

MTTS1

Dimensionality Reduction and Visualization

**Spring 2014
Jaakko Peltonen**

Lecture 3: Linear feature extraction

Feature extraction

- **feature extraction**: (more general) transform the original $\mathbf{x} \in \mathbb{R}^d$ to $\mathbf{z} \in \mathbb{R}^k$ ($k < d$).

Example transformations:

Linear transformations: Principal component analysis (PCA), Linear Discriminant Analysis (LDA), Factor Analysis (FA).

Some *metric learning* methods are similar to linear transformations.

Nonlinear transformations: Self-Organizing Map (SOM), Multidimensional scaling (MDS), manifold embedding methods.

Often based on assuming the data lies on a low-dimensional manifold.

- Today's lecture: linear transformations and kernel-based nonlinear transformations

Principal Component Analysis 1

Definitions:

- PCA finds a low-dimensional linear subspace such that when \mathbf{x} is projected there information loss (here dened as variance) is minimized.
- Finds directions of maximal variance.
- Equivalent to finding eigenvalues and eigenvectors of the covariance matrix.
- Can also be derived probabilistically (see Tipping, Bishop (1999) Mixtures of Probabilistic Principal Component Analyzers. Neural Computation 11: 443-482).

Principal Component Analysis 2

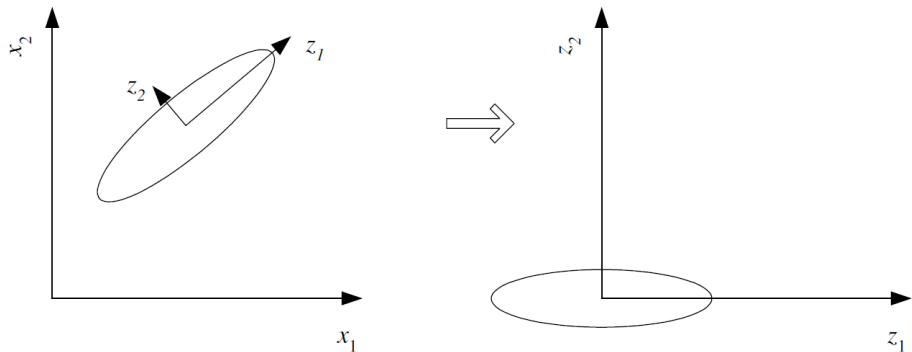


Figure 6.1: Principal components analysis centers the sample and then rotates the axes to line up with the directions of highest variance. If the variance on z_2 is too small, it can be ignored and we have dimensionality reduction from two to one. *From: E. Alpaydm. 2004. Introduction to Machine Learning. Copyright The MIT Press.*

Principal Component Analysis 3

Solution:

- More formally: data $\mathcal{X} = \{\mathbf{x}^t\}_{t=1}^N$, $\mathbf{x}^t \in \mathbb{R}^d$
- Center data: $\mathbf{y}^t = \mathbf{x}^t - \mathbf{m}$, where $\mathbf{m} = \sum_t \mathbf{x}^t / N$
- Compute the covariance matrix $\mathbf{S} = \sum_t \mathbf{y} \mathbf{y}^T / N$
- Diagonalize \mathbf{S} using Spectral Decomposition: $\mathbf{C}^T \mathbf{S} \mathbf{C} = \mathbf{D}$ where
 - \mathbf{C} is an orthogonal (rotation) matrix satisfying $\mathbf{C} \mathbf{C}^T = \mathbf{C}^T \mathbf{C} = \mathbf{1}$ (identity matrix), and
 - \mathbf{D} is a diagonal matrix whose diagonal elements are the eigenvalues $\lambda_1 \geq \dots \geq \lambda_d \geq 0$
- i :th column of \mathbf{C} is the i :th eigenvector.
- Project data vectors \mathbf{y}^t to principal components $\mathbf{z}^t = \mathbf{C}^T \mathbf{y}^t$ (equivalently $\mathbf{y}^t = \mathbf{C} \mathbf{z}^t$).

Principal Component Analysis 4

- Observation: covariance matrix of $\text{fz} \text{tgNt} = 1$ is a diagonal matrix \mathbf{D} whose diagonal elements are the variances.

$$\begin{aligned} \mathbf{S}_z &= \sum_t \mathbf{z} \mathbf{z}^T / N = \sum_t \mathbf{C}^T \mathbf{y} \mathbf{y}^T \mathbf{C} / N \\ &= \mathbf{C}^T \left(\sum_t \mathbf{y} \mathbf{y}^T / N \right) \mathbf{C} = \mathbf{C}^T \mathbf{S} \mathbf{C} = \mathbf{D}, \end{aligned}$$

where the diagonal elements of \mathbf{D} are the variances $\mathbf{D}_{ii} = \sigma_{z_i}^2$

- Eigenvalues $\lambda_i \Leftrightarrow$ variances σ_i^2

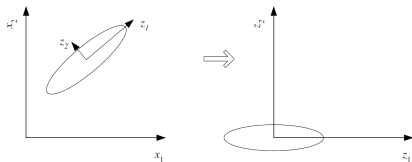


Figure 6.1: Principal components analysis centers the sample and then rotates the axes to line up with the directions of highest variance. If the variance on z_2 is too small, it can be ignored and we have dimensionality reduction from two to one. *From: E. Alpaydm. 2004. Introduction to Machine Learning. Copyright The MIT Press.*

Principal Component Analysis 5

- Idea: in the PC space (\mathbf{z} space), k first principal components explain the data well enough, where $k < d$.
- "Well enough" means here that the **reconstruction error** is small enough. More formally:
- Project the data vectors \mathbf{y}^t into \mathbb{R}^k using $\hat{\mathbf{z}}^t = \mathbf{W}^T \mathbf{y}^t$, where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is a matrix containing the first k columns of \mathbf{C} . ("W = C(:, 1:k)").
- $\hat{\mathbf{z}}^t$ is a representation of \mathbf{y}^t in k dimensions.
- Project $\hat{\mathbf{z}}^t$ back to \mathbf{y}^t space:

$$\hat{\mathbf{y}}^t = \mathbf{W} \hat{\mathbf{z}}^t = \mathbf{W} \mathbf{W}^T \mathbf{y}^t$$

- The average reconstruction error can be shown to be

$$\mathcal{E} = \frac{1}{N} \sum_t \|\mathbf{y}^t - \hat{\mathbf{y}}^t\|^2 = \sum_{i=k+1}^d \lambda_i$$

Principal Component Analysis 6

- Result: PCA is a linear projection of data from \mathbb{R}^d into \mathbb{R}^k such that the average reconstruction error $\mathcal{E} = E \left[\|\mathbf{y} - \hat{\mathbf{y}}\|^2 \right]$ is minimized.
- **Proportion of Variance (PoV) Explained:** $\text{PoV} = \sum_{i=1}^k \lambda_i / \sum_{i=1}^d \lambda_i$
 - Some rules of thumb to find a good k : $\text{PoV} \approx 0.9$ or PoV curve has an "elbow".
- **Dimension reduction:** it may be sufficient to use $\hat{\mathbf{z}}^t$ instead of $\hat{\mathbf{x}}^t$ to train a classifier etc.
- **Visualization:** plotting the data to $\hat{\mathbf{z}}^t$ using $k = 2$ (first thing to do with new data).
- **Data compression:** instead of storing the full data vectors \mathbf{y}_t it may be sufficient to store only $\hat{\mathbf{z}}^t$ and then reconstruct the original data using $\hat{\mathbf{y}}^t = \mathbf{W}\hat{\mathbf{z}}^t$, if necessary.
 - For example, DCT (an approximate of PCA) in JPEG.

Principal Component Analysis 7

Matlab implementation:

- Learning: (suppose samples are columns of X)

```
function [W,m] = pca_learning(X,k)
m = mean(X,2);
[W,~] = eigs(cov(X'), k);
```

- Use: (suppose x_new is a column vector)

```
Z = W' * bsxfun(@minus, X, m);
z_new = W' * (x_new - m);
```

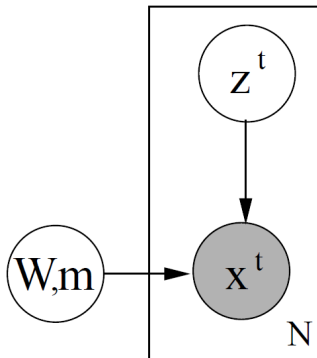
Principal Component Analysis 7

Probabilistic formulation

- Probabilistic model (generative model of the observed data):

$$p(\mathbf{x}^t \mid \mathbf{z}^t, \mathbf{W}, \mathbf{m}, \sigma^2) = N(\mathbf{W}\mathbf{z}^t + \mathbf{m}, \sigma^2 \mathbf{I})$$

- Prior for the hidden variables: $p(\mathbf{z}^t) = N(\mathbf{0}, \mathbf{I})$



Principal Component Analysis 8

Probabilistic formulation

- Probabilistic model (generative model of the observed data):

$$p(\mathbf{x}^t | \mathbf{z}^t, \mathbf{W}, \mathbf{m}, \sigma^2) = N(\mathbf{W}\mathbf{z}^t + \mathbf{m}, \sigma^2 \mathbf{I})$$

- Prior for the hidden variables: $p(\mathbf{z}^t) = N(\mathbf{0}, \mathbf{I})$
- Hidden variables can be marginalized out:

$$\begin{aligned} p(\mathbf{x}^t | \mathbf{W}, \mathbf{m}, \sigma^2) &= \int p(\mathbf{x}^t | \mathbf{z}^t, \mathbf{W}, \mathbf{m}, \sigma^2) p(\mathbf{z}^t) d\mathbf{z}^t \\ &= N(\mathbf{m}, \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}) \end{aligned}$$

Benefits of using the probabilistic formulation:

- a probabilistic way to handle the tailing eigenvalues
- various priors can be applied to \mathbf{W} and \mathbf{m}
- standard algorithms for probabilistic models (e.g. EM)
- provides connection to other probabilistic models

PCA: Toy Example

- Same example used previously for feature selection (forward/backward selection)
- Toy data set consists of 100 10-dimensional vectors from two classes (1 and 0)
- First two dimensions x_1^t and x_2^t : drawn from Gaussian with unit variance and mean of 1 or -1 for the classes 1 and 0 respectively.
- Remaining eight dimensions: drawn from Gaussian with zero mean and unit variance, that is, they contain no information of the class.
- Optimal classifier: if $x_1 + x_2$ is positive the class is 1, otherwise the class is 0.
- Use nearest mean classifier.
- Split data in random into training set of 30+30 items and validation set of 20+20 items

PCA: Toy Example cont.

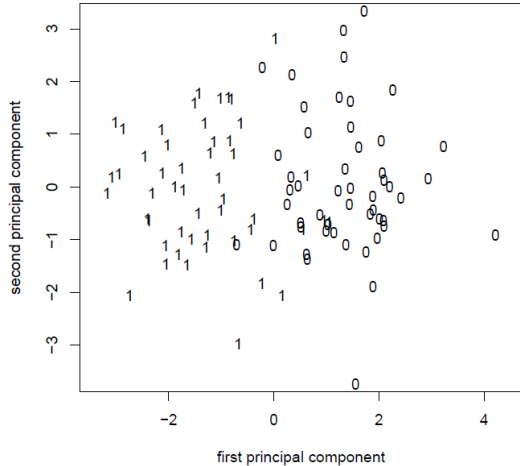
- PCA done to the previous 10-dimensional toy example.
- The first column of **C** shows the special role of x_1 and x_2 :

```
> TOY1.pca$rotation
```

	PC1	PC2	PC3	PC4	PC5
X1	-0.735117401	0.04588148	-0.31630208	0.2695889869	-0.248276945
X2	-0.646092813	0.13874886	0.21251421	-0.2302862081	0.319925262
X3	0.024667479	-0.05161202	-0.28345005	-0.4687697393	0.649227968
X4	0.032852750	-0.18593436	-0.08775101	0.1972183540	0.005126109
X5	0.004581277	-0.31667346	-0.29313421	0.5044335930	0.413148079
X6	-0.040159940	-0.56767619	-0.39726270	-0.1265413915	-0.026557739
X7	-0.075940747	0.18680067	0.17604137	0.0001926329	0.293658046
X8	-0.015784496	0.04059902	-0.18259752	-0.0013653002	-0.006797013
X9	0.086438427	0.23008613	0.17181718	0.5870702809	0.393684023
X10	0.159239249	0.65527667	-0.65509362	-0.0230326636	-0.042639743

	PC6	PC7	PC8	PC9	PC10
X1	0.10503192	0.08322218	0.150235456	0.217513264	0.36781827
X2	-0.27708524	-0.14556192	-0.155014950	-0.299888182	-0.38838070
X3	-0.03599698	-0.01149754	0.307053135	0.134691739	0.40160963
X4	0.11612961	-0.52661040	-0.019556223	-0.695641792	0.37765456
X5	0.11408374	-0.31953556	0.010772758	0.293667271	-0.43260847
X6	-0.13819202	0.46604028	-0.468520436	-0.212504163	-0.02520385
X7	0.76638665	0.15389605	-0.468614434	0.009069721	0.10676435
X8	0.37598438	0.37962657	0.591525091	-0.434049722	-0.37540774
X9	-0.35578987	0.44435407	0.002886506	-0.179093631	0.23304328
X10	-0.09999440	-0.09943693	-0.263815078	-0.093494677	-0.12463066

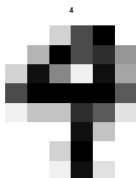
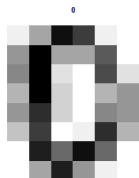
PCA: Toy Example cont.



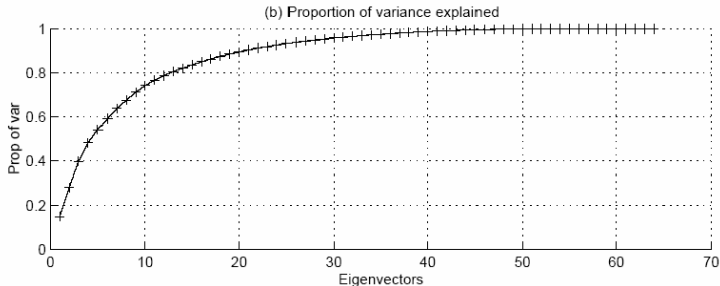
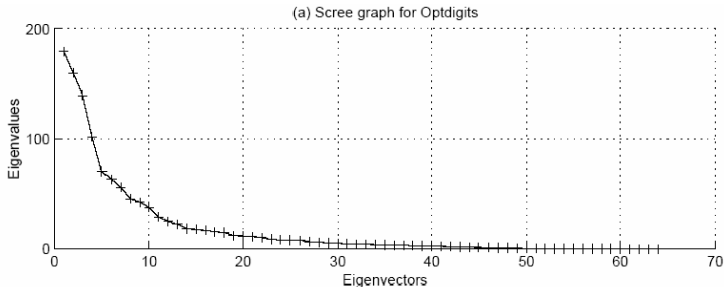
```
> TOY1.pc <- predict(TOY1.pca)
> eqscplot(TOY1.pc[,1:2],type="n",
+          xlab="first principal component",
+          ylab="second principal component")
> text(TOY1.pc[,1:2],labels=as.character(TOY1[, "Class"]))
```

PCA: OptDigits Example

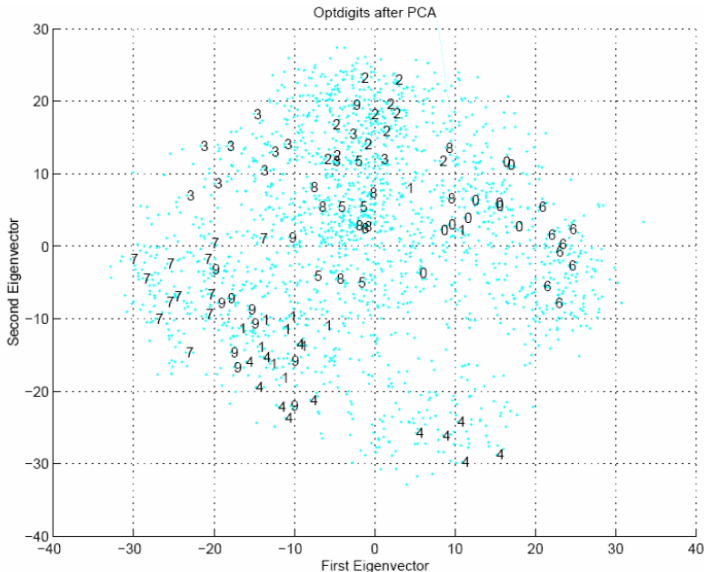
- The OptDigits data set contains 5620 instances of digitized handwritten digits in range 0-9.
- Each digit is a \mathbb{R}^{64} vector: $8 \times 8 = 64$ pixels, 16 grayscales.



PCA: OptDigits Example cont.



PCA: OptDigits Example cont.



PCA: Discussion

- PCA can be used as a preprocessing step for **decorrelation** (notice that the principal components z_i^t are uncorrelated)
- If different components i of the observations, x_i^t , are scaled differently, PCA will concentrate on the largest ones
- Solution: rescale each component i to unit variance before applying PCA
- Probabilistic PCA can be
 - used in case of missing values
 - extended in many ways

Linear Discriminant Analysis 1

- PCA is an unsupervised method (class information is not usually used).
- Linear Discriminant Analysis (LDA) is a **supervised** method for dimensionality reduction in classification problems.
- As PCA, LDA can be accomplished with standard matrix algebra (eigenvalue decompositions etc.). This makes it relatively simple and useful.
- PCA is a good general purpose dimensionality reduction method, LDA is a good alternative if we want to optimize the **separability of classes** in a specific classification task, and are happy with a dimensionality of less than the number of classes ($k < K$).

Linear Discriminant Analysis 2

- Originally introduced for two-class problems, idea: transform the data so that the classes (c_1, c_2) are separated as much as possible

- Within-class scatter matrix $\hat{\Sigma}_w = \sum_i \sum_{\mathbf{x} \in c_i} (\mathbf{x} - \bar{\mathbf{x}}_i)(\mathbf{x} - \bar{\mathbf{x}}_i)'$

where $\bar{\mathbf{x}}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in c_i} \mathbf{x}$ and m_i is the number of samples in c_i

- Between-class scatter matrix

$$\hat{\Sigma}_b = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'$$

- Optimize projection matrix Φ to maximize ratio of between-class to within-class scatter:

$$\mathcal{J}(\Phi) = \frac{|\Phi^T \hat{\Sigma}_b \Phi|}{|\Phi^T \hat{\Sigma}_w \Phi|}$$

Optimized matrix Φ given by eigenvectors of $\hat{\Sigma}_w^{-1} \hat{\Sigma}_b$

Linear Discriminant Analysis 3

- Multi-class case is similar:

- Within-class scatter matrix $\hat{\Sigma}_w = \sum_{i=1}^n \sum_{\mathbf{x} \in c_i} (\mathbf{x} - \bar{\mathbf{x}}_i)(\mathbf{x} - \bar{\mathbf{x}}_i)'$

where $\bar{\mathbf{x}}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in c_i} \mathbf{x}$ and m_i is the number of samples in c_i

- Between-class scatter matrix

$$\hat{\Sigma}_b = \sum_{i=1}^n m_i (\bar{\mathbf{x}}_i - \bar{\mathbf{x}})(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})'$$

- Optimize projection matrix Φ to maximize ratio of between-class to within-class scatter:

$$\mathcal{J}(\Phi) = \frac{|\Phi^T \hat{\Sigma}_b \Phi|}{|\Phi^T \hat{\Sigma}_w \Phi|}$$

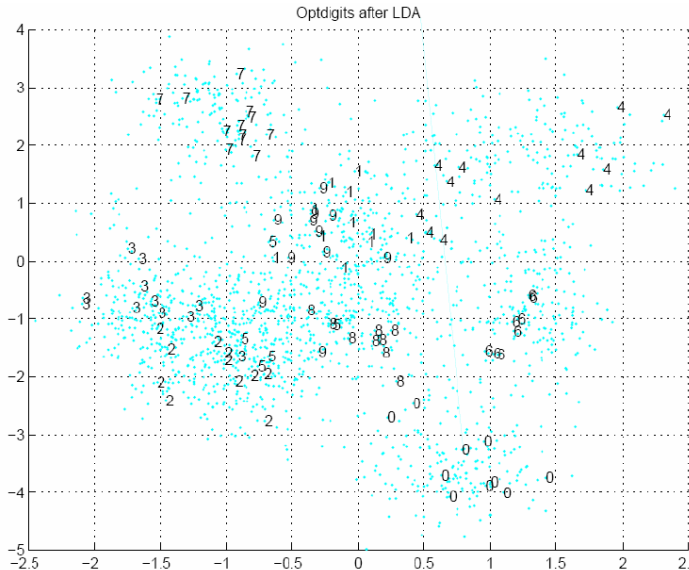
Optimized matrix Φ given by solving the generalized eigenvalue problem

$$\hat{\Sigma}_b \Phi = \lambda \hat{\Sigma}_w \Phi$$

Linear Discriminant Analysis 4

- The rank of the within-class scatter matrix is upper-bounded by $m-n$, and the rank of the between-class scatter matrix is upper bounded by $n-1$. ---> LDA cannot give more projection directions than $n-1$ (number of classes - 1).
- Classification in the low-dimensional space can be done e.g. by finding the nearest class centroid of a new point
- LDA projection maximizes mean-squared distance between classes in the projected space, not the same as minimizing classification error. Pairs of classes that are far apart dominate the LDA criterion, and can leave overlap between the remaining classes.

LDA: OptDigits Example



Independent Component Analysis 1

- Imagine you are in a room with two people talking simultaneously
- Two microphones in different locations are recording the sound
- The microphones record the signals as time series of amplitudes, $x_1(t)$ and $x_2(t)$
- Each signal is a weighed average of the speech signals of the speakers denoted by $s_1(t)$ and $s_2(t)$, so that

$$x_1(t) = a_{11}s_1 + a_{12}s_2$$

$$x_2(t) = a_{21}s_1 + a_{22}s_2$$

where a_{11} , a_{12} , a_{21} , a_{22} are mixing parameters (depending e.g. on distances from microphones to speakers)

- We want to estimate the original sources $s_1(t)$ and $s_2(t)$ using only the recordings $x_1(t)$ and $x_2(t)$.
- Called the **cocktail party problem**.

Independent Component Analysis 2

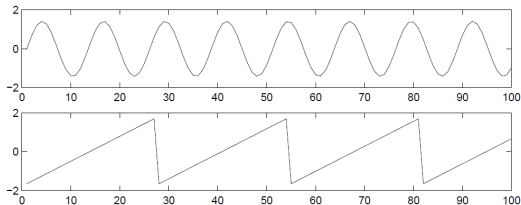


Figure 1: The original signals.

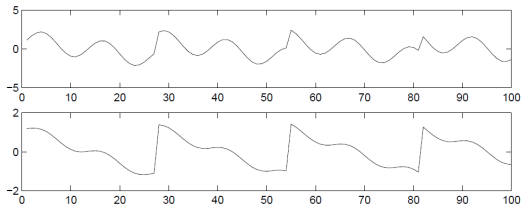


Figure 2: The observed mixtures of the source signals in Fig. 1.

All ICA related figures on these slides are from Hyvärinen and Oja, Independent Component Analysis: Algorithms and Applications, *Neural Networks* 2000

Independent Component Analysis 3

- If we knew the mixing parameters we could solve the sources by a linear matrix equation, but here the mixing parameters are unknown.
- It turns out the source signals and mixing parameters can be solved using assumptions about statistical properties of the source signals. It is enough to assume the sources are **statistically independent**.

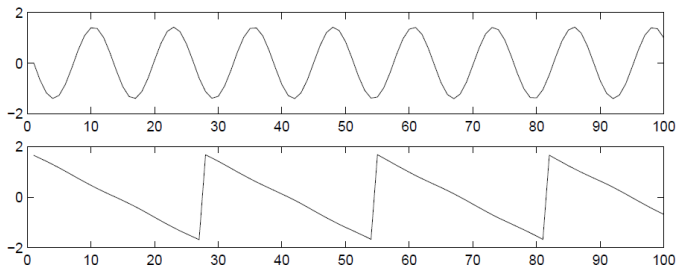


Figure 3: The estimates of the original source signals, estimated using only the observed signals in Fig. 2. The original signals were very accurately estimated, up to multiplicative signs.

Independent Component Analysis 4

- ICA was originally proposed for applications similar to the cocktail party problem, but is now used for a lot of applications, for example analysis of EEG recordings measured from several sensors attached to the scalp.
- ICA can be used for feature extraction. Does not have to be for time series signals.

Independent Component Analysis 5

Basis functions (components) found by ICA from patches of natural images.

Each block in an actual image would be a linear combination of these patches.

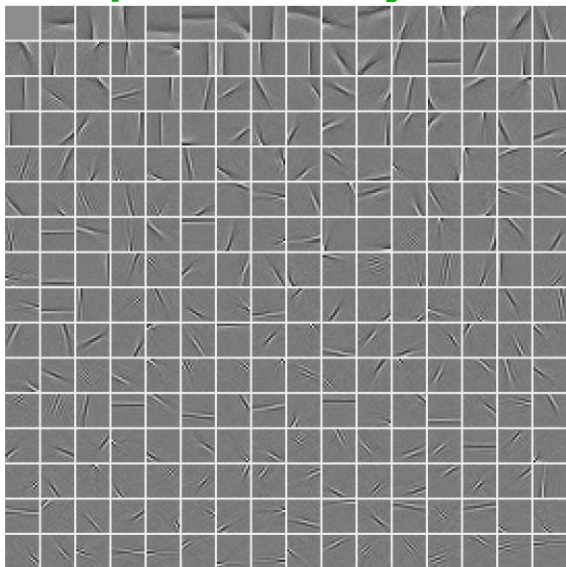


Figure 4: Basis functions in ICA of natural images. The input window size was 16×16 pixels. These basis functions can be considered as the independent features of images.

Independent Component Analysis 6

Definition (generative model):

- Assume we observe n linear mixtures x_1, \dots, x_n of n independent components

$$x_j = a_{j1}s_1 + a_{j2}s_2 + \dots + a_{jn}s_n, \text{ for all } j$$

- Each mixture x_j and each independent component s_k is a random variable, observed values are samples of the random variables
- Assume the mixtures and independent components are zero-mean (can be achieved by subtracting mean from observations)
- In vector notation: $\mathbf{x} = [x_1, \dots, x_n]^T$, $\mathbf{s} = [s_1, \dots, s_n]^T$, matrix \mathbf{A} contains the elements a_{ij} . Denote the j th column of \mathbf{A} by \mathbf{a}_j . Then

$$\mathbf{x} = \mathbf{A}\mathbf{s} = \sum_{i=1}^n \mathbf{a}_i s_i$$

- We want to estimate \mathbf{A} , or equivalently an unmixing matrix \mathbf{W} so that $\mathbf{s} = \mathbf{W}\mathbf{x}$. Closely related to **blind source separation**.

Independent Component Analysis 7

Assumptions:

- To solve the ICA task we assume the underlying sources are **statistically independent**.
- We also assume their distributions are **nongaussian**
- We **do not assume** we know the distributions
- We assume the mixing matrix is square ("as many microphones as voices")

Ambiguities:

- We cannot determine variances of the independent components (ICs): any scaling of a source s_i can be compensated in **A**.
- We cannot determine order of sources (any permutation matrix applied to sources can be compensated in **A**)

Independent Component Analysis 8

Assumptions:

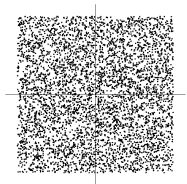
- To solve the ICA task we assume the underlying sources are **statistically independent**.
- We also assume their distributions are **nongaussian**
- We **do not assume** we know the distributions
- We assume the mixing matrix is square ("as many microphones as voices")

Ambiguities:

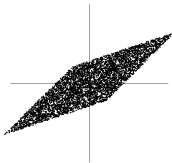
- We cannot determine variances of the independent components (ICs): any scaling of a source s_i can be compensated in **A**.
- We cannot determine order of sources (any permutation matrix applied to sources can be compensated in **A**)

Independent Component Analysis 9

Simple example:



Original sources
distributed in a
square, indep.



Mixed with
 $A = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}$.
Mixtures are
not indep.

How to measure independence?

Theory: y_1, y_2 are independent if

$$p(y_1, y_2) = p_1(y_1)p_2(y_2)$$

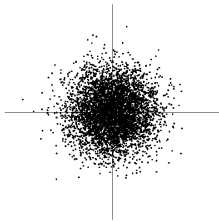
Then, for expectations,

$$E\{h_1(y_1)h_2(y_2)\} = E\{h_1(y_1)\}E\{h_2(y_2)\}$$

Being uncorrelated is not
enough to be independent,
but independent implies uncorrelated.
—> Search for ICs often constrained to
uncorrelated components.

Independent Component Analysis 10

- Why independent components are assumed to be nongaussian: if two variables are IID Gaussian with variance 1, any orthogonal transform of the variables has the same distribution as the original variables \rightarrow ICs could only be estimated up to an orthogonal transform!
- Nongaussianity turns out to be a good criterion for independence: central limit theorem \rightarrow a sum of indep. variables tends towards a Gaussian distribution
- Idea of ICA: optimize an unmixing matrix \mathbf{W} so that the unmixed variables are **as nongaussian as possible!**
- To estimate just one component by $y = \mathbf{w}^T \mathbf{x} = \sum_i w_i x_i$ maximize nongaussianity of $\mathbf{w}^T \mathbf{x}$ with respect to \mathbf{w}
- To find more ICs, maximize again, constrain to be uncorrelated with previous ICs



Independent Component Analysis 11

Measures of nongaussianity:

- **Kurtosis** $\text{kurt}(y) = E\{y^4\} - 3(E\{y^2\})^2$

simple to estimate, nice theoretical properties, e.g.

$\text{kurt}(x_1+x_2) = \text{kurt}(x_1) + \text{kurt}(x_2)$ for independent variables.

But can be sensitive to outliers – not robust.

- **Negentropy** $J(\mathbf{y}) = H(\mathbf{y}_{\text{gauss}}) - H(\mathbf{y})$

where $H(\mathbf{y}) = - \int f(\mathbf{y}) \log f(\mathbf{y}) d\mathbf{y}$ (f : prob. density function)

and $\mathbf{y}_{\text{gauss}}$ is a Gaussian variable with the same mean and covariance matrix as in \mathbf{y} . Idea: Gaussian variables are known to have the highest entropy of all variables with the same second-order statistics. The smaller the entropy of \mathbf{y} is compared to a Gaussian variable, the more nongaussian \mathbf{y} is.

Negentropy is invariant under invertible linear transformations.

Independent Component Analysis 12

- Approximation of negentropy: $J(y) \approx \sum_{i=1}^p k_i [E\{G_i(y)\} - E\{G_i(v)\}]^2$

where k_i are positive constants, v is a zero-mean unit-variance Gaussian variable, and G_i are some nonquadratic functions, e.g.

$$G_1(u) = \frac{1}{a_1} \log \cosh a_1 u, \quad G_2(u) = -\exp(-u^2/2) \quad 1 \leq a_1 \leq 2$$

- **Mutual information:** $I(y_1, y_2, \dots, y_m) = \sum_{i=1}^m H(y_i) - H(\mathbf{y})$

For unit-variance y_i : $I(y_1, y_2, \dots, y_n) = C - \sum_i J(y_i)$ (C: constant)

Minimizing mutual information = maximizing negentropy (when estimates constrained to be uncorrelated)

- **Maximum likelihood:** strongly related to minimization of mutual information. But requires good estimation of densities, at least are they "subgaussian" (kurtosis < 0) or "supergaussian" (> 0)

Independent Component Analysis 13

Computation of ICs (fastICA algorithm):

- Function g for derivative of the nonquadratic function. For G_1, G_2 :
 $g_1(u) = \tanh(a_1 u)$, $g_2(u) = u \exp(-u^2/2)$
- Preprocessing: subtract mean from data; then **whiten** data to have unit covariance matrix: set $\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{x}$ where $\mathbf{E}\{\mathbf{x}\mathbf{x}^T\} = \mathbf{E}\mathbf{D}\mathbf{E}^T$ is the eigenvalue decomposition of the covariance matrix, $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$ and $\mathbf{D}^{-1/2} = \text{diag}(d_1^{-1/2}, \dots, d_n^{-1/2})$
- FastICA:
 1. Choose an initial (e.g. random) weight vector \mathbf{w}
 2. Let $\mathbf{w}^+ = E\{\mathbf{x}g(\mathbf{w}^T\mathbf{x})\} - E\{g'(\mathbf{w}^T\mathbf{x})\}\mathbf{w}$
 3. Let $\mathbf{w} = \mathbf{w}^+ / \|\mathbf{w}^+\|$
 4. If not converged, go to 2.
- Further components: rerun fastICA, decorrelate from previous ICs after each iteration: set $\mathbf{w}_{p+1} = \mathbf{w}_{p+1} - \sum_{j=1}^p \mathbf{w}_{p+1}^T \mathbf{w}_j \mathbf{w}_j$ then renormalize by $\mathbf{w}_{p+1} = \mathbf{w}_{p+1} / \sqrt{\mathbf{w}_{p+1}^T \mathbf{w}_{p+1}}$

Independent Component Analysis 14

- Once ICs have been found they can be used in dimensionality reduction, e.g. by feature selection methods discussed previously. (Or try ranking by nongaussianity (negentropy), etc.)

Some applications of ICA:

- separating artifacts in Magnetoencephalography (MEG) data
- finding hidden factors in financial data like stock portfolios or cashflow data of stores
- representing natural images
- separating user's signal from interfering signals in telecommunications

Kernel based methods

- The methods so far on the lecture all optimize linear transformations of the form $\mathbf{y} = \mathbf{W}\mathbf{x}$.
 - Resulting low-dimensional coordinates $\mathbf{W}\mathbf{x}$ are linear combinations of the original coordinates.
- It is possible to optimize such transformations after first performing a fixed nonlinear mapping $\mathbf{x}_{\text{mapped}} = \mathbf{f}(\mathbf{x})$, so that $\mathbf{y} = \mathbf{W}\mathbf{x}_{\text{mapped}} = \mathbf{W}\mathbf{f}(\mathbf{x})$.
 - If \mathbf{f} is simple to compute, then \mathbf{W} can be learned as usual from the computed $\mathbf{f}(\mathbf{x})$ values.
 - Resulting low-dimensional coordinates $\mathbf{W}\mathbf{f}(\mathbf{x})$ are linear combinations of the $\mathbf{f}(\mathbf{x})$ coordinates but are **nonlinear mappings** of the original \mathbf{x} .
- Problem: computational complexity increases as dimensionality of $\mathbf{f}(\mathbf{x})$ increases.
- **Kernel trick:** it is often enough to be able to compute inner products between $\mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}')$, without explicitly knowing $\mathbf{f}(\mathbf{x})$ or $\mathbf{f}(\mathbf{x}')$

Kernel PCA 1

- Denote the nonlinear feature transformation here by Φ and the set of ℓ transformed input samples by $\Phi(\mathbf{x}_1), \dots, \Phi(\mathbf{x}_\ell)$
- The covariance matrix we need in PCA is then

$$\bar{C} = \frac{1}{\ell} \sum_{j=1}^{\ell} \Phi(\mathbf{x}_j) \Phi(\mathbf{x}_j)^{\top}$$

- We must compute each eigenvalue $\lambda \geq 0$ and eigenvector \mathbf{V} for the matrix satisfying $\lambda \mathbf{V} = \bar{C} \mathbf{V}$.
- It turns out the solutions \mathbf{V} lie in the span of the data: they are linear combinations of $\Phi(\mathbf{x}_1), \dots, \Phi(\mathbf{x}_\ell)$. The eigenvalue problem becomes $\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot \bar{C} \mathbf{V})$ for all $k = 1, \dots, \ell$, where

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) \quad \text{for some coefficients } \alpha_1, \dots, \alpha_\ell$$

- The eigenvalue problem can be written in terms of the coefficients!

Kernel PCA 2

- Inserting the equation for \mathbf{V} in terms of the coefficients, the eigenvalue problem becomes $\ell \lambda \mathbf{K} \boldsymbol{\alpha} = K^2 \boldsymbol{\alpha}$ where \mathbf{K} is an $\ell \times \ell$ inner-product (kernel) matrix: $K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$ $\boldsymbol{\alpha}$ is a column vector with entries $\alpha_1, \dots, \alpha_\ell$
- To solve the eigenvalue problem, solve instead $\ell \lambda \boldsymbol{\alpha} = \mathbf{K} \boldsymbol{\alpha}$ for nonzero eigenvalues.
- Principal component projection directions are normalized to have unit norm, $(\mathbf{V}^k \cdot \mathbf{V}^k) = 1$. Inserting the definitions, that becomes

$$1 = \sum_{i,j=1}^{\ell} \alpha_i^k \alpha_j^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = (\boldsymbol{\alpha}^k \cdot \mathbf{K} \boldsymbol{\alpha}^k) = \lambda_k (\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k)$$

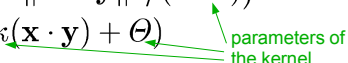
thus the square root of λ_k becomes the normalization factor for $\boldsymbol{\alpha}^k$

- To project a test point \mathbf{x} onto the k :th eigenvector:

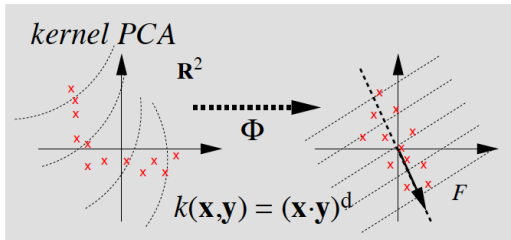
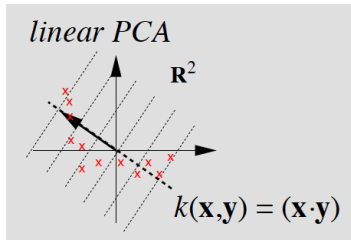
$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{\ell} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}))$$

- None of the operations require the actual transformed features, the inner products between them are enough!

Kernel PCA 3

- Many interesting kernels can be defined that satisfy the properties of an inner product between some transformed features, but the transformed features themselves would be expensive/impossible to compute!
- Polynomial kernel of order d : $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$
(corresponds to a transformation onto all products of d original input values)
- Radial basis function: $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / (2 \sigma^2))$
- Sigmoid kernel: $k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta)$ 
- After the kernels have been computed, computational complexity depends on the size of the kernel matrix but not on the original input dimensionality or the transformed input dimensionality

Kernel PCA 4



Using a nonlinear kernel implicitly causes PCA to be done in a high-dimensional space nonlinearly related to the original features. Dotted lines = contours of constant principal component feature value

Random Projections

- Simple idea: Project using a random matrix \mathbf{W} , e.g. draw each element from a normal distribution
- Mathematical proofs of some good properties
- Very simple and computationally light method
- Surprisingly impressive empirical results
- This has gained popularity in ML research (Keywords: random projections, compressed sensing, extreme learning machine, random features)

References

- Hervé Abdi and Lynne J. Williams. **Principal Component Analysis**. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2, 433-459, 2010
<https://www.utdallas.edu/~herve/abdi-awPCA2010.pdf>
- Aapo Hyvärinen and Erkki Oja. **Independent Component Analysis: Algorithms and Applications**. *Neural Networks*, 13(4-5):411-430, 2000
<http://www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf>
- Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. **Using discriminant analysis for multi-class classification: an experimental investigation**. *Knowl Inf Syst*, 10(4): 453-472, 2006.
<https://users.cs.fiu.edu/~taoli/pub/kais-discriminant.pdf>
- Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. **Kernel principal component analysis**. *Proceedings of ICANN'97*, 1997.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.3580&rep=rep1&type=pdf>
- Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller. **Fisher discriminant analysis with kernels**. *Proceedings of NNSP'99*, 1999.
<http://luthuli.cs.uiuc.edu/~daf/courses/learning/Kernelpapers/00788121.pdf>