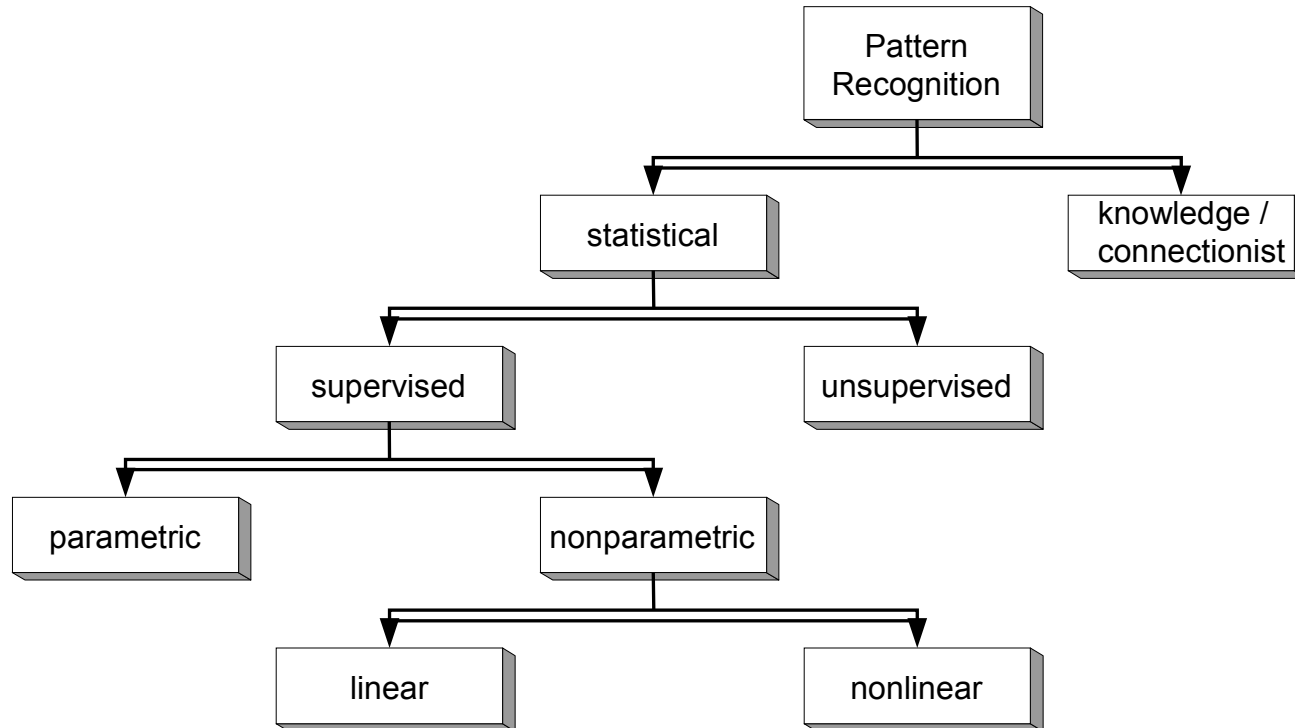


# Pattern Recognition and Classification

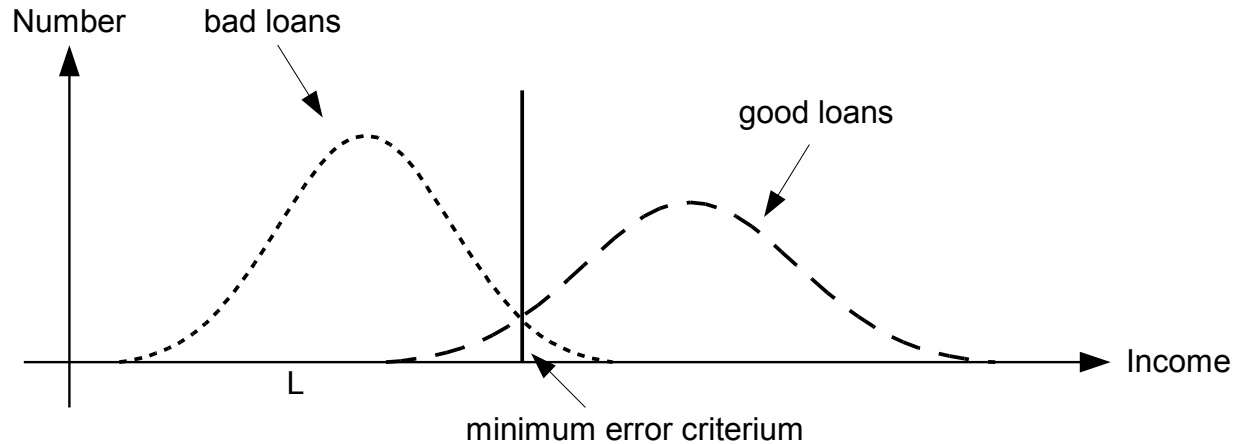
(for speech recognition)

11-751 Speech Recognition  
09-17-2008

# Pattern Recognition



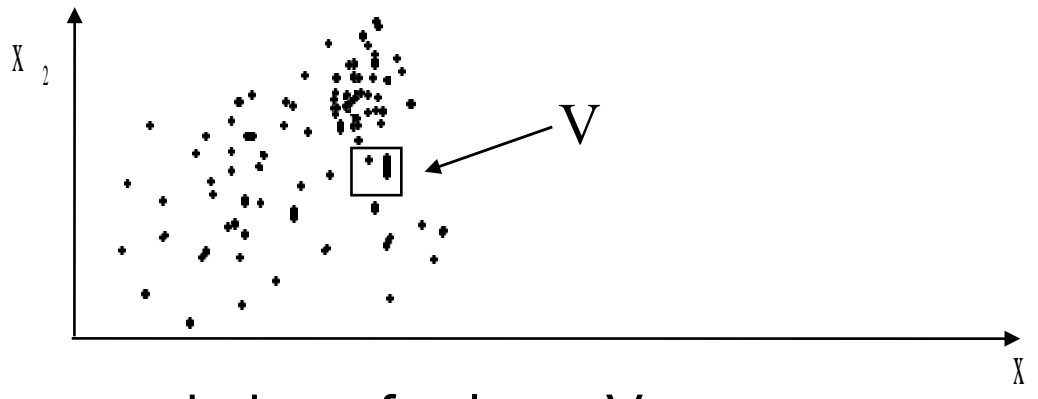
# Parametric - Non-parametric



- **Parametric:**
  - assume underlying probability distribution;
  - estimate the parameters of this distribution.
  - Example: "Gaussian Classifier"
- **Non-parametric:**
  - **Don't assume distribution.**
  - **Estimate probability of error or error criterion directly from training data.**
  - **Examples: Parzen Window, k-nearest neighbor, SVM ...**

# Non-Parametric Techniques: Parzen Windows

- No Assumptions about the distribution are made  
estimate  $p(x)$  directly from data



- Choose a window of volume  $V$
- Count the number of samples that fall inside the window.

- $$p(x) \approx \frac{k / n}{V}$$

$k$  - count  
 $n$  - number of samples

# Parzen Windows

- Problem:  
Volume too large  $\rightarrow$  loose resolution  
Volume too small  $\rightarrow$  erratic, poor estimate

- set  $V_n = 1/\sqrt{n}$

$$\left. \begin{array}{l} \lim_{n \rightarrow \infty} V_n = 0 \\ \lim_{n \rightarrow \infty} k_n = \infty \\ \lim_{n \rightarrow \infty} k_n / n = 0 \end{array} \right\} P_{guess}(x) \rightarrow p(x)$$

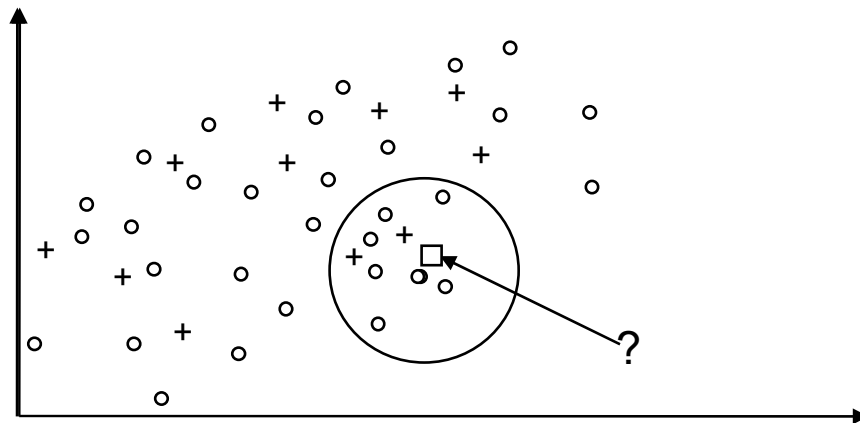
# K-Nearest Neighbors (KNN)

- Idea: Let the volume be a function of the data. Include k-nearest neighbors in estimate.

set  $k = \sqrt{n}$ ; k-nearest neighbor rule for classification

To classify sample x:

- Find k-nearest neighbors of x.
- Determine the class most frequently represented among those k samples (take a vote)
- Assign x to that class.



k = 9  
7 o  
2 +  
classify o  
⇒

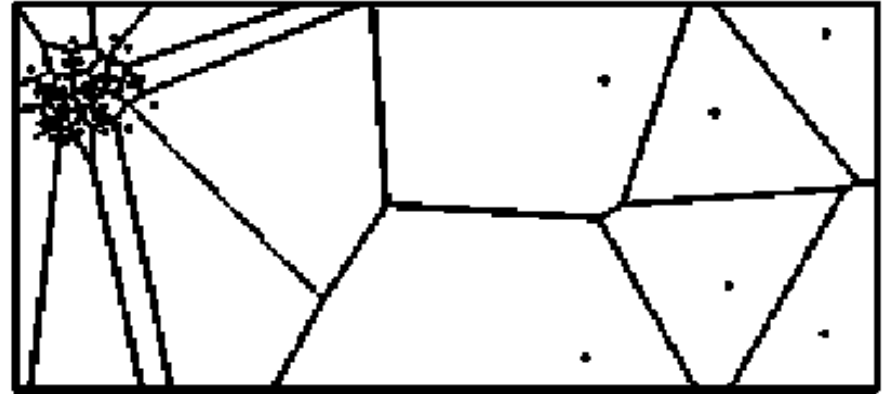
# KNN-Classifer: Problem

- For finite number of samples  $n$ , we want  $k$  to be:
  - large for reliable estimate
  - small to guarantee that all  $k$  neighbors are reasonably close.
- Need training database to be larger.

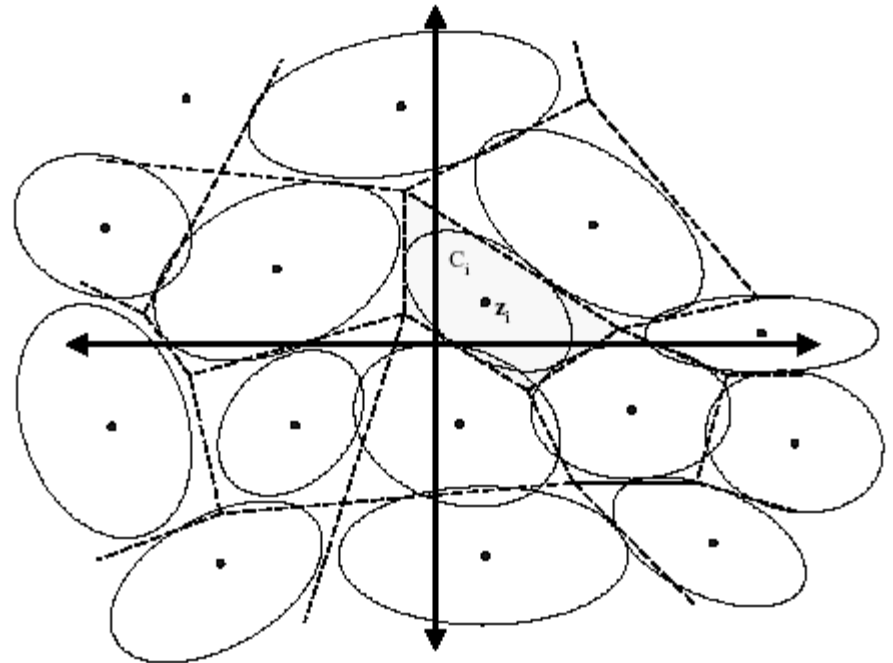
***"There is no data like more data."***

# Disadvantages of KNN-Classifer

Problem:  
different classes have  
different variances  
=> KNN-classifier not optimal



THEREFORE, IF:  
Elements are normally distributed,  
THEN:  
Use Gaussian-classifier



# Distance Measures

- Various Distance Measures
  - Vector-Vector Distance Measures
  - Vector-Class Distance Measures
  - Class-Class Distance Measures
- Distortion Measure
- Gaussian Densities
- Mixtures of Gaussian Densities
- Learning Parameters of Mixtures of Gaussians
- The Expectation Maximization Algorithm for Gaussians Mixtures

# Various Distance Measures

In multidimensional spaces, different distance measures are used:

- Vector-vector-distances:  
distance from vector to its class centroid, or distance between two vectors to decide whether they belong together



- Vector-class-distances:  
distance of vector to its class, i.e. how likely does it belong to class



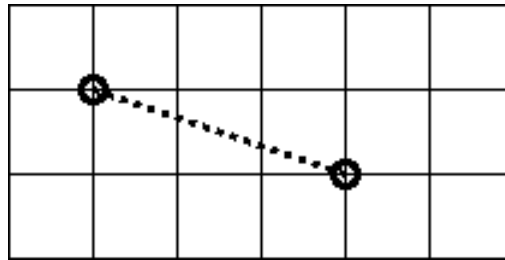
- Class-class-distances  
to decide whether two classes should be the same (clustering)



# Vector-Vector Distance Measures

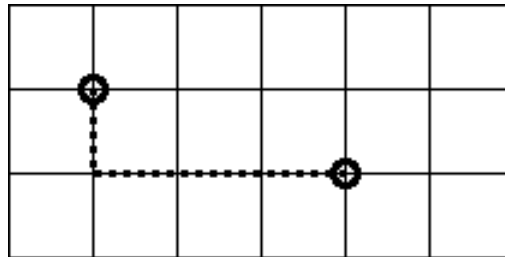
Some common distance measures between  $x$  and  $y$ :

- Hamming Distance: (asymmetric)  $\sum (x_i - y_i)$



- Euclidian Distance: (most commonly used distance)  $\sum (x_i - y_i)^2$

- Cityblock Distance: (saves one expensive multiplication)  $\sum |x_i - y_i|$



# Vector-Class Distance Measures

Some common distance measures between  $x$  and a class of vectors:

- **Mahalanobis** Distance:

$$(x - \mu)^T \Sigma^{-1} (x - \mu)$$

- **Gaussian** Density Value:

$$\frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)}$$

- This value expresses the likelihood of  $x$  belonging to the class whose mean is  $\mu$  and whose variance is  $\Sigma$ .

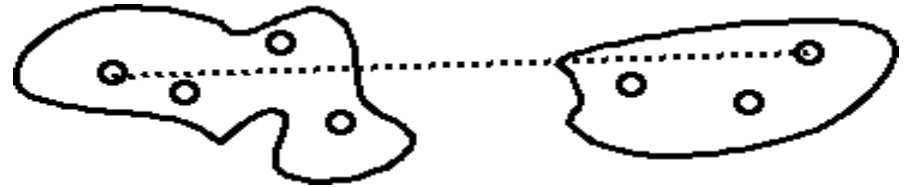
$\Sigma$

# Class-Class Distance Measures

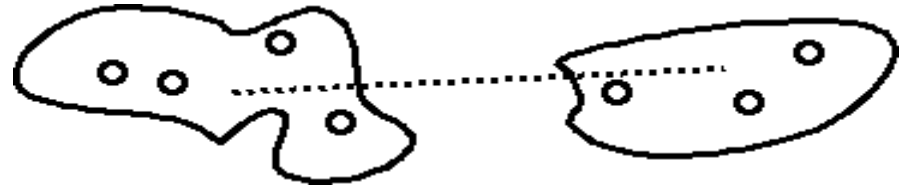
- Nearest neighbor:  $\min |x-y|$



- Farthest neighbor:  $\max |x-y|$



- Distance of means:  $|\mu_x - \mu_y|$



- Entropy:  $P(A) \cdot H(A) + P(B) \cdot H(B) - P(A+B) \cdot H(A+B)$

$H(X) = \sum P(x_i) \log(1/P(x_i))$ ;  $H(X)$  is the amount of information required to specify what symbol has occurred on average

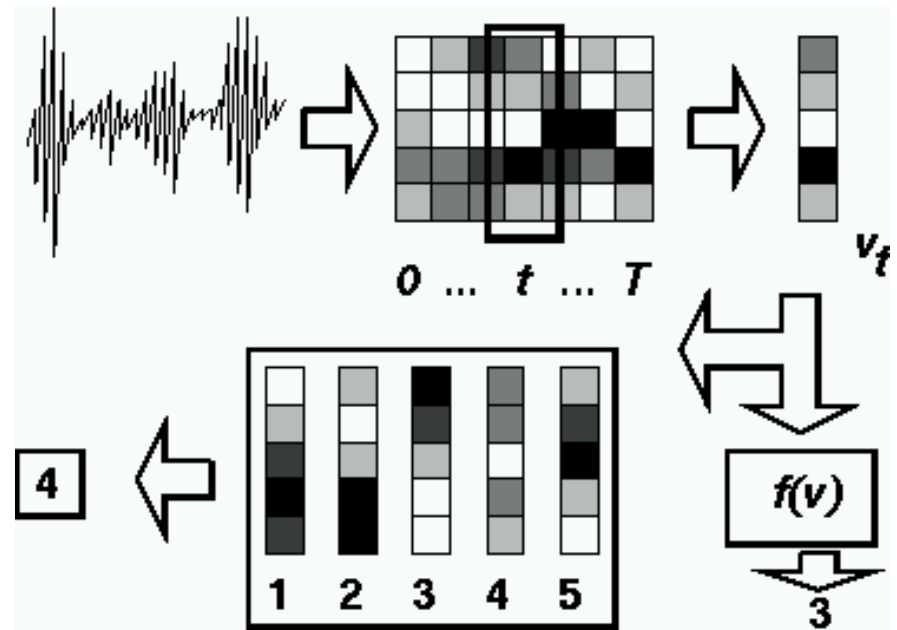
# Unsupervised Classification

## K-Means Algorithm

# Why Vector Quantization?

Source coding: Convert the signal into a sequence of bits that are transmitted over a channel, and then use to reproduce the original signal

Goal: reduce number of bits and achieve a minimal distortion for the given bit rate



- Vector Quantization (VQ) is one of the most efficient source coding tech
- Scalar quantization: Quantization of a single signal value scalar  $\rightarrow$  scalar
- Vector quantization: Joint quantization of multiple signal values  $\rightarrow$  index
- Widely used in speech recognition and synthesis:
  - Dimensionality reduction, data reduction (one index instead of  $\mathbf{R}^n$ )
  - Simple form of classification, discrete acoustic prototypes
  - Robust Signal Processing
- Precompute distances and other functions:  $f(v) \rightarrow g[q(v)]$

# Vector Quantizer Used as a Classifier

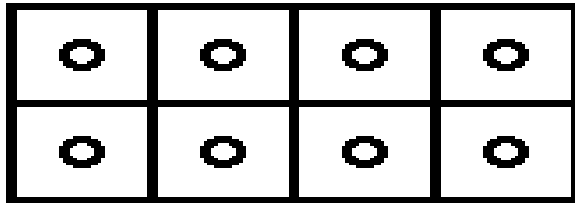
- VQ-indices can be regarded as classification classes
- A class consists of all vectors that are mapped to the same index
- Often:
  - VQ-indices are not indices of final classes
  - VQ-indices are used as a discrete feature for a discrete classifier
  - Discrete classifiers are simple and fast (table-lookup)

# Finding Codebooks of Reference Vectors

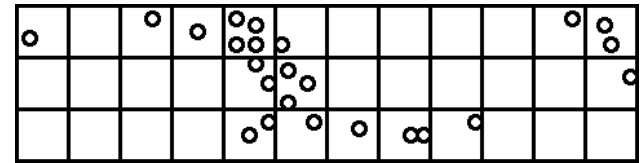
A collection of reference vectors is called a **codebook**.

Ways how to find (good) codebooks (i.e. unsupervised learning):

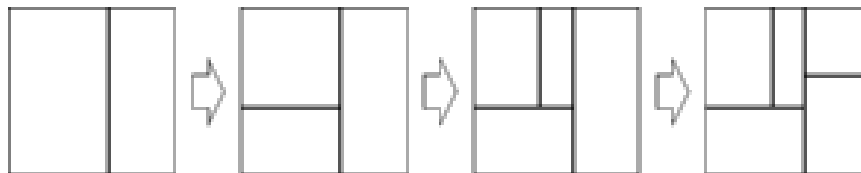
- Simple but suboptimal: use centers of equally sized hypercubes



but?



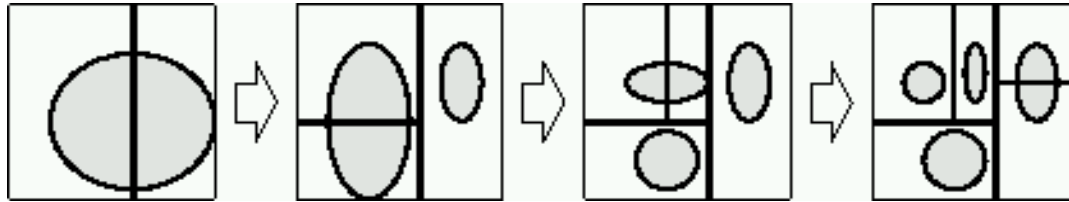
- Possibly independent from data: build tree structure of feature space



- Run the  $k$ -means (or LBG) algorithm

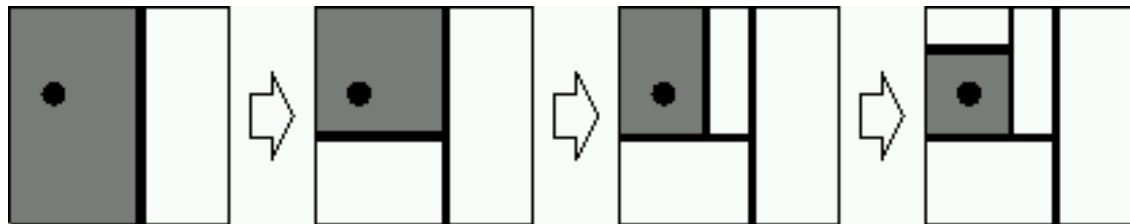
# Finding Codebooks of Reference Vectors

## Tree-Structured Feature Space



1. initialize one node for all data
2. compute the direction of the largest deviation for all nodes
3. split the node with the largest deviation in two (hyperplane through mean)
4. if not yet satisfied then goto step 2

Quantization:



starting at root node, compare vector  $x$  with current node's hyperplane, descend to node that corresponds to the side of the hyperplane on which  $x$  lies

# Finding Codebooks of Reference Vectors

## The $k$ -Means Algorithm

- Step 1 Initialize: given are value of  $k$ , sample vectors  $v_1, \dots, v_T$   
Initialize any  $k$  means (e.g.  $\mu_i = v_i$ )
- Step 2 Nearest-Neighbor classification: Assign every vector  $v_i$  to its class' centroid  $\mu_{f(i)}$
- Step 3 Codebook update: Replace every mean  $\mu_i$  by the mean of all sample vectors that have been assigned to it
- Step 4 Iteration: If not satisfied, yet, then go to step 2

possible stop-criterion:

- A fixed number of iterations
- The average (maximum) distance  $|v_i - \mu_{f(i)}|$  is below a fixed value
- The derivative of the distance is below a fixed value (nothing happens any more)
- Theoretically  $k$ -means can only converge to a local optimum
- Initialization is often critical - repeat  $k$ -means for several codebook sets
- OR: Linde-Buzo-Gray Algorithm, I.e. start with a 1-vector codebook and use splitting algorithm to obtain 2-vector, ...,  $M$ -vector codebook

# **Classification Backup Slides**

# Distortion Measure

- Measures quality of quantization (quantization error)
- Used e.g. as stopping criterion for  $k$ -means algorithm
- Distortion in class  $c$  = expected average distance of a sample vector to its class centroid:  $D_c = E[d(x, c(x))]$
- In practice:  $D_c = ( \sum_x d(x, c(x)) ) / |X|$
- Overall distortion:  $D = \sum_c D_c$
- It is possible to use different distance measures  
most common: Euclidean distance

# Finding Codebooks of Reference Vectors

## Learning Vector Quantization (LVQ)

- Step 1: Given are value of  $k$ , sample vectors  $v_1, \dots, v_T$   
Initialize any  $k$  centroids (e.g.  $\mu_i = v_i$ )
- Step 2: Classification: Assign every vector  $v_i$  its class's centroid  $\mu_{f(i)}$
- Step 3: Update: (a) Move  $\mu_{f(i)}$  towards  $v_i$  by some amount:  
$$\mu'_{f(i)} = \mu_{f(i)} + t \cdot (v_i - \mu_{f(i)})$$
Update: (b) Move  $\mu_{g(i)}$  away from  $v_i$  by some amount:  
$$\mu'_{g(i)} = \mu_{g(i)} - t \cdot (v_i - \mu_{f(i)})$$
- Step 4 Iteration: If not satisfied, yet, then go to step 2

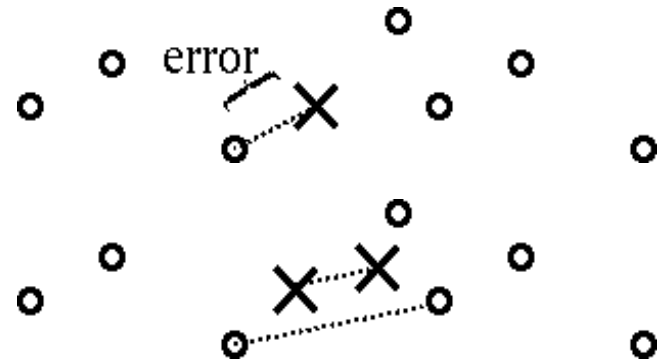
Step 3 (b) only in the **discriminative LVQ2** algorithm.

Then  $g(i)$  is the index of an incorrect but close (the closest) class  
(i.e. supervised learning)

Variation: perform step 3 (and 4) only if  $|v_i - \mu_{f(i)}| < w$

# Distances after Vector Quantization

- The Vector Quantizer is described by
  - A **codebook**: a set of fixed prototype vectors (codeword)
  - A **distortion measure** to match the input vector against each codeword of the codebook
- VQ introduces **quantization error** = distance between vector and its class representative
- Distance between vectors can differ a lot from the distances between their class representatives
- The quantization error depends on the underlying distance measure
- The goal of a good vector quantizer should be to minimize the quantization error

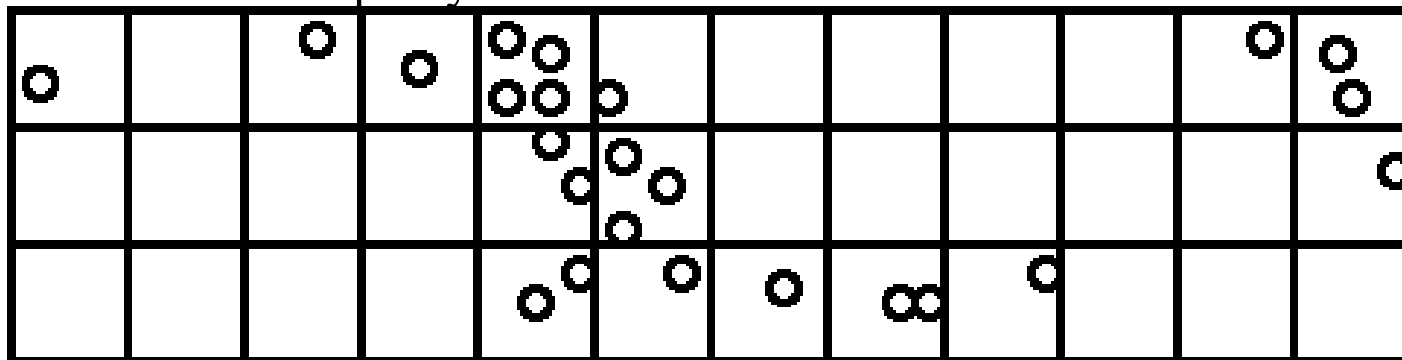


# The Vector Quantization Step

Most straightforward approach:  
partition the vector space into equally sized hypercubes:

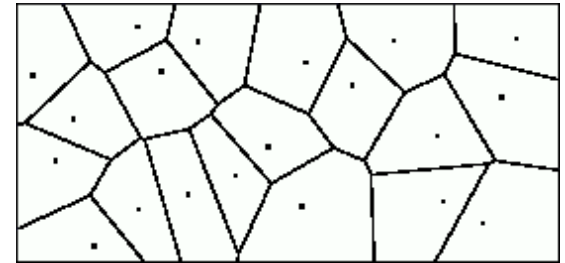
1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36

but what if data is not equally distributed:



# Voronoi Regions or the Nearest Neighbor Classifier

- Instead of using uniform quantization hypercubes, use variably sized areas
- Choose a number of prototypical reference vectors
- The set of points to which a reference vector is the closest is called the vector's **Voronoi region**
- Classification of a vector  $v$  is done by assigning it the **nearest neighbor** of all reference vectors, i.e. the reference vector of the Voronoi region in which  $v$  lies
- Questions to be solved:
  - What distance measure to use to decide nearest neighbor
  - Can we find the nearest neighbor faster than computing the distances to all reference vectors



# Speeding Up the Nearest-Neighbor Search

Often: many reference vectors (16 ... 1024) per class,  
many classes ( $10^4$  ...  $10^5$ )  
high dimensionality ( $d = 16$  ... 48)  
=> huge computational effort for nearest neighbor search

Two different principles:

- Exact classification:

The nearest neighbor is always found

- Abort distance computation when possible
- Build hierarchical tree structure of the feature space

- Approximative classification:

Sometimes only one of the nearest neighbors is found

- Use simple but incorrect distance measure
- Build hierarchy of reference vectors

# Speeding Up Nearest Neighbor Search

Don't compute full distance:  $(x[1]-r_l[1])^2 + (x[2]-r_l[2])^2 + \dots + (x[d]-r_l[d])^2$

if  $(x[1]-r_m[1])^2 + \dots + (x[i]-r_m[i])^2 > (x[1]-r_n[1])^2 + \dots + (x[d]-r_n[d])^2$

then there is no need to compute  $(x[i+1]-r_m[i+1])^2 \dots (x[d]-r_n[d])^2$

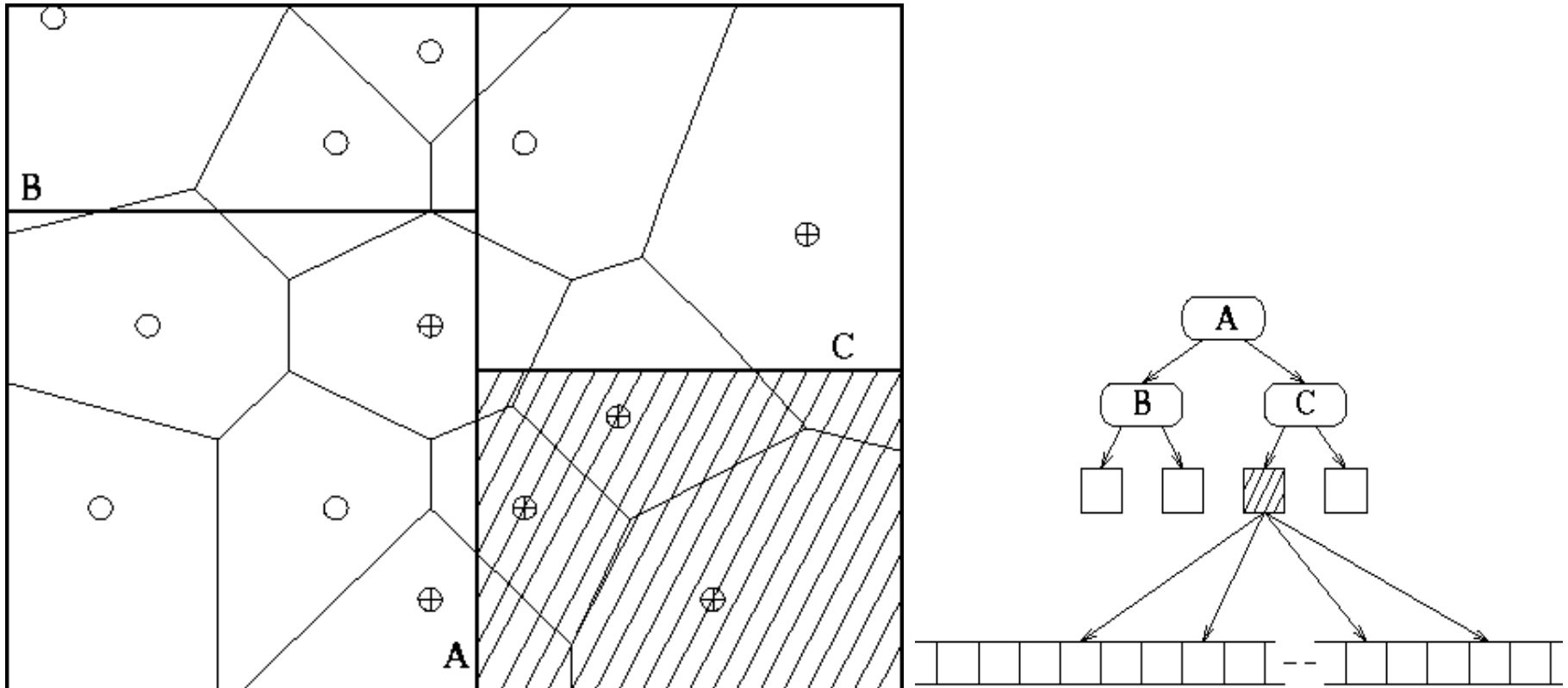
$\Rightarrow$  abort distance computation for reference vector  $r_m$

# Speeding Up Nearest Neighbor Search

Idea: Descend hierarchical tree structure.

One descent-step is only one scalar comparison with a hyperplane.

Eventually the leaves of the tree contain a (small) list of nearest neighbor candidates:



# Learning Parameters of Mixtures of Gaussians

Maximum likelihood estimation of a single parameter is easy:

We have sample vectors  $x_1, \dots, x_T$ , we want to maximize  $\prod_x p(x|c_x)$

where we have the standard Gaussian parameters  $\Sigma_{c_x}$  and  $\mu_{c_x}$ .

When  $c_x$  is given then life is easy

BUT:

When optimizing Gaussian Mixtures, we additionally have to optimize the class-assignment function and the mixture weights.

Unfortunately, the optimal class assignment  $c_x$  depends on the choice of the  $\Sigma$  and  $\mu$ , and the optimal values for the  $\Sigma$  and  $\mu$  depend on the class assignment function.

⇒ No analytic way, solve the problem by iterative procedure

**Expectation Maximization (EM) Algorithm**

# The Expectation Maximization Algorithm for Gaussian Mixtures

given: a Gaussian Mixture

$$P(x_t|s_j) = \sum_{k=1}^{n_j} c_{jk} \cdot \frac{1}{\sqrt{(2\pi)^n |\Sigma_{jk}|}} e^{-\frac{1}{2}(x_t - \mu_{jk})^T \Sigma_{jk}^{-1} (x_t - \mu_{jk})}$$

and training data  $x_{t=1..T}$

Now let  $E[x_t \in k] =: \gamma_{tk}$ , i.e. the contribution of Gaussian  $k$  to  $p(x_t|s_j)$

The the update rules for the Gaussian mixture coefficients are:

$$\overline{c_k} = \frac{1}{T} \sum_{t=1}^T \gamma_{tk}$$

$$\overline{\mu_k} = \frac{1}{\sum_t \gamma_{tk}} \sum_{t=1}^T \gamma_{tk} x_t$$

$$\overline{\Sigma_k} = \frac{1}{\sum_t \gamma_{tk}} \sum_{t=1}^T \gamma_{tk} (x_t - \mu_k)(x_t - \mu_k)^T$$

# Decision Function $g(x)$

$g(x) > 0 \Rightarrow$  Class A  
 $g(x) < 0 \Rightarrow$  Not class A  
 $g(x) = 0 \Rightarrow$  No decision

$$g(\vec{x}) = \sum_{i=1}^n w_i x_i + w_0 = \vec{w}^T \vec{x} + w_0$$

$$x = (x_1, \dots, x_n)^T$$

$$w = (w_1, \dots, w_n)^T$$

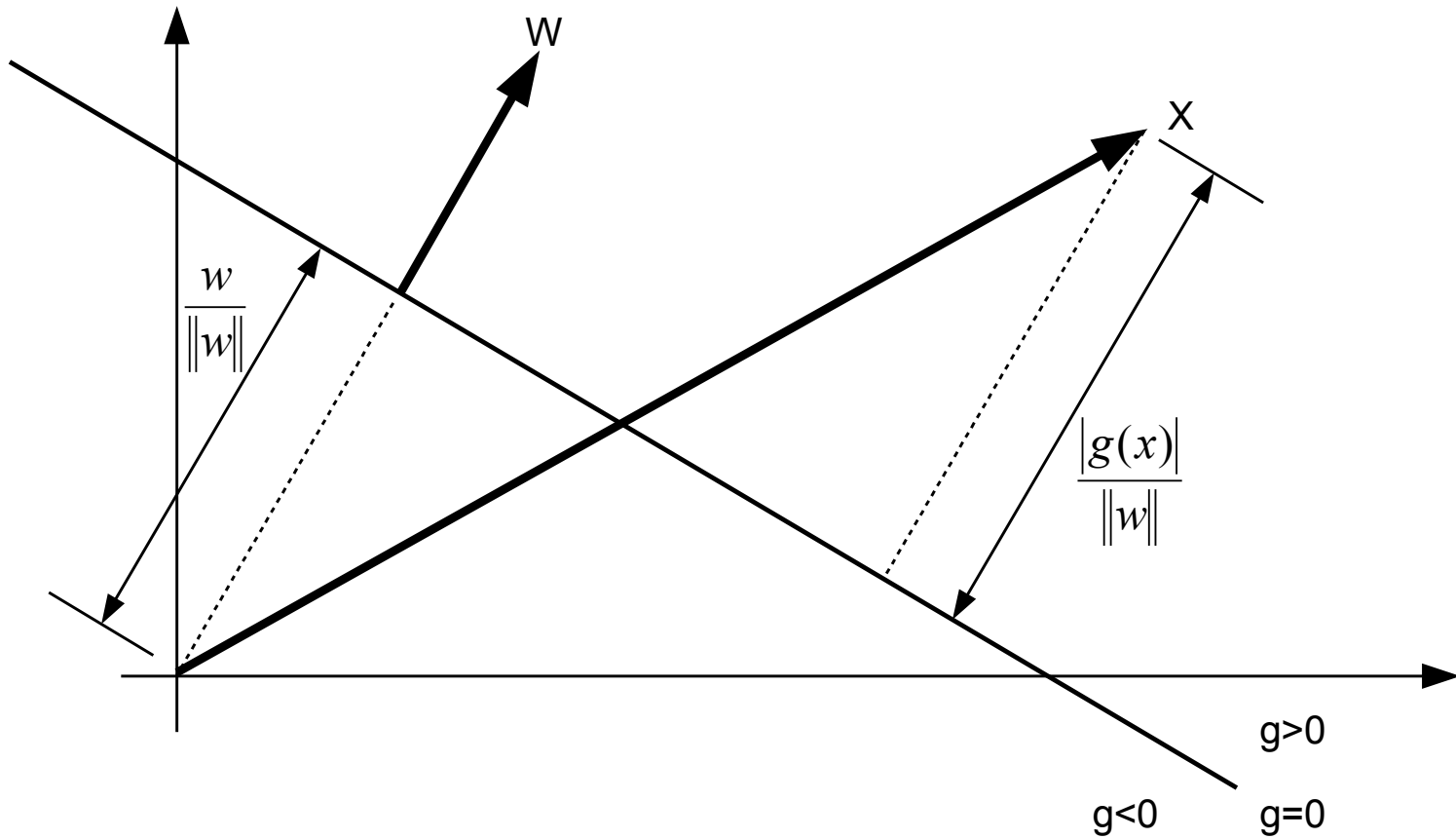
$$w_0$$

Feature vector

Weight vector

Threshold weight

# Linear Discriminant Function I



# Linear Discriminant Functions II

Hyperplane H: 
$$g(\vec{x}) = \sum_{i=1}^n w_i x_i + w_0 = \vec{w}^T \vec{x} + w_0 = 0$$

$$\Rightarrow \vec{x} = q \frac{\vec{w}}{\|\vec{w}\|} + r \frac{\vec{w}}{\|\vec{w}\|} + \vec{x}_p$$

Vector  $q \frac{\vec{w}}{\|\vec{w}\|}$  equals: 
$$g\left(q \frac{\vec{w}}{\|\vec{w}\|}\right) = 0 = q \|\vec{w}\| + w_0 \Rightarrow \boxed{q = -\frac{w_0}{\|\vec{w}\|}}$$

And with 
$$g(\vec{x}) = \vec{w}^T q \frac{\vec{w}}{\|\vec{w}\|} + \vec{w}^T r \frac{\vec{w}}{\|\vec{w}\|} + \vec{w}^T \vec{x}_p + w_0 = -w_0 + r \|\vec{w}\| + w_0$$

We get 
$$\boxed{r = \frac{g(\vec{x})}{\|\vec{w}\|}}$$

# Fisher-Linear Discriminat

- Dimensionality Reduction;
- Project a set of multidimensional points onto a line  $y = \vec{w}^T \vec{x}$
- Fisher Discriminant is function that maximizes criterion

$$g(x) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2},$$

where  $\tilde{m}_i = \frac{1}{n_i} \sum_{y \in Y_i} y$  sample mean for projected

samples  $\tilde{s}_i^2 = \sum_{y \in Y_i} (y - \tilde{m}_i)^2$  scatter for projected samples

# Fisher Linear Discriminant

Fisher's linear discriminant:

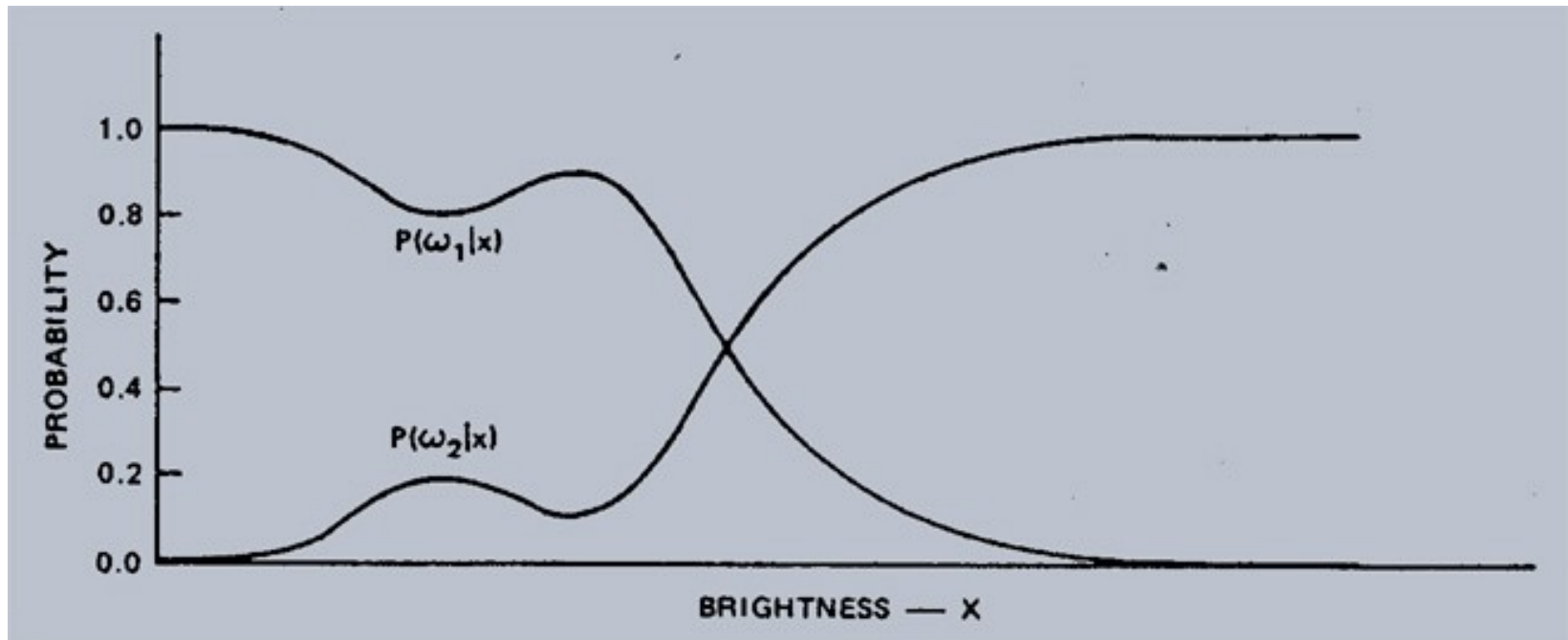
$$\vec{w} = S_w^{-1} (\vec{m}_1 - \vec{m}_2)$$

$$S_w = S_1 + S_2$$

(within-class scatter matrix)

$$S_i = \sum_{x \in X_i} (\vec{x} - \vec{m}_i)(\vec{x} - \vec{m}_i)^t$$

# A posteriori probabilities



# Risk

- May refuse to make decision in ambiguous cases (→ margin)
- Estimate cost of each decision (some more costly than others)

$$\Omega = \{\omega_1, \dots, \omega_s\} \text{ s states of nature}$$
$$A = \{\alpha_1, \dots, \alpha_a\} \text{ a possible actions}$$

# Risk

## Loss function

$\lambda (\alpha_i / \omega_j)$       loss of action  $\alpha_i$  , given state  $\omega_j$

$$P (\omega_j / \vec{X}) = \frac{p (\vec{X} / \omega_j) P (\omega_j)}{p (\vec{X})}$$

Expected loss of taking action  $\alpha_i$

$$R (\alpha_i / \vec{X}) = \sum_{j=1}^s \lambda (\alpha_i / \omega_j) P (\omega_j / \vec{X})$$

↑  
conditional risk

# Risk

$$R(\alpha_i / \vec{x}) = \sum_{j=1}^s \lambda(\alpha_i / \omega_j) P(\omega_j / \vec{x})$$

Minimize expected loss by selecting  $\alpha_i$  action that minimizes conditional risk.

Two category case:

$$R(\alpha_1 / \vec{x}) = \lambda_{11} P(\omega_1 / \vec{x}) + \lambda_{12} P(\omega_2 / \vec{x})$$

$$R(\alpha_2 / \vec{x}) = \lambda_{21} P(\omega_1 / \vec{x}) + \lambda_{22} P(\omega_2 / \vec{x})$$

# Risk

decide  $\omega_1$   
if

$$R(\alpha_1 / \vec{x}) < R(\alpha_2 / \vec{x})$$

$\omega_1$   
decide

$$(\lambda_{21} - \lambda_{11}) P(\omega_1 / \vec{x}) > (\lambda_{12} - \lambda_{22}) P(\omega_2 / \vec{x})$$

if

$$\omega_1 (\lambda_{21} - \lambda_{11}) p(\vec{x} / \omega_1) P(\omega_1) > (\lambda_{12} - \lambda_{22}) p(\vec{x} / \omega_2) P(\omega_2)$$

decide  
if  $\omega_1$

$$\frac{p(\vec{x} / \omega_1)}{p(\vec{x} / \omega_2)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \cdot \frac{P(\omega_2)}{P(\omega_1)}$$


Likelihood  
Ratio



independent  
of  $x$

decide  
if

# Minimum Error Rate Classification

- Decision rule to minimize error rate
- Define zero-one loss function

$$\lambda(\alpha_i / \omega_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \quad i, j = 1 \dots c$$

$$\begin{aligned} R(\alpha_i / \vec{x}) &= \sum_{j=1}^c \lambda(\alpha_i / \omega_j) P(\omega_j / \vec{x}) \\ &= \sum_{j \neq i} P(\omega_j / \vec{x}) \\ &= 1 - P(\omega_i / \vec{x}) \end{aligned}$$

# Minimum Error Rate Classification

- To minimize risk and the average probability of error, select  $i$  that maximizes posterior  $P(\omega_i / \mathbf{X})$
- Decide  $\omega_i$  if  $P(\omega_i / \mathbf{X}) > P(\omega_j / \mathbf{X})$  for all  $j \neq i$

# Gaussian Classifier

Univariate Normal Density:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{\vec{x} - \vec{\mu}}{\sigma}\right)^2\right]$$
$$\sim N(\vec{\mu}, \sigma^2)$$

Multivariate Density:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu})^t \Sigma^{-1} (\vec{x} - \vec{\mu})\right]$$
$$\sim N(\vec{\mu}, \Sigma)$$

$$g_i(x) = -\frac{1}{2}(\mathbf{x} - \mu_i)^t \Sigma_i^{-1} (\mathbf{x} - \mu_i) - \frac{d}{2} \log(2\pi)$$
$$- \frac{1}{2} \log |\Sigma_i| + \log P(\omega_i)$$

# Decision Boundaries for a Minimum-Distance Classifier

