feature at each time

- Sometimes these can be combined in one algorithm; for feature extraction, they usually are

Criteria

1. The optimal criterion: the final performance of the entire system, calculated using cross-validation
   
   For any practical application, too expensive!

2. Approximate performance predictors:
   calculate the performance of an easy-to-use model, that gives an indication of how well a more powerful model may perform

3. General desired properties:
   decorrelation or local structure preservation, e.g. for visualisation

Feature extraction

- **Unsupervised**
  - \( \{x\} \)
  - Linear

- **Supervised**
  - \( \{x\}, \{y\} \)
  - Linear
  - Nonlinear
**Linear feature extraction**

- Unsupervised:
  - Principal Component Analysis (PCA)
- Supervised:
  - Canonical Correlation Analysis (CCA)
  - Partial Least Squares (PLS)
  - Linear Discriminant Analysis (LDA)

**PCA is the most widely used feature extraction method**

**Principal component analysis**

- Principal component analysis (PCA, 1901):
  - find directions in data...
  - which retain as much variation as possible
  - which make projected data uncorrelated
  - which minimise squared reconstruction error

**Principal component analysis (2)**

- Given a set of measurements $x_1, \ldots, x_p$ (with zero mean, i.e. $\mu = 0$), we look for a linear combination

$$
\xi_j = \sum_{j=1}^p A_{ij} x_j
$$

- $A$ is a matrix of coefficients, which
  - has orthonormal columns
  - maximises the variance of the $\xi_j$'s

**PCA derivation**

- To find the first variable,

$$
\xi_1 = \sum_{j=1}^p A_{1j} x_j = a_1^T x
$$

choose $a_1$ to maximise the variance of $\xi_1$, with the constraint that the length of $a_1$ is 1:

$$
a_1^T a_1 = |a_1|^2 = 1
$$

- $\text{var}(\xi_1) = E[\xi_1^2] - E[\xi_1] E[\xi_1] = a_1^T \Sigma a_1$

where $\Sigma = \text{cov}(x_j, x_k)$

**PCA derivation (2)**

- Maximise $f(a_1) = \text{var}(\xi_1) = a_1^T \Sigma a_1$

subject to the constraint $c(a_1) = a_1^T a_1 - 1 = 0$

- Use Lagrange multiplier: in the optimum,

$$
\nabla f(a_1) - \nu \nabla c(a_1) = 0
$$

$$
\Sigma a_1 - \nu a_1 = 0
$$

- Remember from last week:

solutions to $(\Sigma - \lambda I) e_j = \Sigma e_j - \lambda e_j = 0$

are the eigenvectors $e_j$ and corresponding eigenvalues $\lambda_j$, $j = 1, \ldots, p$

**PCA derivation (3)**

- Which eigenvector should we choose for $\nu$?

$$
\Sigma a_1 - \nu a_1 = 0
$$

$$
\text{var}(\xi_1) - \nu \cdot 1 = 0
$$

$$
\nu = \lambda_1 = \text{largest eigenvalue}
$$

$a_1$ is the first principal component

$a_1$ is the first eigenvector

$a_1$ is the corresponding eigenvector

$a_1$ is the first principal component
PCA derivation (4)
• For the next principal component, maximise
  \[ f(a_2) = \text{var}(\xi_2) = a_2^T \Sigma a_2 \]
subject to the constraints
  \[ |a_2| = 1 \quad : c_1(a_2) = a_2^T a_2 - 1 = 0 \]
  \[ a_2 \perp a_1 \quad : c_2(a_2) = a_2^T \Sigma a_1 = a_2^T a_1 = 0 \]
• Similar derivation shows that the second principal component is the eigenvector corresponding to the second largest eigenvalue
  • Etcetera...

Intrinsic dimensionality
• Number of dimensions “truly used by the data”
• Depends on point of view (neighbourhood size!)

PCA example (2)
• For image data, principal components can also be interpreted...
  most often occurring variations between digits

Intrinsic dimensionality
• Number of dimensions “truly used by the data”
• Depends on point of view (neighbourhood size!)

PCA derivation (6)
• PCA: for \( \xi = A^T x \), to keep \( d \) dimensions, choose
  \[ A = (e_1 \ e_2 \ ... \ e_j) \]
where the \( e_i \) are eigenvectors of the covariance matrix \( \Sigma \), sorted by the accompanying eigenvalues \( \lambda_j > \lambda_j > ... \)
• To choose \( d \), inspect the retained variance, \( \sum_{j=1}^d \lambda_j \)
  or the ratio \( \sum_{j=1}^d \lambda_j / \sum_{j=1}^\infty \lambda_j \)
• Intrinsic dimensionality: \( d \) for which 90%-95% variance is retained

PCA example
• e.g. NIST digits: 2000 samples, \( p = 256 \)

PCA tips & tricks
• Derivation supposes mean of data is zero, so it should be removed: \( x' = (x - \mu) \)
• PCA is sensitive to scaling (e.g. length in cm has a much larger variance than length in m), so it’s best to standardise: \( x' = (x - \mu) / \sigma \)
PCA tips & tricks (2)

- PCA used for reconstructing data:
  \[ x_r = A^T (x - \mu) \]
  since \((AA^T)^{-1} = (AA^T)^T = AA^T\)

PCA conclusions

- Principal component analysis:
  - **global** and **linear** (but can use mixtures, see later)
  - **unsupervised** (but can be performed on average
    per-class covariance matrix: \texttt{km} in PRTools)
  - needs a **lot of data**
    to estimate \(\Sigma\) well!
  - Danger:
    criterion is not necessarily related to the goal; might discard important directions

Independent component analysis (ICA)

- Model: \(\xi = Ax\)
- PCA finds vectors such that projected data is **uncorrelated**:
  \(p(x, y) = 0\)
- ICA finds vectors such that projected data is **independent**:
  \(p(f(x), g(y)) = 0\), \(\forall f(.), g(.)\)
- Related to the Blind Source Separation (BSS) problem:
  unmixing discrete (sound) sources from a number of observed linear combinations

ICA (2)

- Many different algorithms; most are iterative, optimising non-Gaussianity of projected data:
  - Central Limit Theorem: sums of i.i.d. features will approximately be Gaussian distributed...
  - ...therefore, finding a projection which is as non-Gaussian as possible will likely find a single feature

ICA (3)

- FastICA: widely used algorithm
  - Entropy: \(H(\xi) = - \int \xi \log(\xi) d\xi\)
  - Negentropy: \(J(\xi) = H(\xi_{\text{Gauss}}) - H(\xi)\)
    with \(\xi_{\text{Gauss}}\) a Gaussian r.v. with the same covariance matrix as \(\xi\)
  - Approximate negentropy: assume \(x\) zero mean and pre-whitened (by PCA, or \(x' = C^{-\frac{1}{2}}x\)):
    \[ J_G(a) = [E(G(x'a)) - E(G(\nu))]^2 \]
    with \(x' \sim \mathcal{N}(0,1)\) r.v. and \(G\) a contrast function, e.g. \(G(x) = x^4\) (kurtosis!)

ICA (4)

- FastICA iteratively maximises \(J_G(a)\) and discards \(a\)
- Fixed-point:
  - \(J_G(a) = [E(G(a'x)) - E(G(\nu))]^2\)
    is optimal when \(E(G(a'x))\) optimal
  - Constrain \(|a|^2 = 1\): Lagrange multiplier
  - In optimum: \(E(xG'(a'x)) - \beta a = 0\)
- Algorithm:
  - \(a^* = E(xG'(a'x)) - E(G'(a'x)) a\)
  - \(a_{n+1} = a^n / \|a^n\|\)
ICA (5)
• On image data ICA gives more localised basis vectors than PCA; e.g. on 12×12 pixel windows, from natural scenes:

Supervised linear feature extraction
• If a desired output y (or label ω) is present for each x, supervised criteria can be used
• Three illustrations:
  • Partial Least Squares (PLS)
  • Canonical Correlation Analysis (CCA)
  • Linear Discriminant Analysis (LDA)

Partial Least Squares (PLS)
• Reconstruction:
  • {x} and {y} both assumed stochastic, treated equally
• Regression:
  • {y} depends on {x}; only {y} is assumed stochastic

PLS (2)
• Feature extraction for regression:
  • Simplest: Principal Component Regression (PCR)
    • Perform PCA: maximise \( \text{var}(a_1^T x) \)
    • Perform regression in low-dimensional space
  • Partial Least Squares (PLS):
    • Find vector \( a_1 \) for \( x \) that maximises \( \text{cov}(a_1^T x, y) \)

Canonical Correlation Analysis (CCA)
• Find basis vectors \( a_i \) for \( x \) and \( b_i \) for \( y \) such that the projections \( \hat{x} = a_i^T x \) and \( \hat{y} = b_i^T y \) are maximally correlated:

\[
\rho(\hat{x}, \hat{y}) = \frac{\mathbb{E}[\hat{x}\hat{y}]}{\sqrt{\mathbb{E}[\hat{x}^2]\mathbb{E}[\hat{y}^2]}} = \frac{\mathbb{E}[a_i^T x y^T h]}{\sqrt{\mathbb{E}[a_i^T x x^T a_i]\mathbb{E}[h y y^T h]}} = \frac{a_i^T \Sigma_{x y} h}{\sqrt{a_i^T \Sigma_x a_i + h_i^T \Sigma_h h_i}}
\]
• Constraints, algorithm like PCA

Linear discriminant analysis (LDA)
• Also known as Fisher mapping (fisherm)
• Minimise within-class scatter \( S_w = \sum_{i=1}^{C} \frac{N_i}{N} \sum (x_i - m_i)(x_i - m_i)^T \)
• Maximise between-class scatter \( S_b = \sum_{i=1}^{C} \frac{N_i}{N} (m_i - m)(m_i - m)^T \)

\[
\text{Weighted average} = S_w
\]
LDA (2)
• Find basis vector $a_1$ for $\{x\}$ such that in the projections, the classes are maximally separated
• Choose $a_1$ to maximise Fisher criterion:
  $$J_F(a_1) = \frac{a_1^T S_B a_1}{a_1^T S_W a_1}$$
• Solution: eigenanalysis on $S_W^{-1} S_B$
• LDA is equal to CCA if class labels are stored as vectors, e.g. $\omega = 1 \rightarrow y = (1 \ 0 \ ... \ 0)^T$, $\omega = 2 \rightarrow y = (0 \ 1 \ ... \ 0)^T$, etc.

LDA (3)
- Original
- Decorrelated
- Sphered ($S_W^{-1}$)
- PCA on means ($S_B$)
- Restored
- Projected data

LDA (4)
- Map down to a maximum of $c-1$ dimensions (why?)
- Example: NIST digits

LDA (5)
• To avoid fitting noise, can do PCA first
• If system underdetermined ($n \leq p$), first doing PCA is required

Linear feature extraction
• General approach: find eigenvectors $a$ and eigenvalues $\lambda$ of $C^{-1} B a = \lambda a$, with...
  $$B \quad C$$
  $$\begin{pmatrix}
  \Sigma_{xx} & I \\
  \Sigma_{yx} & 0
  \end{pmatrix}$$
  $$\begin{pmatrix}
  \Sigma_{xx} & \Sigma_{yx} \\
  0 & 0
  \end{pmatrix}$$
  $$\begin{pmatrix}
  \Sigma_{xx} & \Sigma_{yx} \\
  0 & 0
  \end{pmatrix}$$
  $$S_W \quad S_B$$

Feature extraction
- Unsupervised
- Supervised
- Linear
- Nonlinear
Nonlinear feature extraction

- Large collection of possible mappings, but not all applicable to all problems
- Usually need an optimisation algorithm
- Here: only unsupervised methods

Kernel PCA (KPCA)

- Use kernel trick, like in support vector classifier
- In effect: add results of nonlinear operations on features as features, and apply standard PCA
- Example: \( K(x,y) = (x^T y + 1)^d \)
- Similarly: kernel LDA, kernel CCA, ...

Topographic mapping

- Potential problem: overfitting
- Solution: first cluster data locally ("code vectors")
- EM-type problem:
  - Cluster membership of each object
  - Manifold fit of each cluster
**Principal curve (PC)**

- Principal component: minimise deviation from line
- Principal curve: minimise deviation from a curve
- Need to assume a family of curves, e.g. B-splines, Bézier-curves, piecewise linear...
- Training: using expectation-maximisation (EM), alternating fitting PDFs and regression by smoothing


**Self-organising map (SOM)**

- Assume clusters lie on curve or grid, and fit this to data (cross between clustering and feature extraction)
- Training: iteratively, pick a random training sample and move the nearest grid point and its neighbours towards it (form of simulated annealing)


**Generative Topographic Mapping**

- GTM: more principled version of SOM, “mixture-of-Gaussians” + explicit mapping


**Mixture of subspaces**

- Mixture model:
  \[ p(x) = \sum_{i=1}^{k} p(x|z_i) \pi_i \]
  \[ p(x|z) = N(A z + \mu, \Psi) \]
  \[ p(z) = N(0, I) \]


**Autoregressive neural network**

- Feedforward neural networks predicting their input
- Bottleneck layer: feature extraction
- Criterion like PCA (reconstruction error)
- Training: like standard NNs (back-propagation, ...)


**Autoregressive neural network (2)**

- With multiple hidden layers: nonlinear feature extraction

---

**Embedding**

- Find new representation directly, such that some properties (e.g., distances between samples) are preserved as well as possible

---

**Isomap**

- Euclidean distance not suitable for preserving topology...
  - construct neighbourhood graph, e.g., connect each object $x$ to its $k$ nearest neighbours
  - calculate distance $D_{ij}$ between $x_i$ and $x_j$ as shortest path over neighbourhood graph
  - perform classical scaling using $D$

---

**Locally linear embedding (LLE)**

- Globally preserve local structure around each sample
  - **Step I**: for each sample $x_i$, find weights $w$ that best reconstruct it linearly from its $k$ neighbours, $x_{n(1)}...x_{n(k)}$
  - Minimise: $E_I(w) = \sum_{i=1}^{N} \sum_{j=1}^{k} w_j (x_i - x_{n(j)})^2$

---

**Locally linear embedding (2)**

- To calculate $w$ for sample $x_i$:
  \[
  E_I^{(i)}(w) = \left| x_i - \sum_{j=1}^{k} w_j x_{n(j)} \right|^2 = \sum_{j=1}^{k} w_j^2 (x_i - x_{n(j)})^2 = w^T Q w
  \]
  with $\sum_{j=1}^{k} w_j = 1$, solution is: $w = Q^{-1} Q^{-1} Q^{-1}$

  where $Q$ is the $k \times k$ local Gram matrix,
  \[
  Q_{ij} = (x_i - x_{n(j)})^T (x_i - x_{n(m)}) \quad \text{or} \quad Q_{ij} = \frac{1}{2} (D_{i,n(j)} + D_{i,n(m)} - D_{n(j),n(m)})
  \]
  (note: often, $Q = Q + \epsilon I$)

---

**Locally linear embedding (3)**

- Result of Step I:
  sparse $N \times N$ matrix $W$, with $W_{i,n(j)} = w_{i,j}^{(i)}$
Locally linear embedding (4)

- **Step II**: find a projection $z_i$ for each sample $x_i$ by minimising

$$
\mathcal{E}_{II}(Z) = \sum_{i=1}^{N} \left( x_i - \sum_{j=1}^{k} w_{ij} z_{(j)} \right)^2
$$

$$
= \text{tr}(Z(I - W)^T(I - W)Z) = \text{tr}(MZ^2)
$$

where $Z$ contains the $z_i$ as its columns

- Constraints: $\frac{1}{N} \sum_{i=1}^{N} z_i = \frac{1}{k} Z 1 = 0$ and $\frac{1}{N} ZZ^T = I$

- Rayleigh-Ritz theorem: solutions are eigenvectors of $M$ corresponding to smallest eigenvalues

- Discard smallest eigenvalue to constrain $\frac{1}{N} \sum_{i=1}^{N} z_i = 0$

Summary

- Feature selection and extraction: useful for visualisation, necessary because of curse of dimensionality

- Feature extraction:
  - linear v nonlinear
  - supervised v unsupervised

- PCA is the most important feature extraction method