
COMP 435 Spring 2010

Lecture 05

Machine Learning I

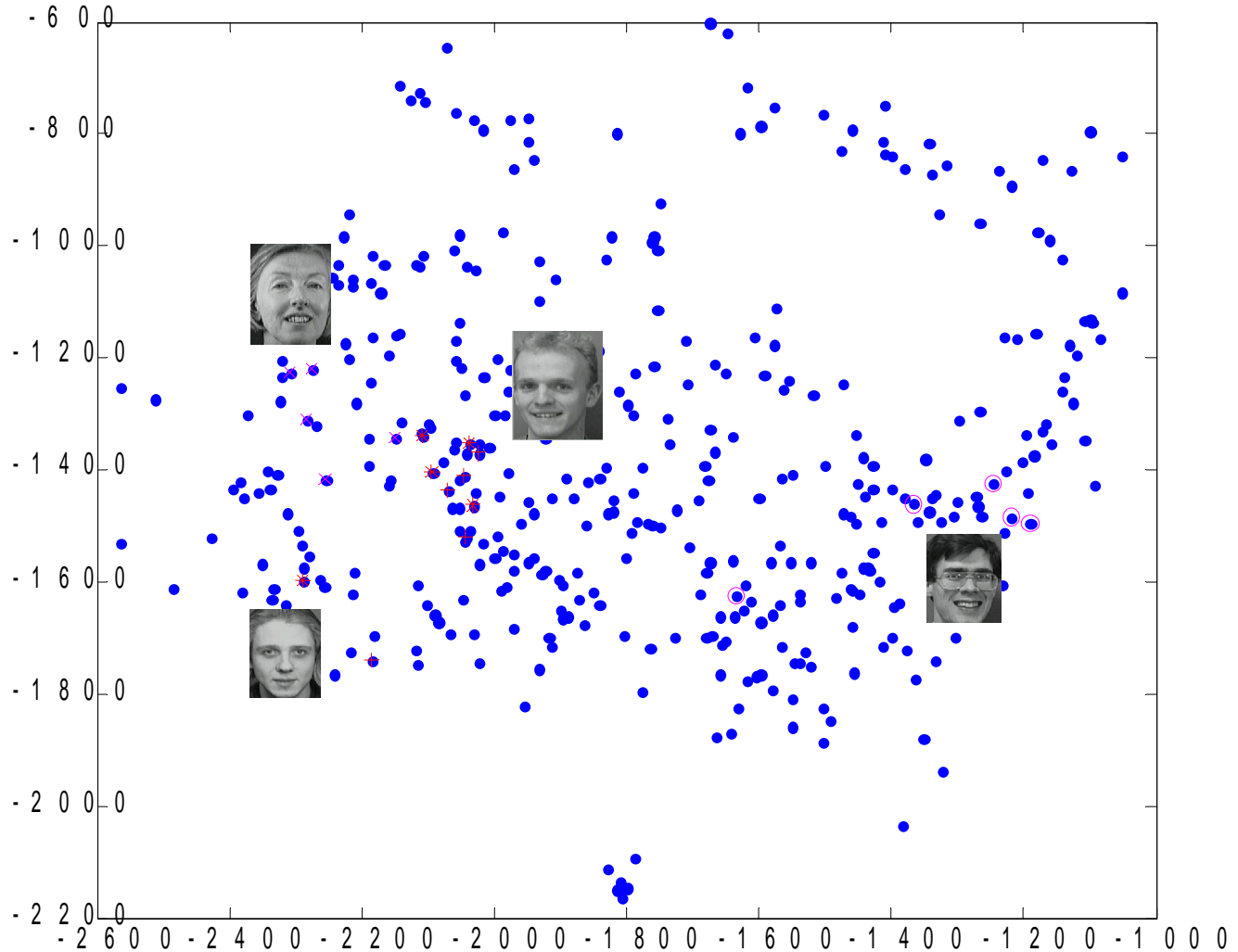
Re-Cap: Lecture 4

- Principal Component Analysis - PCA
 - » Mathematical Formulation
 - » Numerical Solution & Matlab Implementation
- Discrete Cosine Transform (DCT)
 - » Formulation
 - » Numerical Solution & Matlab Implementation
- Linear Discriminant Analysis (LDA), aka Fisher Discriminant Analysis
 - » Formulation
 - » Numerical Solution & Matlab Implementation

Why Transform ?

- Project (hi-dimensional) sensor data to (low-dimensional) feature space, s.t. the patterns belonging to the same class are closer to each other

$$Y=AX$$



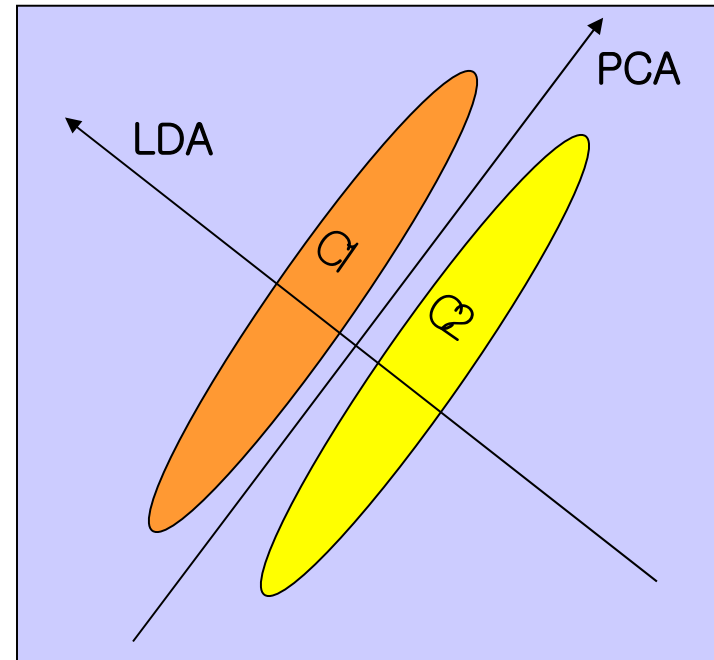
PCA vs LDA

- **PCA**

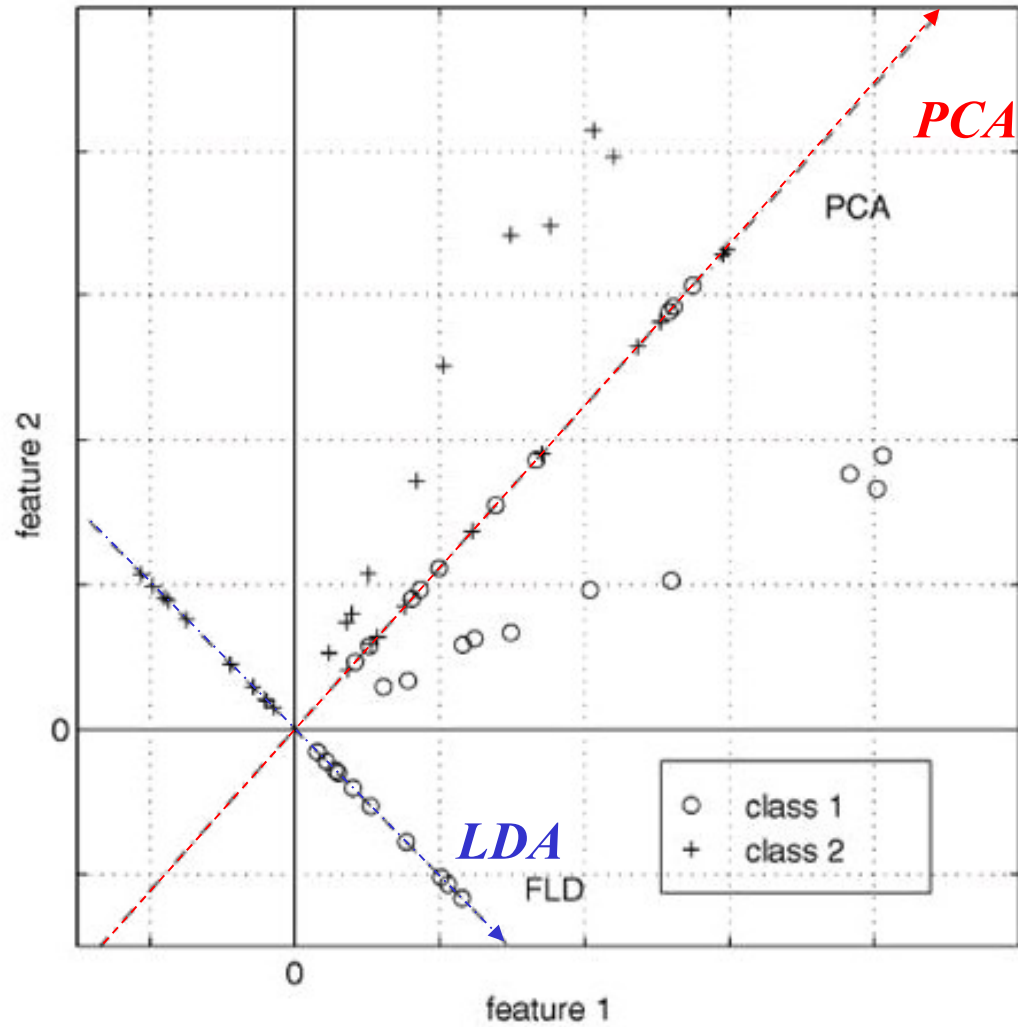
- Finds the best approximation of the data in a low dimensional space
- Computed by eigen decomposition of the covariance matrix of the data

- **LDA**

- Finds the projection that maximizes the inter-class scatter over intra-class scatter
- Computed by a generalized eigen problem

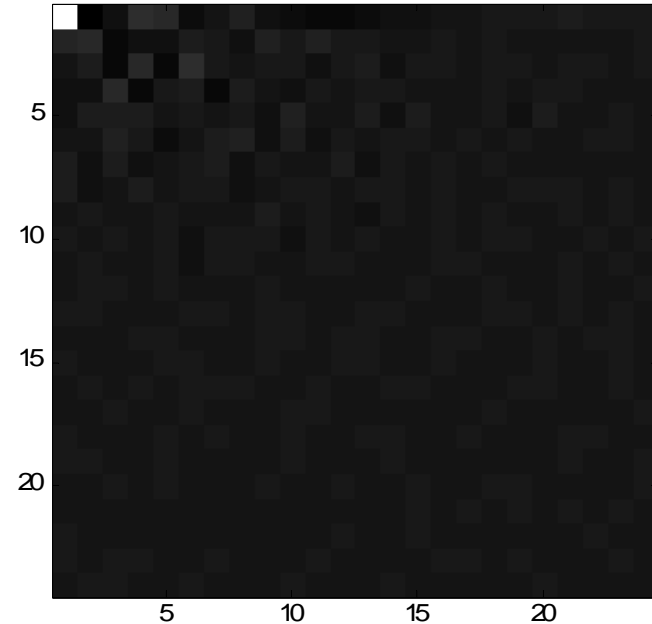
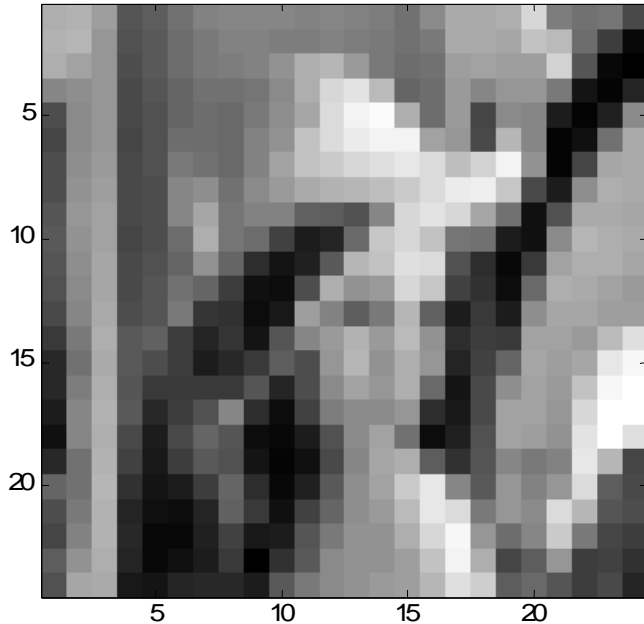


PCA vs LDA



DCT

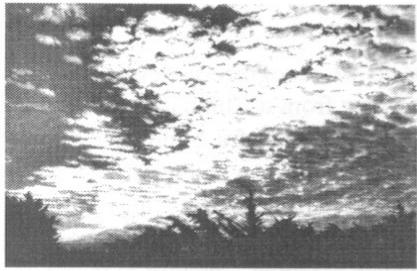
- A good approximation of PCA for a large class of signals
 - Matlab: for an $h \times w$ points DCT: `im2=dct2(im, h, w)`



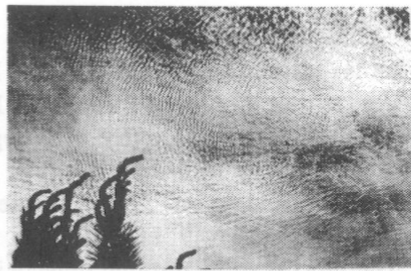
- Notice that small number of co-efficients can represent the original signal

Pattern Recognition in Biometric Problems

Patterns



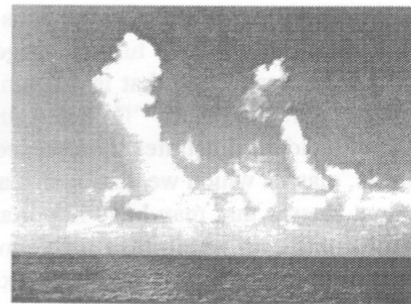
(a)



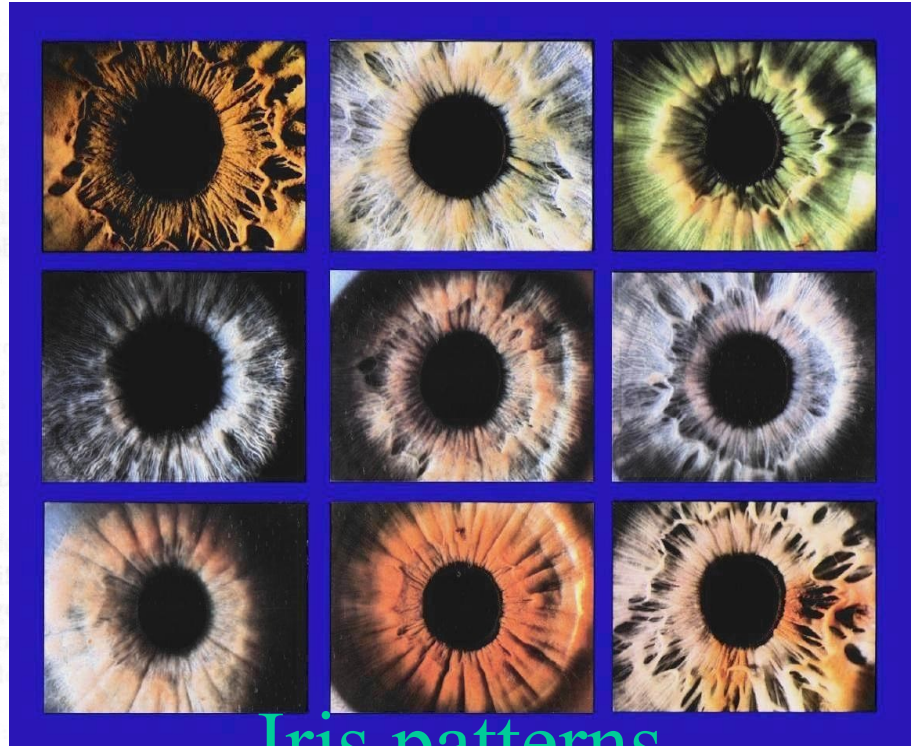
(b)



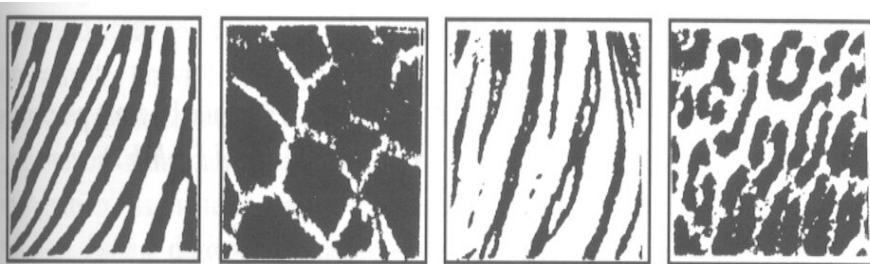
(c) Cloud patterns



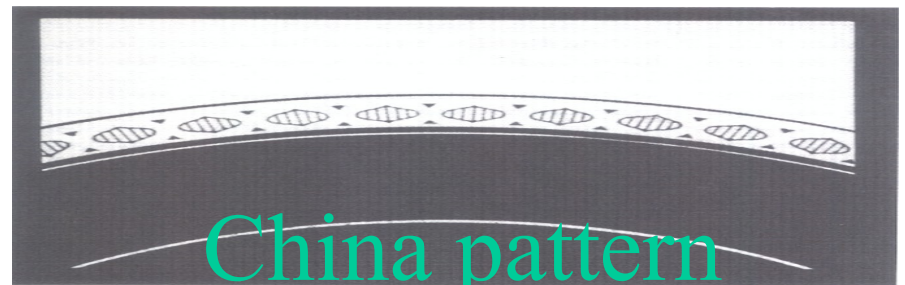
(d)



Iris patterns



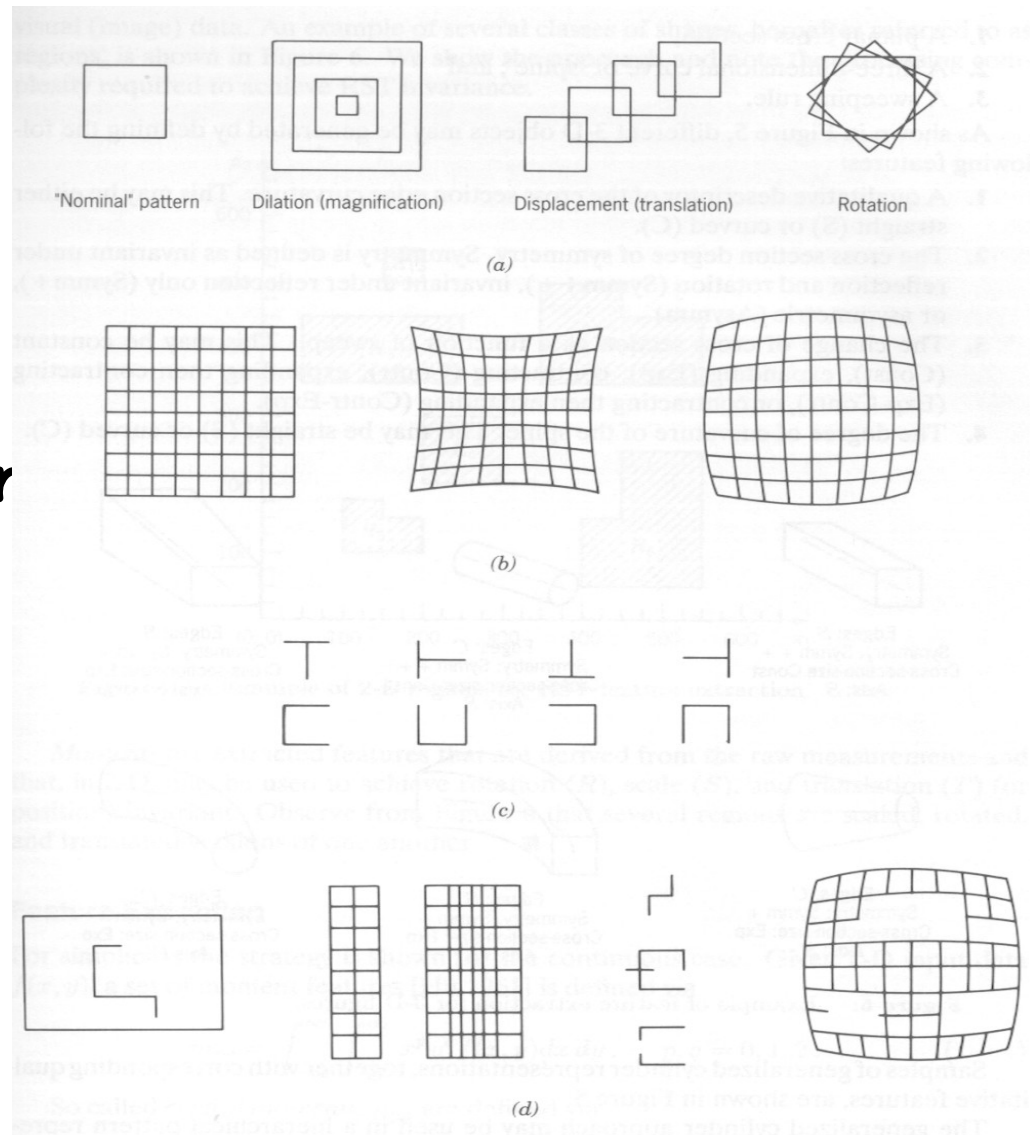
Animal coat patterns



China pattern

Intra-Class Pattern Distortions

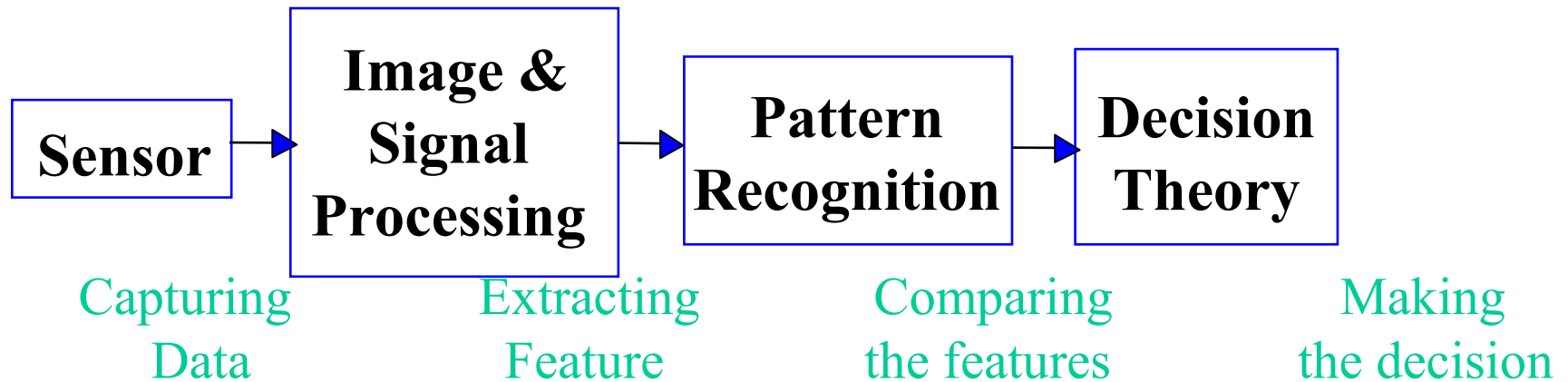
- Translation
- Rotation
- Scaling
- Nonlinear distortions



What is Pattern Recognition ?

- **Given a template, recognizes its most likely label, or answering “who this template it belongs ?”**
- **Typically having two phases**
 - In training phases, training data and labels are collected in the enrollment process, and then features/templates are extracted, and decision boundaries trained.
 - In recognition phase, given an unknown template, trying to answer which label is the closest.

PR System Diagram



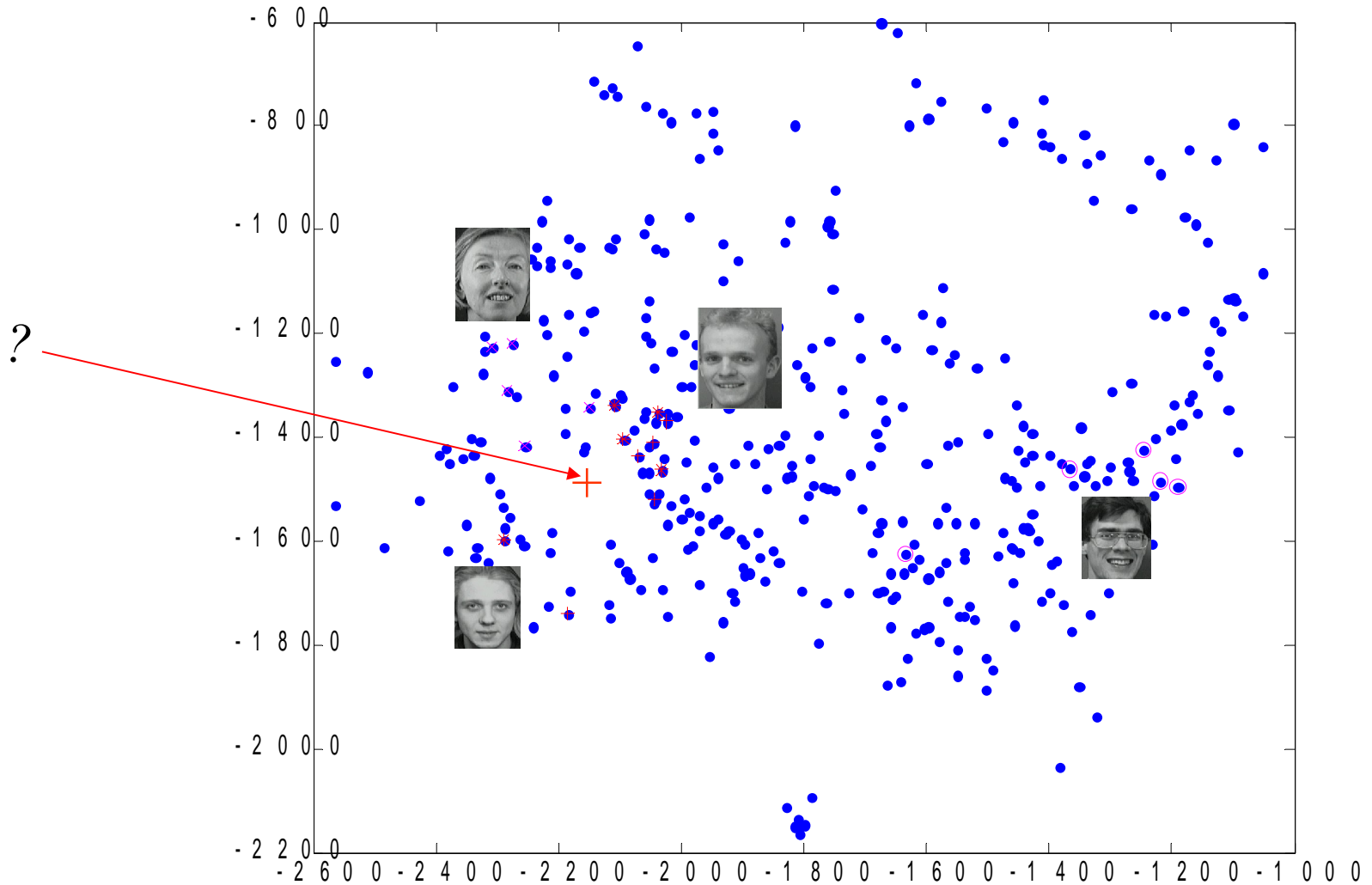
Machine Learning

- **The task of machine learning is to**
 - Have a mathematical representation of relationship between templates and labels
 - Then for a given template, assign the most likely label.
 - There's a geometrical and statistical view to this problem
 - » Geometrical: decision boundaries, nearest neighbours
 - » Statistical: modeling template as some distribution condition on labels.
- **In this class, we will cover 3 types**
 - NN classifier
 - Bayesian Classifier
 - SVM

Nearest Neighbor Classifier

NN Classifier

- Given Unknown Pattern, what is its label ?



1-NN Classifier

- 1-Nearest Neighbor Classifier:
 - ▶ The labelled dataset $\mathcal{D}_n = \{(\mathbf{x}_1, \theta_1), \dots, (\mathbf{x}_n, \theta_n)\}$
 - ▶ where $\theta_i \in \{\omega_1, \dots, \omega_c\}$ is the class label for \mathbf{x}_i .
 - ▶ The input \mathbf{x} and its nearest neighbor \mathbf{x}'

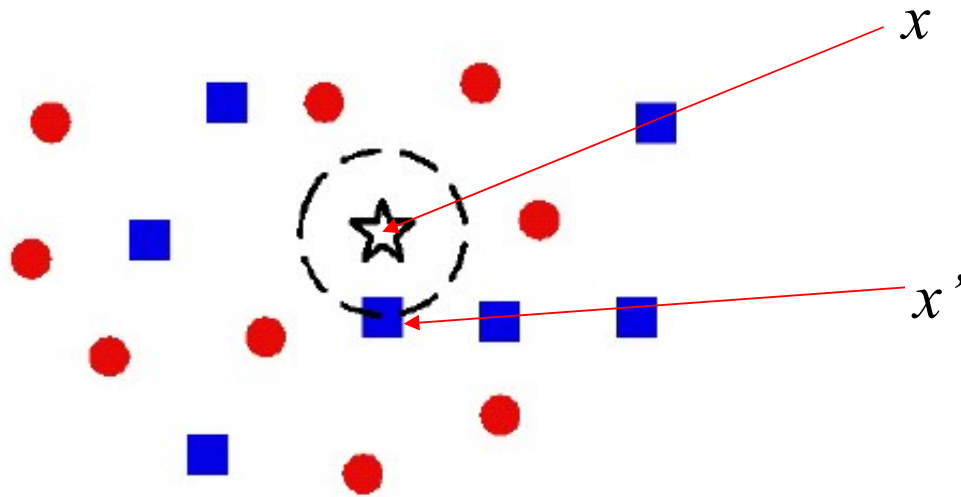
$$\mathbf{x}' = \arg \min_{\mathbf{x}_i \in \mathcal{D}_n} \text{dist}(\mathbf{x}, \mathbf{x}_i)$$

- ▶ The 1-nearest-neighbor decision rule

- Then the label of \mathbf{x} is given as,

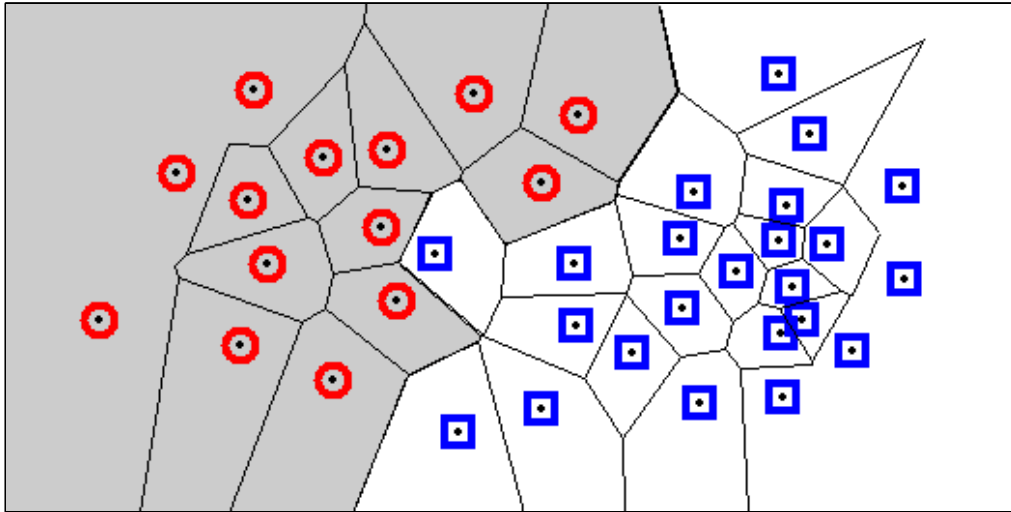
$$\theta(\mathbf{x}) = \theta(\mathbf{x}')$$

1-NN Illustration



Then, x is recognized as belonging to the blue class

1-NN Decision Regions



Each cell contains one sample, and every location within the cell is closer to that sample than to any other sample.

A Voronoi diagram divides the space into such cells.

Every query point will be assigned the classification of the sample within that cell. The *decision boundary* separates the class regions based on the 1-NN decision rule.

Knowledge of this boundary is sufficient to classify new points.

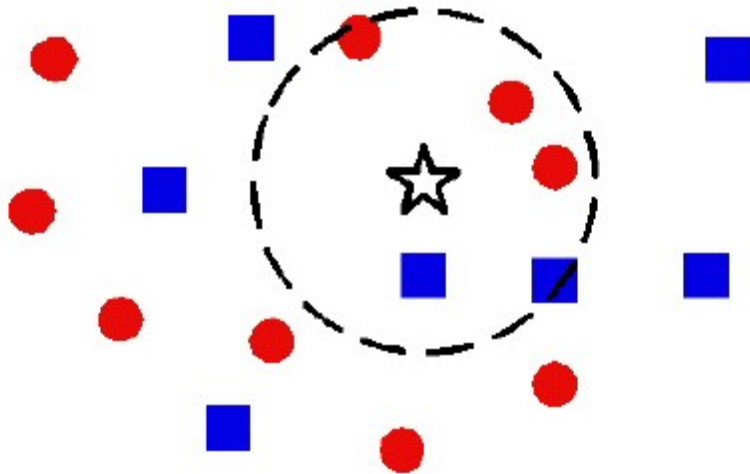
The boundary itself is rarely computed; many algorithms seek to retain only those points necessary to generate an identical boundary.

K-NN Classifier

- K-Nearest Neighbor classifier:

- ▶ The labelled dataset $\mathcal{D}_n = \{(\mathbf{x}_1, \theta_1), \dots, (\mathbf{x}_n, \theta_n)\}$
- ▶ where $\theta_j \in \{\omega_1, \dots, \omega_c\}$ is the class label for \mathbf{x}_j .
- ▶ For an input \mathbf{x} , find its k nearest neighbors
- ▶ The k -nearest-neighbor decision rule

$\mathbf{x} \rightarrow \{majority\ class\ label\ of\ the\ k\ nearest\ neighbors\}$



$$\theta(x) = \bullet$$

Distance Metrics in NN Classifiers

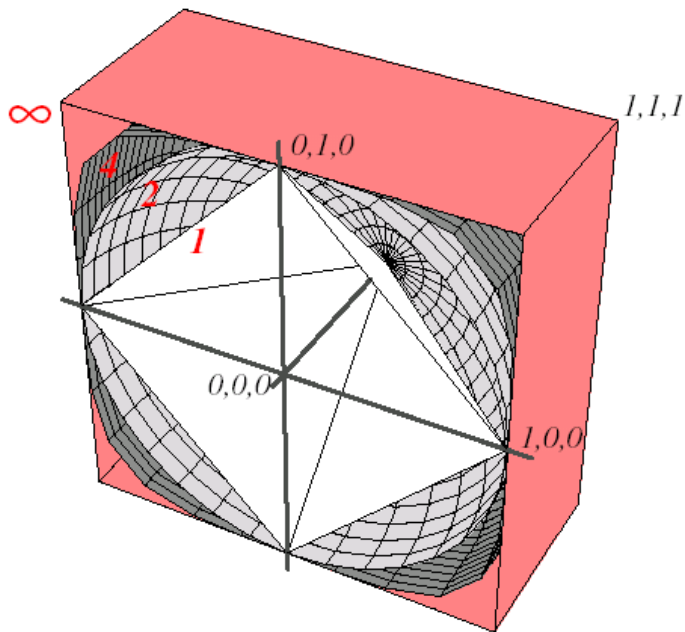
- Given the data set, distance metrics used directly affects the performance of NN classifiers
- Properties of Metric $\text{dist}(x_j, x_k)$
 - Non-negative, $\text{dist}(x_j, x_k) \geq 0$
 - Reflexivity, $\text{dist}(x_j, x_k) = 0$, iff $x_j = x_k$.
 - Symmetry, $\text{dist}(x_j, x_k) = \text{dist}(x_k, x_j)$
 - Triangular in-equality: $\text{dist}(x_j, x_k) + \text{dist}(x_j, x_t) \geq \text{dist}(x_k, x_t)$

Minkowski Metric

• L_k norm:

$$L_k(x, y) = \left(\sum_{i=1}^d |x_i - y_i|^k \right)^{1/k}$$

- Where x, y are d -dimensional vectors
- When $k=2$, it is known as Euclidean distance
- When $k=1$, it is known as Manhattan distance



2-dimensional sphere with different L_k metrics

Obviously, different metric will define different local neighborhood.

Metric & Subspace

- Mahalanobis Distance:

$$\text{dist}(x_1, x_2) = \sqrt{(x_1 - x_2)^T Q (x_1 - x_2)}, \quad x_1, x_2 \in R^d$$

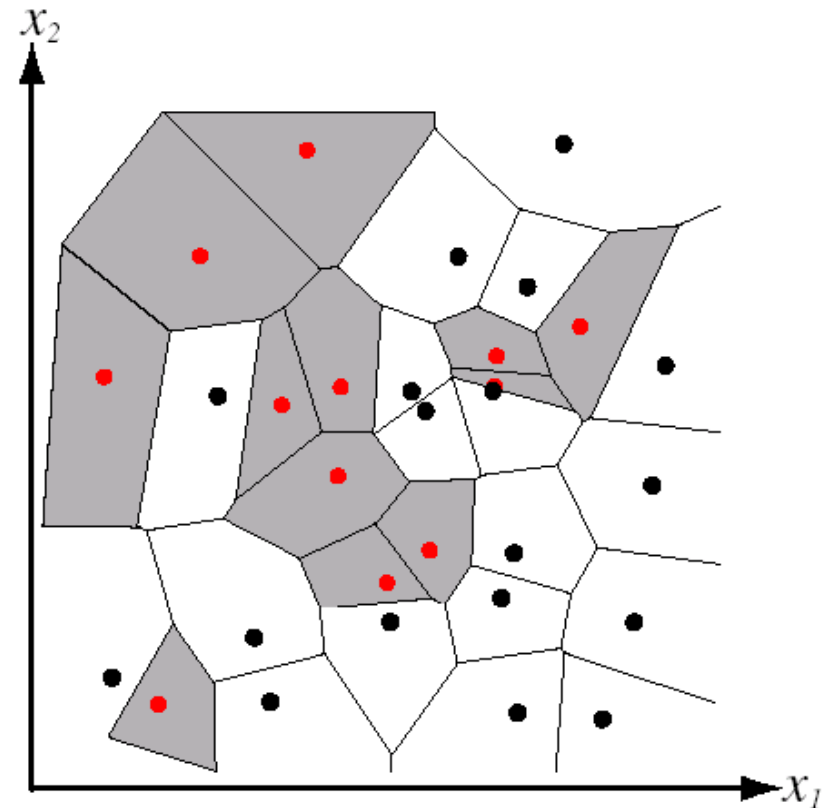
- Where Q is positive definite.
- In PCA/LDA, if the subspace learnt is A, i.e, $Y=AX$, then

$$\text{dist}(y_1, y_2) = \sqrt{(Ax_1 - Ax_2)^T Q (Ax_1 - Ax_2)} = \sqrt{(x_1 - x_2)^T A^T A (x_1 - x_2)}$$

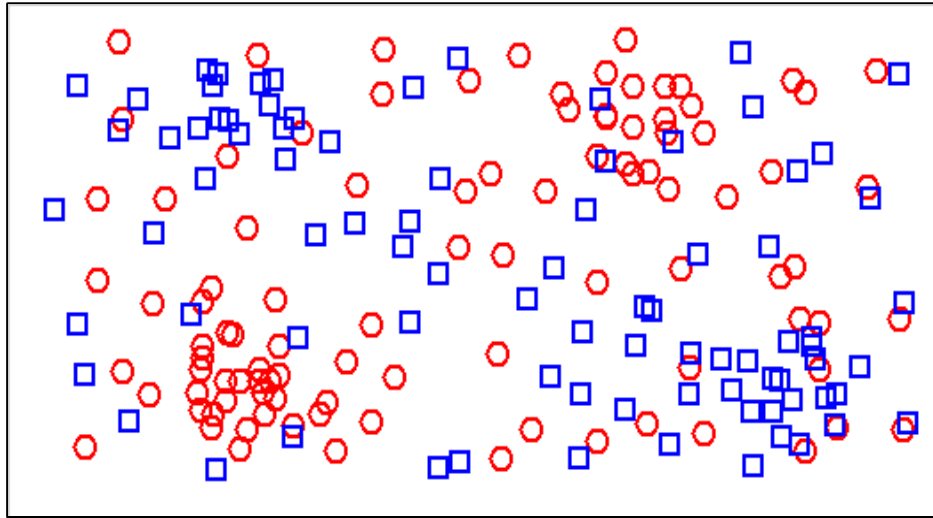
- i.e, the Euclidean distance in Y, is the Mahalanobis distance in X, with $Q = A^T A$.

Challenges to the NN classifier

- Large data set and complex NN decision boundaries
- Cost for storage and computing are high.
- Need to reduce the data set
- To make the recognition even better
- How to do it ?
 - Editing



Dataset Reduction: Editing

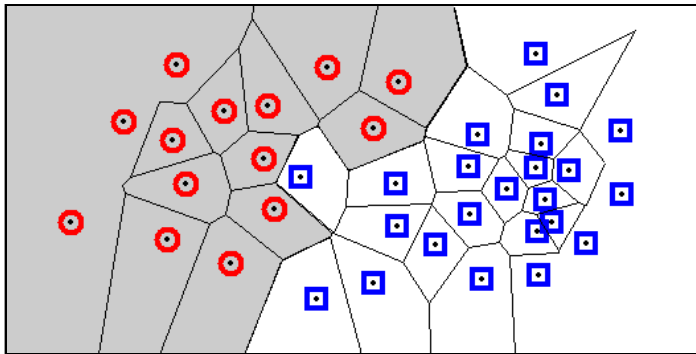


- Training data may contain noise, overlapping classes
 - starting to make assumptions about the underlying distributions
- Editing seeks to remove noisy points and produce smooth decision boundaries – often by retaining points far from the decision boundaries
- Results in homogenous clusters of points

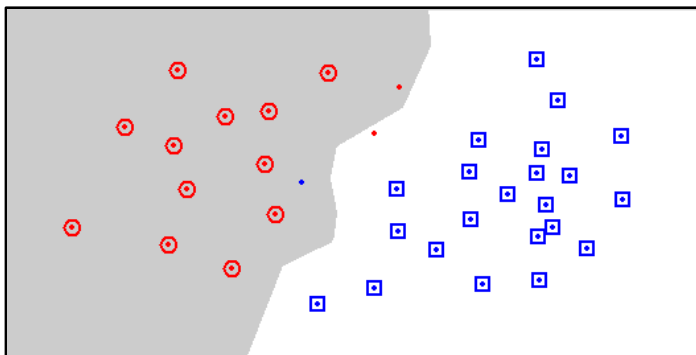
Wilson Editing

- Wilson 1972
- Remove points that do not agree with the majority of their k nearest neighbours

Earlier example

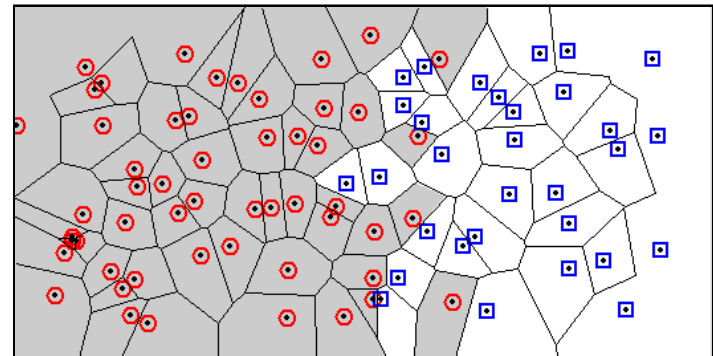


Original data

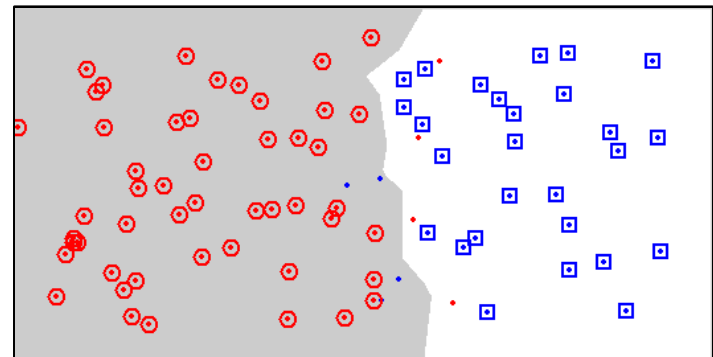


Wilson editing with $k=7$

Overlapping classes



Original data

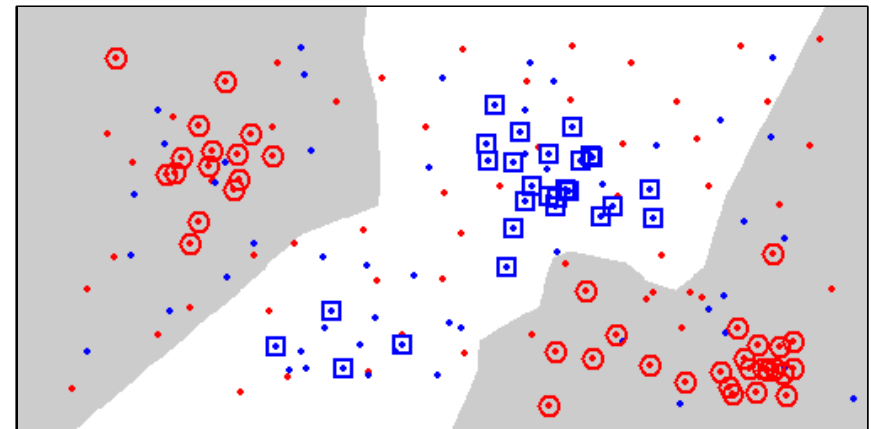


Wilson editing with $k=7$

Multi-edit

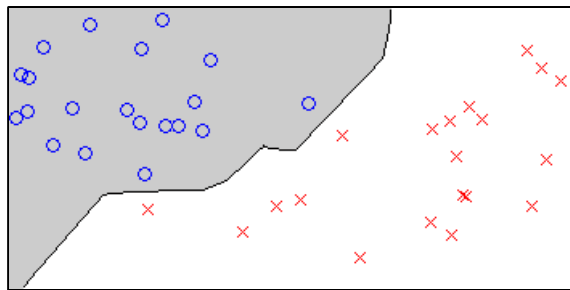
- Multi-edit [Devijer & Kittler '79]
 - Repeatedly apply Wilson editing to random partitions
 - Classify with the 1-NN rule
- Approximates the error rate of the Bayes decision rule

1. **Diffusion:** divide data into $N \geq 3$ random subsets
2. **Classification:** Classify S_i using 1-NN with $S_{(i+1) \bmod N}$ as the training set ($i = 1..N$)
3. **Editing:** Discard all samples incorrectly classified in (2)
4. **Confusion:** Pool all remaining samples into a new set
5. **Termination:** If the last I iterations produced no editing then end; otherwise go to (1)

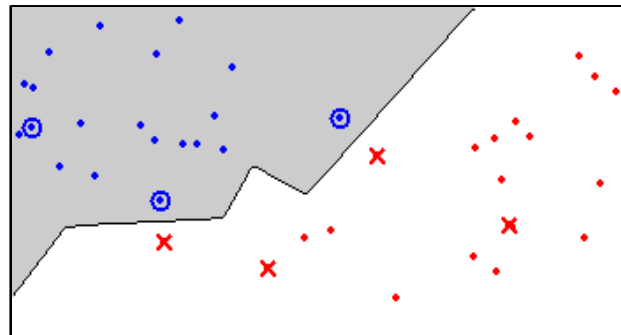


Condensing

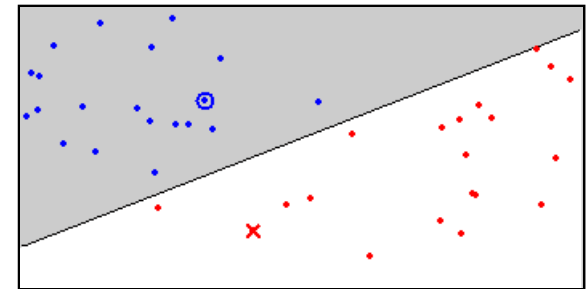
- Aim is to reduce the number of training samples
- Only keep data samples that are needed to define the decision boundary
- This is reminiscent of a Support Vector Machine
- ***Decision Boundary Consistent*** - a subset whose nearest neighbour decision boundary is identical to the boundary of the entire training set
- ***Consistent Set*** --- - the smallest subset of the training data that correctly classifies all of the original training data
- ***Minimum Consistent Set*** - smallest consistent set



Original data



Condensed data



Minimum Consistent Set

Condensing

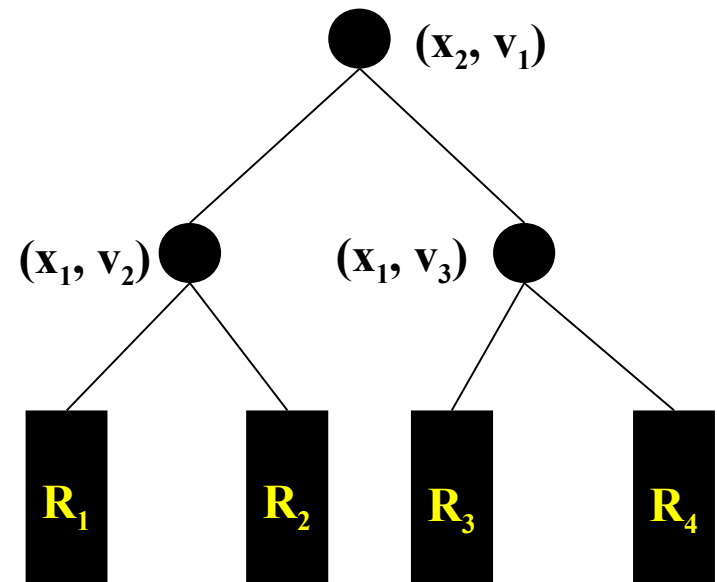
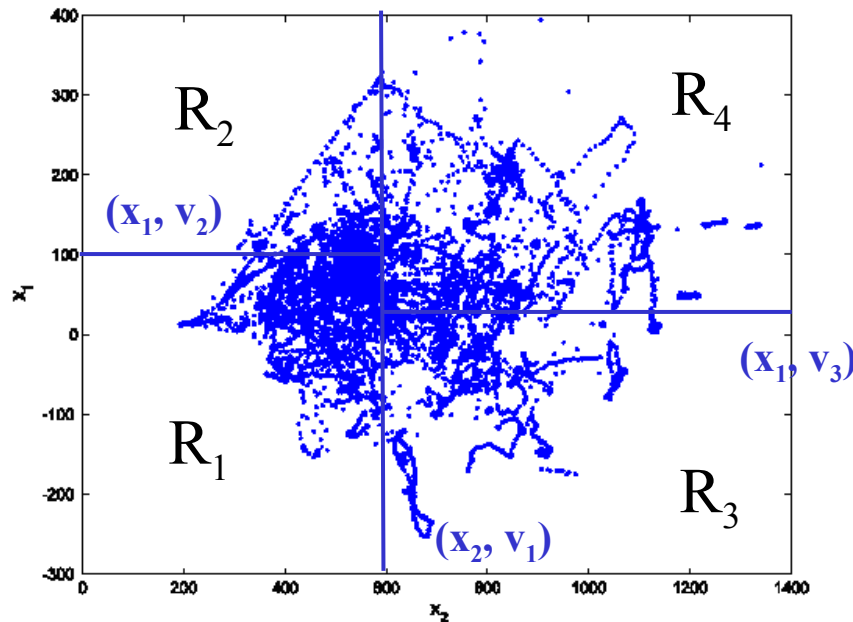
- The core idea is that only training data samples close to the decision boundary are important
- Will have a more thorough discussion with the Support Vector Machine

Indexing to Improve NN performance

- Another way to deal with the data set size is to do indexing
- Partition the data set into a hierarchical structure with an indexing data structure.
- At the time of classification, do a NN search first, then apply k-NN rules

Top-down Iterative data partition scheme

- Project data to the axis with the largest variance
- Split into Left and Right set at median value
- Store cutting plane and value, as well as Min Bounding Box (MBB) at each node

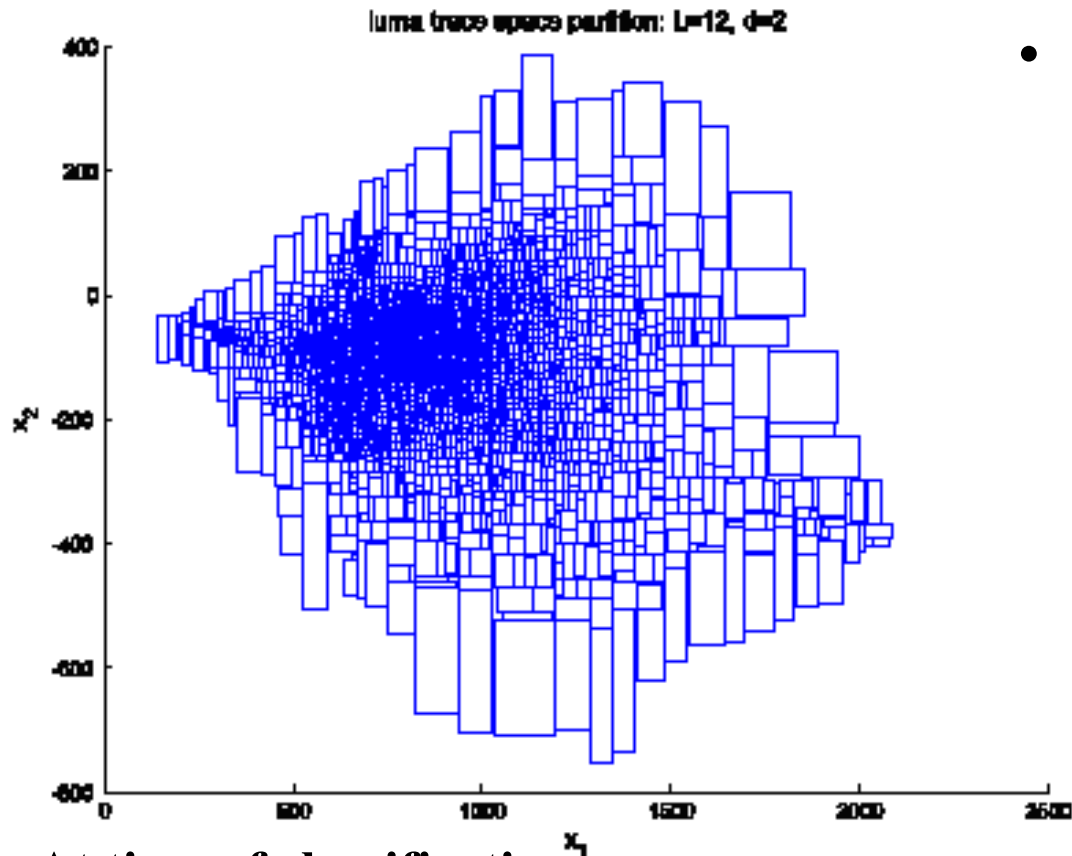


Kd-Tree: L=2

- At retrieval time, query clip is traversing the tree by MBB intersections and splits

Index structure

- Kd-tree partition of video data space.



At time of classification

- A locality of query point x is identified from the data set
- K-NN rule applied to find the label for x

- Example:
 - 5 hours of video frames
 - An index tree of 12 levels, and 4096 leaf nodes levelMBBs are plotted. Each node has about 132 frames

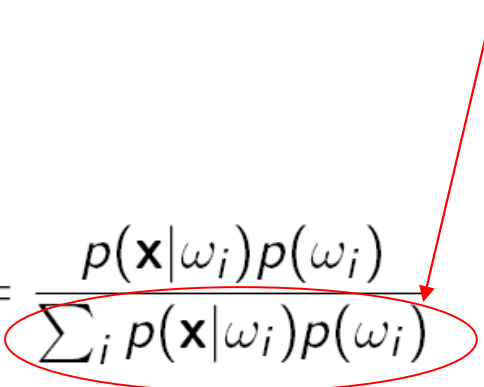
Bayesian Classifier with Gaussian Mixture Data

Bayesian Decision

- Based on probabilistic modeling of patterns
- Definitions: Likelihood, Prior, Posterior:

- ▶ $p(\mathbf{x}|\omega_i)$ Likelihood
- ▶ $p(\omega_i)$ Prior
- ▶ $p(\omega_i|\mathbf{x})$ Posterior
- ▶ Bayes Rule

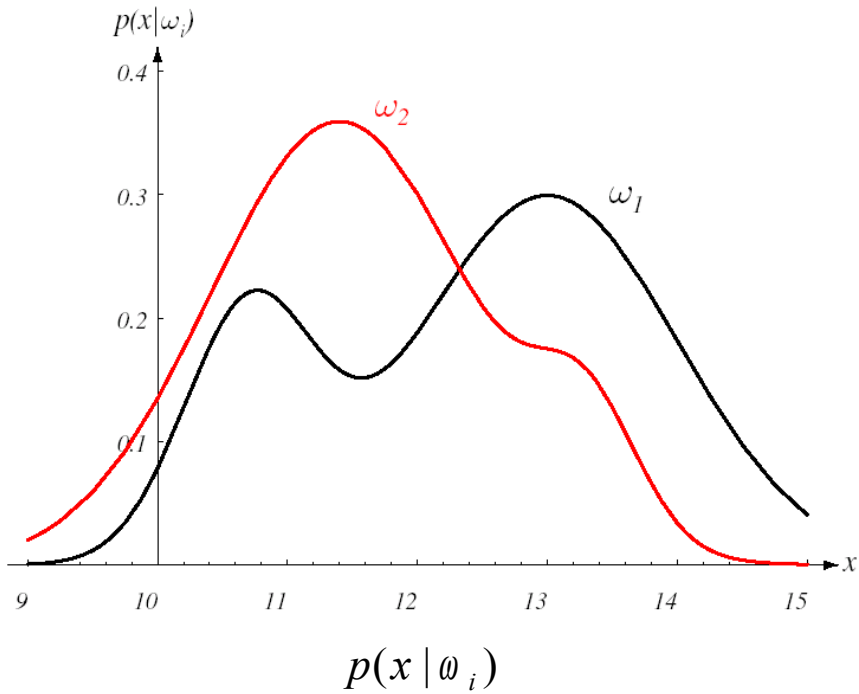
Normalizing const

$$p(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)p(\omega_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\omega_i)p(\omega_i)}{\sum_j p(\mathbf{x}|\omega_j)p(\omega_j)}$$


- ▶ In other words

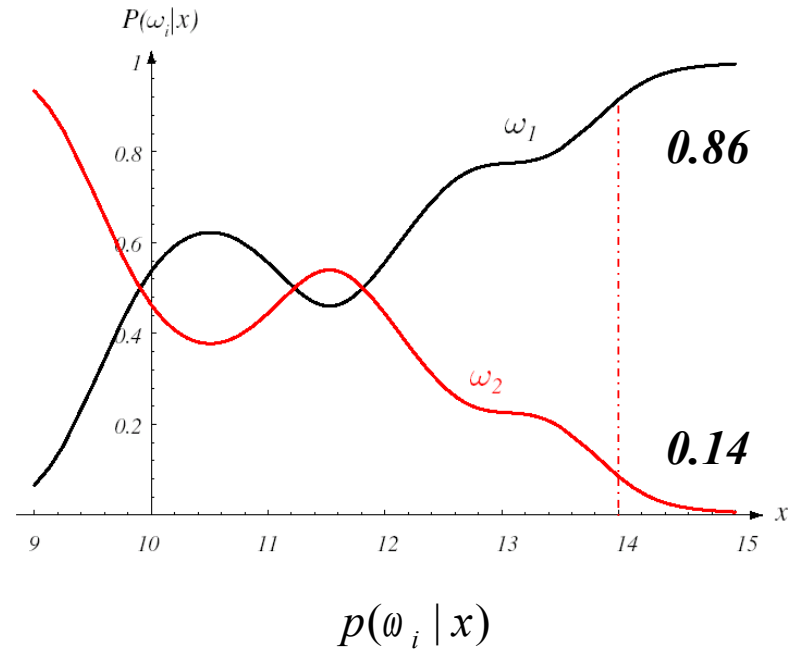
posterior \propto likelihood \times prior

Example



**Given Likelihood functions
And Prior:**

$$p(\omega_1) = 1/3, \quad p(\omega_2) = 2/3$$



**Posterior functions
Should sum to 1**

**Given the evidence, the prob
Of belonging to certain class**

Bayesian Decision Rule

- Min error rate classification:
 - Let action a_k be classifying x to class k :
 - Let the action loss function be:

$$\lambda(\alpha_k|\omega_i) = \begin{cases} 0 & i = k \\ 1 & i \neq k \end{cases} \quad i, k = 1, \dots, c$$

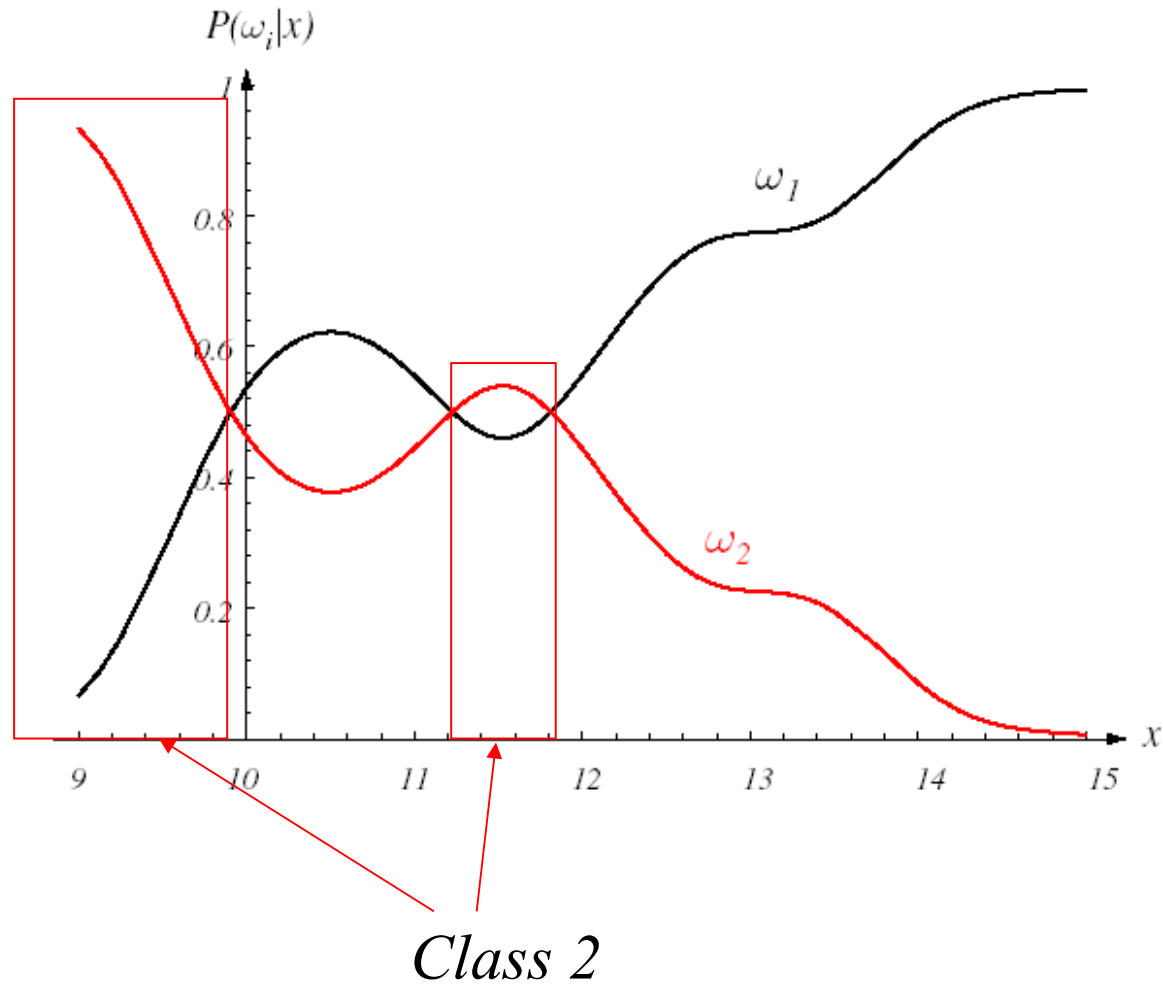
- Which means if correct decision, no loss, otherwise, loss is 1
- The error rate, or conditional risk is therefore:

$$R(\alpha_k|\mathbf{x}) = \sum_{i \neq k} P(\omega_i|\mathbf{x}) = 1 - P(\omega_k|\mathbf{x})$$

- This gives us the Bayesian decision rule:

$Decide \omega_k \text{ if } P(\omega_k|\mathbf{x}) > P(\omega_i|\mathbf{x}) \quad \forall i \neq k$

Example



Bayesian Classifier

- Definitions:

- ▶ Discriminant function: $g_i(\mathbf{x})$, $i = 1, \dots, C$, assigns ω_i to \mathbf{x}

- ▶ Classifier

$$x \rightarrow \omega_i \text{ if } g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \forall j \neq i$$

- ▶ Examples:

$$g_i(\mathbf{x}) = P(\omega_i|\mathbf{x})$$

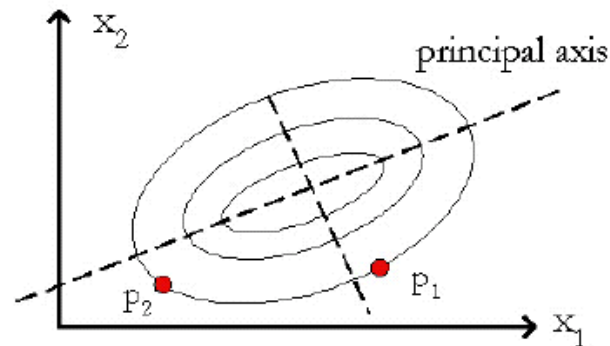
$$g_i(\mathbf{x}) = P(\mathbf{x}|\omega_i)P(\omega_i)$$

$$g_i(\mathbf{x}) = \ln P(\mathbf{x}|\omega_i) + \ln P(\omega_i)$$

d-dimensional Likelihood functions

- Completely characterized by its mean and dxd covariance:

$$p(\mathbf{x}) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$



- ▶ The principal axes (the direction) are given by the eigenvectors of the covariance matrix $\boldsymbol{\Sigma}$
- ▶ The length of the axes (the uncertainty) is given by the eigenvalues of $\boldsymbol{\Sigma}$

Discriminant Function for Gaussian Distribution

- The likelihood is characterized by the Gaussian parameters: mean and covariance”

Minimum-error-rate classifier

$$g_i(\mathbf{x}) = \ln p(\mathbf{x}|\omega_i) + \ln p(\omega_i)$$

When using Gaussian densities, it is easy to see:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma_i| + \ln p(\omega_i)$$

Case I: $\Sigma_j = \sigma^2 \mathbf{I}$

$$g_i(\mathbf{x}) = -\frac{\|\mathbf{x} - \mu_i\|^2}{2\sigma^2} + \ln p(\omega_i) = -\frac{1}{2\sigma^2}[\mathbf{x}^T \mathbf{x} - 2\mu_i^T \mathbf{x} + \mu_i^T \mu_i] + \ln p(\omega_i)$$

Notice that $\mathbf{x}^T \mathbf{x}$ is a constant. Equivalently we have

$$g_i(\mathbf{x}) = -\left[\frac{1}{\sigma^2}\mu_i\right]^T \mathbf{x} + \left[-\frac{1}{2\sigma^2}\mu_i^T \mu_i + \ln p(\omega_i)\right]$$

This leads to a *linear discriminant function*

$$g_i(\mathbf{x}) = \mathbf{W}_i^T \mathbf{x} + \mathbf{W}_{i0}$$

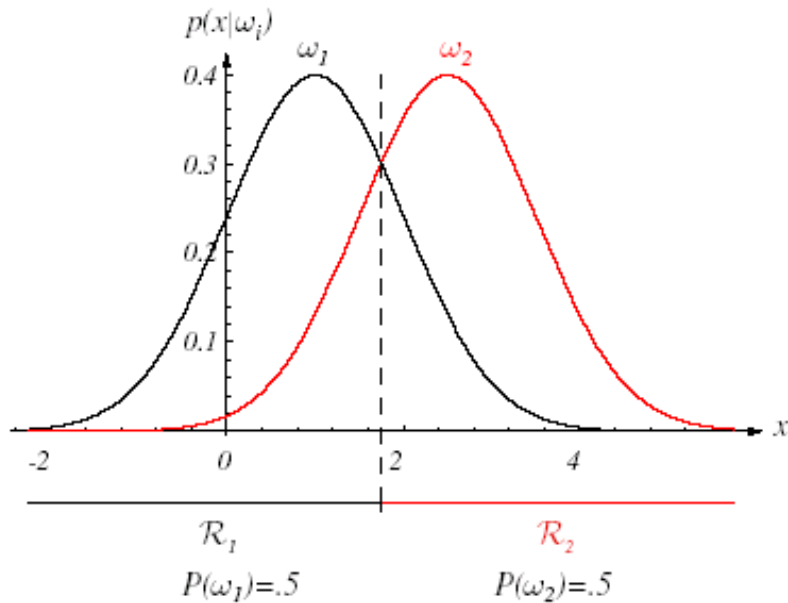
At the decision boundary $g_i(\mathbf{x}) = g_j(\mathbf{x})$, which is linear:

$$\mathbf{W}^T (\mathbf{x} - \mathbf{x}_0) = 0,$$

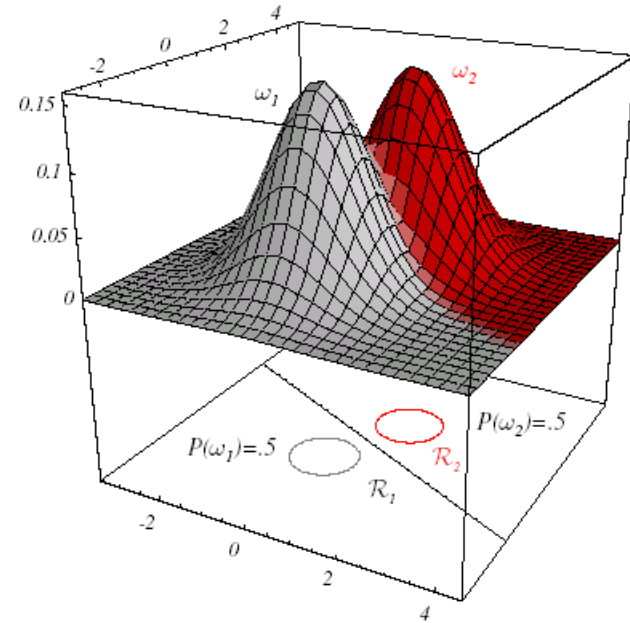
where $\mathbf{W} = \mu_i - \mu_j$ and

$$\mathbf{x}_0 = \frac{1}{2}(\mu_i + \mu_j) - \frac{\sigma^2}{\|\mu_i - \mu_j\|^2} \ln \frac{p(\omega_i)}{p(\omega_j)} (\mu_i - \mu_j)$$

Example



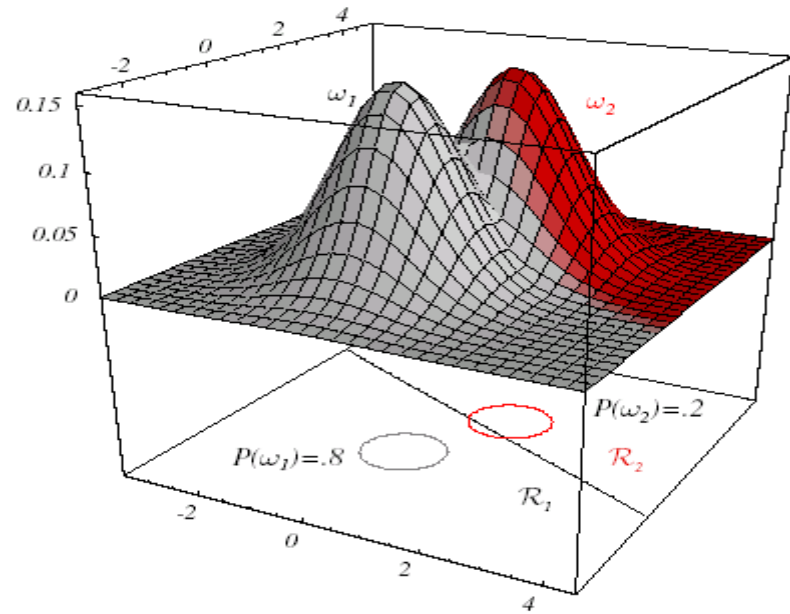
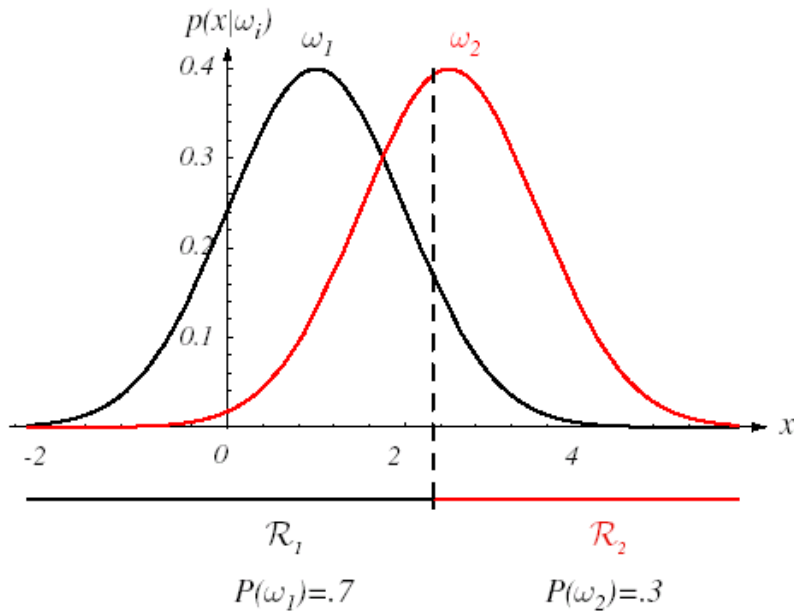
1-d case



2-d case

Notice that the decision boundary is linear hyperplane of Dimension $d-1$, perpendicular to the line connecting 2 means

Example



The class prior prob will have effects on the decision boundary locations

Case II: $\Sigma_i = \Sigma$

- Classes have identical, but arbitrary covariance
- Discriminant function:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma^{-1}(\mathbf{x} - \mu_i) + \ln p(\omega_i)$$

Similarly, we can have an equivalent one:

$$g_i(\mathbf{x}) = (\Sigma^{-1} \mu_i)^T \mathbf{x} + \left(-\frac{1}{2} \mu_i^T \Sigma^{-1} \mu_i + \ln(\omega_i)\right)$$

Case II: $\Sigma_i = \Sigma$

The discriminant function and decision boundary are still linear:

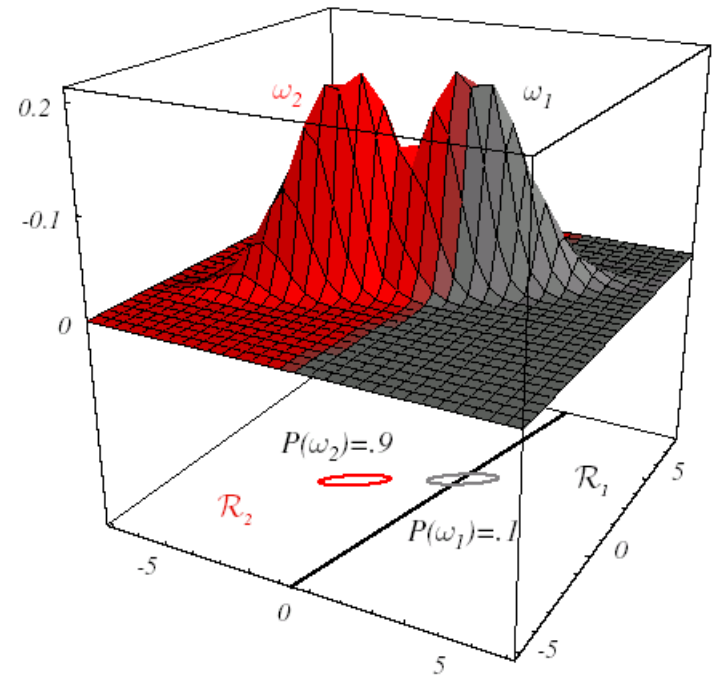
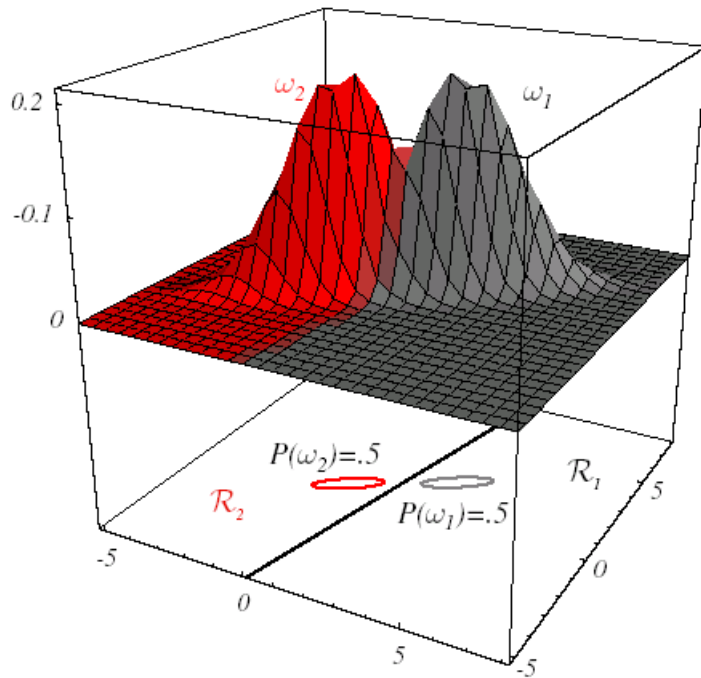
$$\mathbf{W}^T (\mathbf{x} - \mathbf{x}_0) = 0$$

$$\text{where } \mathbf{W} = \Sigma^{-1}(\mu_i - \mu_j) \quad \text{and}$$

$$\mathbf{x}_0 = \frac{1}{2}(\mu_i + \mu_j) - \frac{\ln p(\omega_i) - \ln p(\omega_j)}{(\mu_i - \mu_j)^T \Sigma^{-1}(\mu_i - \mu_j)} (\mu_i - \mu_j)$$

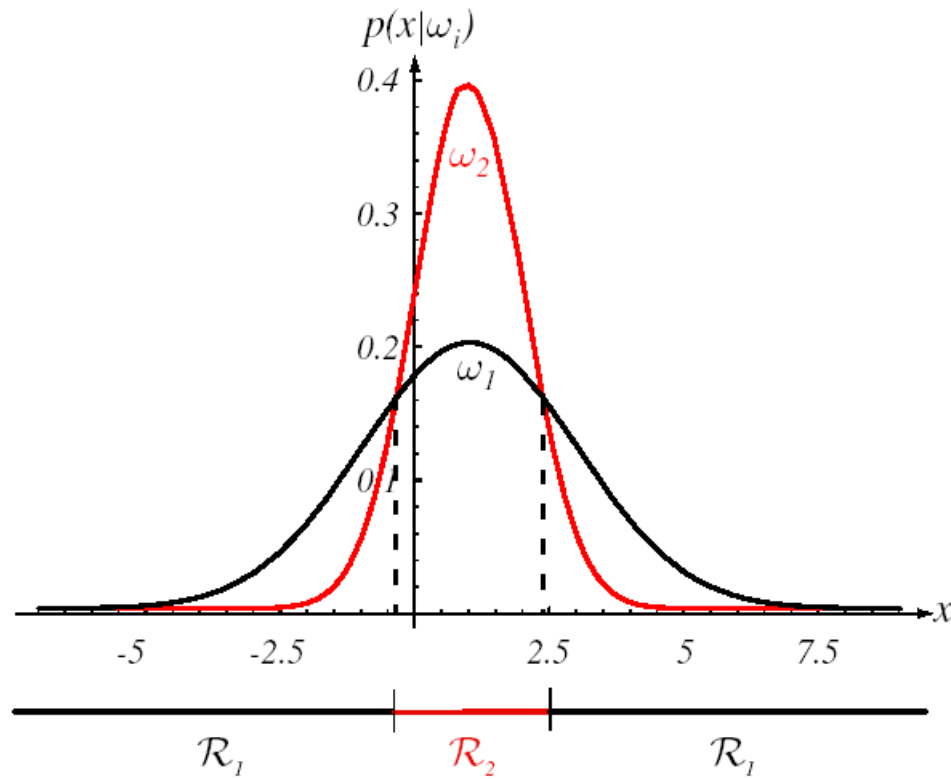
Note: Compared with Case I, the Euclidean distance is replaced by Mahalanobis distance. The boundary is still linear, but the hyperplane is no longer orthogonal to $\mu_i - \mu_j$.

Examples



- 2 likelihood prob functions have the same covariance (or shape)
- Decision boundary is still linear
- Not orthogonal to the line connecting two means anymore.
- Prior still moves the decision boundary

Example-non simply connected decision region



*2 classes have the same variance in likelihood
Decision region are not a simple line.*

Case III: $\Sigma_i = \text{arbitrary}$

- Arbitrary covariance for each class:
 - Decision function:

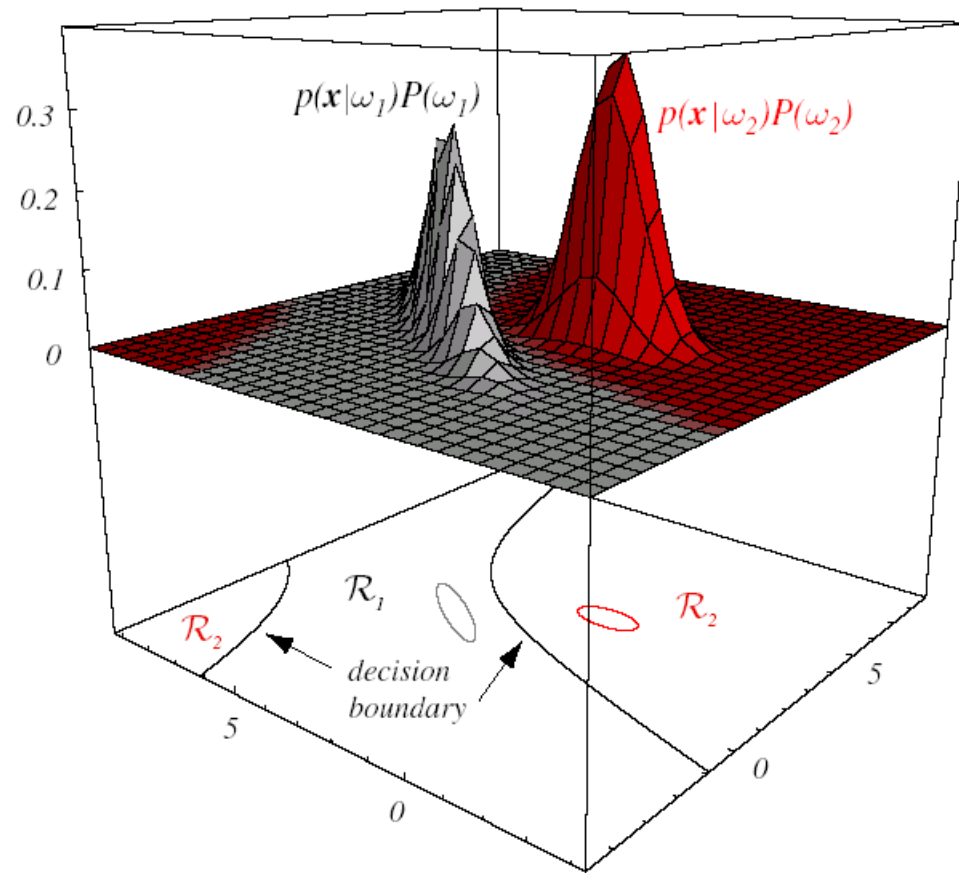
$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{A}_i \mathbf{x} + \mathbf{W}_i \mathbf{x} + \mathbf{W}_{i0},$$

where

$$\begin{aligned} \mathbf{A}_i &= -\frac{1}{2} \Sigma_i^{-1} \\ \mathbf{W}_i &= \Sigma_i^{-1} \mu_i \\ \mathbf{W}_{i0} &= -\frac{1}{2} \mu_i^T \Sigma_i^{-1} \mu_i - \frac{1}{2} \ln |\Sigma_i| + \ln p(\omega_i) \end{aligned}$$

Note: The decision boundary is no longer linear! It is hyperquadrics.

Example



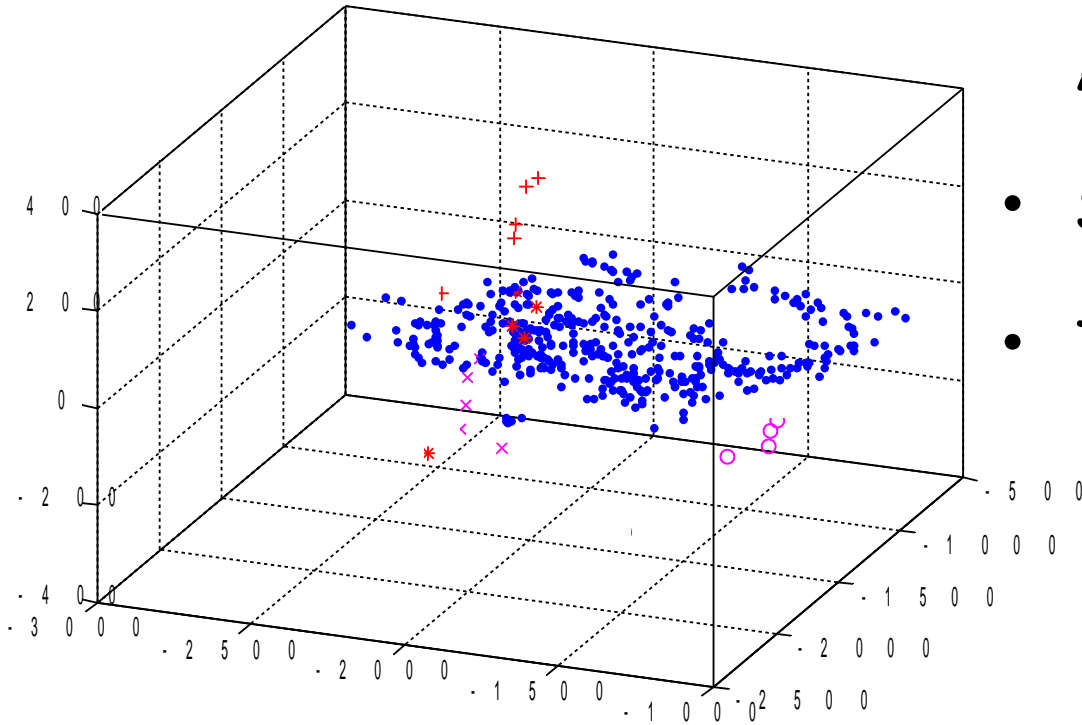
Matlab Implementation

CLASS = CLASSIFY(SAMPLE, TRAINING, GROUP) classifies each row of the data in **SAMPLE** into one of the groups in **TRAINING**. **SAMPLE** and **TRAINING** must be matrices with the same number of columns.

- **GROUP** is a grouping variable for **TRAINING**. Its unique values define groups, and each element defines which group the corresponding row of **TRAINING** belongs to. **GROUP** can be a categorical variable, numeric vector, a string array, or a cell array of strings.
- **TRAINING** and **GROUP** must have the same number of rows.
- **CLASSIFY** treats NaNs or empty strings in **GROUP** as missing values, and ignores the corresponding rows of **TRAINING**.
- **CLASS** indicates which group each row of **SAMPLE** has been assigned to, and is of the same type as **GROUP**.

-
- **[rec_lbls, err]=classify(x1, x0, y0, method)**
 - X1 - test data, unknown, in $m \times d$ format, row vectors
 - X0 - training data, in $n \times d$ format, row vectors, with known labels
 - Y0 - training data labels, in $n \times 1$ format
 - Method: 'linear' - roughly the case 2 , 'quadratic', roughly case 3.
 - **Useful in face recognition project.**

Classify



- Eigen face projected AT&T data set
- Selected 4 subjects
- Try recognition with
 - K-NN classifier (homework-2)
 - Bayesian min error classifier with Gaussian modeling

Use built-in MVN classifier

*Recognition results with
eigen face projection
-12 dimensions*

*- quadratic (case 3) 3 out of 4 correct
- linear (case 2): 2 out of 4 correct*

```
test ids: 5 12 20 32
```

```
recognized ids, linear : 40 35 20 32
```

```
recognized ids, quadratic : 5 35 20 32 >>
```

```
61 % projection
62 kDim = 12;
63 A1=double(EigFaces(:, indx([1:3])));
64 x = double(faces)*A1;
65
66 % prepare labels
67 nSubj=40; nImages = 10;
68 faceIds=zeros(nSubj*nImages, 1);
69 for k=1:nSubj
70     offs=(k-1)*nImages;
71     faceIds(offs+1:offs+nImages) = k;
72 end
73 % total 8 query cases
74 qSubj=[5 12 20 32]; qId = 4;
75 % pull query faces: qx
76 for k=1:length(qSubj)
77     qOffs(k) = 10*(qSubj(k)-1)+qId;
78     qx(k,:)=x(qOffs(k), :);
79 end
80 % training samples: tx
81 % setdiff(a, b), a must contain b !
82 txOffs = setdiff([1:nSubj*nImages], qOffs);
83 tx = x(txOffs,:);
84 tx_labels = faceIds(txOffs);
85
86 [rec_labels1, err]=classify(qx, tx, tx_labels, 'linear');
87 [rec_labels2, err]=classify(qx, tx, tx_labels, 'quadratic');
88 fprintf('\n test ids: '); fprintf('%d ', qSubj);
89 fprintf('\n recognized ids, linear : '); fprintf('%d ', rec_labels1);
90 fprintf('\n recognized ids, quadratic : '); fprintf('%d ', rec_labels2);
```

Summary

- **K-Nearest Neighbor (kNN) classifier**
 - Intuitive solution.
 - Data set complexity presents challenges to computing and storage
 - Editing/Condensation to reduce data set
 - Indexing to speed up access
- **Bayesian Classifier with Gaussian Distribution Modeling**
 - Bayesian rule: posterior proportional to likelihood \times prior
 - Min Error Rate Classification
 - Applied to Gaussian Distribution Case:
 - » Identical covariance among classes - linear decision boundaries
 - » Arbitrary covariance - quadratic boundaries