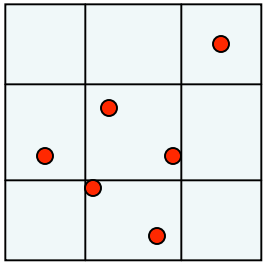
Feature Selection/Extraction

Dimensionality Reduction

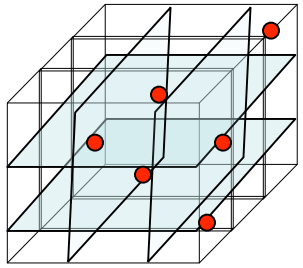
Feature Selection/Extraction

- Solution to a number of problems in Pattern Recognition can be achieved by choosing a better feature space.
- **Problems and Solutions:**
  - Curse of Dimensionality:
    - #examples needed to train classifier function grows exponentially with #dimensions.
    - Overfitting and Generalization performance
  - What features best characterize class?
    - What words best characterize a document class
    - Subregions characterize protein function?
  - What features critical for performance?
      - Subregions characterize protein function?
  - Inefficiency
    - Reduced complexity and run-time
  - Can't Visualize
    - Allows 'intuiting' the nature of the problem solution.

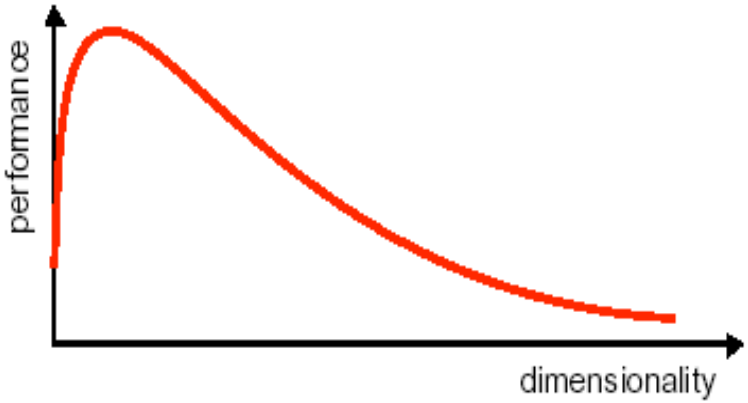
Curse of Dimensionality



Same Number of examples
Fill more of the available space
When the dimensionality is low



- **Implications of the curse of dimensionality**
  - Exponential growth with dimensionality in the number of examples required to accurately estimate a function
- **In practice, the curse of dimensionality means that**
  - For a given sample size, there is a maximum number of features above which the performance of our classifier will degrade rather than improve
    - In most cases, the information that was lost by discarding some features is compensated by a more accurate mapping in lower-dimensional space



Selection vs. Extraction

- **Two general approaches for dimensionality reduction**
  - Feature extraction: Transforming the existing features into a lower dimensional space
  - Feature selection: Selecting a subset of the existing features without a transformation

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature selection}} \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \\ x_{i_M} \end{bmatrix} \qquad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}\right)$$

- **Feature extraction**
  - PCA
  - LDA (Fisher's)
  - Nonlinear PCA (kernel, other varieties
  - 1st layer of many networks

Feature selection (Feature Subset Selection)

Although FS is a special case of feature extraction, in practice quite different

- FSS searches for a subset that minimizes some cost function (e.g. test error)
- FSS has a unique set of methodologies

Feature Subset Selection

Definition

Given a feature set $\boldsymbol{x}=\{x_i \mid i=1\ldots N\}$

find a subset $\boldsymbol{x_M}=\{x_{i1}, x_{i2}, \ldots, x_{iM}\}$, with $M<N$, that optimizes an objective function $J(Y)$, e.g. P(correct classification)

Why Feature Selection?

- Why not use the more general feature extraction methods?

Feature Selection is necessary in a number of situations

- Features may be expensive to obtain
- Want to extract meaningful rules from your classifier
- When you transform or project, measurement units (length, weight, etc.) are lost
- Features may not be numeric (e.g. strings)

Implementing Feature Selection

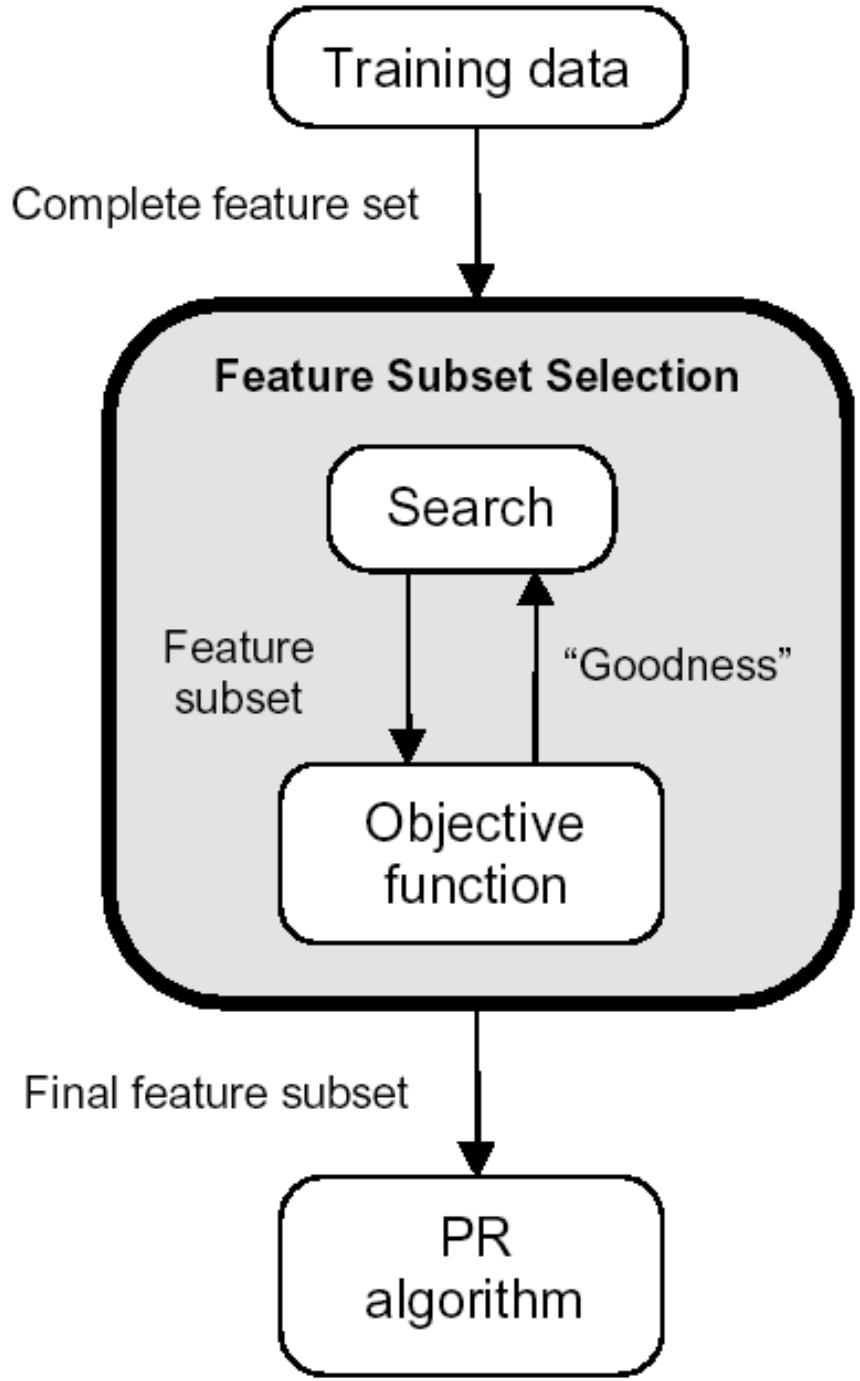
- **Feature Subset Selection requires**
  - A search strategy to select candidate subsets
  - An objective function to evaluate these candidates
- **Search Strategy**
  - Exhaustive evaluation of feature subsets involves $\binom{N}{M}$ combinations for a fixed value of M, and 2^N combinations if M must be optimized as well
    - This number of combinations is unfeasible, even for moderate values of M and N, so a search procedure must be used in practice
    - For example, exhaustive evaluation of 10 out of 20 features involves 184,756 feature subsets; exhaustive evaluation of 10 out of 20 involves more than 10^{13} feature subsets
  - A search strategy is therefore needed to direct the FSS process as it explores the space of all possible combination of features

Objective Function

The objective function evaluates candidate subsets and returns a measure of their “goodness”.

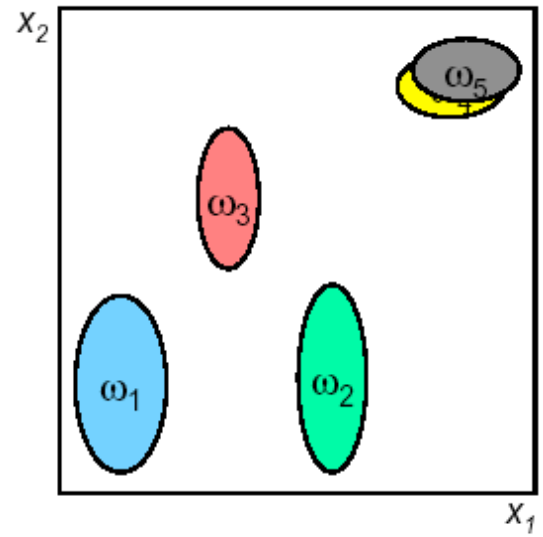
This feedback is used by the search strategy to select new candidates.

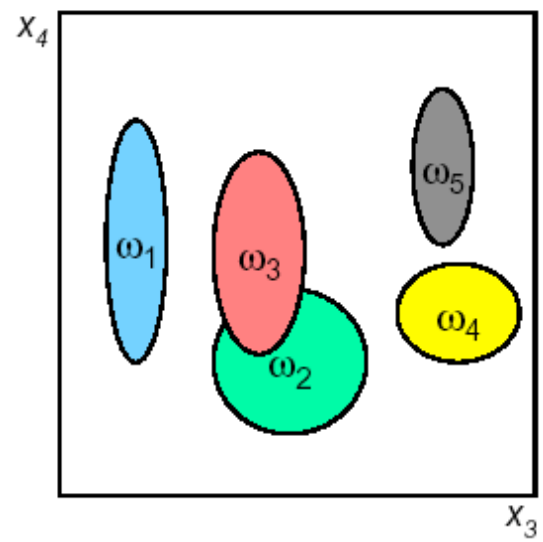
Simple Objective function: Cross-validation error rate.



Naïve sequential feature selection

- **One may be tempted to evaluate each individual feature separately and select those M features with the highest scores**
  - Unfortunately, this strategy will VERY RARELY work since it does not account for feature dependence
- **An example will help illustrate the poor performance that can be expected from this naïve approach**
  - The figures show a 4-dimensional pattern recognition problem with 5 classes. Features are shown in pairs of 2D scatter plots
  - The objective is to select the best subset of 2 features using the naïve sequential feature selection procedure
  - Any reasonable objective function will rank features according to this sequence: $J(x_1)>J(x_2)\approx J(x_3)>J(x_4)$
    - x_1 is, without a doubt, the best feature. It clearly separates ω_1, ω_2, ω_3 and $\{\omega_4, \omega_5\}$
    - x_2 and x_3 have similar performance, separating classes in three groups
    - x_4 is the worst feature since it can only separate ω_4 from ω_5, the rest of the classes having a heavy overlap
  - The optimal feature subset turns out to be $\{x_1, x_4\}$, because x_4 provides the only information that x_1 needs: discrimination between classes ω_4 and ω_5
  - However, if we were to choose features according to the individual scores $J(x_k)$, we would choose x_1 and either x_2 or x_3, leaving classes ω_4 and ω_5 non separable
    - This naïve strategy fails because it does not take into account the interaction between features

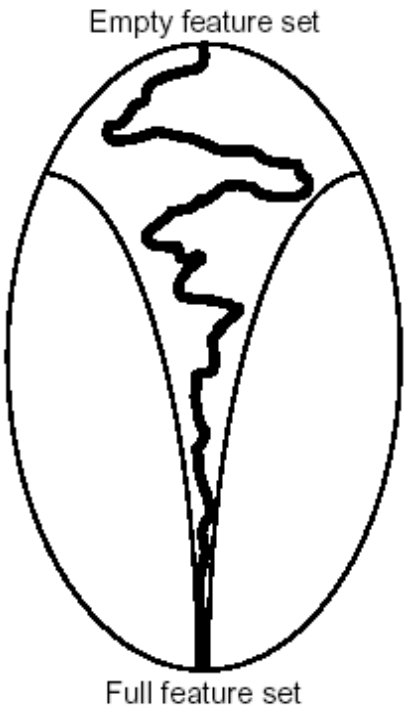




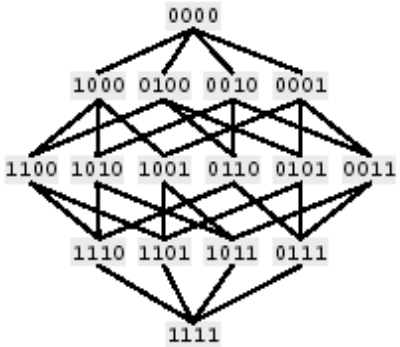
Sequential Forward Selection (SFS)

- **Sequential Forward Selection is the simplest greedy search algorithm**
  - Starting from the empty set, sequentially add the feature x^+ that results in the highest objective function $J(Y_k+x^+)$ when combined with the features Y_k that have already been selected
- **Algorithm**

1. Start with the empty set $Y=\{\varnothing\}$
2. Select the next best feature $x^+ = \underset{x \in X - Y_k}{\operatorname{argmax}}\left[J(Y_k + x)\right]$
3. Update $Y_{k+1}=Y_k+x$; $k=k+1$
4. Go to 2



- **Notes**
  - SFS performs best when the optimal subset has a small number of features
    - When the search is near the empty set, a large number of states can be potentially evaluated
    - Towards the full set, the region examined by SFS is narrower since most of the features have already been selected
  - The search space is drawn like an ellipse to emphasize the fact that there are fewer states towards the full or empty sets
    - As an example, the state space for 4 features is shown. Notice that the number of states is larger in the middle of the search tree
    - The main disadvantage of SFS is that it is unable to remove features that become obsolete after the addition of other features



Feature Extraction

- **Two approaches are available to perform dimensionality reduction**
  - **Feature extraction**: creating a subset of new features by combinations of the existing features
  - **Feature selection**: choosing a subset of all the features (the ones more informative)

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature selection}} \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \\ x_{i_M} \end{bmatrix} \qquad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}\right)$$

- **The problem of feature extraction can be stated as**
  - Given a feature space $\mathbf{x}_i \in \mathbf{R}^N$ find a mapping $\mathbf{y=f(x):R^N \rightarrow R^M}$ with $\mathbf{M<N}$ such that the transformed feature vector $y_i \in R^M$ preserves (most of) the information or structure in R^N.
  - An **optimal** mapping **y=f(x)** will be one that results in **no increase in the minimum probability of error**
    - This is, a Bayes decision rule applied to the initial space R^N and to the reduced space R^M yield the same classification rate

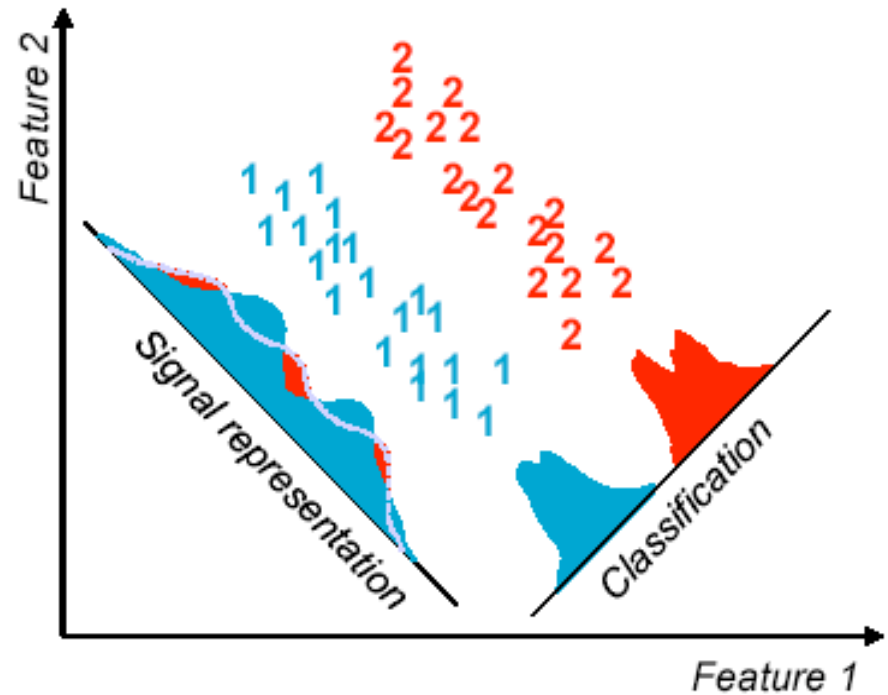
In general, the optimal mapping y=f(x) will be a non-linear function

- However, there is no systematic way to generate non-linear transforms
- The selection of a particular subset of transforms is problem dependent
- For this reason, feature extraction is commonly limited to linear transforms: ***y=Wx***

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & & w_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Signal representation versus classification

- **The selection of the feature extraction mapping *y=f(x)* is guided by an objective function that we seek to maximize (or minimize)**
- **Depending on the criteria used by the objective function, feature extraction techniques are grouped into two categories:**
  - **Signal representation**: The goal of the feature extraction mapping is to represent the samples accurately in a lower-dimensional space
  - **Classification**: The goal of the feature extraction mapping is to enhance the class-discriminatory information in the lower-dimensional space
- **Within the realm of linear feature extraction, two techniques are commonly used**
  - Principal Components Analysis (PCA)
    - uses a signal representation criterion
  - Linear Discriminant Analysis (LDA)
    - uses a signal classification criterion



PCA Derivation: Minimizing Reconstruction Error

Any point in $\mathbf{R}^n$ can perfectly reconstructed in a new Orthonormal basis of size *n*.

$$\mathbf{x} = \mathbf{U}\mathbf{y}, \quad \text{such that} \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

$$\mathbf{x} = \left[\vec{u}_1 \middle| \vec{u}_2 \middle| \cdots \middle| \vec{u}_n\right]\mathbf{y} = \sum_{i=1:n} y_i \vec{u}_i$$

$$\hat{\mathbf{x}}(m) = \left[\vec{u}_1 \middle| \vec{u}_2 \middle| \cdots \middle| \vec{u}_m\right]\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Define a reconstruction based on the 'best' *m* vectors $\mathbf{x}(m)$

$$\hat{\mathbf{x}} = \left[\vec{u}_1 \middle| \vec{u}_2 \middle| \cdots \middle| \vec{u}_m\right]\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} + \left[\vec{u}_{m+1} \middle| \vec{u}_{m+2} \middle| \cdots \middle| \vec{u}_n\right]\begin{bmatrix} y_{m+1} \\ \vdots \\ y_n \end{bmatrix}$$

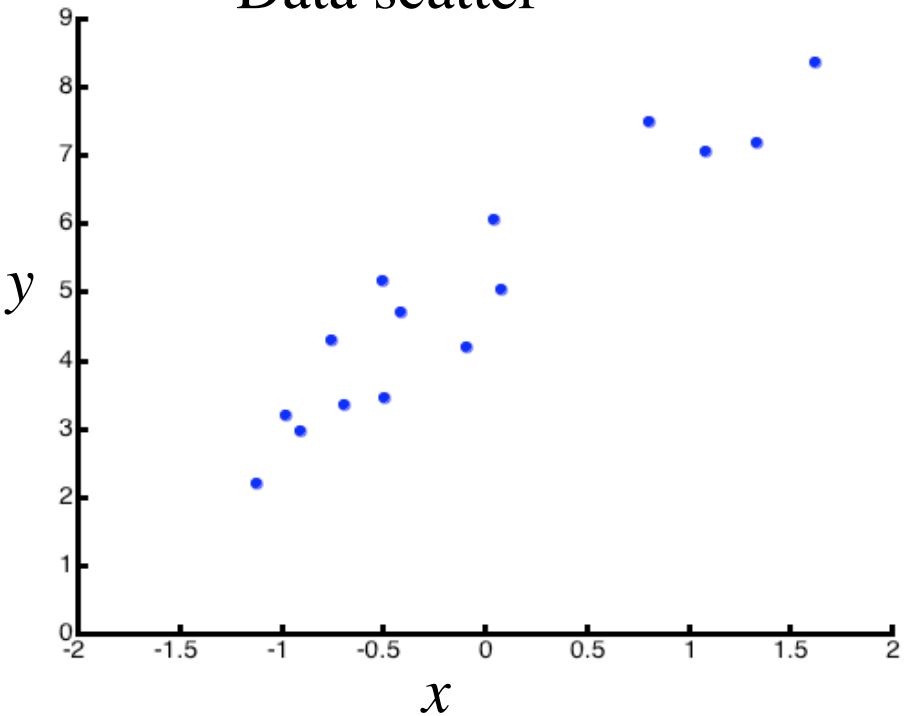
$$\hat{\mathbf{x}} = \mathbf{U}_m\mathbf{y}_m + \mathbf{U}_d\mathbf{b} = \hat{\mathbf{x}}(m) + \hat{\mathbf{x}}_{discard}$$

Goal: Find an orthonormal basis of m vectors, $m<n$ that minimizes *Reconstruction error.*

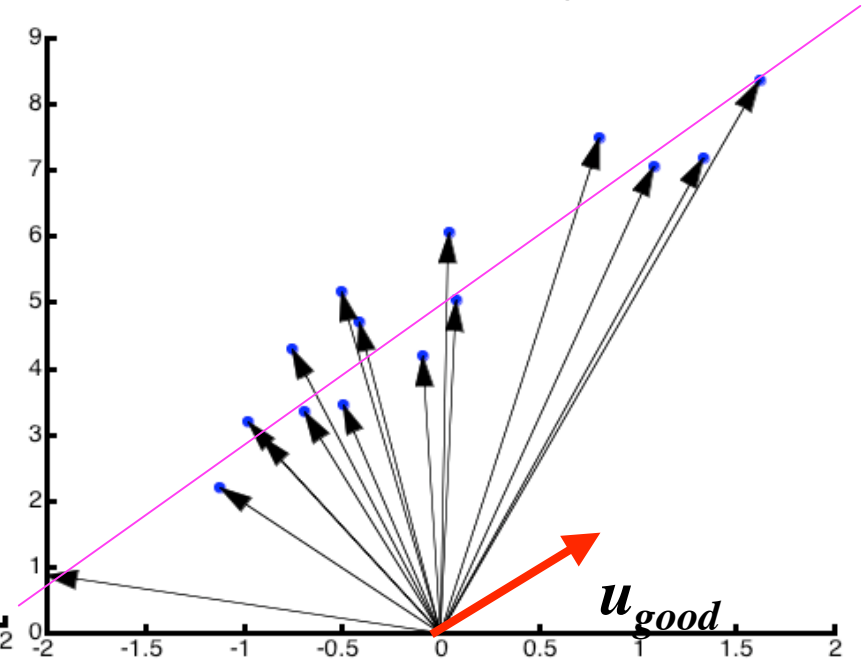
$$Err^2_{recon} = \sum_{k=1:Nsamples} \left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)^T \left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)$$

Visualizing Reconstruction Error

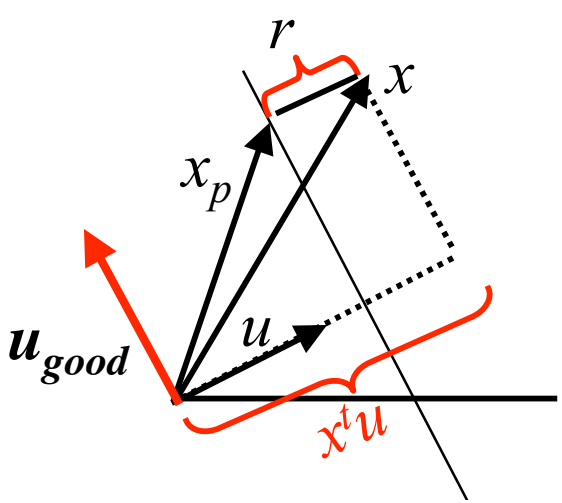
Data scatter



Data as 2D vectors



Solution involves finding directions *u* which minimize the perpendicular distances and removing them



Goal: Find basis vectors u_i and constants b_i minimize reconstruction error

Rewriting the error….

$$\Delta\mathbf{x}(m) = \mathbf{x} - \hat{\mathbf{x}}(m) = \sum_{i=1:n} y_i \vec{u}_i - \left(\sum_{i=1:m} y_i \vec{u}_i + \sum_{i=(m+1):n} b_i \vec{u}_i \right) = \sum_{i=(m+1):n} (y_i - b_i)\vec{u}_i$$

$$Err_{recon}^2 = E\left[\left\| \Delta\mathbf{x}(m) \right\|^2 \right] = E\left[\sum_{j=(m+1):n} (y_j - b_j)\vec{u}_j \sum_{i=(m+1):n} (y_i - b_i)\vec{u}_i \right]$$

$$= E\left[\sum_{j=(m+1):n} \sum_{i=(m+1):n} (y_i - b_i)(y_j - b_j)\vec{u}_i^T \vec{u}_j \right]$$

$$= E\left[\sum_{i=(m+1):n} (y_i - b_i)^2 \right] = \sum_{i=(m+1):n} E\left[(y_i - b_i)^2 \right]$$

Solving for b….

$$\frac{\partial Err}{\partial b_i} = 0 = \frac{\partial}{\partial b_i} \sum_{i=(m+1):n} E\left[(y_i - b_i)^2 \right] = 2(E[y_i] - b_i) \quad \Rightarrow \quad b_i = E[y_i]$$

Therefore, replace the discarded dimensions y_i's by their expected value.

Now rewrite the error replacing the b_i

$$\sum_{i=(m+1):n} E\left[(y_i - E[y_i])^2\right] = \sum_{i=(m+1):n} E\left[(\mathbf{x}^T \vec{u}_i - E[\mathbf{x}^T \vec{u}_i])^2\right]$$

$$= \sum_{i=(m+1):n} E\left[(\mathbf{x}^T \vec{u}_i - E[\mathbf{x}^T \vec{u}_i])^T (\mathbf{x}^T \vec{u}_i - E[\mathbf{x}^T \vec{u}_i])\right]$$

$$= \sum_{i=(m+1):n} E\left[\vec{u}_i^T (\mathbf{x}^T - E[\mathbf{x}^T])^T (\mathbf{x}^T - E[\mathbf{x}^T]) \vec{u}_i\right]$$

$$= \sum_{i=(m+1):n} E\left[\vec{u}_i^T (\mathbf{x} - E[\mathbf{x}])(\mathbf{x} - E[\mathbf{x}])^T \vec{u}_i\right]$$

$$= \sum_{i=(m+1):n} \vec{u}_i^T E\left[(\mathbf{x} - E[\mathbf{x}])(\mathbf{x} - E[\mathbf{x}])^T\right] \vec{u}_i$$

$$\boxed{= \sum_{i=(m+1):n} \vec{u}_i^T \mathbf{C} \vec{u}_i}$$

$\mathbf{C}$ is the covariance matrix for $\mathbf{x}$

Thus, finding the best basis u_i involves minimizing the quadratic form,

$$Err = \sum_{i=(m+1):n} \vec{u}_i^T \mathbf{C} \vec{u}_i$$

subject to the constraint $\| u_i \|=1$

Using Lagrangian Multipliers we form the constrained error function:

$$Err = \sum_{i=(m+1):n} \vec{u}_i^T \mathbf{C} \vec{u}_i + \lambda_i \left(1 - \vec{u}_i^T \vec{u}_i\right)$$

$$\frac{\partial Err}{\partial \vec{u}_i} = \frac{\partial}{\partial \vec{u}_i} \sum_{i=(m+1):n} \vec{u}_i^T \mathbf{C} \vec{u}_i + \lambda_i \left(1 - \vec{u}_i^T \vec{u}_i\right) = 0$$

$$= \frac{\partial}{\partial \vec{u}_i} \left(\vec{u}_i^T \mathbf{C} \vec{u}_i + \lambda_i \left(1 - \vec{u}_i^T \vec{u}_i\right)\right) = 2\mathbf{C}\vec{u}_i - 2\lambda_i \vec{u}_i = 0$$

Which results in the following Eigenvector problem

$$\mathbf{C}\vec{u}_i = \lambda_i \vec{u}_i$$

Plugging back into the error:

$$Err = \sum_{i=(m+1):n} \vec{u}_i^T \mathbf{C} \vec{u}_i + \lambda_i \left(1 - \vec{u}_i^T \vec{u}_i\right)$$

$$Err = \sum_{i=(m+1):n} \vec{u}_i^T \left(\lambda_i \vec{u}_i\right) + 0 = \sum_{i=(m+1):n} \lambda_i$$

Thus the solution is to discard the m-n *smallest eigenvalue eigenvectors.*

PCA summary:

1) Compute data covariance
2) Eigenanalysis on covariance matrix
3) Throw out smallest eigenvalue eigenvectors

Problem: How many to keep?

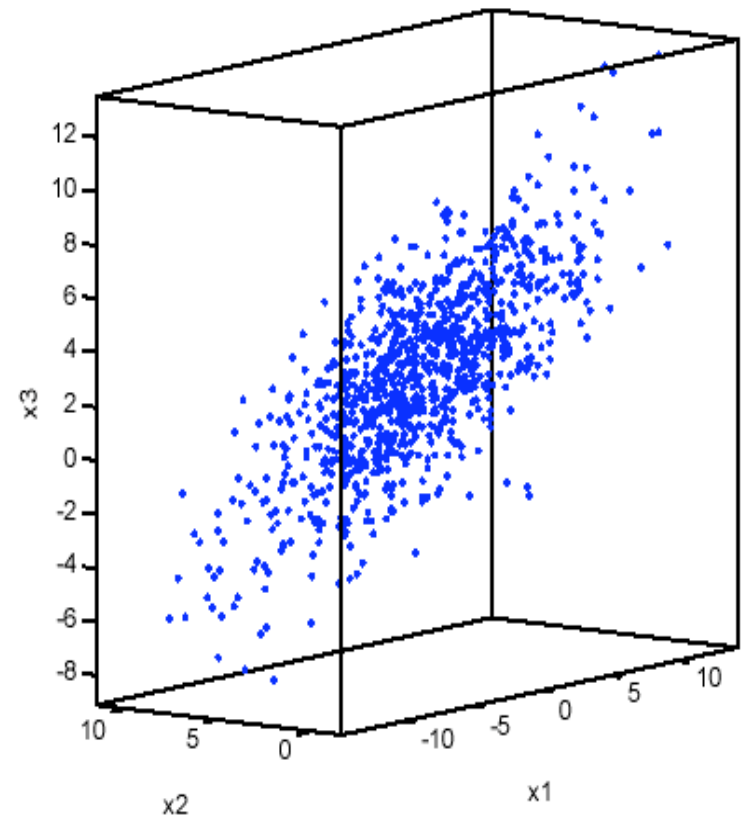
Many criteria.

e.g. % total data variance:

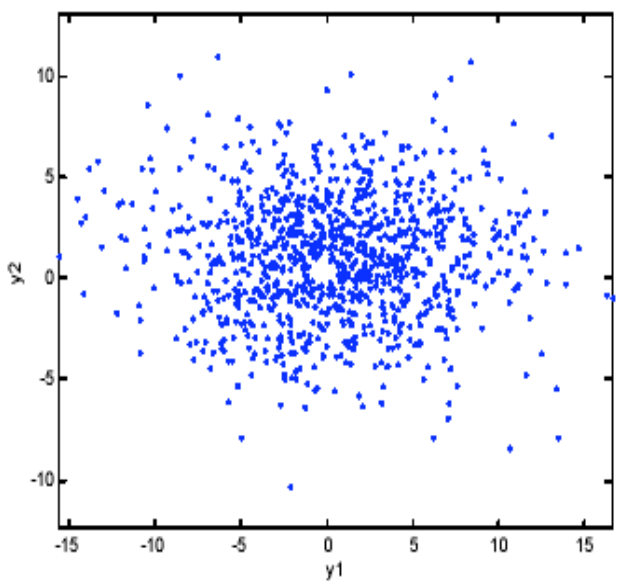
$$\max(m) \ni \frac{\sum_{i=(m+1):n} \lambda_i}{\sum_{i=1:n} \lambda_i} < \varepsilon$$

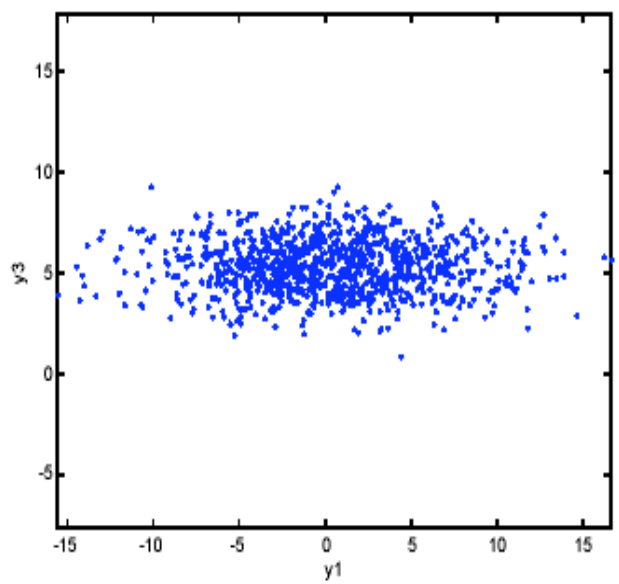
- **In this example we have a three-dimensional Gaussian distribution with the following parameters**

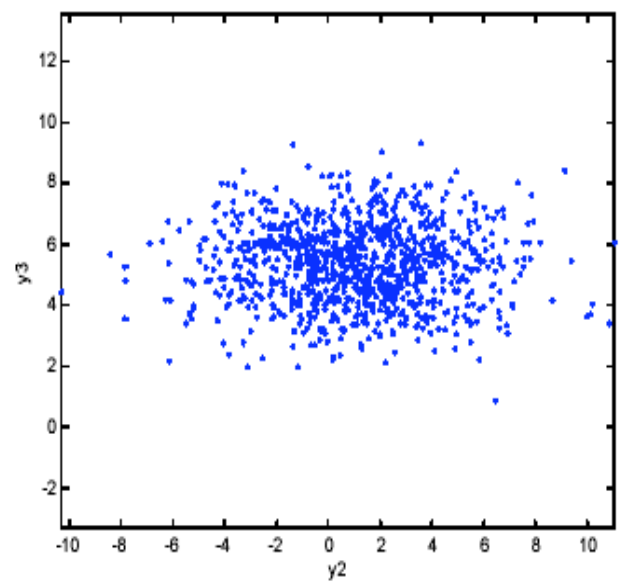
$$\mu = [0\ 5\ 2]^T \text{ and } \Sigma = \begin{bmatrix} 25 & -1 & 7 \\ -1 & 4 & -4 \\ 7 & -4 & 10 \end{bmatrix}$$



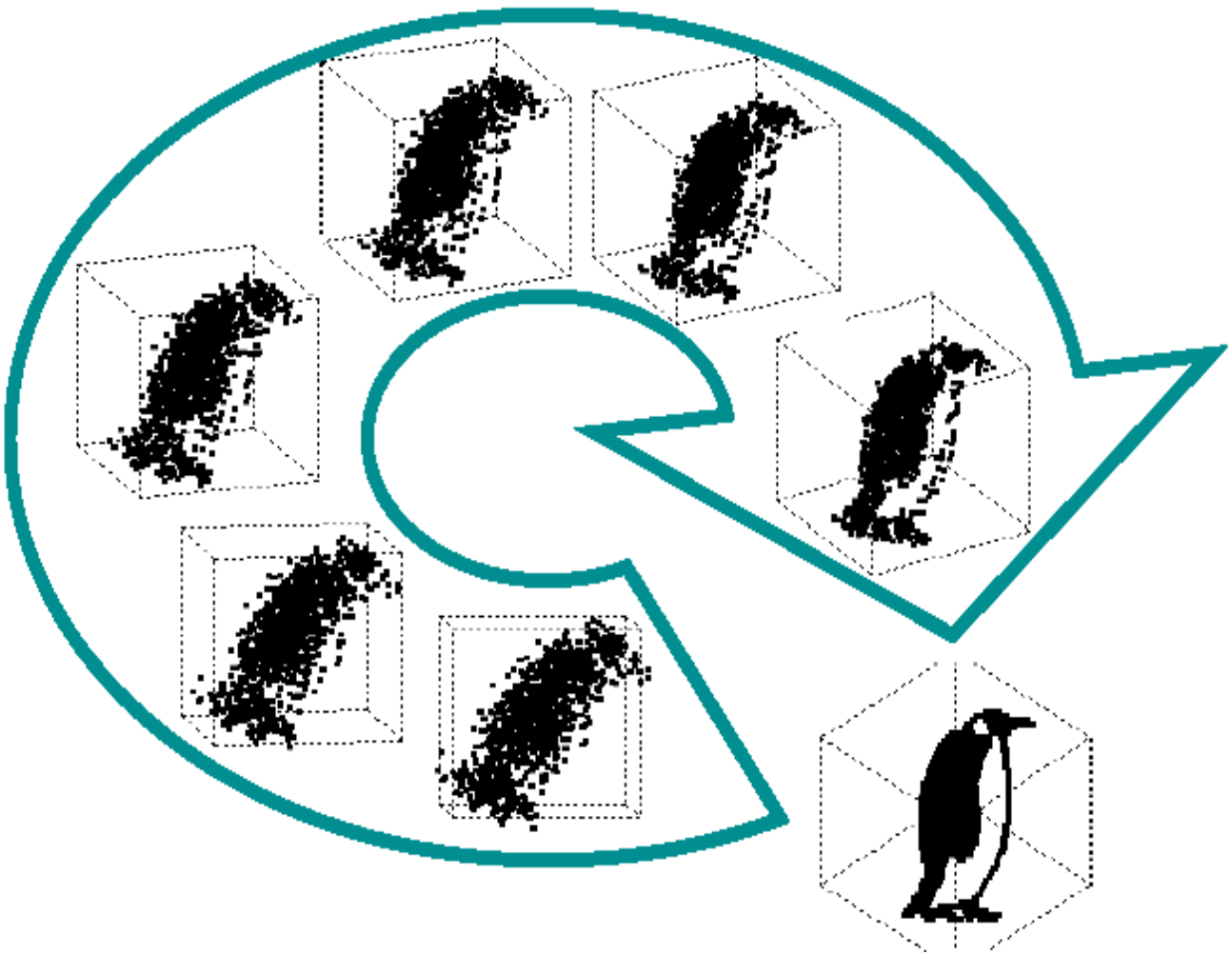
- **The three pairs of principal component projections are shown below**
  - Notice that the first projection has the largest variance, followed by the second projection
  - Also notice that the PCA projections de-correlates the axis

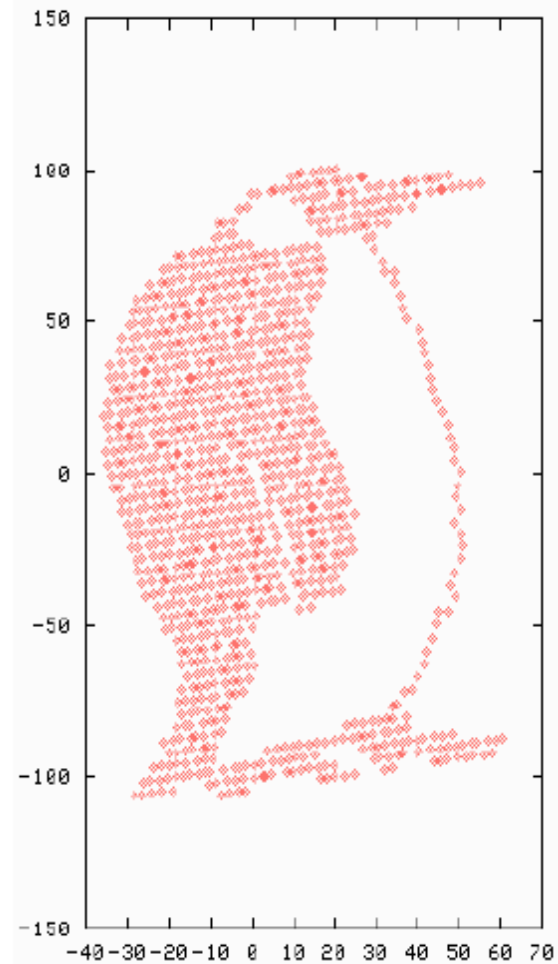




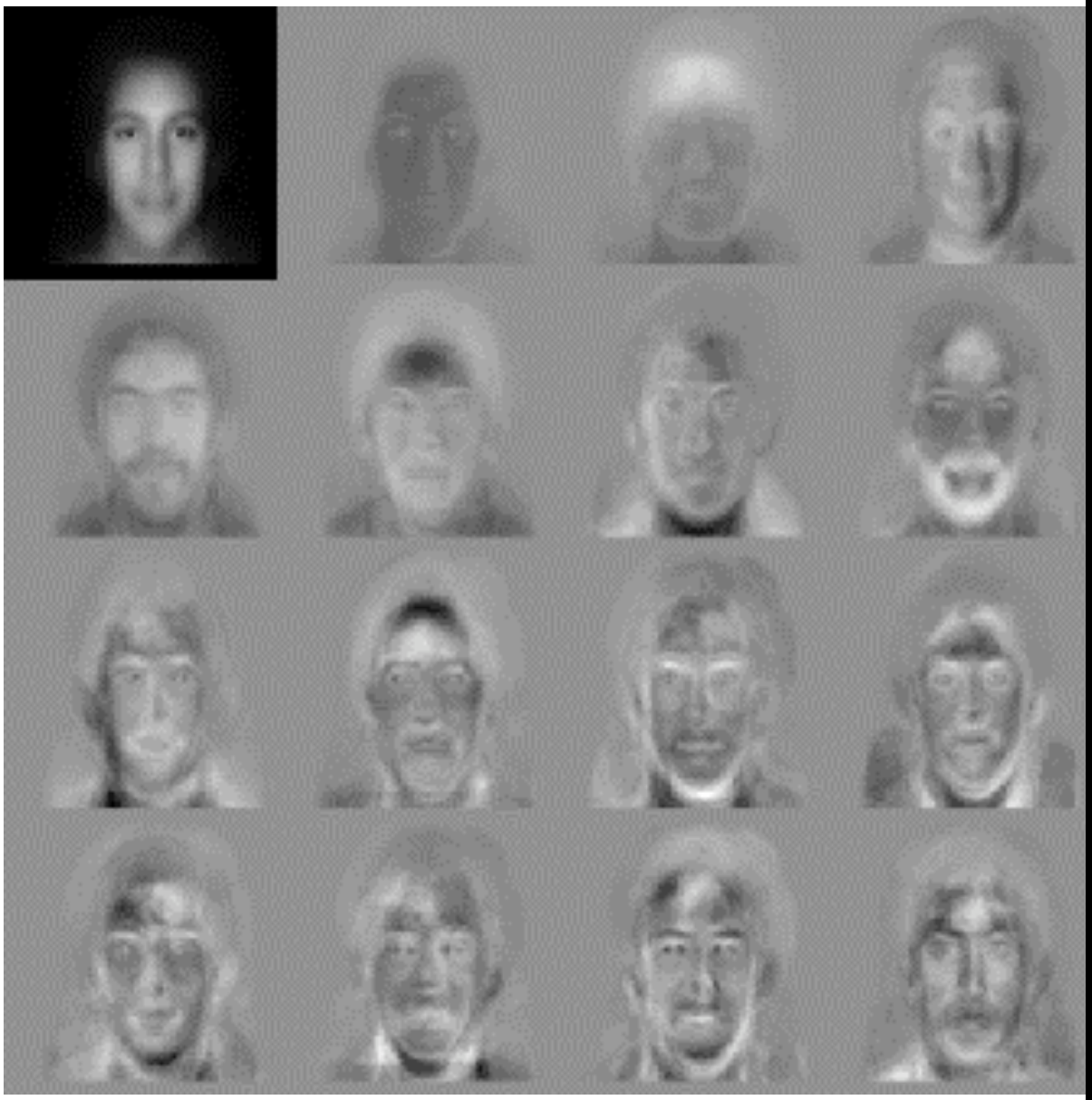


- **This example shows a projection of a three-dimensional data set into two dimensions**
  - Initially, except for the elongation of the cloud, there is no apparent structure in the set of points
  - Choosing an appropriate rotation allows us to unveil the underlying structure. (You can think of this rotation as "walking around" the three-dimensional set, looking for the best viewpoint)
- **PCA can help find such underlying structure. It selects a rotation such that most of the variability within the data set is represented in the first few dimensions of the rotated data**
  - In our three-dimensional case, this may seem of little use
  - However, when the data is highly multidimensional (10's of dimensions), this analysis is quite powerful

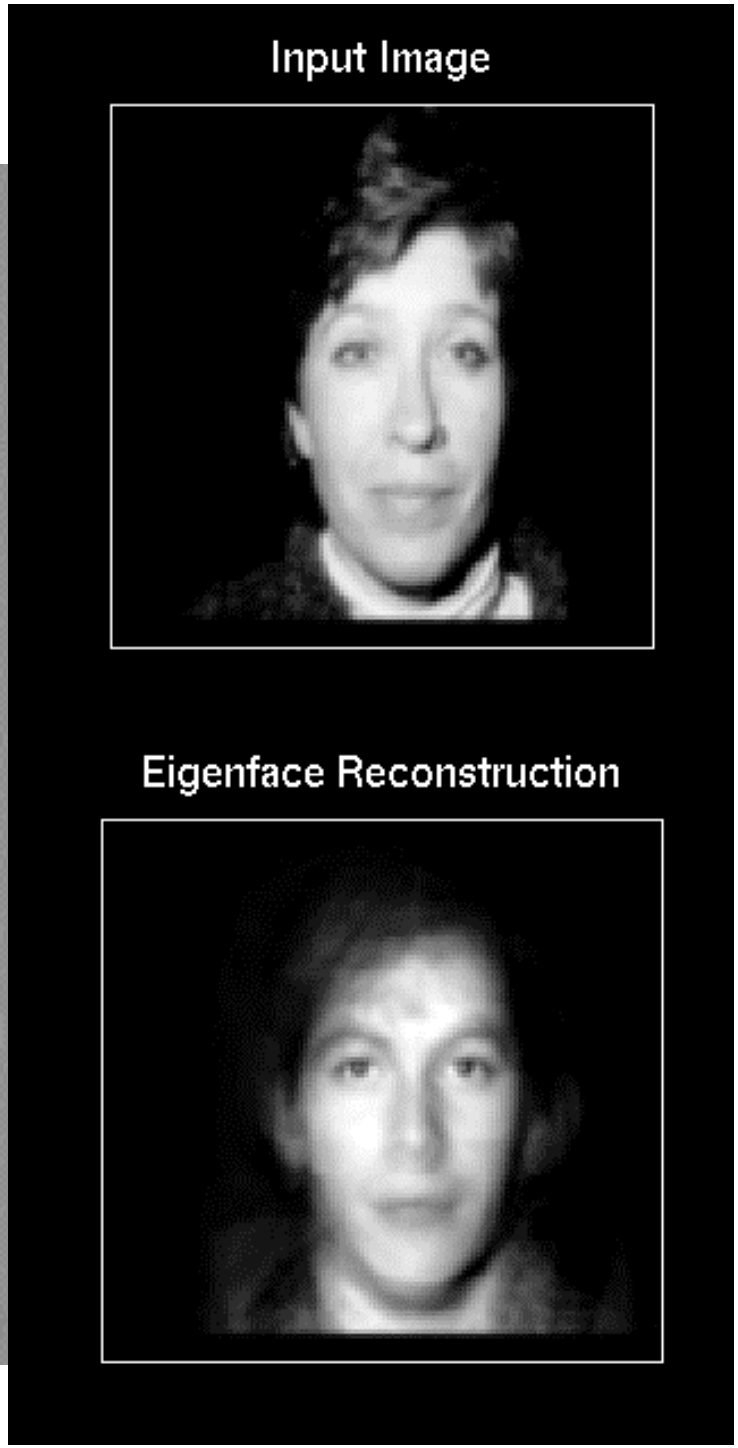




PCA on aligned face images



http://www-white.media.mit.edu/vismod/demos/facerec/basic.html



Extensions: ICA

- Find the 'best' linear basis, minimizing the statistical dependence between projected components

Problem:

***Find c* hidden ind. sources x_i**

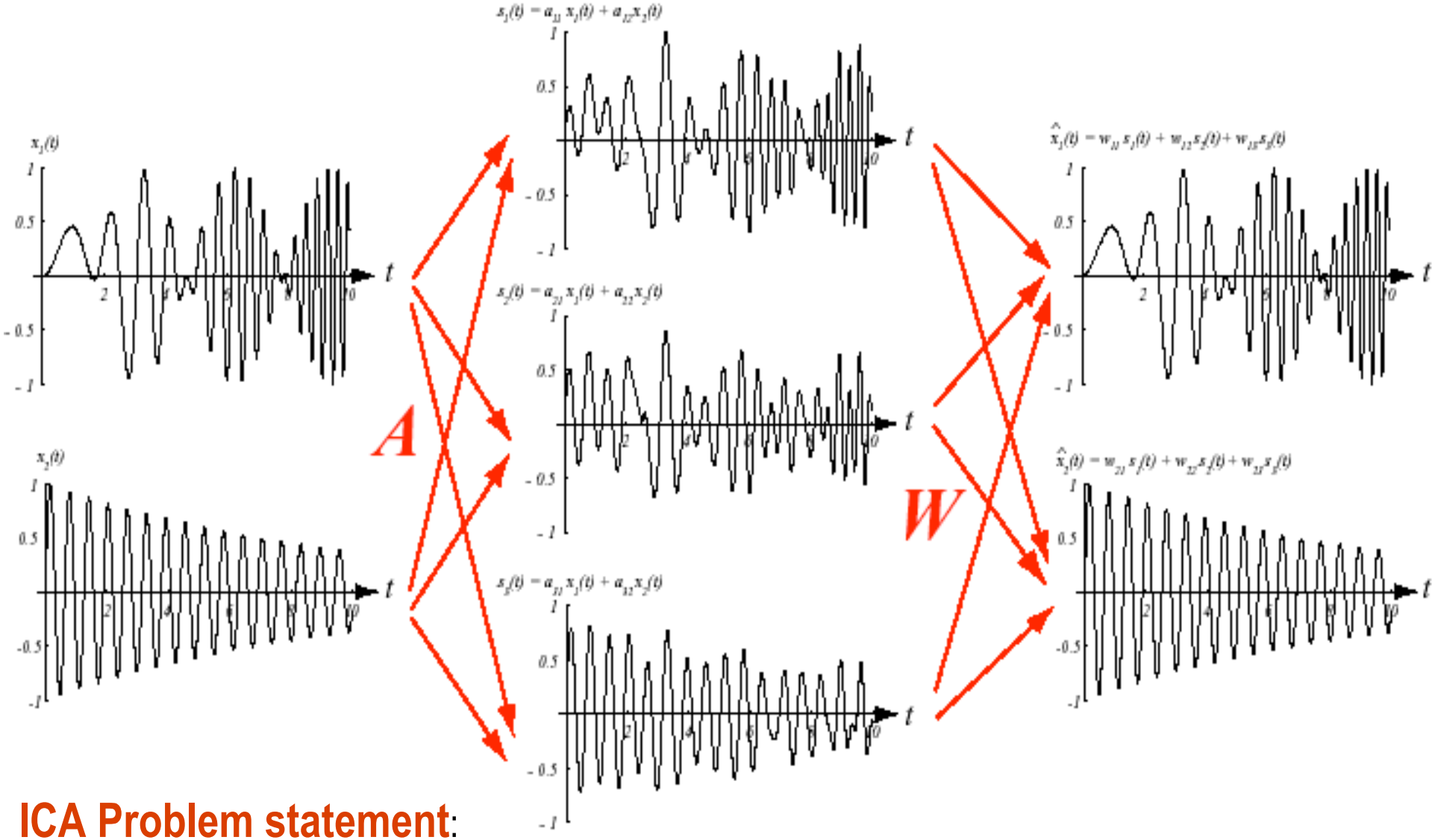
$$p(\mathbf{x}(t)) = \prod_{i=1}^{c} p(x_i(t)).$$

Suppose that a d-dimensional data (or sensor) vector is observed at each moment,

Observation Model

$$\mathbf{y}(t) = \mathbf{A}\mathbf{x}(t),$$

where $\mathbf{A}$ is a $c \times d$ scalar matrix, and below we shall require $d \geq c$.



ICA Problem statement:

Recover the source signals from the sensed signals. More specifically, we seek a real matrix **W** such that **z**(t) is an estimate of **x**(t):

$$\mathbf{z}(t) = \mathbf{W}\mathbf{y}(t) = \mathbf{W}\mathbf{A}\mathbf{x}(t),$$

We approach the determination of $\mathbf{A}$ by maximum-likelihood techniques. We use an estimate of the density, parameterized by $\mathbf{a}$ $\hat{p}(\mathbf{y};\ \mathbf{a})$ and seek the parameter vector $\mathbf{a}$ that minimizes the diffrerence between the source distribution and the estimate. That is, $\mathbf{a}$ is the basis vectors of $\mathbf{A}$ and thus $\hat{p}(\mathbf{y};\ \mathbf{a})$ is an estimate of the $p(\mathbf{y})$.

This difference can be quantified by the Kullback-Liebler divergence:

$$\begin{aligned} D(p(\mathbf{y}), \hat{p}(\mathbf{y};\ \mathbf{a})) &= D(p(\mathbf{y})||\hat{p}(\mathbf{y};\ \mathbf{a})) \\ &= \int p(\mathbf{y}) log \frac{p(\mathbf{y})}{\hat{p}(\mathbf{y};\ \mathbf{a})} d\mathbf{y} \\ &= H(\mathbf{y}) - \int p(\mathbf{y}) log \hat{p}(\mathbf{y};\ \mathbf{a}) d\mathbf{y} \end{aligned} \tag{94}$$

The log-likelihood is

$$l(\mathbf{a}) = \frac{1}{n} \sum_{i=1}^{n} log \hat{p}(\mathbf{x}_i;\ \mathbf{a}). \tag{95}$$

Solve via:

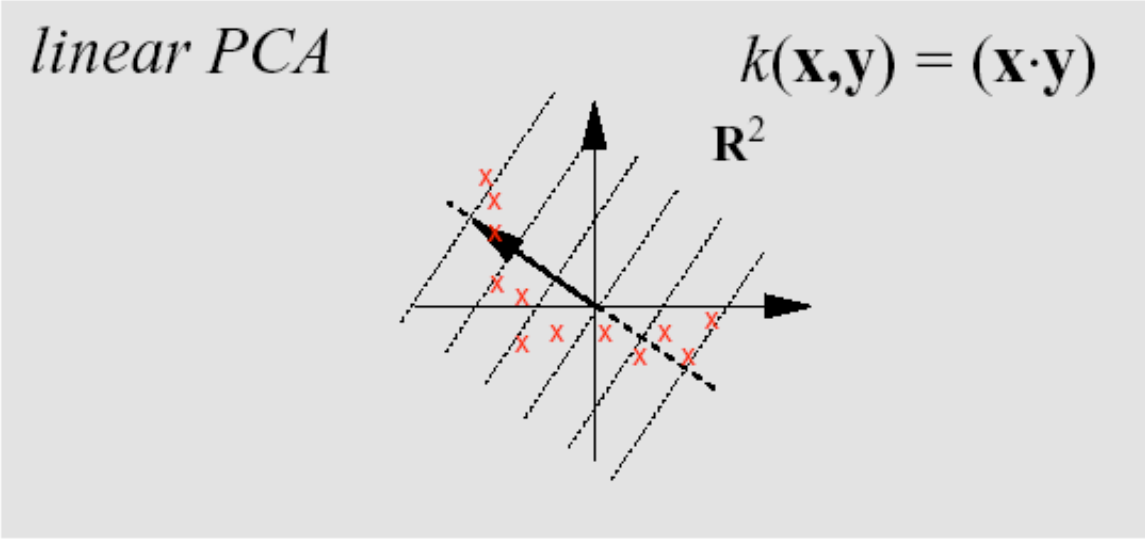
$$\frac{\partial l(\mathbf{a})}{\partial \mathbf{W}} = -\frac{\partial}{\partial \mathbf{W}} D(p(\mathbf{x})||\hat{p}(\mathbf{z})).$$

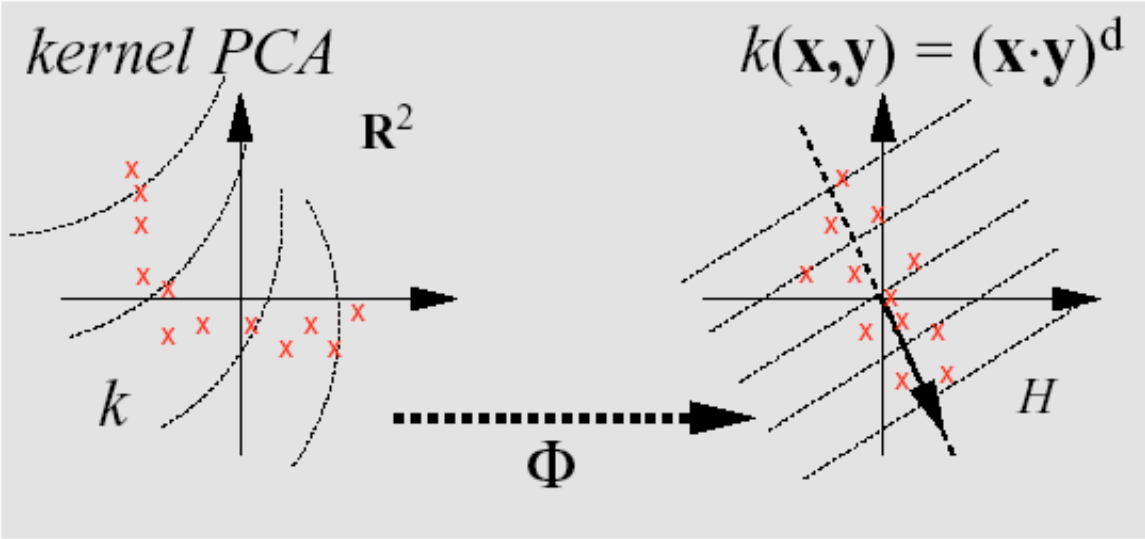
Depending on density assumptions, ICA can have easy or hard solutions

- Gradient approach
- Kurtotic ICA: Two lines matlab code.
  - http://www.cs.toronto.edu/~roweis/kica.html
- `yy` are the mixed measurements (one per column)
- `W` is the unmixing matrix.

```
% W = kica(yy);
xx = sqrtm(inv(cov(yy')))*(yy-repmat(mean(yy,2),1,size(yy,2)));
[W,ss,vv] = svd((repmat(sum(xx.*xx,1),size(xx,1),1).*xx)*xx');
```

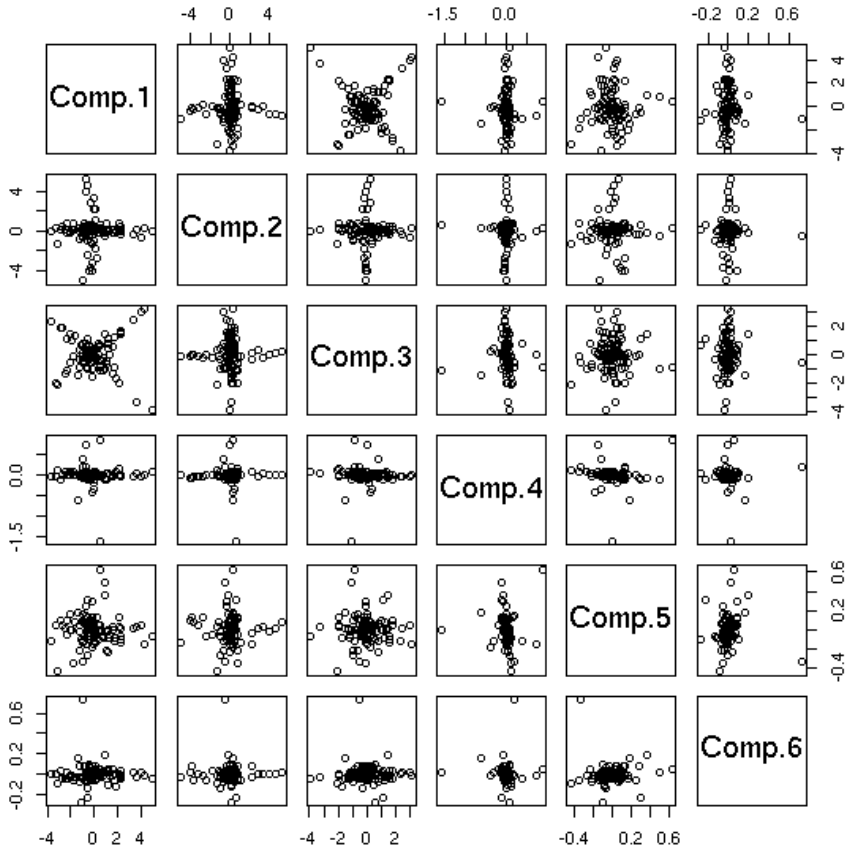
Kernel PCA





Using Non-linear components

- Principal Components Analysis (PCA) attempts to efficiently represent the data by finding orthonormal axes which maximally decorrelate the data
- Makes Following assumptions:
  - · Sources are Gaussian
  - · Sources are independent and stationary (iid)



Extending PCA

Rewriting PCA in terms of dot products

First, we need to remember that the eigenvectors lie in the span of $x_1 \dots x_n$ **Proof**: Substituting equation 4 into 5, we get

$$C\boldsymbol{v} = \frac{1}{m}\sum_{j=1}^{m} x_j x_j^{\mathsf{T}} \boldsymbol{v} = \lambda \boldsymbol{v}$$

Thus,

$$\begin{aligned} \boldsymbol{v} = & \quad \tfrac{1}{m\lambda}\textstyle\sum_{j=1}^{m} x_j x_j^{\mathsf{T}} \boldsymbol{v} \\ = & \quad \tfrac{1}{m\lambda}\textstyle\sum_{j=1}^{m} (x_j \cdot \boldsymbol{v}) x_j \end{aligned}$$

Show that $(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{v} = (\boldsymbol{x}\cdot\boldsymbol{v})\boldsymbol{x}$

$$
\begin{aligned}
(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{v} &= \begin{pmatrix} x_1x_1 & x_1x_2 & \dots & x_1x_M \\ x_2x_1 & x_2x_2 & \dots & x_2x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_Mx_1 & x_Mx_2 & \dots & x_Mx_M \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{pmatrix} \\
&= \begin{pmatrix} x_1x_1v_1 + x_1x_2v_2 + \ldots + x_1x_Mv_M \\ x_2x_1v_1 + x_2x_2v_2 + \ldots + x_2x_Mv_M \\ \vdots \\ x_Mx_1v_1 + x_Mx_2v_2 + \ldots + x_Mx_Mv_M \end{pmatrix} \\
&= \begin{pmatrix} (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_1 \\ (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_2 \\ \vdots \\ (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_M \end{pmatrix} \\
&= \begin{pmatrix} x_1v_1 + x_2v_2 + \ldots + x_Mv_M \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \qquad = (\boldsymbol{x}\cdot\boldsymbol{v})\boldsymbol{x}
\end{aligned}
$$

If we first send the data into another space,

$$\Phi : \mathcal{X} \to \mathcal{H}, \mathrm{x} \mapsto \Phi(\mathrm{x})$$

Then, assuming we can center the data (i.e., $\sum_{k=1}^{m} \Phi(x_k) = 0$ – this is shown in the appendix), we can write the covariance matrix

$$C = \frac{1}{m} \sum_{j=1}^{m} \Phi(x_j)\Phi(x_j)^{\mathsf{T}}$$

Which can be diagonalized with nonnegative eigenvalues satisfying

$$\lambda \boldsymbol{V} = C\boldsymbol{V}$$

$$Cv = \lambda v = \lambda \sum_{i=1}^{m} \alpha_i \Phi(x_i)$$

Substituting

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_j \Phi(x_i) K(x_i, x_j) = m\lambda \sum_{j=1}^{m} \alpha_j \Phi(x_i)$$

where $K(x_i, x_j)$ is an inner-product kernel defined by

$$K(x_i, x_j) = \Phi(x_i)^{\mathsf{T}} \Phi(x_i)$$

To express the relationship entirely in terms of the inner-product kernel, we premultiply both sides by $\Phi(x_k)^{\mathsf{T}}$ and

rewrite the expression as the eigenvalue problem

$$\boldsymbol{K}\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha}$$

Kernel PCA algorithm

$$K_{ij} = k(x_i, x_j)$$

Eigenanalysis

$$(m\lambda)\vec{\alpha} = K\vec{\alpha}$$

$$K = A\Lambda A^{-1}$$

Enforce

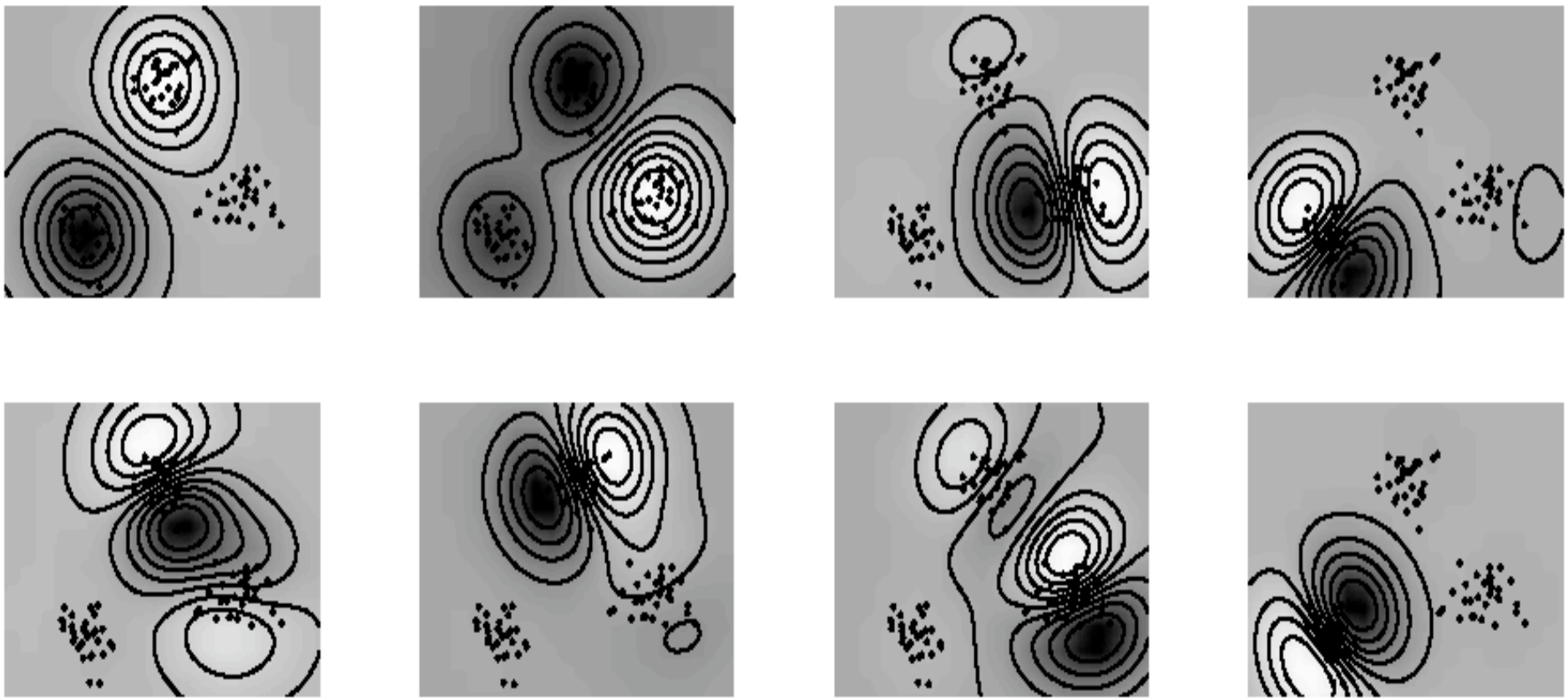
$$\lambda_n \left\| \vec{\alpha}^n \right\|^2 = 1$$

Compute Projections

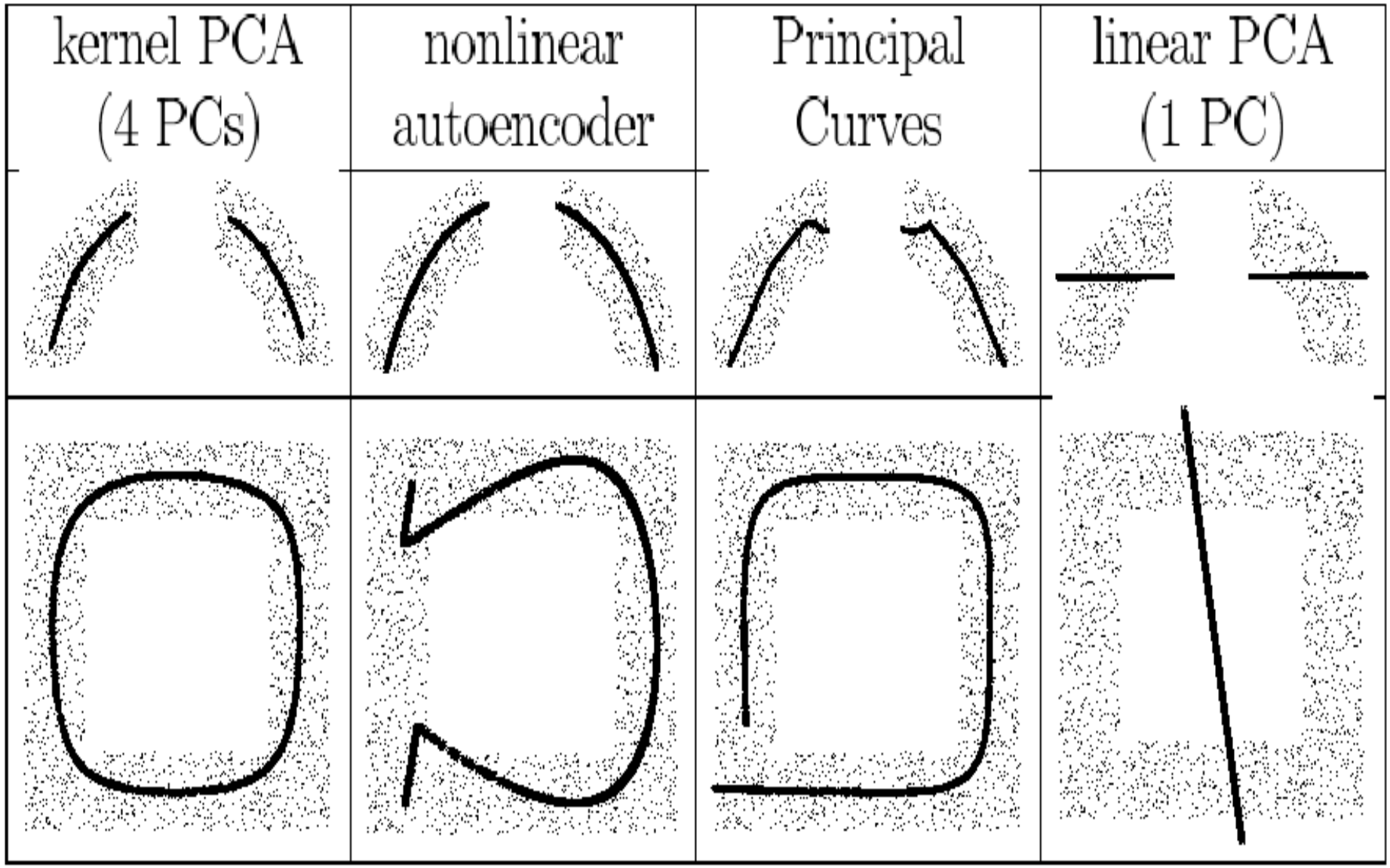
$$y_n = \sum_{i=1}^{m} \alpha_i^j k(x_i, x)$$

Toy Example with Gaussian Kernel

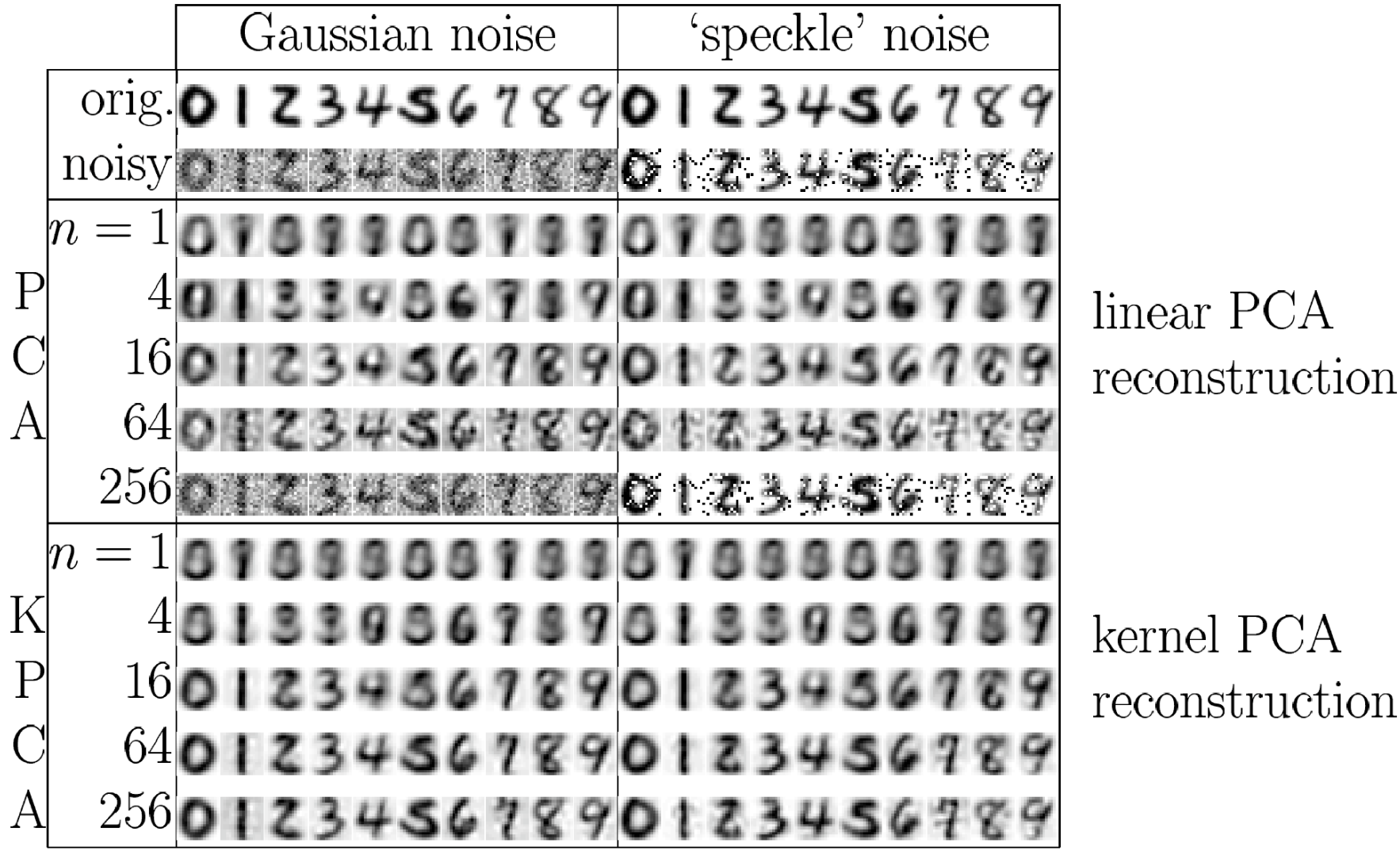
$$k(x, x') = \exp\left(-\|x - x'\|^2\right)$$



Comparison of Different Algorithms



Denoising of USPS Digits



Another application: face modeling [46].

Probabilistic Clustering

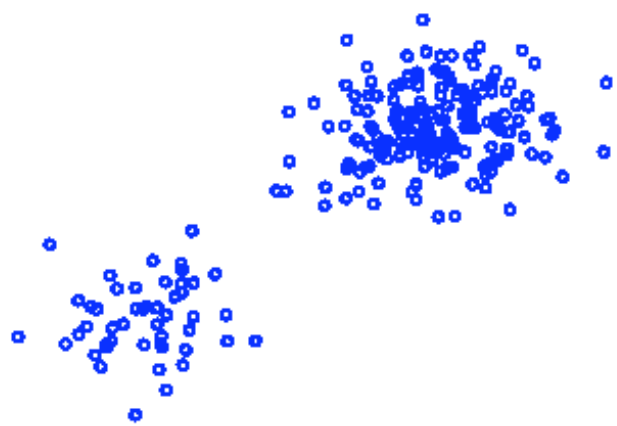
EM, Mixtures of Gaussians, RBFs, etc

Multi-variate density estimation

- A mixture of Gaussians model

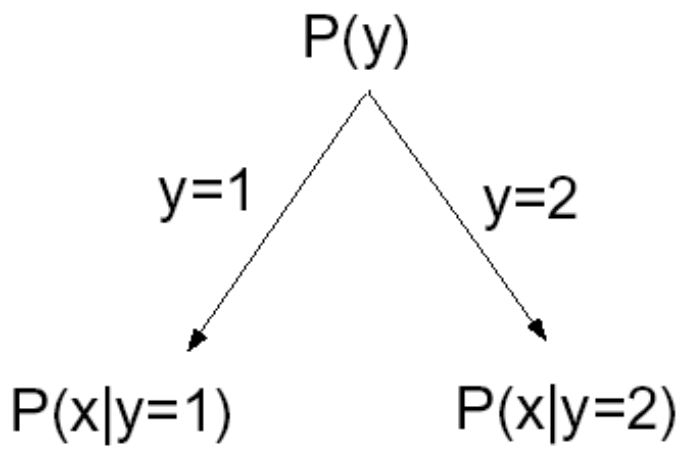
$$p(\mathbf{x}|\theta) = \sum_{i=1}^{k} p_j \, p(\mathbf{x}|\mu_j, \Sigma_j)$$

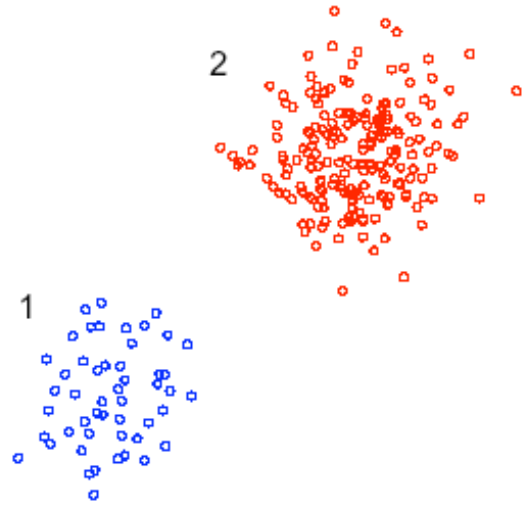
where $\theta = \{p_1, \ldots, p_k, \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k\}$ contains all the parameters of the mixture model. $\{p_j\}$ are known as *mixing proportions or coefficients*.



Mixture density

- Data generation process:





$$
\begin{aligned}
p(\mathbf{x}|\theta) &= \sum_{j=1,2} P(y=j) \cdot p(\mathbf{x}|y=j) \qquad \text{(generic mixture)} \\
&= \sum_{j=1,2} p_j \cdot p(\mathbf{x}|\mu_j, \Sigma_j) \qquad \text{(mixture of Gaussians)}
\end{aligned}
$$

- Any data point $\mathbf{x}$ could have been generated in two ways

Mixture density

- If we are given just $\mathbf{x}$ we don't know which mixture component this example came from

$$p(\mathbf{x}|\theta) = \sum_{j=1,2} p_j p(\mathbf{x}|\mu_j, \Sigma_j)$$

- We can evaluate the posterior probability that an observed $\mathbf{x}$ was generated from the first mixture component

$$\begin{aligned} P(y=1|\mathbf{x},\theta) &= \frac{P(y=1)\cdot p(\mathbf{x}|y=1)}{\sum_{j=1,2} P(y=j)\cdot p(\mathbf{x}|y=j)} \\ &= \frac{p_1\, p(\mathbf{x}|\mu_1,\Sigma_1)}{\sum_{j=1,2} p_j\, p(\mathbf{x}|\mu_j,\Sigma_j)} \end{aligned}$$

But only if we are given the distributions and prior

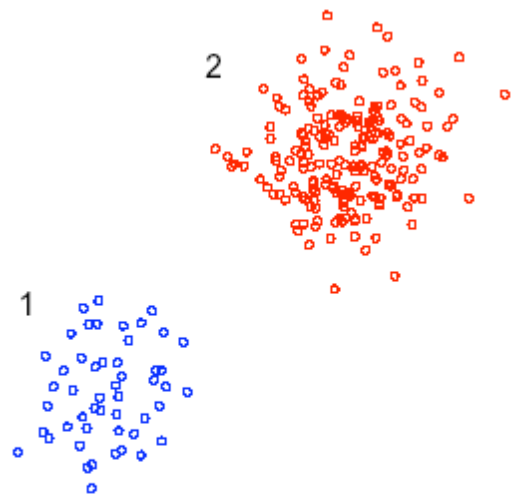
- This solves a *credit assignment* problem

Mixture density estimation

- Suppose we want to estimate a two component mixture of Gaussians model.

$$p(\mathbf{x}|\theta) = p_1\, p(\mathbf{x}|\mu_1, \Sigma_1) + p_2\, p(\mathbf{x}|\mu_2, \Sigma_2)$$

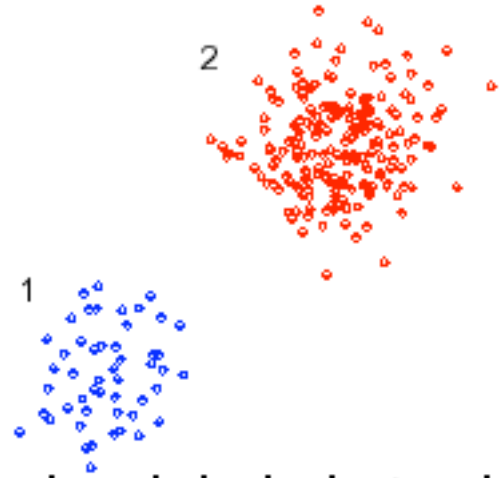
- If each example $\mathbf{x}_i$ in the training set were labeled $y_i = 1, 2$ according to which mixture component (1 or 2) had generated it, then the estimation would be easy.



- Labeled examples $\Rightarrow$ no credit assignment problem

Mixture density estimation

When examples are already assigned to mixture components (labeled), we can estimate each Gaussian independently



- If $\hat{n}_j$ is the number of examples labeled j, then for each $j = 1, 2$ we set

$$\hat{p}_j \leftarrow \frac{\hat{n}_j}{n}$$

$$\hat{\mu}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i:y_i=j} \mathbf{x}_i$$

$$\hat{\Sigma}_j \leftarrow \frac{1}{\hat{n}_j} \sum_{i:y_i=j} (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T$$

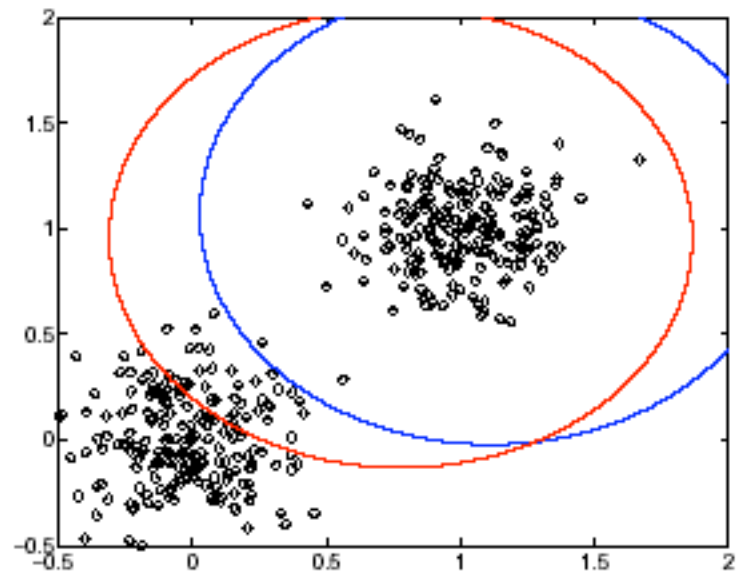
Mixture density estimation: credit assignment

- Of course we don't have such labels ... but we can guess what the labels might be based on our current mixture distribution

- We get soft labels or posterior probabilities of which Gaussian generated which example:

$$\hat{p}(j|i) \leftarrow P(y_i = j|\mathbf{x}_i, \theta)$$

where $\sum_{j=1,2} \hat{p}(j|i) = 1$ for all $i = 1, \ldots, n$.



- When the Gaussians are almost identical (as in the figure), $\hat{p}(1|i) \approx \hat{p}(2|i)$ for almost any available point $\mathbf{x}_i$.

  Even slight differences can help us determine how we should modify the Gaussians.

The EM algorithm

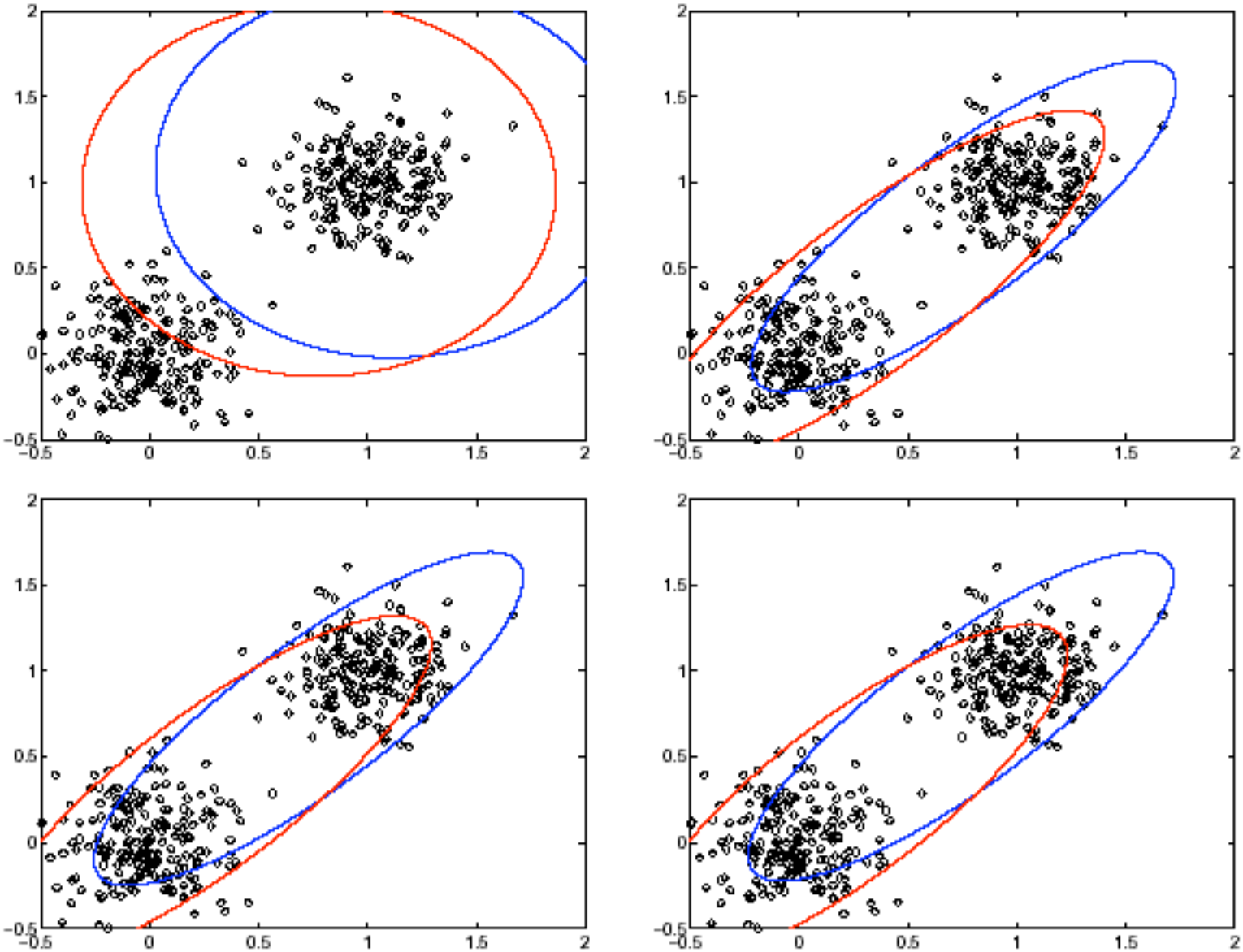
E-step: softly assign examples to mixture components

$\hat{p}(j|i) \leftarrow P(y_i = j|\mathbf{x}_i, \theta)$, for all $j = 1, 2$ and $i = 1, \ldots, n$

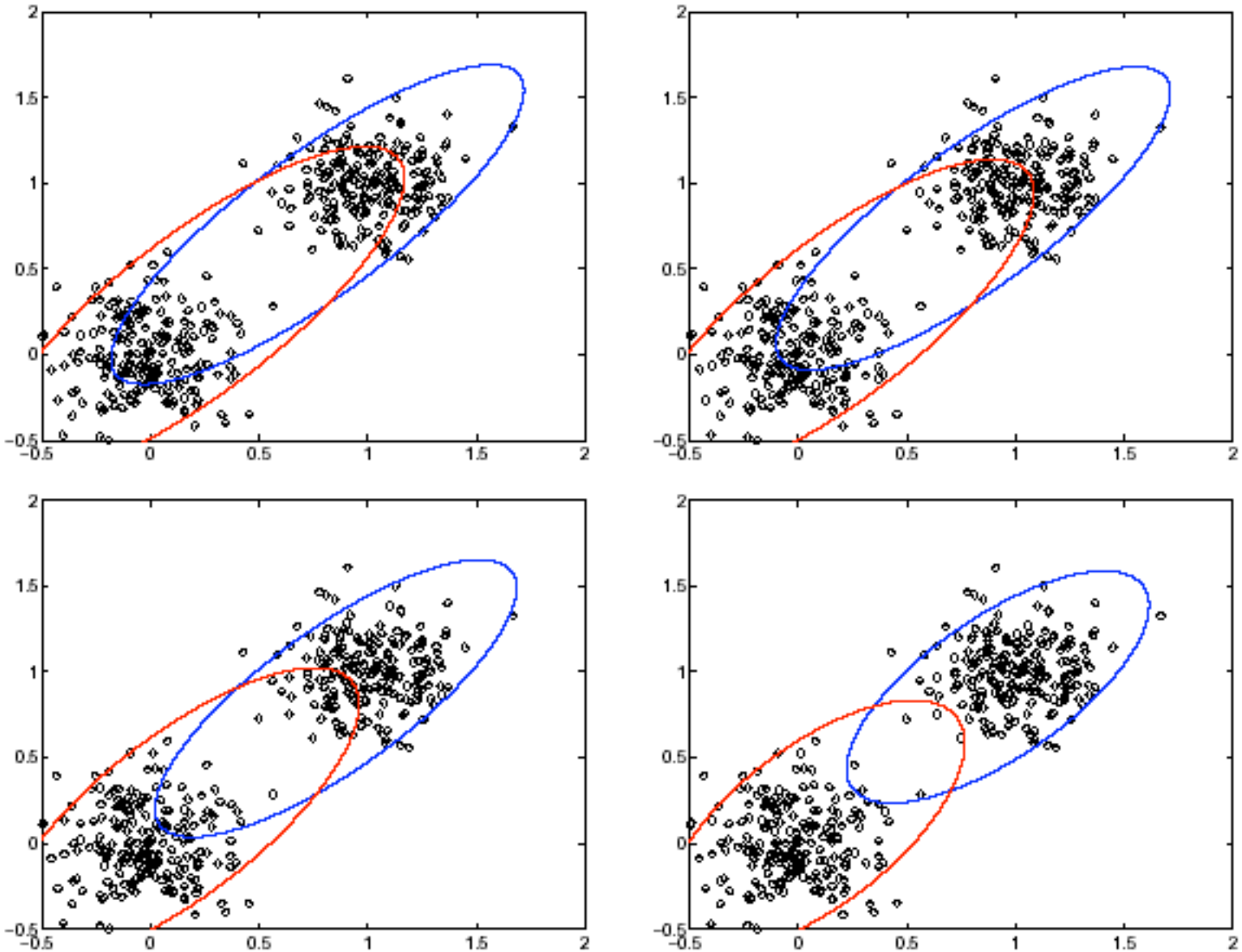
M-step: re-estimate the parameters (separately for the two Gaussians) based on the soft assignments.

$$\begin{aligned}
\hat{n}_j &\leftarrow \sum_{i=1}^{n} \hat{p}(j|i) = \text{Soft \# of examples labeled } j \\
\hat{p}_j &\leftarrow \frac{\hat{n}_j}{n} \\
\hat{\mu}_j &\leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j|i)\, \mathbf{x}_i \\
\hat{\Sigma}_j &\leftarrow \frac{1}{\hat{n}_j} \sum_{i=1}^{n} \hat{p}(j|i)\, (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^T
\end{aligned}$$

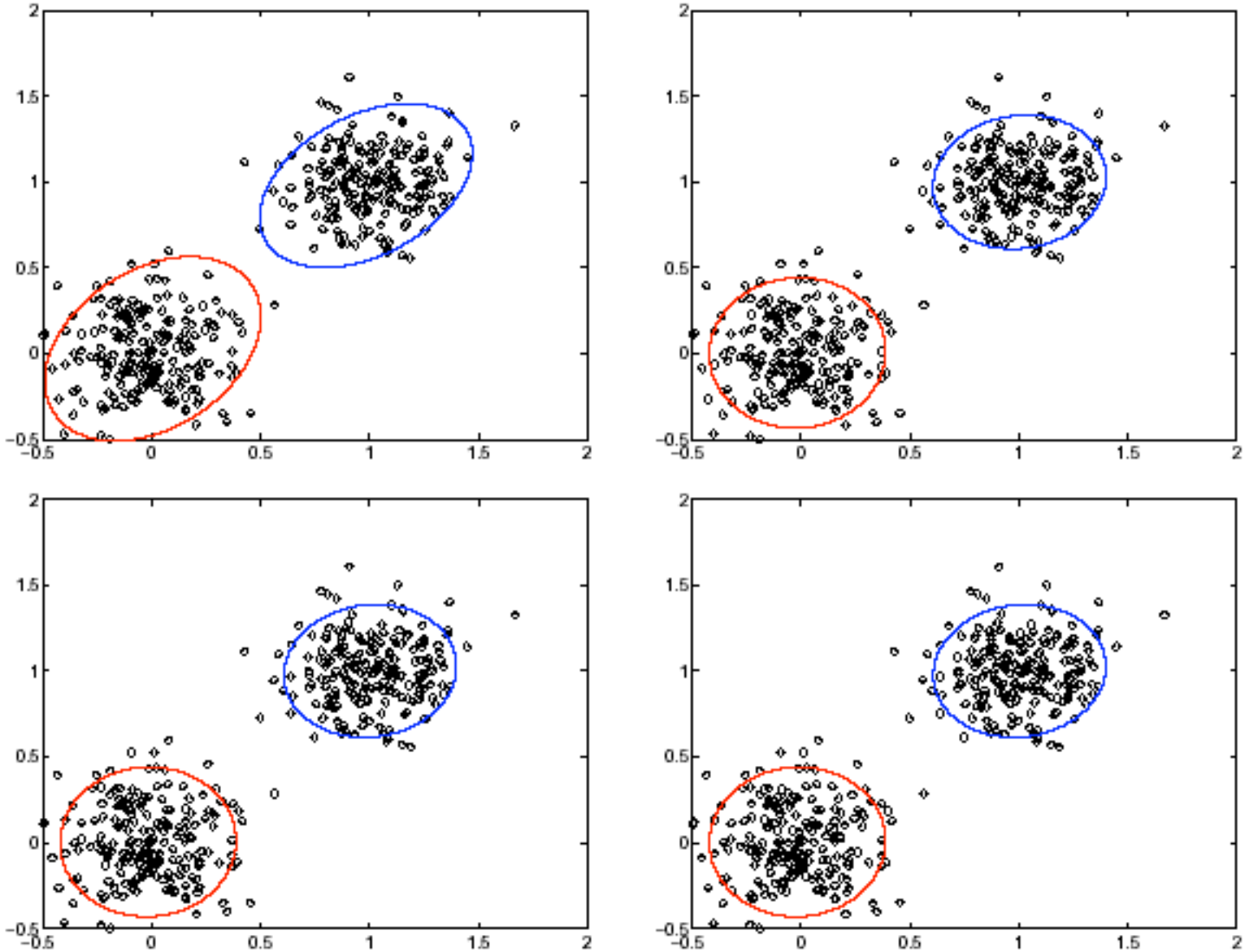
Mixture density estimation: example



Mixture density estimation



Mixture density estimation

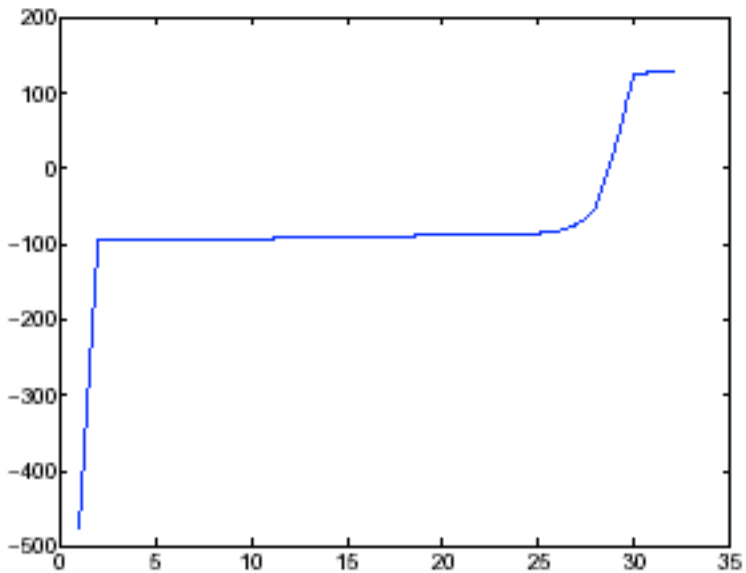


The EM-algorithm

- Each iteration of the EM-algorithm *monotonically* increases the (log-)likelihood of the n training examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

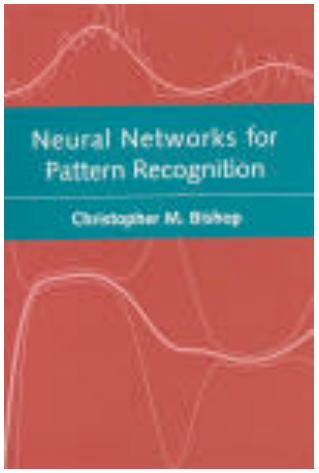
$$\log p(\text{ data } |\theta) = \sum_{i=1}^{n} \log \left(\overbrace{p_1\, p(\mathbf{x}_i|\mu_1, \Sigma_1) + p_2\, p(\mathbf{x}_i|\mu_2, \Sigma_2)}^{p(\mathbf{x}_i|\theta)} \right)$$

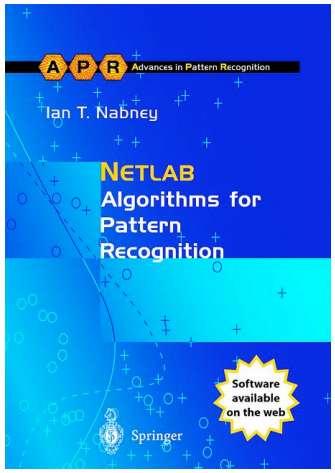
where $\theta = \{p_1, p_2, \mu_1, \mu_2, \Sigma_1, \Sigma_2\}$ contains all the parameters of the mixture model.

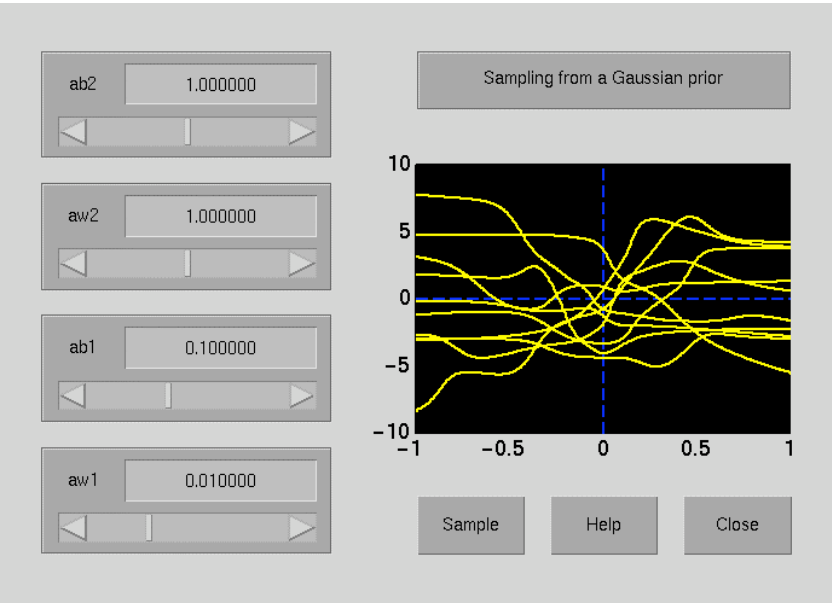


NETLAB

http://www.ncrg.aston.ac.uk/netlab/

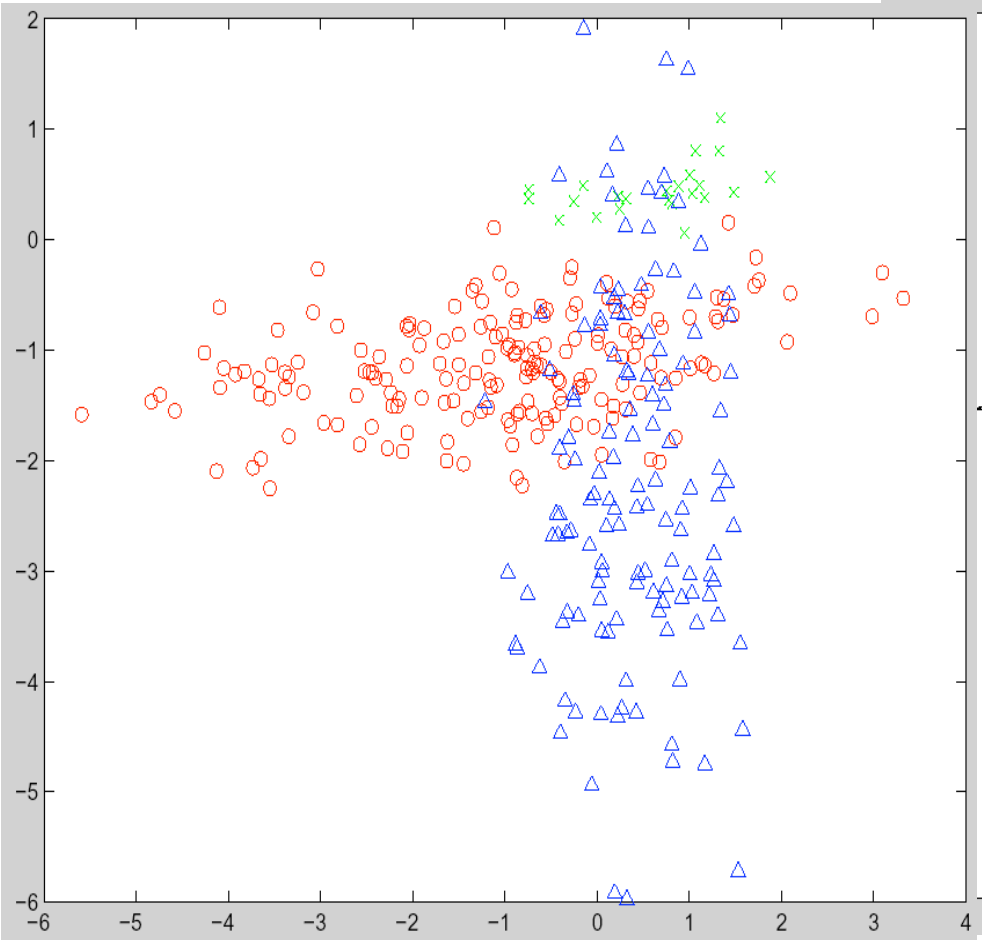






* PCA
* Mixtures of probabilistic PCA
* Gaussian mixture model with EM training
* Linear and logistic regression with IRLS
* Multi-layer perceptron with linear, logistic and softmax outputs and error functions
* Radial basis function (RBF) networks with both Gaussian and non-local basis functions
* Optimisers, including quasi-Newton methods, conjugate gradients and scaled conj grad.
* Multi-layer perceptron with Gaussian mixture outputs (mixture density networks)
* Gaussian prior distributions over parameters for the MLP, RBF and GLM including multiple hyper-parameters
* Laplace approximation framework for Bayesian inference (evidence procedure)
* Automatic Relevance Determination for input selection
* Markov chain Monte-Carlo including simple Metropolis and hybrid Monte-Carlo
* K-nearest neighbour classifier
* K-means clustering
* Generative Topographic Map
* Neuroscale topographic projection
* Gaussian Processes
* Hinton diagrams for network weights
* Self-organising map

Data sampled from
Mixture of 3 Gaussians



Spectral Clustering