INTRODUCTION

Pattern Recognition (PR)

- Statistical PR
- Syntactic PR
- Fuzzy logic PR
- Neural PR

Example - basketball players and jockeys

We will keep practical applicability in mind:

- Are PR techniques suitable to the problem?
- Can we develop useful models and determine model parameters?
- Are there available formal tools and applicable heuristics?
- Does a computationally practical solution exist?

Applications

- Image processing, analysis, machine vision
- Seismic analysis
- Radar signal analysis
- Face recognition
- Speech recognition
- Fingerprint identification
- Character recognition
- Medical diagnosis
- etc.

PR process:

information reduction
information labeling
information mapping

C class membership space
P pattern space
F measurement space

G relations between classes and patterns
M relations between patterns from P and measurements from F

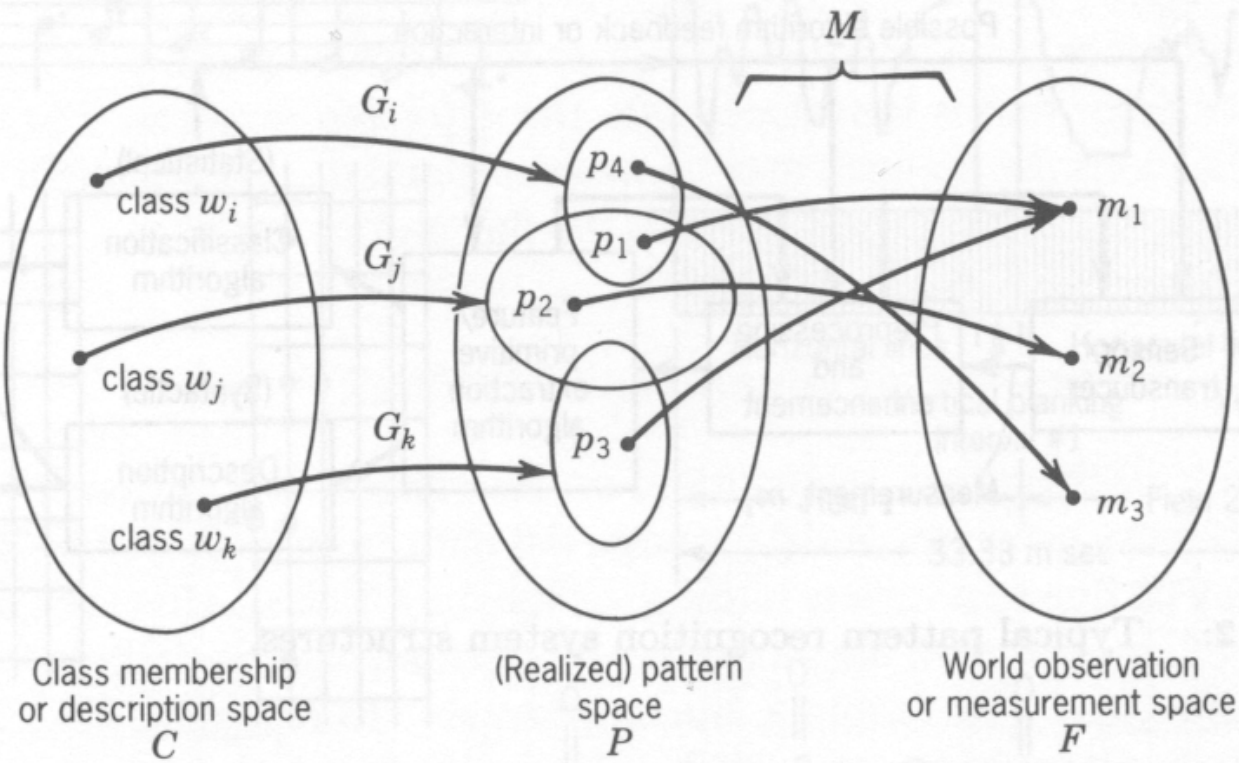


Figure 1: Mappings in an abstract representation of pattern generation/classification/interpretation systems.

PR problem (StatPR and SyntPR):
Given measurements m_i, we look for a method to identify and invert mappings M and G_i for all i.

Unfortunately, these mapping are not functions and are not "onto" ==> are not invertible.

Different patterns may have the same measurements ==> ambiguity.

- M reflects our view of the world ... good measurements are more likely to produce good classification.
- Patterns from the same class are "close" in the P space.
- Measurements from the same class are (often) not close in the F space.

Example ... red and blue cars are close in P; while red and blue color are far in F.

100% correct classification may not be feasible.

Apples vs. hand-grenades example ... sometimes it is also useful to consider the cost of mis-classification.

Structure of a typical PR system

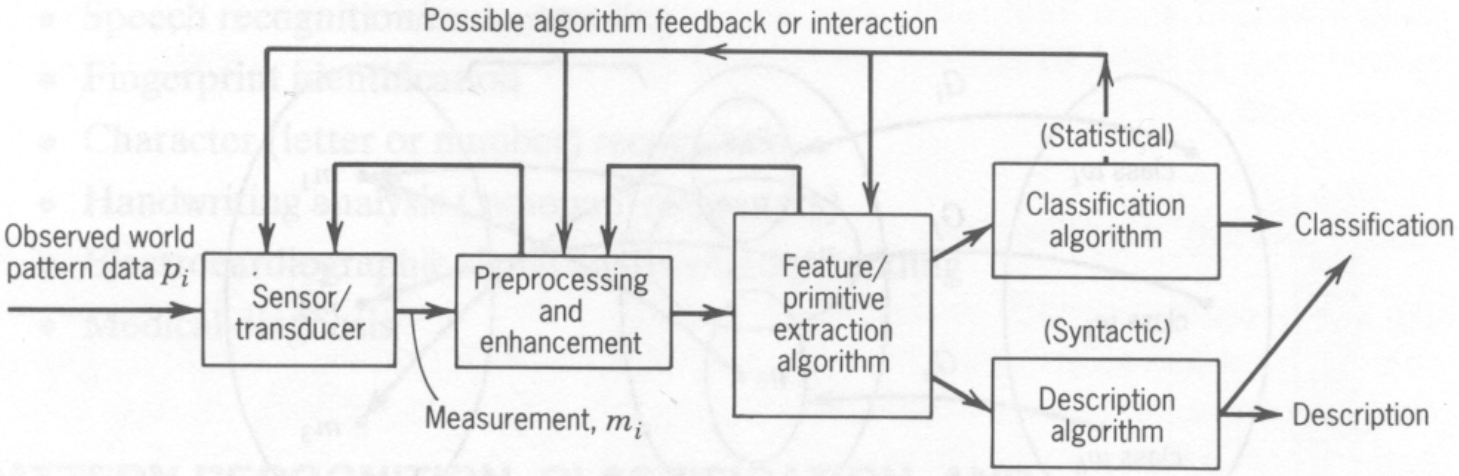


Figure 2: Typical pattern recognition system structures.

Patterns and Features:

Pattern ... a set of measurements, often in a vector form (StatPR) or graph/grammar form (SyntPR).

Features ... Any extractable measurements used.

- numerical size
- symbolic color
- complex primitives

Feature extraction - measurements extracted from data.

- may require significant computational effort (e.g., extracting shape properties of 3D objects)
- extracted features may be “noisy” ... may have errors

Extracted features:

- computationally feasible
- good discriminative power
- good descriptive power

Feature selection - selection of features from the set of available features.

Pattern Distortion

Measurements may be “noisy” ... color varies with lighting, shape varies with viewing angle, etc.

- Features should be invariant to such changes

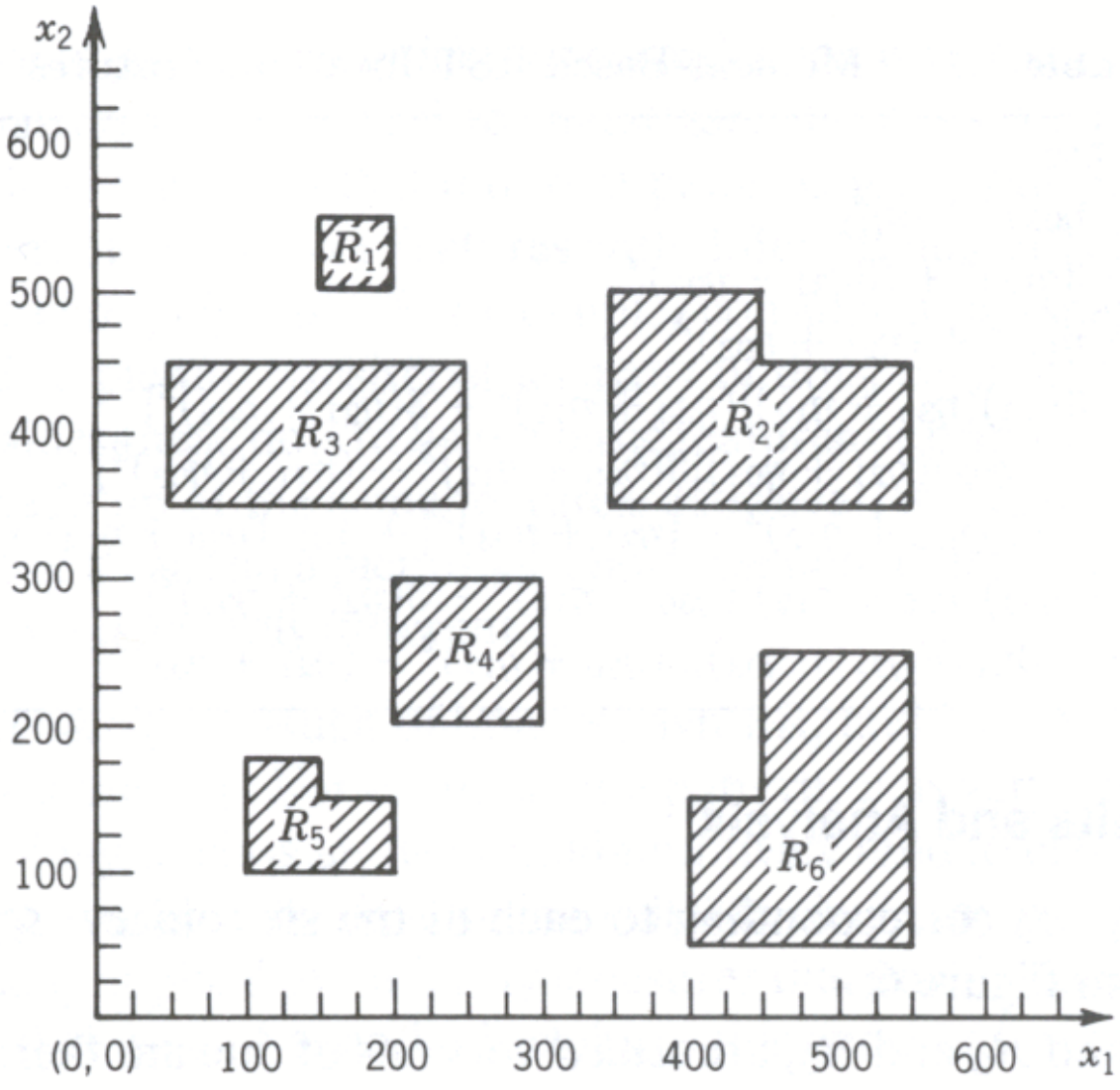


Figure 6: Example of 2-D regions for RST feature extraction.

RST-invariant moments (well-known 7 features based on statistical central moments)

Table 1.1: Moment-Based RST Invariant Features

$$\phi_1 = \eta_{20} + \eta_{02}$$
$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$
$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$
$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

ϕ_i invariant to RST transforms

Table 1.2: Moment-Based Features ϕ_i for R_j $i = 1, 2, \ldots 7$; $j = 1, 2, \ldots 6$

Feature	R_1	R_2	R_3	R_4	R_5	R_6
ϕ_1	$1.67E-01$	$1.94E-01$	$2.083E-01$	$1.67E-01$	$1.94E-01$	$1.94E-01$
ϕ_2	$0.00E+00$	$6.53E-03$	$1.56E-02$	$0.00E+00$	$6.53E-03$	$6.53E-03$
ϕ_3	$0.00E+00$	$1.02E-03$	$0.00E+00$	$0.00E+00$	$1.02E-03$	$1.02E-03$
ϕ_4	$0.00E+00$	$4.56E-05$	$0.00E+00$	$0.00E+00$	$4.56E-05$	$4.56E-05$
ϕ_5	$0.00E+00$	$4.25E-09$	$0.00E+00$	$0.00E+00$	$4.25E-09$	$4.25E-09$
ϕ_6	$0.00E+00$	$1.70E-06$	$0.00E+00$	$0.00E+00$	$1.70E-06$	$1.70E-06$
ϕ_7	$0.00E+00$	$-8.85E-09$	$0.00E+00$	$0.00E+00$	$-8.85E-09$	$-8.85E-09$

Concept of Similarity

Patterns from one class are similar to each other.

However, quantification of similarity is often difficult.

Feature Vector and Feature space

Feature vector $\underline{x}$ ... d-dimensional

Feature space R^d ... if features are unconstrained
subspace of R^d ... if features are constrained

Feature vectors ... used in StatPR, NeurPR

Definitions

- Classification assigns input data to one or more of c prespecified classes based on extraction of significant features or attributes and the analysis of these attributes
- Recognition the ability to classify
- Don't know class a dummy c+1st class
- Description structural description of the input pattern
- Class set of patterns known to originate from the same source in C.
- Noise resulting from non-ideal circumstances
    - measurement errors
    - feature extraction errors
    - training data errors

The key to success is often to identify suitable attributes (features, descriptions) and to form a good measure of similarity and an associated matching process.

Classifiers, Discriminant functions

(StatPR, NeurPR)

Nonoverlapping regions in R^d. Border of each region is a decision boundary.
Discriminant functions $g_i(\underline{x})$ partition R^d, i=1,..,c.

Decision rule:
Assign $\underline{x}$ to class w_m (Region R_m)

if $g_m(\underline{x}) > g_i(\underline{x})$ for all i; $i \neq m$

Then, $g_k(\underline{x}) = g_l(\underline{x})$ defines a decision boundary between classes k and l.

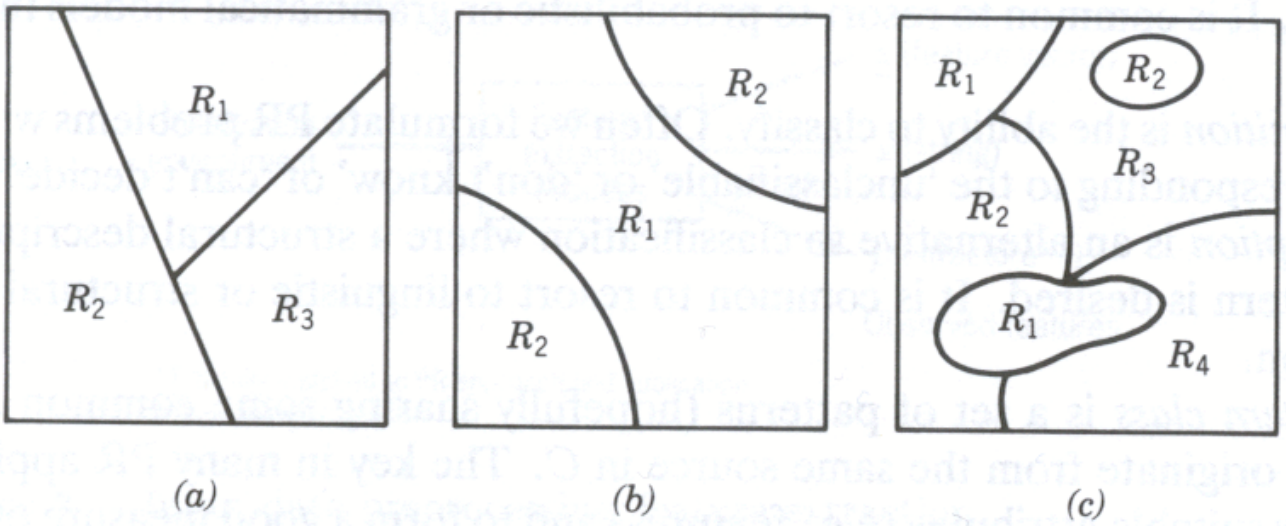


Figure 9: Sample decision regions (2-D example).
(a) Linear (piecewise).
(b) Quadratic (hyperbolic).
(c) (Relatively) general.

$$g_1(\underline{x}) = \| \underline{x} - \underline{x}_1 \|$$

$$g_2(\underline{x}) = \| \underline{x} - \underline{x}_2 \|$$

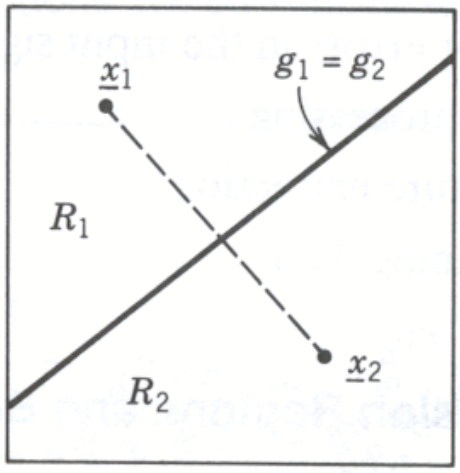


Figure 10: Discriminant function and corresponding decision regions (minimum distance classifier).
(a) Discriminant functions.
(b) Corresponding partition of R^2.

Training and Learning in PR Systems

Maximum available (and useful) information should be employed in the designed PR system. Supervised learning approaches serve as an example.

Training set H - a pair of a pattern & class info.
In Synt PR we also need a set of counter-examples.

Although, unsupervised approaches exist - cluster analysis.

PR approaches

Statistical
Structural (Syntactic)
Fuzzy
Neural

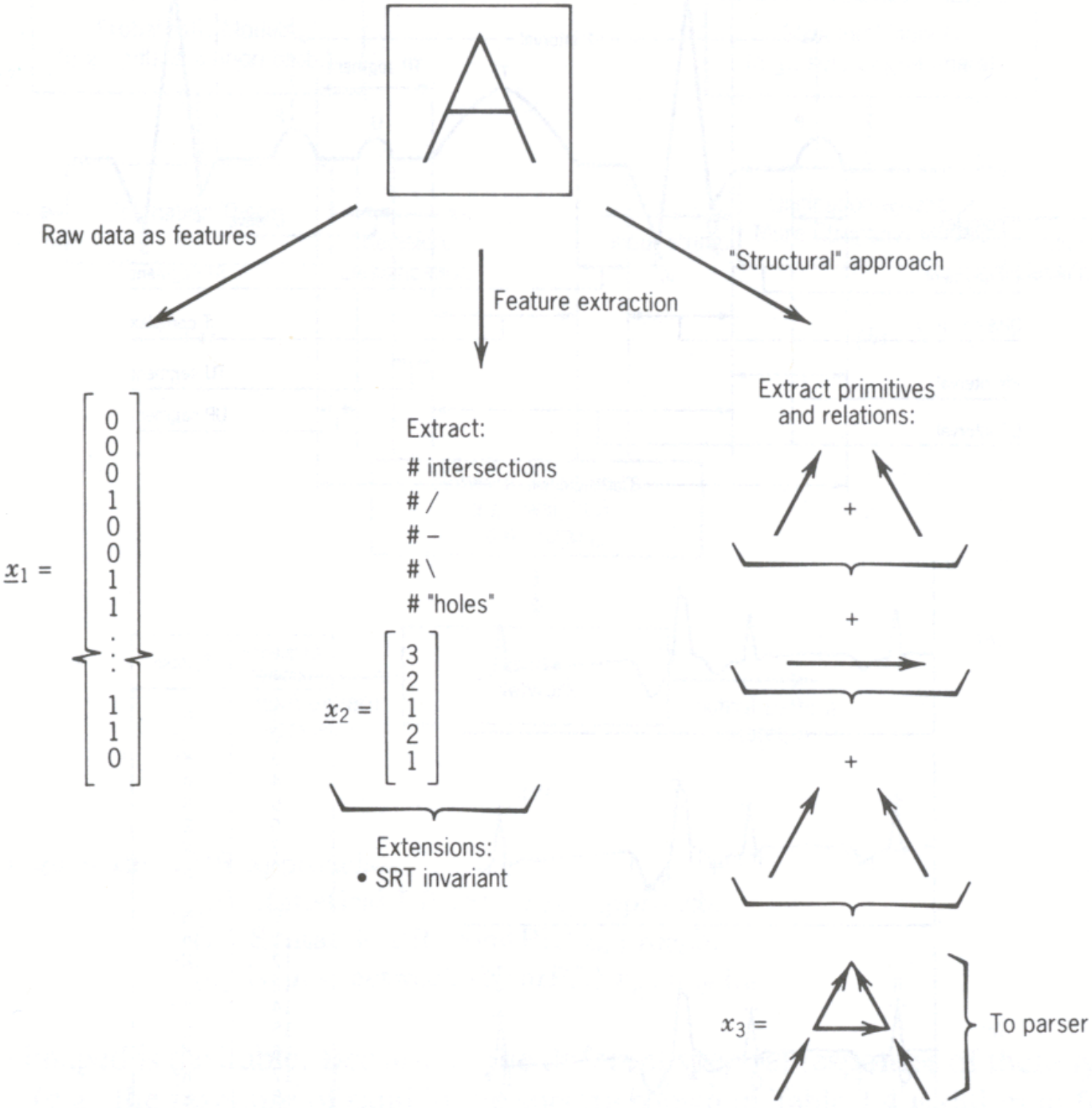


Figure 14: Example of PR approaches.
(a) Input pattern.
(b) Possible feature extraction approaches.

Table 1.3: Comparing StatPR, SyntPR, and NeurPR Approaches

	StatPR	**SyntPR**	**NeurPR**
1. Pattern Generation (Storing) Basis	Probabilistic Models	Formal Grammars	Stable State or Weight Array
2. Pattern Classification (Recognition/ Description) Basis	Estimation/ Decision Theory	Parsing	Based on (Predictable) Properties of NN
3. Feature Organization	Feature Vector	Primitives and Observed Relations	Neural Input or Stored States
4. Typical Learning (Training) Approaches			
Supervised:	Density/distribution estimation (usually parametric)	Forming grammars (heuristic or grammatical inference)	Determining NN system parameters (e.g., weights)
Unsupervised:	Clustering	Clustering	Clustering
5. Limitations	Difficulty in expressing structural information	Difficulty in learning structural rules	Often little semantic information from network

Procedures for PR system engineering

Step 1: Study the classes of patterns under consideration to develop possible characterizations. This includes assessments of (quantifiable) pattern structure and probabilistic characterizations, as well as exploration of possible within-class and interclass similarity/dissimilarity measures. In addition, possible pattern deformations or invariant properties and characterization of 'noise' sources should be considered at this point.

Step 2: Determine the availability of feature/measurement data.

Step 3: Consider constraints on desired system performance and computational resources (e.g., parts/minute, classification accuracy).

Step 4: Consider the availability of training data.

Step 5: Consider the availability of suitable and known PR techniques (e.g., StatPR, SyntPR, and clustering). an overall PR system structure.

Step 6: Develop a PR system simulation. This may involve choosing models, grammars, or network structures.

Step 7: Train the system.

Step 8: Simulate system performance.

Step 9: Iterate among the above steps until desired performance is achieved.

Statistical PR

Approaches to statistical classifier design

- calculating a posteriori probabilities from a priori probabilities … Bayesian Theory
- minimizing classification losses

Both strategies can be implemented using discriminant functions

---

Bayesian theory
- fundamental statistical approach
- quantifying trade-offs between classification decisions using probability and costs accompanying such decisions

Specific example – classifying sea bass and salmon

fish appears randomly on a belt, let w denote state of nature
$w=w_1$ … sea bass
$w=w_2$ … salmon

state of nature is unpredictable, therefore must be probabilistically described

How likely is it that salmon appears?

… there is some a priori probability
$P(w_1)$ that next fish on conveyor is a sea bass
$P(w_2)$ that next fish on conveyor is a salmon

assuming no other fish can appear, then
$P(w_1) + P(w_2) = 1$

A priori probabilities may be known in advance, e.g., based on the time of the year, location, etc.

Assume that we must make a decision without "seeing" the fish …

# of classes c=2, no features ... d=0, a priori probabilities of classes $P(w_1)=0.9$, $P(w_2)=0.1$

Discriminant function:
if $P(w_1) > P(w_2)$ … choose w_1 otherwise choose w_2

P(error) =
$P(\text{choose } w_2|w_1)P(w_1) + P(\text{choose } w_1|w_2)P(w_2)$

if we always choose w_1 ...

$P(\text{error}) = 0 * 0.9 + 1 * 0.1 = 0.1$

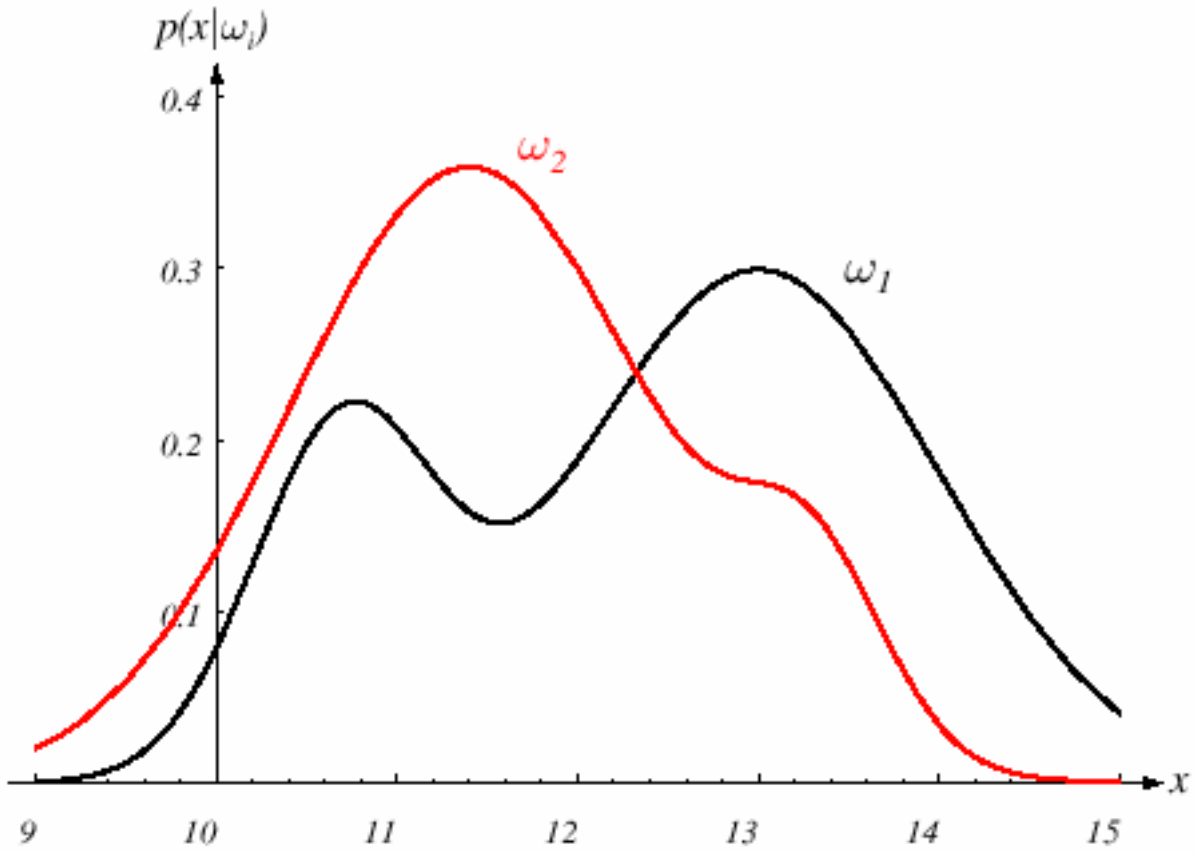
Probability of error ... 10% ==> minimal error

Works well if $P(w_1) >> P(w_2)$,
not well at all if $P(w_1) = P(w_2)$.

To improve classification correctness, we use features that can be measured on fish.

Assume a single continuous feature x,
x is a continuous random variable,
p(x|w) is *class-conditional probability density function,*
its distribution depends on the state of nature (class) w.

Difference between $p(x|w_1)$ and $p(x|w_2)$ … difference in feature value between populations of sea bass and salmon:



Assume that we know a priori probabilities $P(w_i)$ and densities $p(x|w_i)$, suppose that we measure x …
… what can be said about w … the status of nature = classification of fish?

Joint probability density of having a pattern from category w_j that has a feature value x:

$$p(w_j , x) = P(w_j|x)\, p(x) = p(x|w_j)\, P(w_j)$$

==> Bayes rule:

$$P(w_j|x) = p(x|w_j)[P(w_j) / p(x)]$$

and here (for 2 classes):
$$p(x) = p(x|w_1)\, P(w_1) + p(x|w_2)\, P(w_2)$$

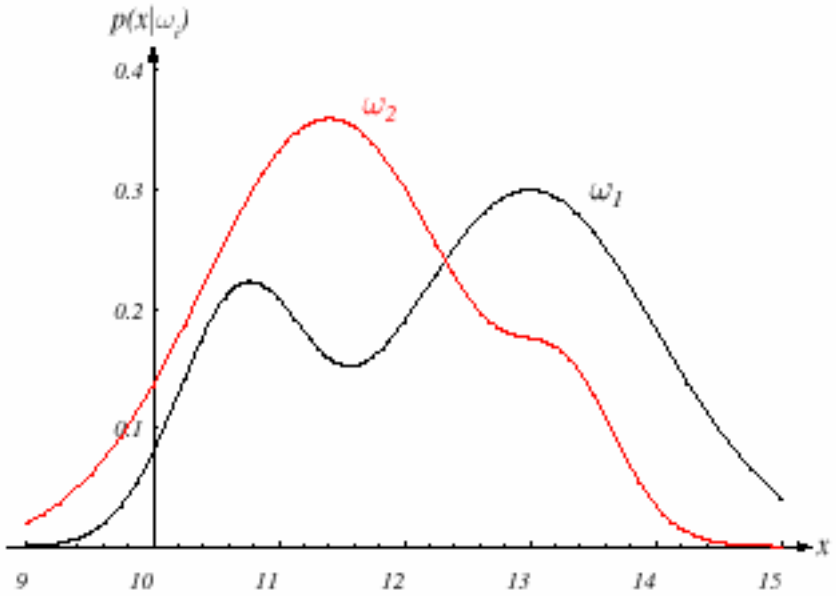
Bayes rule is informally:

$$posterior = (likelihood \times prior)/(evidence)$$

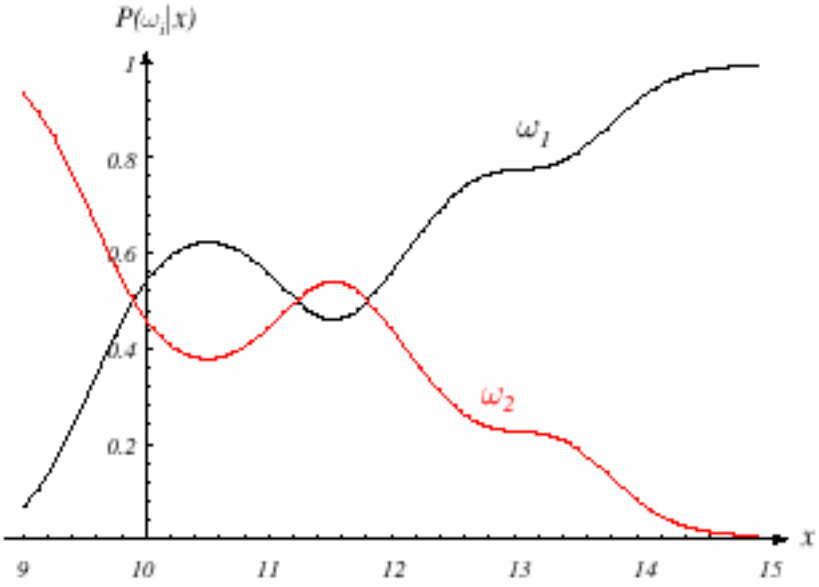
$p(x|w_j)$ is likelihood of w_j with respect to x
==> other things being equal, x is more likely to happen for class w_j

p(x) is common to all class conditional probabilities and can be eliminated
p(x) is mostly a scaling factor and can be omitted for classification purposes (scales sum of a posteriori probs to 1)

Likelihood graph:



if $P(w_1) = 2/3$ and $P(w_2) = 1/3$ then:



==> by observing x, *a priori* probability $P(w_j)$ can be converted to *a posteriori* probability $P(w_j|x)$.

Error assessment

P(error) = P(x is assigned to a wrong class)

for c=2

$P(error) = P(w_2|w_1)P(w_1) + P(w_1|w_2)P(w_2)$

Classification error:

Minimization of error:
deciding w_1 for $P(w_1|x) > P(w_2|x)$
and deciding w_2 otherwise

Minimization of the classification error:

choose w_1 if $p(x|w_1)P(w_1) > p(x|w_2)P(w_2)$
choose w_2 if $p(x|w_1)P(w_1) < p(x|w_2)P(w_2)$

Rigorous solution to the problem ... calculating a posteriori probabilities using a priori probabilities:

Special cases:
if $p(x|w_1) = p(x|w_2)$
… decision only depends on prior probabilities

if $P(w_1) = P(w_2)$
… decision only depends on likelihood

Generalization:

- more than one feature
- more than 2 classes
- allow other action than classification
- introducing loss function more general than probability of error

more features than 1
→ replaces scalar x with a vector x from a d-dimensional feature space R^d

more classes than 2
→ deciding w_k for $P(w_k|x) > P(w_m|x)$ for all m<>k

allow other action than classification
.. → allows not to classify = don't know class

loss function
→ weighting decision costs (apples/hand-grenades)

$[w_1, w_2, \ldots w_c]$ … finite set of c classes

$[\alpha_1, \alpha_2, \ldots \alpha_a]$ … finite set of *a* possible actions

$\lambda(\alpha_i|w_j)$ … loss function
- loss resulting from taking action α_i when the class is w_j

x … d-component feature vector (random variable)

$p(\mathbf{x}|w_j)$ … state-conditional probability density function where w_j is the true class assignment

$P(w_j)$ … A priori probability of class w_j

A posteriori probability from Bayes formula:

$$P(w_j|\mathbf{x}) = p(\mathbf{x}|w_j)\, P(w_j) / p(\mathbf{x})$$

where

$$p(\mathbf{x}) = \Sigma_{j=1\ldots c}\, p(\mathbf{x}|w_j)\, P(w_j)$$

Let x be classified as belonging to w_i. Then

$$P(\text{error}|x) = \Sigma_{k=1}^{c}{}_{(k\neq i)} \quad P(w_k|x)$$

However, Bayes formulation of the classification error may not always represent the risks well.

...

Remember the example of apples vs. hand-grenades

General Measures of Classification Risk

Loss function

	class 1	class 2	...	class c
class 1	λ_{11}	λ_{12}		λ_{1c}
class 2	λ_{21}	λ_{22}		λ_{1c}
...				
class c	λ_{c1}	λ_{c2}		λ_{cc}

Then, the expected classification risk (conditional risk) for feature vector x is:

$$R(\alpha_i|\mathbf{x}) = \Sigma_{\mathbf{j=1...c}}\, \lambda(\alpha_i|w_j)\, P(w_j|\mathbf{x})$$

For each particular observation **x**, the classification risk can be minimized by choosing action a_i that minimizes the conditional risk.

<u>for 2 classes … action dependent on **x** [action $\alpha(\mathbf{x})$]</u>

$$R[\alpha(x) \rightarrow \alpha_1] = R(\alpha_1|\mathbf{x}) = \lambda_{11}\, P(w_1|x) + \lambda_{12}\, P(w_2|x)$$

$$R[\alpha(x) \rightarrow \alpha_2] = R(\alpha_2|\mathbf{x}) = \lambda_{21}\, P(w_1|x) + \lambda_{22}\, P(w_2|x)$$

λ_{11} and λ_{22} are "rewards" for correct classification,

λ_{12} and λ_{21} are losses from incorrect classification

For c classes, the expected risk $R[\alpha(x)]$ that x will be classified incorrectly (using the total probability theorem):

$$R[\alpha(x)] = \int R[\alpha(x)|x]\, p(x)\, dx$$

Therefore, minimizing the conditional risk $R[\alpha(x)|x]$ also minimizes the expected risk $R[\alpha(x)]$.

Usually, $\lambda_{ii} = 0$, no loss associated with a correct classification.

Consider unit loss functions $\lambda_{ij} = 1$ for $i \neq j$

... all errors are equally costly.

Then, (unit losses)

$$R[\alpha(x) \rightarrow \alpha_i] = \Sigma_{k=1}^{c} \lambda_{ik}\, P(w_k|x) =$$

$$= \Sigma_{k \neq i}\, P(w_k|x) = 1 - P(w_i|x)$$

$\Rightarrow$ to minimize the conditional risk, the decision rule has to choose the α_i that maximizes $P(w_i|x)$, that is to choose the maximum a posteriori probability.

$\Rightarrow$

MAXIMUM A POSTERIORI PROBABILITY CLASSIFIER

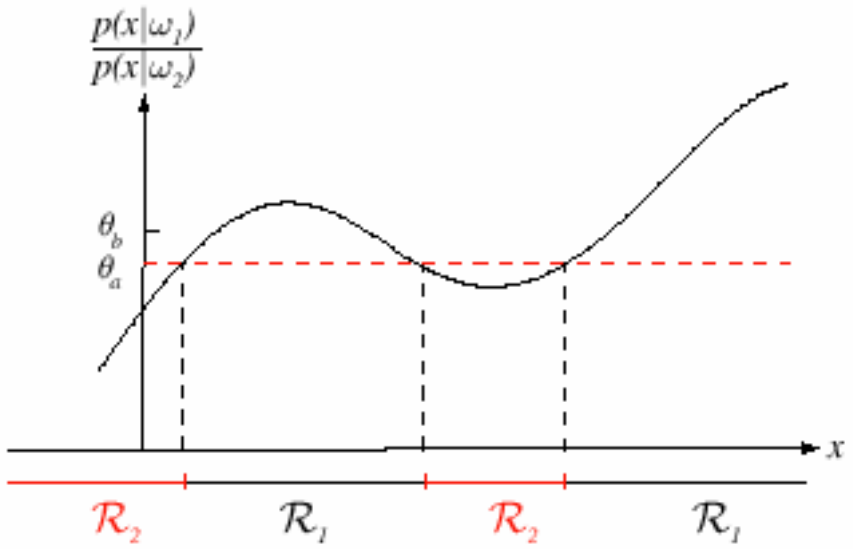
choose w_i

$$P(w_i|x) > P(w_k|x) \quad \text{for all } k \neq i$$

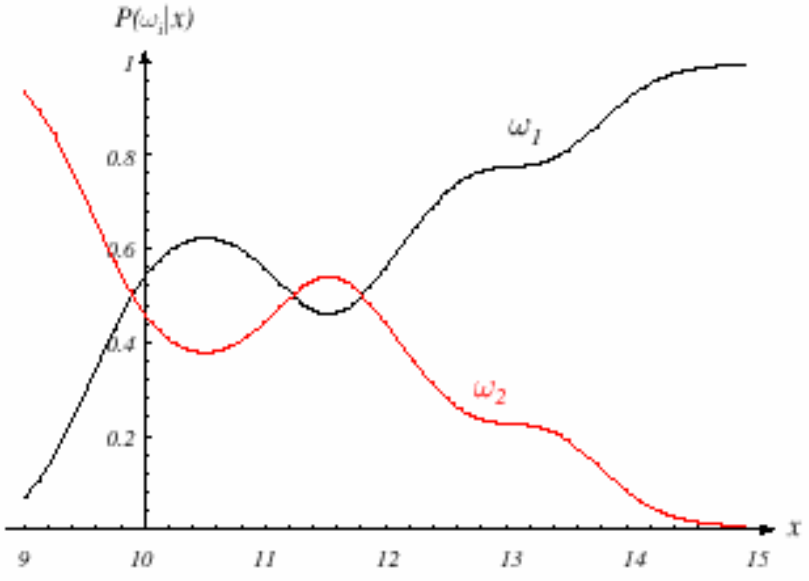
For general formulation of risk:

choose α_i

$$R(\alpha_i|x) > R(\alpha_k|x) \quad \text{for all } k \neq i$$

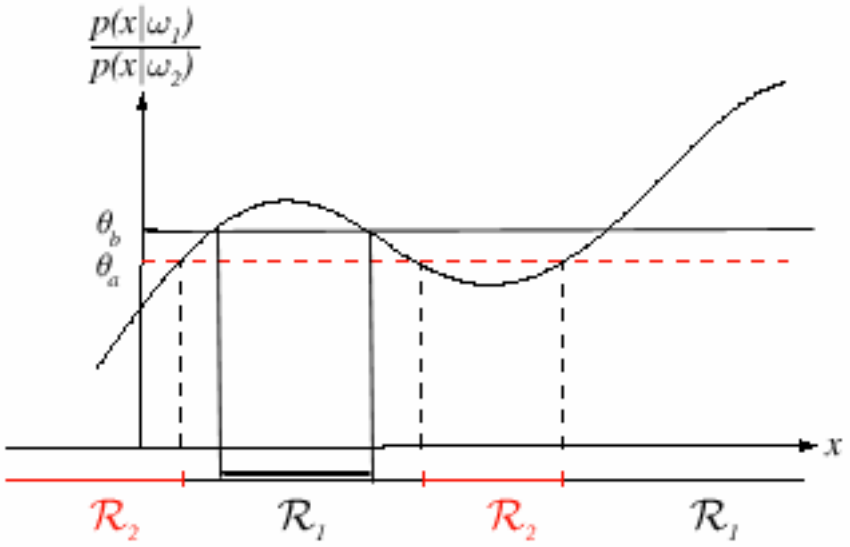


y-axis shows the ratio of a priori probabilities
Using zero-one loss functions λ_{ij}.= 1 for i≠j, decision boundaries are identical to those based on a posteriori probabilities (below)



decision function is dependent on loss functions λ

If classification errors of misclassifying w_1 are greater than for w_2, threshold increases and region for class w_1 shrinks:



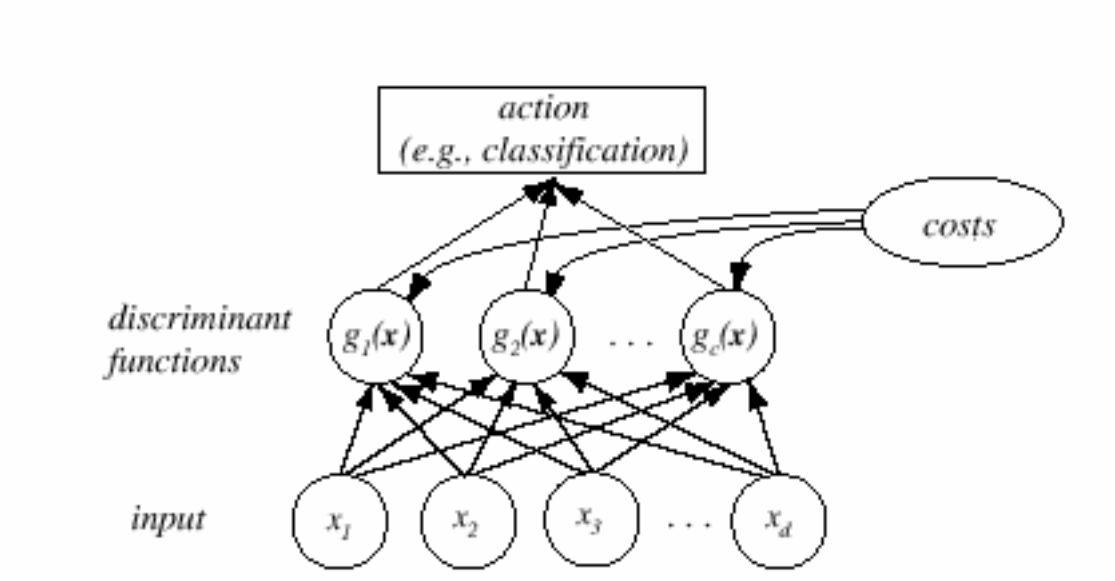
Classifiers, discriminant functions, decision surfaces

Assign feature vector x to class w_m

if $g_m(\underline{x}) > g_i(\underline{x})$ for all i; $i \neq m$

Then, $g_k(\underline{x}) = g_l(\underline{x})$ defines a decision boundary between classes k and l.

Classifier can be viewed as a machine computing discriminant functions:



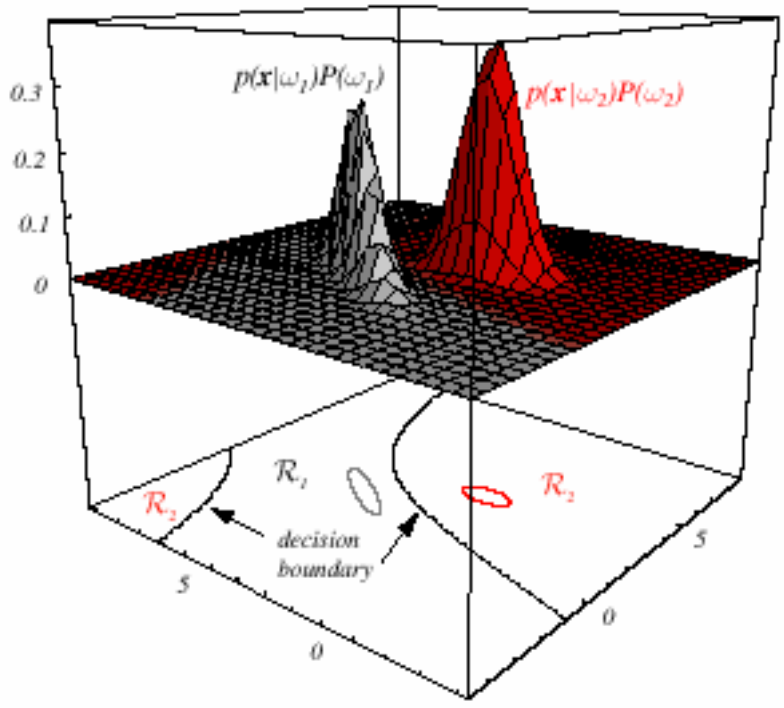
Bayes classifier fits this representation well

$$g_i(x) = - R(\alpha_i|x)$$

and maximum of discriminant function corresponds to minimum risk

or directly

$$g_i(x) = P(w_i|x)$$



Two-class case (dichotomy):

$$g(x) = g_1(x) – g_2(x)$$

Bayes classifier behavior is determined by
1) conditional density $p(x|w_i)$
2) a priori probability $P(w_i)$

Normal (Gaussian) probability density function was studied extensively

Why normal?
- it is well behaved and analytically tractable
- it is an apprpriate model for values randomly distributed around mean μ.

Gaussian models for $p(x|w_1)$

- multidimensional Gaussian distribution

x is a d-dimensional vector $d \times 1$

μ is the mean vector

Σ is the covariance matrix $d \times d$

➔ complete specification of p(x)
using $d+d(d+1)/2$ parameters

Having class-specific mean vectors μ_i and class-specific covariance matrices Σ_i, class-dependent density functions $p(x|w_i)$ can be calculated.

How to estimate mean vectors μ_i and covariance matrices Σ_i? It will come later when we discuss training.

Discriminant functions in this case:

i-th class:

$$g_i(x) = P(w_i|x)$$

Classification ... finding the largest discrimination function:

More generally, any monotonically increasing function of $g_i(x)$ can be used as a valid discrimination function. Assuming equal a priori probabilities,

$$P(w_i|x) = p(x|w_i)$$

and

$$g_i(x) = \log[p(x|w_i)]$$

Let's assume equal Σ for all classes, different mean vectors μ.

General multivariate normal density:

$$p(x) = 1/((2\pi)^{d/2} |\sum|^{1/2}) \ \exp[-1/2 \ (x-\mu)^T \Sigma^{-1} (x-\mu)]$$

➔

➔

$$g_i'(x) = -1/2\ (x-\mu_i)^T \Sigma^{-1} (x-\mu_i) - (d/2)\log(2\pi) - (1/2)\log|\Sigma|$$

class-independent biases can be eliminated

What remains is the squared distance of feature vector x from the i-th mean vector μ_i weighted by the inverse of the covariance matrix

For $\Sigma = I$, Euclidean norm results.

$g'_i(x)$ is largest if $(x-\mu_i)^T \Sigma^{-1} (x-\mu_i)$ is smallest

... minimum distance concept for the same Σ for all classes ...

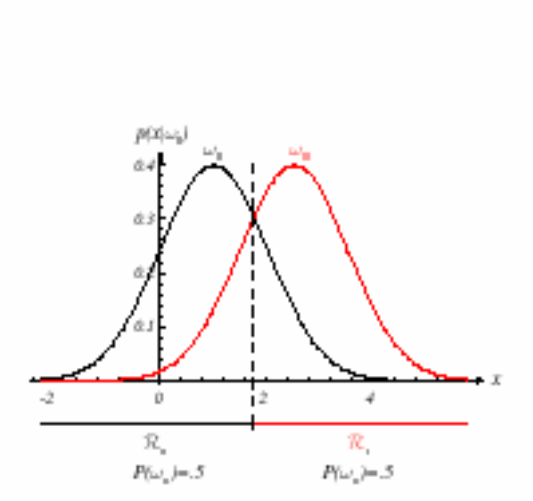
Linear discriminant functions

$$g_i(x) = w_i^T x + w_{i0}$$

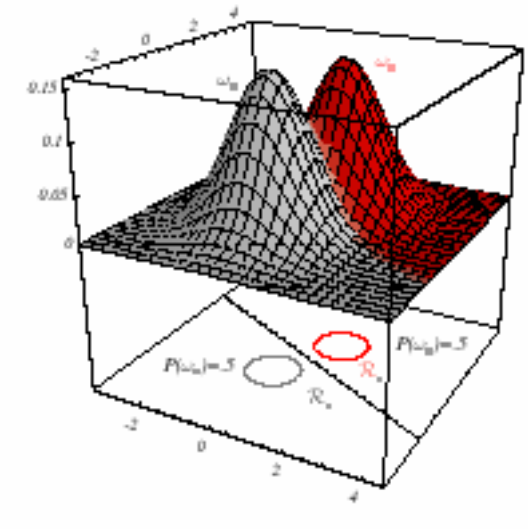
$$w_{i0} = -1/2\ \mu_i^T \mu_i \qquad \text{and} \qquad w_i = \mu_i$$

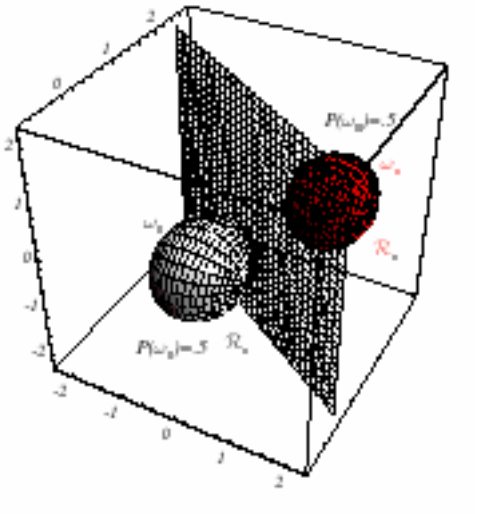
Thus, μ_i is a template for class w_i.

1-D case:



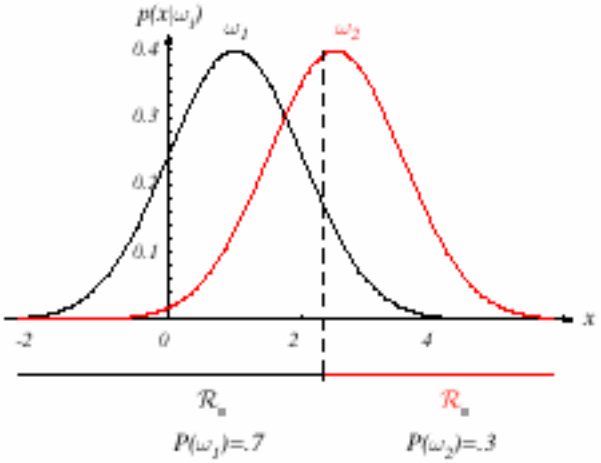
2-D and 3-D cases:

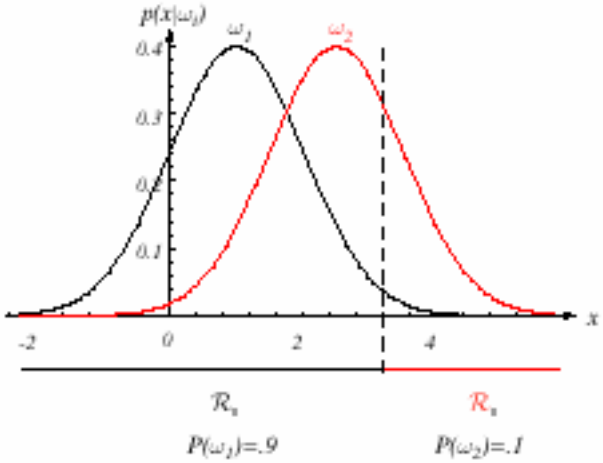


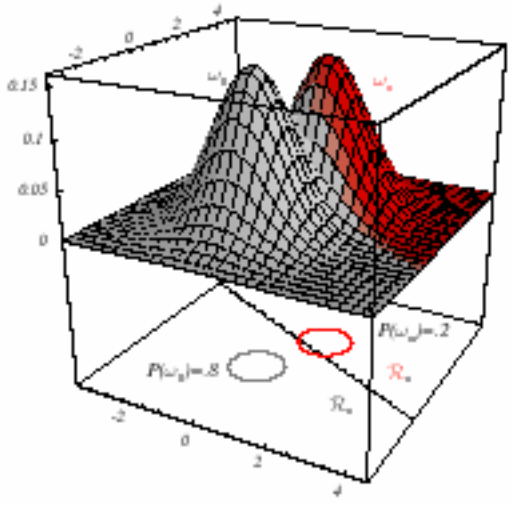


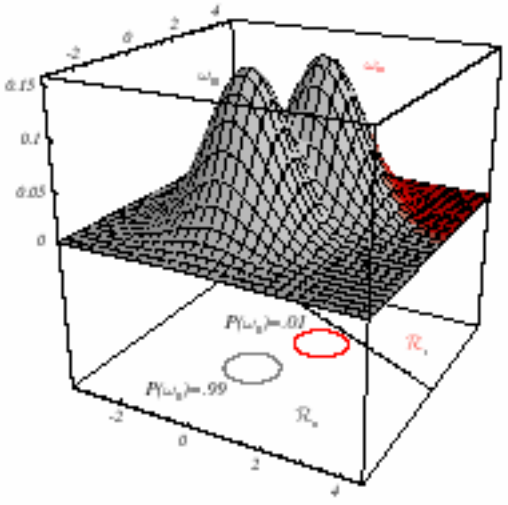
➔ linear discrimination functions.

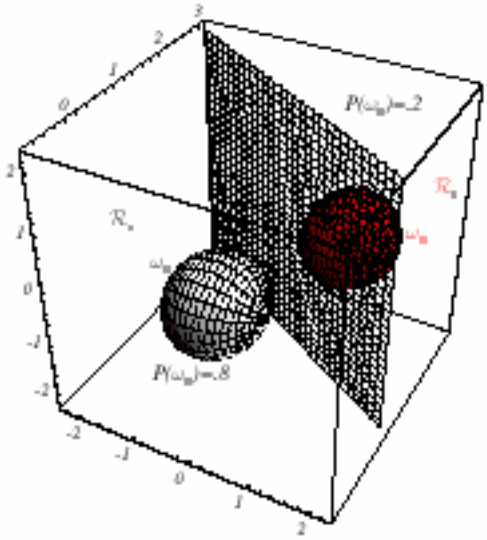
(However, Σ = I is not necessary to achieve linear discriminant functions.)

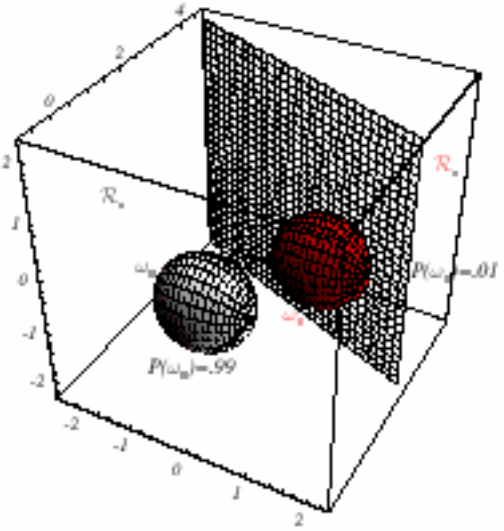

p(x|ω_i)
ω_1
ω_2
0.4
0.3
0.2
0.1
x
-2
0
2
4
R_1
R_2
P(ω_1)=.7
P(ω_2)=.3


p(x|ω_i)
ω_1
ω_2
0.4
0.3
0.2
0.1
x
-2
0
2
4
R_1
R_2
P(ω_1)=.9
P(ω_2)=.1


0.15
0.1
0.05
0
P(ω_2)=.2
P(ω_1)=.8
R_1
R_2


0.15
0.1
0.05
0
P(ω_2)=.01
P(ω_1)=.99
R_1
R_2

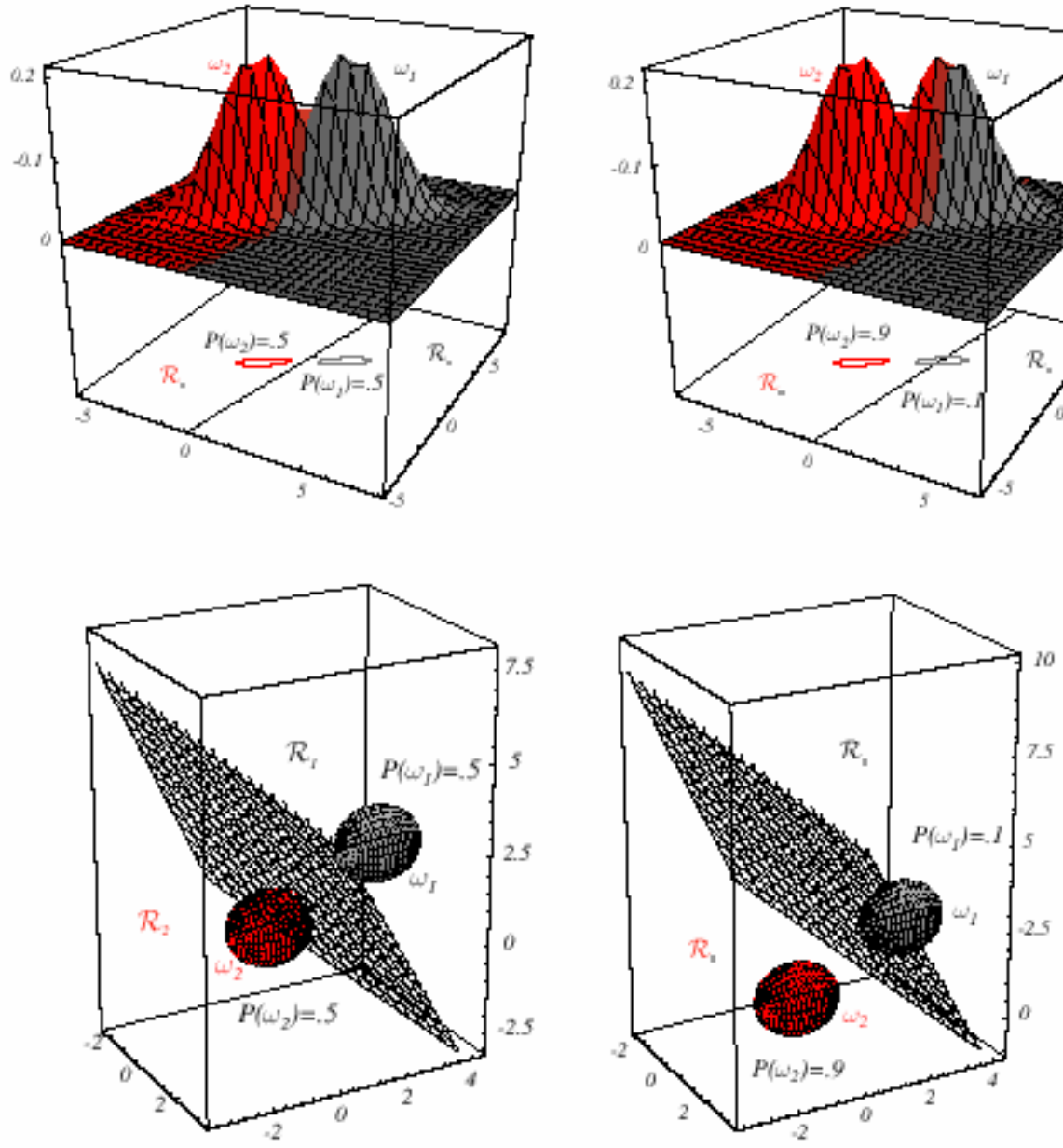

P(ω_2)=.2
P(ω_1)=.8
R_1
R_2


P(ω_2)=.01
P(ω_1)=.99
R_1
R_2

$\Sigma_I = \Sigma$ = covariance matrices identical but arbitrary

$$g_i'(x) = - (x-\mu_i)^T \Sigma^{-1} (x-\mu_i)$$

➔ measure squared
Mahalanobis distance $(x-\mu_i)^T \Sigma^{-1} (x-\mu_i)$
from x to to each of the mean vectors
assign class according to the nearest mean



Generalized result ... Σ_i is class dependent (=arbitrary)

$$g_i(x) = -1/2(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i) - (d/2)\log(2\pi) - (1/2)\log|\Sigma_i|$$

or, using the Bayes formula:

$$g_i(x) = \log[P(w_i|x)p(x)] = \log[p(x|w_i)] + \log[P(w_i)]$$

If components are uncorrelated with equal variance σ^2, i.e.,

$$\Sigma_i = \sigma^2 I$$

and after eliminating the class-independent bias $\log(\Sigma_i)$,

$$g_i(x) = -(1/(2\sigma^2))(x-\mu_i)^T (x-\mu_i) + \log[P(w_i)]$$

In that case, loci of constant $||x-\mu_i||^2$ are hyperspheres, each centered at the class mean μ_i.

Previously, $\Sigma=I$, now assume:

unequal variances

uncorrelated components

The covariance matrix for class i:

$$\Sigma_i = \begin{matrix} \sigma_{11}^2 & 0 & 0 & \dots & 0 \\ 0 & \sigma_{22}^2 & & & 0 \\ 0 & 0 & & \dots & 0 \\ 0 & 0 & & \dots & 0 \\ 0 & 0 & 0 & & \sigma_{dd}^2 \end{matrix}$$

$$\Sigma^{-1}_i = \begin{matrix} 1/\sigma_{11}^2 & 0 & \dots & 0 \\ 0 & 1/\sigma_{22}^2 & & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1/\sigma_{dd}^2 \end{matrix}$$

Thus, the decision rule yields a weighted distance classifier ... more emphasis on features with smaller σ_{ii}^2.

Decision surfaces are hyperellipses (hyperquadrics).

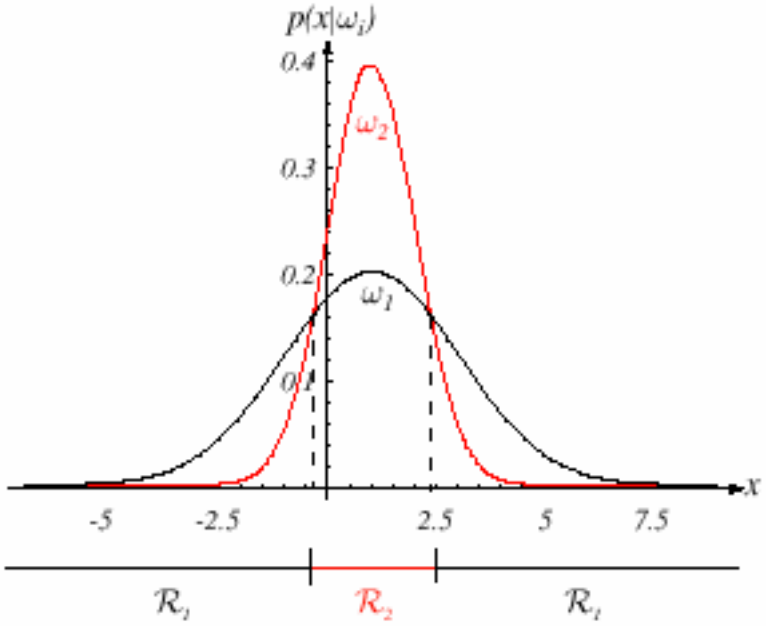


FIGURE 2.13. Non-simply connected decision regions can arise in one dimensions for Gaussians having unequal variance. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

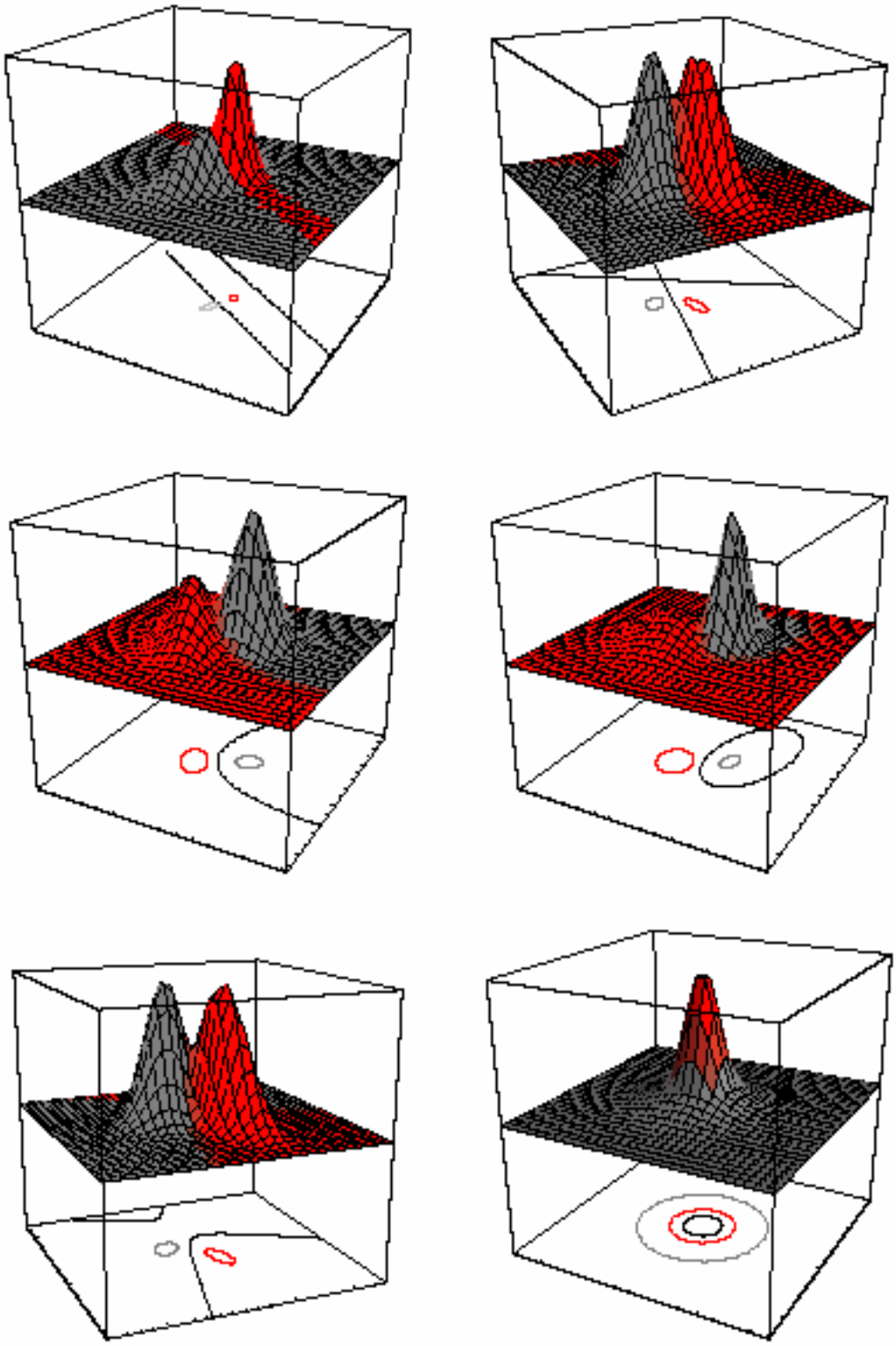


FIGURE 2.14. Arbitrary Gaussian distributions lead to Bayes decision boundaries that are general hyperquadrics. Conversely, given any hyperquadric, one can find two Gaussian distributions whose Bayes decision boundary is that hyperquadric. These variances are indicated by the contours of constant probability density. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

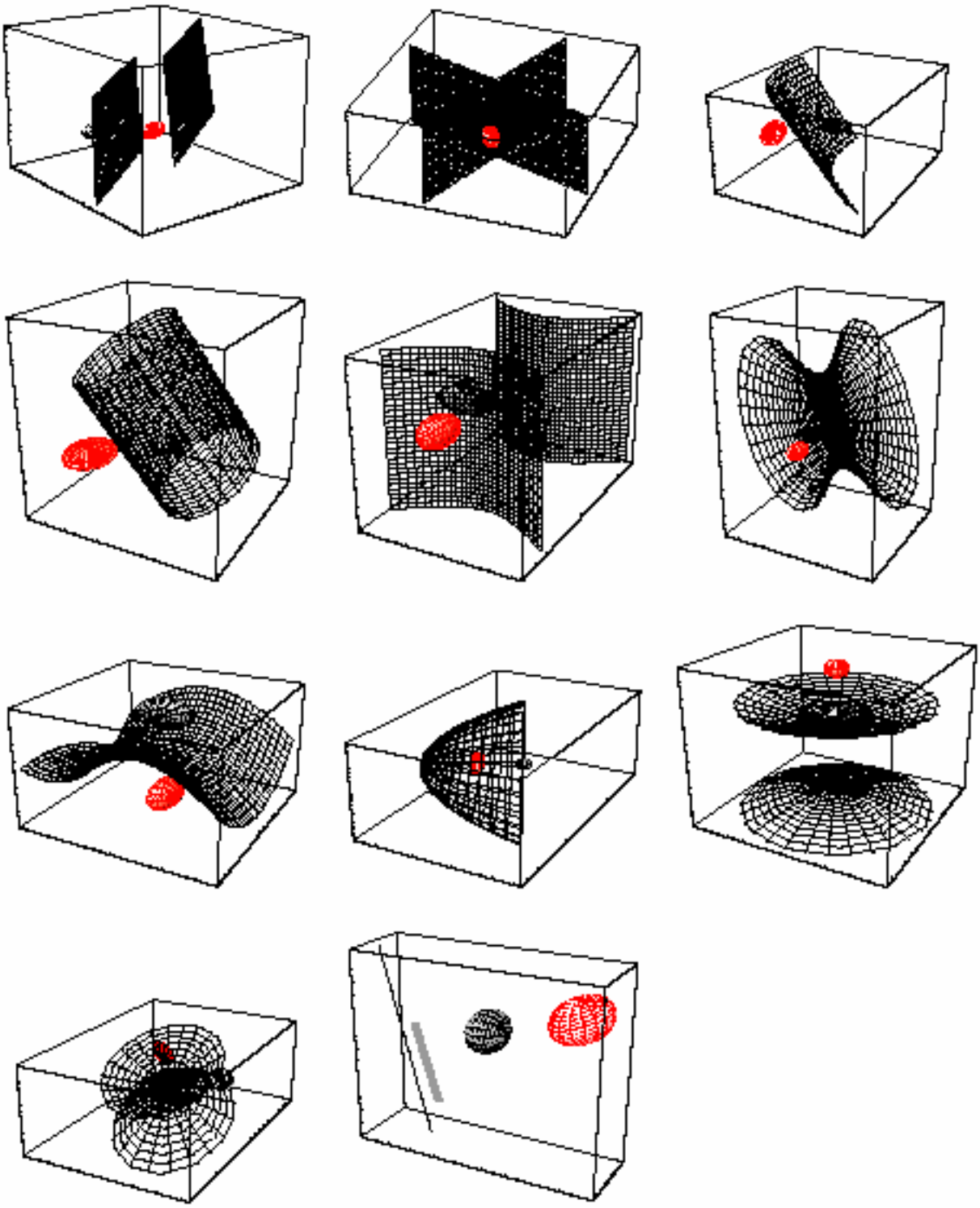


FIGURE 2.15. Arbitrary three-dimensional Gaussian distributions yield Bayes decision boundaries that are two-dimensional hyperquadrics. There are even degenerate cases in which the decision boundary is a line. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Bayes decision theory – discrete features

until now, feature vector x was from R^d

frequently, x can only have one of *m* discrete values

➔ probability density functions is singular and integrals

$$\int p(x|w_j)\, dx$$

must be replaced by summation over all values of x in discrete distribution

$$\Sigma P(x|w_j)$$

Bayes formula then uses probabilities instead of probability densities:

$$P(w_j|\mathbf{x}) = P(\mathbf{x}|w_j)\, P(w_j) / P(\mathbf{x})$$

where

$$P(\mathbf{x}) = \Sigma_{j=1\ldots c}\, P(\mathbf{x}|w_j)\, P(w_j)$$

Definition of conditional risk remains unchanged

➔ so the fundamental Bayes decision rule is still the same.

Extensions:

How to determine parameters of the probability density functions (PDF)?

May be based on training samples from a training set H.
(Coming soon)

Maximum likelihood classification is not the only one.

A) Nearest neighbor classification
B) Decision Trees
C) Unsupervised learning approaches

Chapter 3
Supervised Learning

Maximum likelihood classification

$$P(w_i|x) = p(x|w_i)\, P(w_i)$$

requires knowledge of $p(x|w_i)$, $P(w_i)$, c

always easy for $P(w_i)$, c

frequently difficult for $p(x|w_i)$, especially if dimensionality is high ➔ never enough samples

... If we can assume that the distribution is normal
→ Gaussian densities, we need μ_i, Σ_i for each class
→ much simpler than estimating an unknown function.

This information must be derived from the training set.

Assume that the FORM of densities is known, then we have to ESTIMATE the density parameters.

Parameter Estimation

A) Maximum likelihood estimation
assuming that parameters are fixed but unknown

B) Bayesian estimation
assumes random variables with known a priori distributions

Maximum Likelihood Estimation

Training set in the form of c subsets $H_1, H_2, ..., H_c$

Samples are assumed to be generated by the density function for class i ... probability density $p(x|w_i)$.

Let the set of parameters to be estimated be denoted Θ_i

In the Gaussian case, $\Theta_i = [\mu_i, \Sigma_i]$.

Let's denote the parameter dependency $p(x|w_i,\Theta_i)$.

where Θ_i is unknown but fixed ... not a random vector

Additionally, assume that H_i gives
no information about Θ_j if $i \neq j$
➔ we can work with each class separately.

➔ we have c separate problems as follows (we can remove class dependency from notation, since problems are separated):

A set of *H* training samples drawn independently from the probability density $p(x|\Theta)$ and the goal is to estimate the unknown parameter vector Θ.

General case is described on p. 86-87,
… but special Gaussian case is more frequently used …

Gaussian case

Consider samples drawn from a multivariate normal population with mean μ and covariance matrix Σ.

A) Only μ is unknown

$$\mu = 1/n \, \Sigma_{k=1}^{n} x_k$$

➔ maximum likelihood estimate for the unknown population mean is the arithmetic average of training samples.

In n-D .. μ is centroid of a cloud of samples.

B) μ and Σ are unknown

This is a more usual case than case A.

Univariate case .. estimate μ and σ^2

➔ $\mu = 1/n \sum_{k=1}^{n} x_k$ (same as before)

➔ $\sigma^2 = 1/n \sum_{k=1}^{n} (x_k - \mu)^2$

Multivariate case … estimate μ and Σ

➔ $\mu = 1/n \sum_{k=1}^{n} x_k$ (same as before)

➔ $\Sigma = 1/n \sum_{k=1}^{n} (x_k - \mu)(x_k - \mu)^T$

Bias

The maximum-likelihood estimate of σ^2 is biased
➔ the expected value over all data sets is not equal true variance

Check the above statement for n=1:

$$\Sigma = 1/n \sum_{k=1}^{n} (x_k - \mu)(x_k - \mu)^T = 0$$

while the entire distribution has a non-zero covariance matrix ➔ result cannot be trusted for small values of n

Such estimators are *asymptotically unbiased* … give correct results for large-enough training sets.

Elementary unbiased estimator:

$$\Sigma_{unbiased} = 1/(n-1) \sum_{k=1}^{n} (x_k - \mu)(x_k - \mu)^T$$

If estimator is unbiased for all distributions (like the one right above) ➔ absolutely unbiased estimator

Existence of 2 estimators ➔ neither of them is perfect

Another problem is that if we have an unreliable model for underlying distributions, the classifier based on maximum likelihood will not be optimal ➔ model selection is crucial.

Bayesian Parameter Estimation

Objective: Given H_i, form $P(w_i|x, H_i)$
(obviously, a posteriori probability depends on the training set)

Using the Bayes formula, the BEST density parameter is chosen based on the maximum a posteriori probability of the parameter.

However, reaching the solution in a general case is not straightforward.

In special cases of Normal distribution, the situation is much easier

A) Known Σ, unknown μ

How to calculate μ_i for class i from the training set?

a) Max. likelihood: Use the recursive formula:

$$\mu_i(k+1) = 1/(k+1)\,[k\,\mu_i(k) + x_{k+1}]$$

b) Using the Bayes approach:
assume the a priori covariance matrix for class i:

$$\text{known } \Sigma_i = (1/a)\,\sigma_i$$

Then, the recursive formula

$$\mu_i(k+1) = [(a+k)/(a+k+1)]\mu_i(k) + [1/(a+k+1)]x_{k+1}$$

Parameter a represents the confidence in the a priori estimate of μ. In training, it specifies the number of steps in which we believe more in the a priori estimate than in the measured mean.

For a=0, μ = mean(x)

B) Unknown Σ, known μ

a) Max. likelihood; recursive formula:

$$\Sigma_i(k+1) = 1/(k+1)\,[k\,\Sigma_i(k) + (x_{k+1} - \mu_i)(x_k - \mu_i)^T]$$

b) Bayes approach:

Let K be the number of samples in the training set. Let $\Phi_i(0)$ be the a priori estimate of the covariance matrix. Let $\Sigma_i(K)$ be calculated as in a) above.

Then,

$$\Phi_i(K) = [b\,\Phi_i(0) + K\,\Sigma_i(K)] / [b + K]$$

and $\Phi_i(K)$ is considered the Bayes estimate of $\Sigma_i(K)$.

Parameter b represents the confidence in the a priori estimate of Σ.

C) Unknown Σ, unknown μ (most often the case)

a) Max. likelihood; recursive formula:

$$\mu_i(k+1) = 1/(k+1)\,[k\,\mu_i(k) + x_{k+1}]$$

$$\Sigma_i(k+1) = 1/k\,[(k-1)\,\Sigma_i(k) + \\ + (x_{k+1} - \mu_i(k+1))(x_{k+1} - \mu_i(k+1))^T + \\ + k(\mu_i(k) - \mu_i(k+1))(\mu_i(k) - \mu_i(k+1))^T\,]$$

b) Bayes approach:

Let K be the number of samples in the training set.
Let $\mu_i(0)$ be the a priori estimate of the mean vector for class i.

Let $\Phi_i(0)$ be the a priori estimate of the covariance matrix.

Using $\mu_i(K)$ and $\Sigma_i(K)$ calculated according to the maximum likelihood formulae in A) and B), respectively:

$$M_i(K) = [a\ \mu_i(0) + K\ \mu_i(K)] / [a + K]$$

$$\Phi_i(K) = [1/ (b + K)]$$
$$\{ \quad b\ \Phi_i(0) + a\ \mu_i(0)\ \mu_i(0)^T + (K\text{-}1)\ \Sigma_i(K) +$$
$$+ K\ \mu_i(K)\ \mu_i(K)^T - (a+K)\ M_i(K)\ M_i(K)^T \quad \}$$

$M_i(K)$ is considered the Bayes estimate of $\mu_i(K)$.
$\Phi_i(K)$ is considered the Bayes estimate of $\Sigma_i(K)$.

Parameters a,b represent the confidence in the a priori estimates of μ and Σ.

When do Maximum Likelihood and Bayes methods differ?

- identical for infinite numbers of training samples
- but # of training samples is finite, so which approch is better and when?

Criteria:
- computational complexity
  (Max. likelihood is easier … differential calculus, no multidimensional integration as may be needed by Bayes techniques)
- interpretability
  (max. likelihood gives single best model,
  Bayesian methods give a weighted average of models = more difficult to understand)
- confidence in prior information
  e.g., in form of $p(x|w_i,\Theta_i)$ … Bayesian methods use more such a priori information

➔ if a priori information is reliable,
  Bayesian approach is typically better

➔ Max Likelihood and Bayesian approaches are identical for uniform priors = no specific prior info

Sources of error in final classification system:

1) Bayes (or Indistinguishability) error
   error due to overlapping densities $p(x|w_i)$
   → can never be eliminated

2) Model error
   error caused by having an incorrect model
   ➔ better model is needed but is it known?
   model error in maximum likelihood
   and Bayes methods rarely differ

3) Estimation error
   consequence of having a finite sample
   → can be decreased by increasing the training sample

There are theoretical and methodological arguments supporting Bayesian methods
- but in reality, maximum likelihood estimation is simpler and leads to classifiers nearly as accurate as the more-difficult-to-design Bayes classifiers.

Problems of Dimensionality

- classification problems with tens or hundreds of features are common

- let's assume that no features are intentionally redundant

→ how classification accuracy depends on dimensionality and amount of training data

→ what is computational complexity of classifier design

A) Accuracy, dimension, training sample size

- the most useful are features for which the difference between means is large relative to the standard deviations
➔ Mahalanobis distance *r* is used, the larger the distance, the better classification

$$r^2 = (\mu_1 - \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2)$$

so for conditionally independent case when

$$\Sigma = \text{diag} (\sigma_1^2, \sigma_2^2, \ldots , \sigma_d^2)$$

… squared Mahalanobis distance is

$$r^2 = \Sigma_{i=1}^{d} [(\mu_{i1} - \mu_{i2}) / \sigma_i]^2$$

➔ each feature contributes to an increase of r^2 … improve classification

- but it also says that no feature is useless as long as its means differ … BUT …

… BUT …

beyond certain point, adding new features frequently causes decrease of performance, not improvement.

B) Computational Complexity

$$g_i(x) = -1/2(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i) - (d/2)\log(2\pi) - (1/2)\log|\Sigma_i|$$

^ O(dn)

^ **O(nd^2)**

^ O(1)

^ **O(d^2n)**

➔ this is done for each class

→ Overall complexity for Bayes <u>learning</u> O(cd^2n)

➔ earlier we saw that we need large training samples
→ obvious "cost" associated with large *n*

→ Classification complexity is much lower
[computing $(x-\mu_i)$] → O(d) complexity= linear

plus

multiplication of inverse covariance matrix by separation vector … $O(d^2)$

plus

do it for all c classes … for small "c" it is still **$O(d^2)$**

→ MUCH less costly than learning

C) Overfitting

If a number of available samples is not sufficient, the number of features may be too large to be supported by the small sample set ➔ overfitting occurs, classifier does not generalize, rather it memorizes the training set.

Solutions:
- reduce dimensionality of feature set
    - remove features
    - combine features (K-L transform)
- assume that all c classes have the same covariance matrix and pool available data
- improve estimate of Σ

Problems of Parametric Approaches:

- the form of the distribution is not known
- the form of the distribution does not fit known estimation approaches

Nonparametric Approaches

1) Direct estimation of $p(x|w_i)$
   based on a generalized multidimensional histogram

2) Direct estimation of $P(w_j|x)$

3) Transform of the Feature Space
   hope that learning will be easier in the transformed space

Nonparametric Density Estimation

divide the feature space in regions R

vector x falling in region R ...

$$P(x \in R) = \int_R p(x')dx' \cong P$$

Over R, P represents smoothed measure of density p(x). Assuming that p(x) is constant over R, estimating P results in the estimate of p(x).

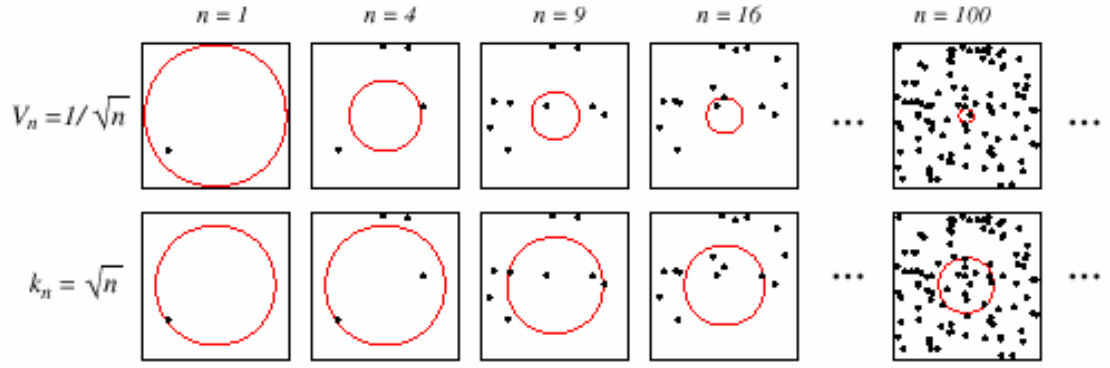


FIGURE 4.2. There are two leading methods for estimating the density at a point, here at the center of each square. The one shown in the top row is to start with a large volume centered on the test point and shrink it according to a function such as $V_n = 1/\sqrt{n}$. The other method, shown in the bottom row, is to decrease the volume in a data-dependent way, for instance letting the volume enclose some number $k_n = \sqrt{n}$ of sample points. The sequences in both cases represent random variables that generally converge and allow the true density at the test point to be calculated. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Assume n independently drawn samples of PDF p(x) that characterize class w_i - samples are available in the training set H_i.

Probability that k out of n of these samples fall into region R is given by binomial distribution … for large "n", the binomials peak strongly at true probability.

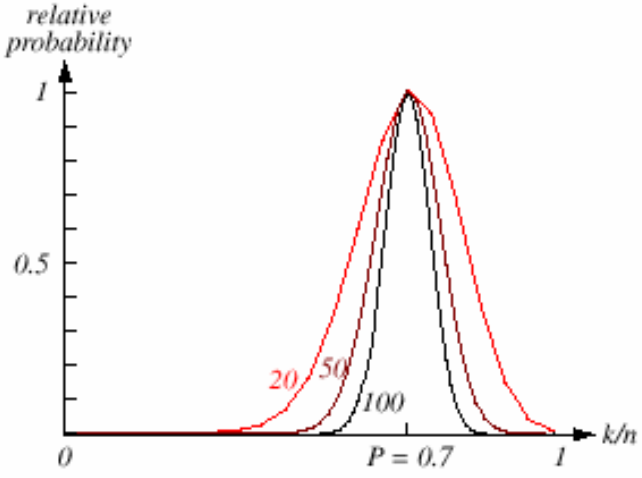


FIGURE 4.1. The relative probability an estimate given by Eq. 4 will yield a particular value for the probability density, here where the true probability was chosen to be 0.7. Each curve is labeled by the total number of patterns *n* sampled, and is scaled to give the same maximum (at the true probability). The form of each curve is binomial, as given by Eq. 2. For large *n*, such binomials peak strongly at the true probability. In the limit $n \to \infty$, the curve approaches a delta function, and we are guaranteed that our estimate will give the true probability. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Example:

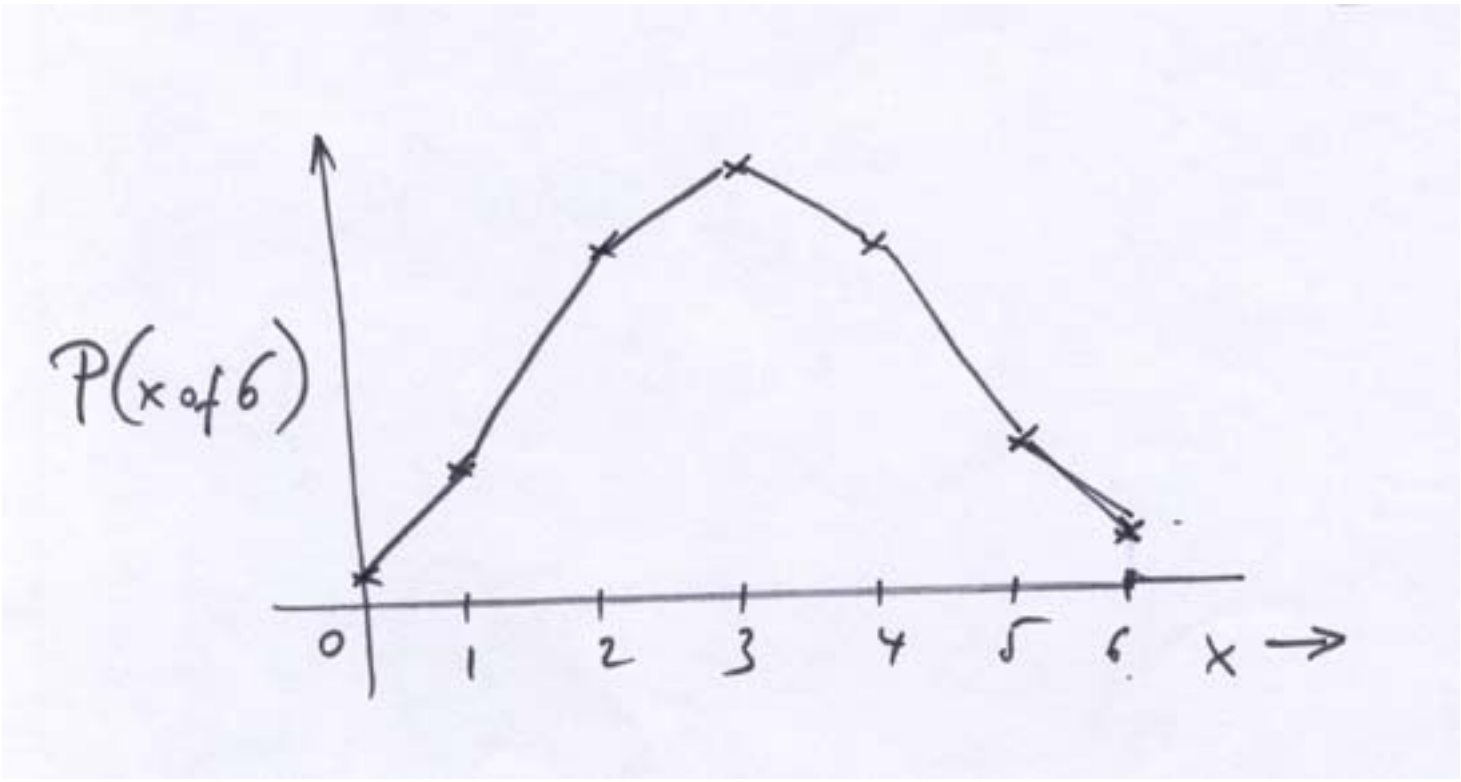
6 coins - how many heads ...

P(0 of 6) = 1/64
P(1 of 6) = 6/64
P(2 of 6) = 15/64
P(3 of 6) = 20/64
P(4 of 6) = 15/64
P(5 of 6) = 6/64
P(6 of 6) = 1/64

Obviously, assuming P=1/2 …

$\mu = 6\ P = 3$

$\sigma^2 = nP\ (1\text{-}P) = 6\ (1/2)\ (1 - 1/2) = 3/2$



P(k of n vectors $\in$ R) is large for k $\cong$ nP. It is small otherwise. Let's not consider the small numbers ...

Therefore, it is likely that the observed number of vectors falling into region R is only the result of the mean - therefore, $k_{observed} = nP$.

Thus, estimate for P:

$$P = k_{observed}/n$$

Therefore, for class w_i, using training set H_i, if n samples fall in region R with volume V_n,
we estimate the PDF as

$$p_n(x) = k_n / (n V_n)$$

We are interested in convergence of p(x) as $n \rightarrow \infty$.

Volume V of the region R cannot be too small since then the estimates p(x) would vary from region to region ... we need the smoothing effect.

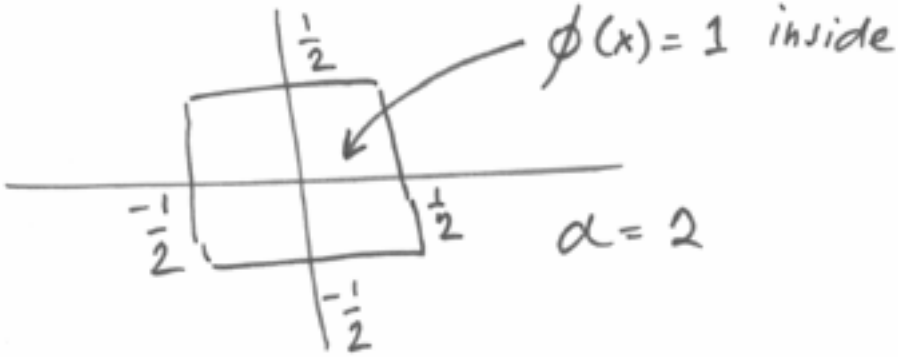
What is the optimal size? 2 strategies:
1) shrinking regions ... Parzen windows
2) growing regions ... Nearest neighbor method

Parzen Windows

Let the regions R_n be d-dimensional hypercubes with edge-dimension h_n. Let $V_n = (h_n)^d$ be centered at x.

Using d-dimensional step function, the d-dimensional feature space is divided into the hypercube and the rest of space thus facilitating to count the samples in regions R.

$$\phi(x) = \begin{cases} 1 & |x_i| \le \frac{1}{2} \quad i = 1, 2, \dots d \\ 0 & \text{otherwise} \end{cases}$$



$$\phi_2(x) = \phi\left(\frac{x - x_i}{h_n}\right)$$

translated and scaled

↳ origin x_i ↳ side h_n

Training set with n samples ... the number of samples in the hypercube region centered at x is
(for "0" outside of the cube and "1" inside of the cube)

$$k_n = \sum_{i=1}^{n} \phi\left(\frac{x - x_i}{h_n}\right)$$

$$p_n(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{\phi[(x - x_i)/h_n]}{V_n}$$

If we view ϕ as an interpolation function for $p_n(x)$ and define

$$\delta_n(x) = \frac{1}{V_n} \phi\left(\frac{x}{h_n}\right)$$

$p_n(x)$ can be calculated as

$$p_n(x) = \frac{1}{n} \sum_{i=1}^{n} \delta_n(x - x_i)$$

If h_n is large ... heavy smoothing, insensitive to local variations in x

If h_n is small ... sharp peaks of $\delta_n(x)$ at x=0, if the training set is not infinite, we have not covered entire R^d ... erroneous estimates of $p_n(x)$ ... erroneous results.

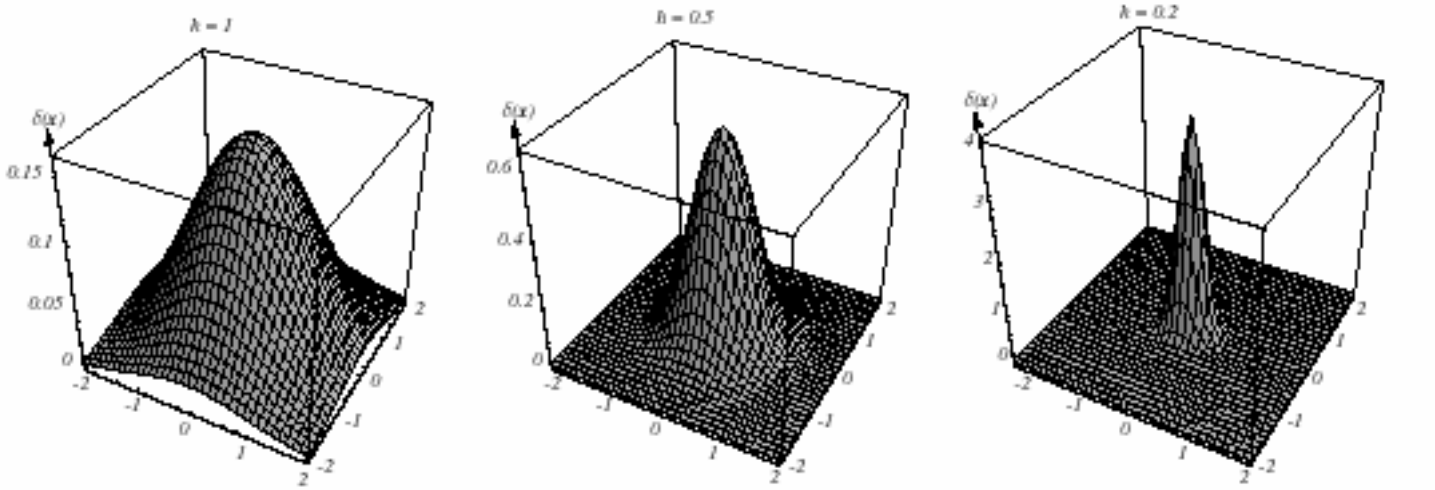


FIGURE 4.3. Examples of two-dimensional circularly symmetric normal Parzen windows for three different values of *h*. Note that because the $\delta(\mathbf{x})$ are normalized, different vertical scales must be used to show their structure. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

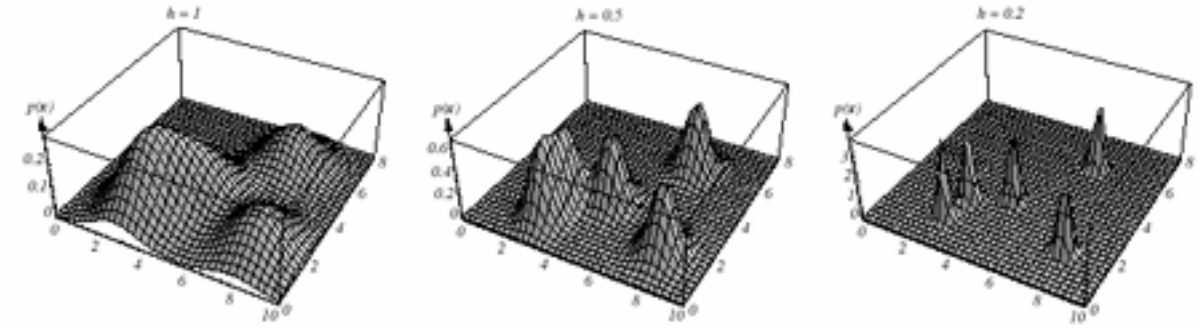
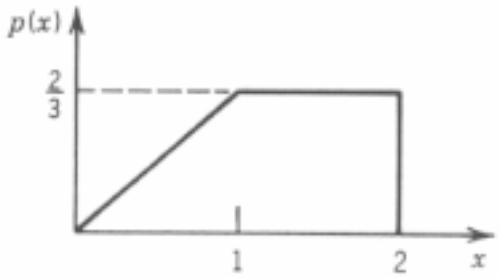


FIGURE 4.4. Three Parzen-window density estimates based on the same set of five samples, using the window functions in Fig. 4.3. As before, the vertical axes have been scaled to show the structure of each distribution. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.



(a)

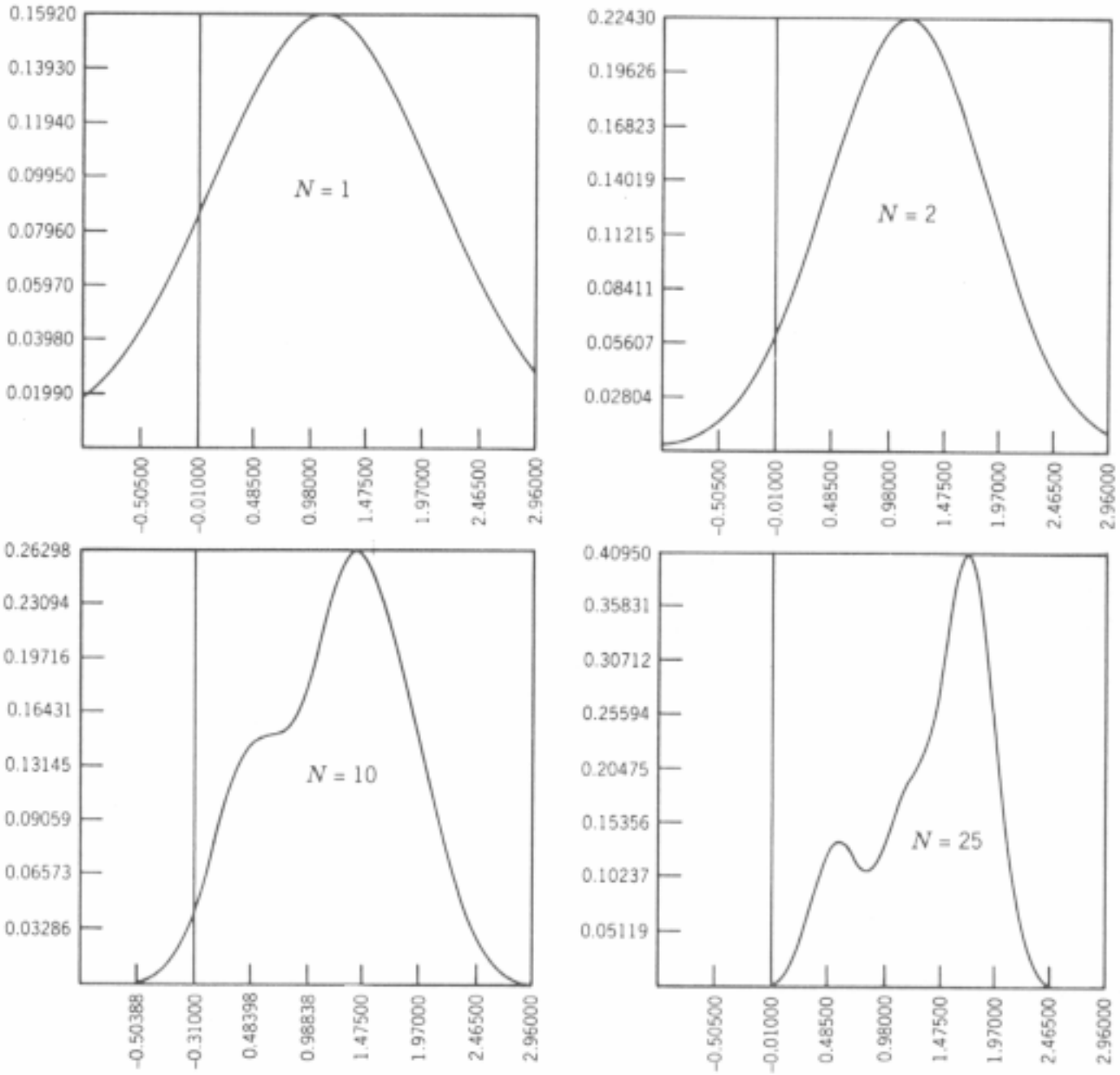
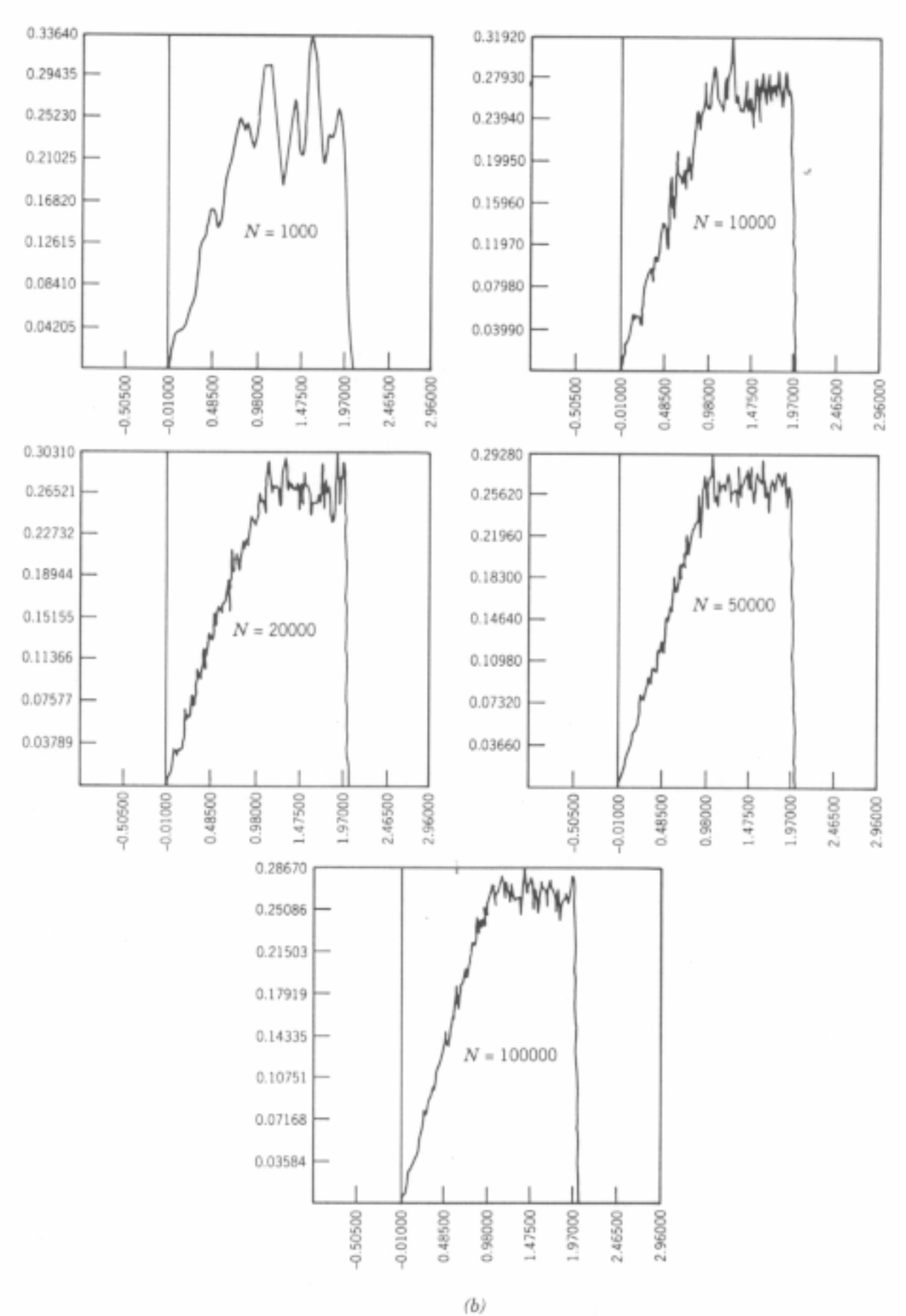


Figure 7: Nonparametric estimation of 1-D density. $\left(\text{y-axis scaled by } \frac{1}{\sqrt{2\pi}}\right)$



(b)

Figure 7 (cont.): (b) Estimates for various values of N

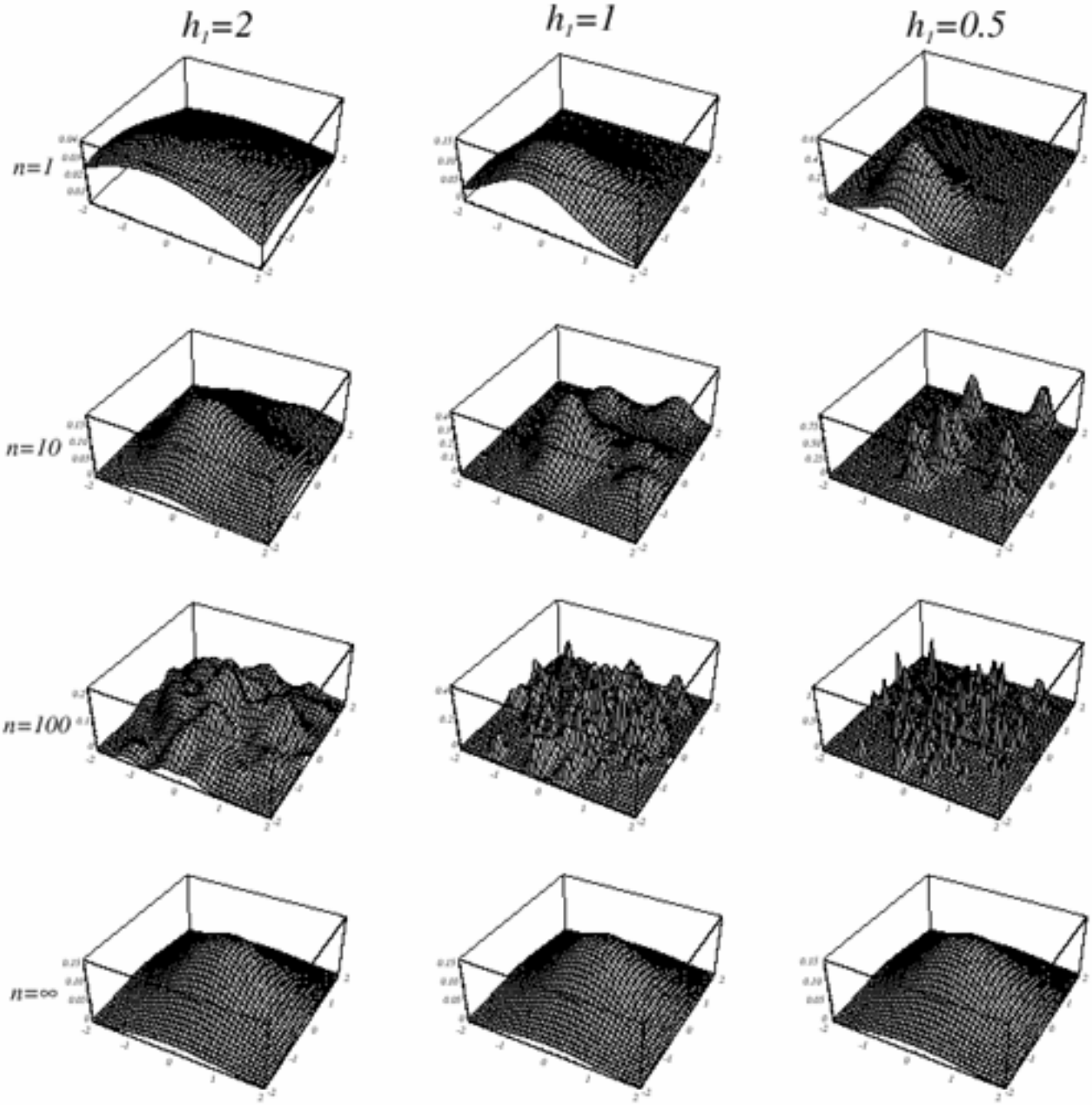


FIGURE 4.6. Parzen-window estimates of a bivariate normal density using different window widths and numbers of samples. The vertical axes have been scaled to best show the structure in each graph. Note particularly that the $n = \infty$ estimates are the same (and match the true distribution), regardless of window width. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley

➔ many samples are required to get a good estimate

How to shrink the regions?

E.g., using *iteration-driven volume determination*, select starting volume V_0, shrink with increased number of samples in the training set

$$V_n = V_0 / \text{sqrt}(n)$$

However, again, everything depends on the selection of V_0. Additionally, if data do not cover the feature space equally, many regions will not contain samples and the estimate of p(x) will be erroneous.

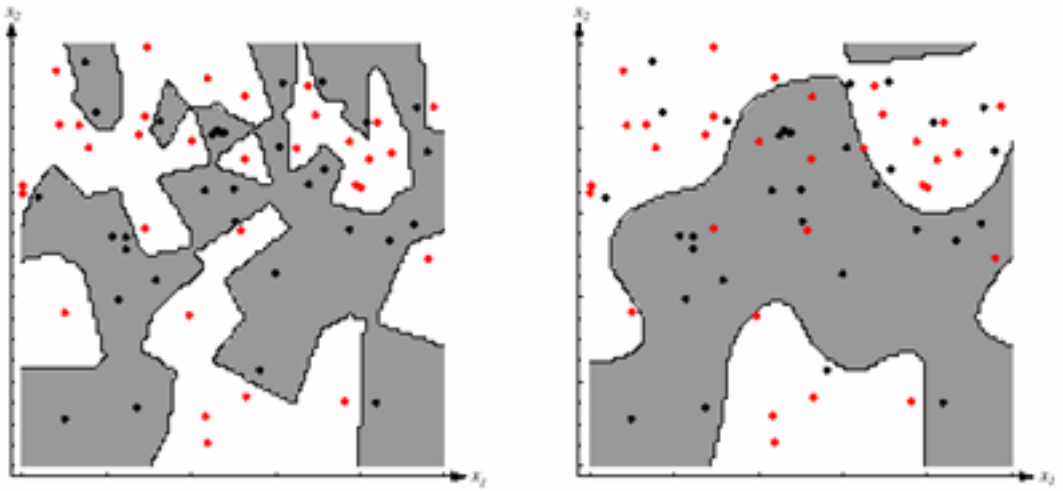


FIGURE 4.8. The decision boundaries in a two-dimensional Parzen-window dichotomizer depend on the window width *h*. At the left a small *h* leads to boundaries that are more complicated than for large *h* on same data set, shown at the right. Apparently, for these data a small *h* would be appropriate for the upper region, while a large *h* would be appropriate for the lower region; no single window width is ideal overall. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

➔ no single size is optimal …

Maybe, the region size should be a function of the sample data:

k-NN Nonparametric Estimation

1) form some number of regions, each centered around a location $x \in R^d$.

Then, increase region's size until it contains k_n nearest neighbor samples, k_n is a function of n.

If the density around x is high - the region will be small and vice versa.

Important ... we are using single-class training sets H_i

Direct estimation of $P(w_i|x)$

suppose we captured k_i samples from class w_i in region R with volume V … joint probability $p_n(x,w_i)$

$$p_n(x, w_i) = \frac{k_i / n}{V}$$

Bayes rule:

$$P_n(w_i|x)\, p_n(x) = p_n(x, w_i)$$

or

$$P_n(w_i|x) = \frac{p_n(x, w_i)}{\sum_{j=1}^{c} p_n(x, w_j)} = \frac{\frac{k_i/n}{V}}{\sum_{j=1}^{c} \frac{k_j/n}{V}} =$$

$$= \frac{k_i}{k}$$

k … total number of samples in all classes in volume V

NNR Nearest neighbor rule

keep in memory all samples from the training set

1-NNR

in the classification stage, assign the class of the nearest labeled sample from the training set

Efficient algorithms to find the NN - preordering of samples

3-NNR (k-NNR)

examine labels of k nearest neighbor samples, decision made based on higher number of samples from a particular class

- odd k helps if we have 2 classes - avoids ties.

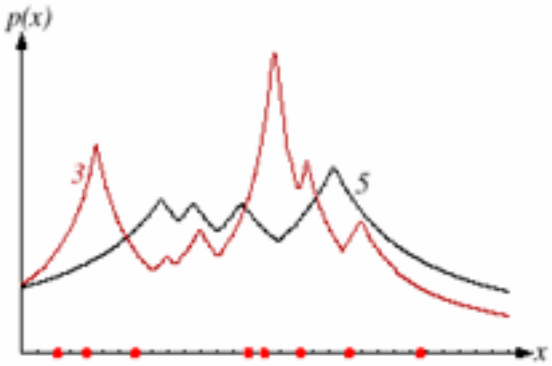


FIGURE 4.10. Eight points in one dimension and the k-nearest-neighbor density estimates, for $k = 3$ and 5. Note especially that the discontinuities in the slopes in the estimates generally lie *away* from the positions of the prototype points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

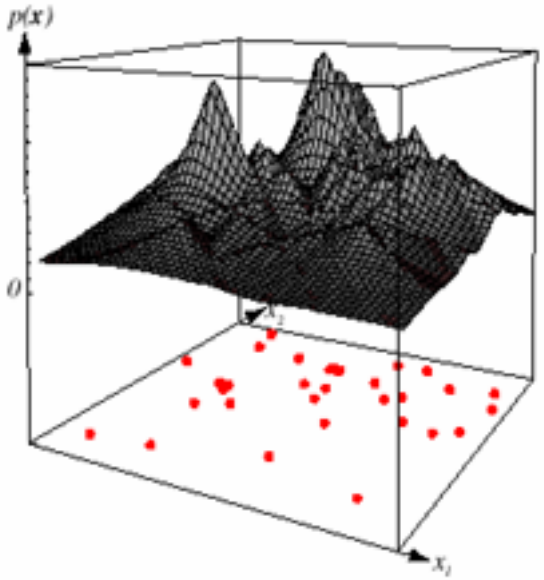


FIGURE 4.11. The k-nearest-neighbor estimate of a two-dimensional density for $k = 5$. Notice how such a finite n estimate can be quite "jagged," and notice that discontinuities in the slopes generally occur along lines away from the positions of the points themselves. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

➔ note discontinuities far from samples

Voronoi tesselation of the d-dimensional space …

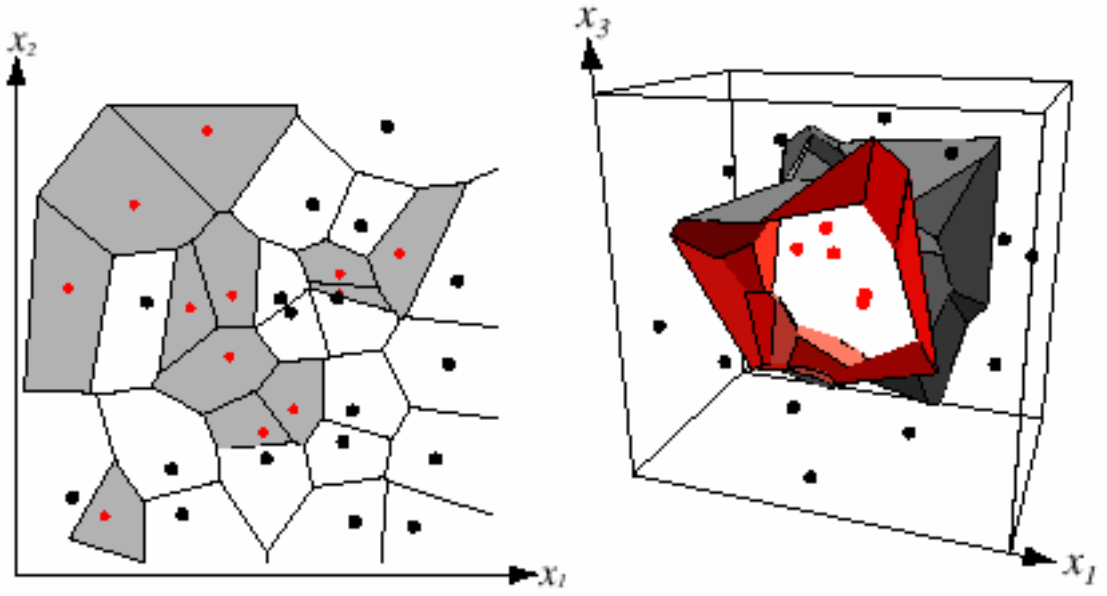


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

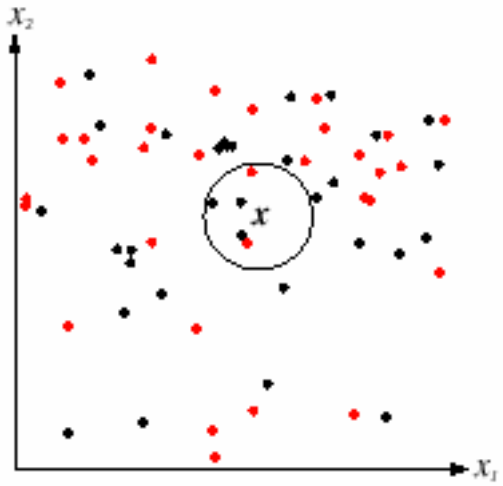


FIGURE 4.15. The k-nearest-neighbor query starts at the test point **x** and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point **x** would be labeled the category of the black points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Metrics and Nearest Neighbor Classification

NN-classification relies on a distance function – metric

Properties of a metric D(.,.):

- nonnegativity
- reflexivity
- symmetry
- triangle inequality

a,b,c … 3 vectors

1) $D(a,b) >= 0$
2) $D(a,b) = 0 \quad iff \quad a=b$
3) $D(a,b) = D(b,a)$
4) $D(a,b) + D(b,c) > D(a,c)$

Example:
Euclidean distance in d-dimensions is a metric

The usual Euclidean metric is not invariant to scaling, especially to non-uniform scaling for each coordinate.

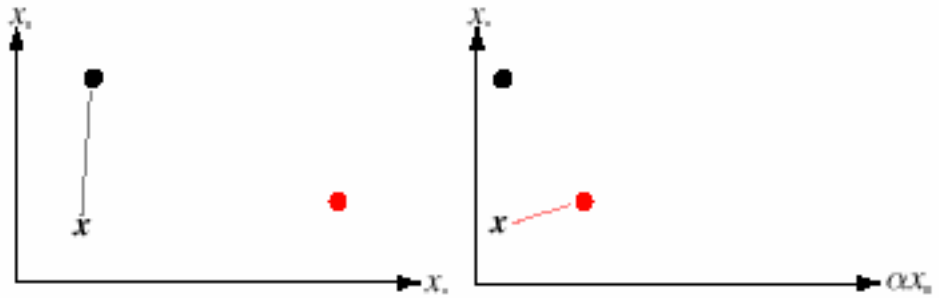


FIGURE 4.18. Scaling the coordinates of a feature space can change the distance relationships computed by the Euclidean metric. Here we see how such scaling can change the behavior of a nearest-neighbor classifer. Consider the test point **x** and its nearest neighbor. In the original space (left), the black prototype is closest. In the figure at the right, the x_1 axis has been rescaled by a factor 1/3; now the nearest prototype is the red one. If there is a large disparity in the ranges of the full data in each dimension, a common procedure is to rescale all the data to equalize such ranges, and this is equivalent to changing the metric in the original space. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Minkowski metric (also L_k norm):

$$L_k(a,b) = \left(\sum_{i=1}^{d} |a_i - b_i|^k \right)^{1/k}$$

L_1 norm = city block distance
L_2 norm = Euclidean distance
L_∞ norm … maximum of distances between projections of a,b vectors onto each of d coordinate axes

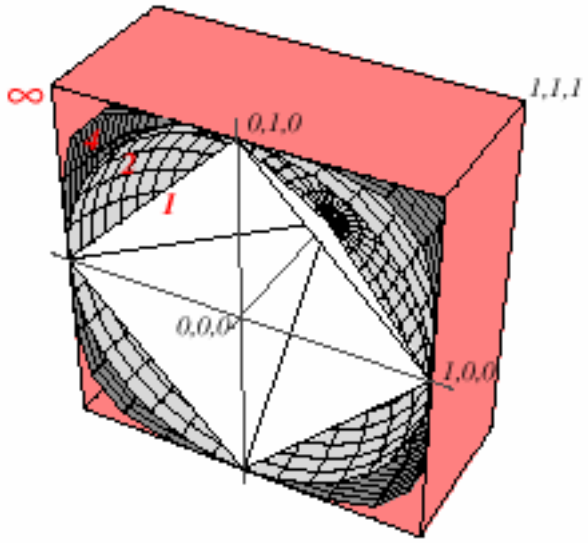


FIGURE 4.19. Each colored surface consists of points a distance 1.0 from the origin, measured using different values for k in the Minkowski metric (k is printed in red). Thus the white surfaces correspond to the L_1 norm (Manhattan distance), the light gray sphere corresponds to the L_2 norm (Euclidean distance), the dark gray ones correspond to the L_4 norm, and the pink box corresponds to the L_∞ norm. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley &

Tanimoto metric … distance between two sets:

$$D_T(S_1, S_2) = (n_1 + n_2 - 2\,n_{12}) / (n_1 + n_2 - n_{12})$$

n_1 … number of elements in set S_1
n_2 … number of elements in set S_2
n_{12} … number of elements in $S_1 \cap S_2$

… Problems in which two patterns are either the same or different with no natural notion of graded similarity.

Tangent Distance

- typical problems of metrics used in Nearest Neighbor classifiers is the lack of invariance

Example below … shift of "5" causes 2 "fives" be more dissimilar then non-shifted "5" and "8" when Euclidean distance of gray values in corresponding pixels is used.

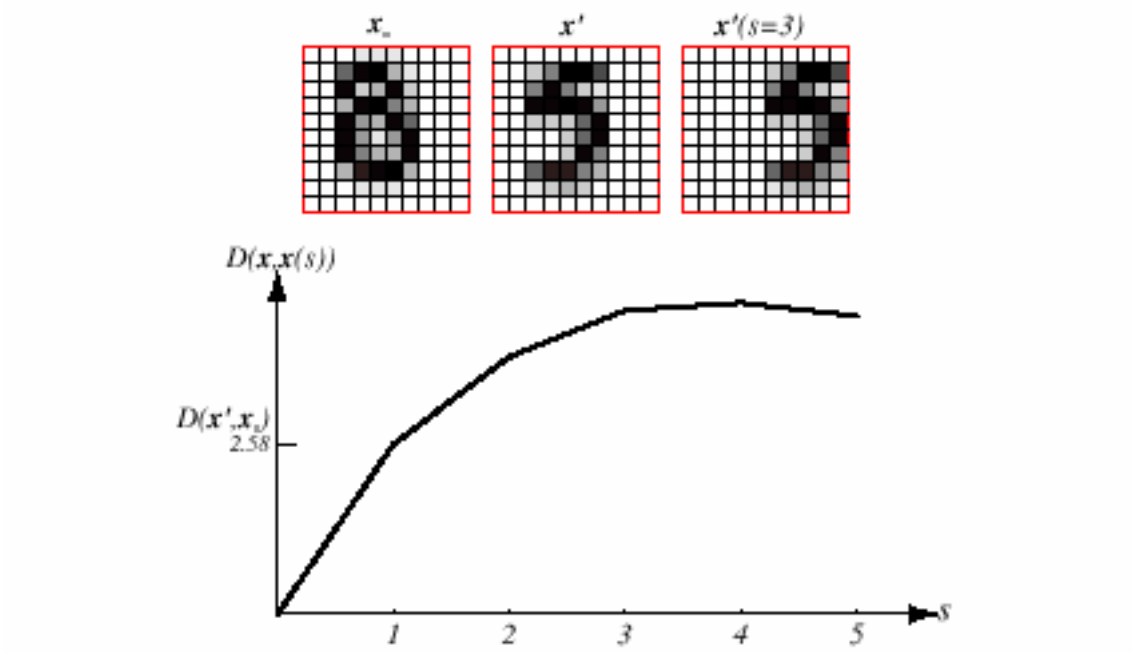


FIGURE 4.20. The uncritical use of Euclidean metric cannot address the problem of translation invariance. Pattern $\mathbf{x}'$ represents a handwritten 5, and $\mathbf{x}'(s = 3)$ represents the same shape but shifted three pixels to the right. The Euclidean distance $D(\mathbf{x}', \mathbf{x}'(s = 3))$ is much larger than $D(\mathbf{x}', \mathbf{x}_8)$, where $\mathbf{x}_8$ represents the handwritten 8. Nearest-neighbor classification based on the Euclidean distance in this way leads to very large errors. Instead, we seek a distance measure that would be insensitive to such translations, or indeed other known invariances, such as scale or rotation. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley &

Similarly, Euclidean distance is not invariant to rotation, scaling

Since transforming the patterns first is computationally prohibitive, tangent distance is often used.

Let there be *r* transforms applicable to the problem, e.g., translation, rotation, shear, scale, line thinning, etc.

==> perform linearly independent combination of transforms ==> construct new prototypes.

Transforms are expensive, but transformation only needs to be done once – during training.

Thus, a matrix of Tangent Vectors is created for each training samples according to all possible transforms

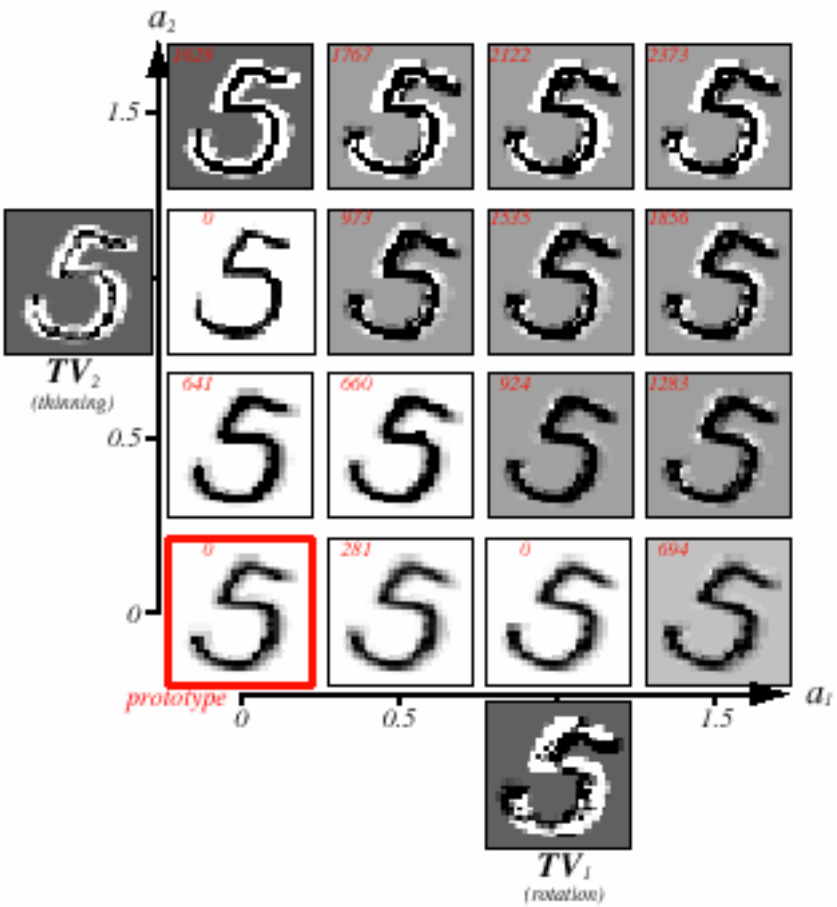


FIGURE 4.21. The pixel image of the handwritten 5 prototype at the lower left was subjected to two transformations, rotation, and line thinning, to obtain the tangent vectors $\mathbf{TV}_1$ and $\mathbf{TV}_2$; images corresponding to these tangent vectors are shown outside the axes. Each of the 16 images within the axes represents the prototype plus linear combination of the two tangent vectors with coefficients a_1 and a_2. The small red number in each image is the Euclidean distance between the tangent approximation and the image generated by the unapproximated transformations. Of course, this Euclidean distance is 0 for the prototype and for the cases $a_1 = 1, a_2 = 0$ and $a_1 = 0, a_2 = 1$. (The patterns generated with $a_1 + a_2 > 1$ have a gray background because of automatic grayscale conversion of images with negative pixel values.) From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Tangent distance is (one-sided tangent distance)

$$D_{tan}(x',x) = \min_a [\| (x' + Ta) - x \|]$$

➔ optimization of "**a**"

minimal Euclidean distance from x to tangent space of x'.

= we search for point in the tangent space that is closest to a test point x = the linear approximation of the ideal.

Not too difficult since the distance we minimize is a quadratic function (shown in red below).

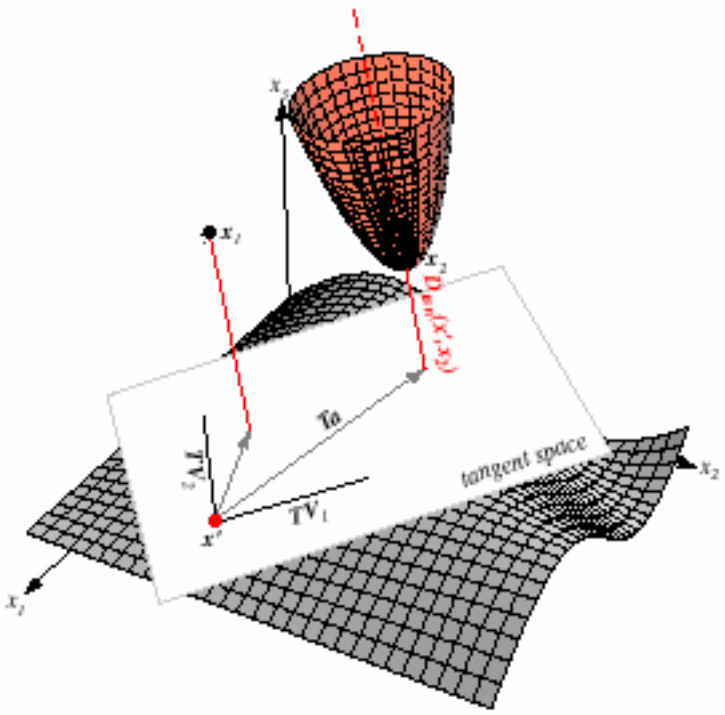


FIGURE 4.22. A stored prototype $\mathbf{x}'$, if transformed by combinations of two basic transformations, would fall somewhere on a complicated curved surface in the full d-dimensional space (gray). The tangent space at $\mathbf{x}'$ is an r-dimensional Euclidean space, spanned by the tangent vectors (here $\mathbf{TV}_1$ and $\mathbf{TV}_2$). The tangent distance $D_{tan}(\mathbf{x}', \mathbf{x})$ is the smallest Euclidean distance from $\mathbf{x}$ to the tangent space of $\mathbf{x}'$, shown in the solid red lines for two points, $\mathbf{x}_1$ and $\mathbf{x}_2$. Thus although the Euclidean distance from $\mathbf{x}'$ to $\mathbf{x}_1$ is less than that to $\mathbf{x}_2$, for the tangent distance the situation is reversed. The Euclidean distance from $\mathbf{x}_2$ to the tangent space of $\mathbf{x}'$ is a quadratic function of the parameter vector $\mathbf{a}$, as shown by the pink paraboloid. Thus simple gradient descent methods can find the optimal vector $\mathbf{a}$ and hence the tangent distance $D_{tan}(\mathbf{x}', \mathbf{x}_2)$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Feature Extraction and Feature Selection

Decision making success is closely related to the amount of information available.

Thus, feature vectors should contain as complete a description as possible ... this would increase the number of features

No features are for free - feature calculation costs, classification costs,

In reality, we have to compromise between the error caused by incomplete feature description and complexity of the description/classification stages.

Thus, selection or extraction of features is of high importance ... use a minimal number of informative features

How to identify informative features?

This part is not formalized

No theory exists specifying which features should be measured on objects to improve classification success rate

Compared to the well formalized and optimal approaches for classifier setup, the situation is quite interesting:

We can find optimal solution for the classifier design having some amount of descriptive information about objects

however ...

We are not able to guarantee that the available amount of descriptive information about objects will be sufficient for satisfactory classification performance

Feature Extraction

Let's have N descriptive features (N-dimensional pattern space)

Then, a feature extraction is represented by a linear or non-linear transform that maps any N-dimensional vector x into an M-dimensional vector y, M<N.

Feature Selection

Special case of Feature Extraction where the M components of y are a subset of the N components of x. Interestingly, feature extraction is usually easier than feature selection.

However, extraction often does not really solve the problem, if we transform the original feature space, we still need all the original features to calculate the new vectors y in the transformed space ... we can save on classifier complexity, cannot save on feature calculation.

Karhunen-Loève Expansion

New vectors y minimize a mean-square error criterion with respect to original pattern vectors x.

K patterns from class *A* characterized by *m* features:

$x_k = [\, x_1, x_2, \ldots, x_m]^T$

choose n orthonormal vectors e_i

$i=1, \ldots, n$

$n <= m$

$$\tilde{x}_k = \Sigma_{i=1}^{n} c_{ki}\ e_i \qquad (*)$$

PDF FILE WITH K-L DERIVATION EQUATIONS TO BE INSERTED HERE

Properties of K-L Transform

+ properties:

1. For n-dimensional space, it offers minimal distance error
2. Feature vectors after K-L transform are not correlated
3. Features are decreasingly important with increasing eigen number, unimportant features can be dropped
4. If we decide to improve approximation accuracy (decrease distance error), it is not necessary to recompute everything, just add some more features.

- properties

1. K-L expansion determines best descriptive features, not best discriminative features
2. Thus, large eigen numbers may not represent the best features w.r.t. classification
3. Despite the fact that we select a smaller number of final features, we still have to calculate them all.

Feature Extraction using Ratio of Scatter

Transform the feature space so that the ratio

(Inter-class scatter / Intra-class scatter)

is maximal.

This ratio is inversely proportional to probability of incorrect classification.

Individual features do not contribute equally to the value of the ratio

Thus, we can disregard features that do not contribute substantially to the value of the ratio.

Inter-class scatter is defined as

PDF FILE WITH RATIO OF SCATTER DERIVATION EQUATIONS TO BE INSERTED HERE

Linear Discriminant Functions

so far… we assumed that forms of underlying distributions were known
→ we used the samples (training sets) to estimate parameters of distributions.

Now - let's assume that we know the forms of discriminant functions g(x)
→ we will use the samples (training sets) to estimate parameters of g(x).

Training error:
average error incurred in classifying training set samples

Caution!
Low training error does NOT guarantee good classifier performance on test sets

Linear discrimination functions and surfaces

$$g(x) = w^T x + w_0$$

w ... weight vector
w_0 … bias, threshold weight

Two-category case:

ω_1 … assign if $g(x) > 0$
ω_2 … assign if $g(x) < 0$

no decision if $g(x) = 0$

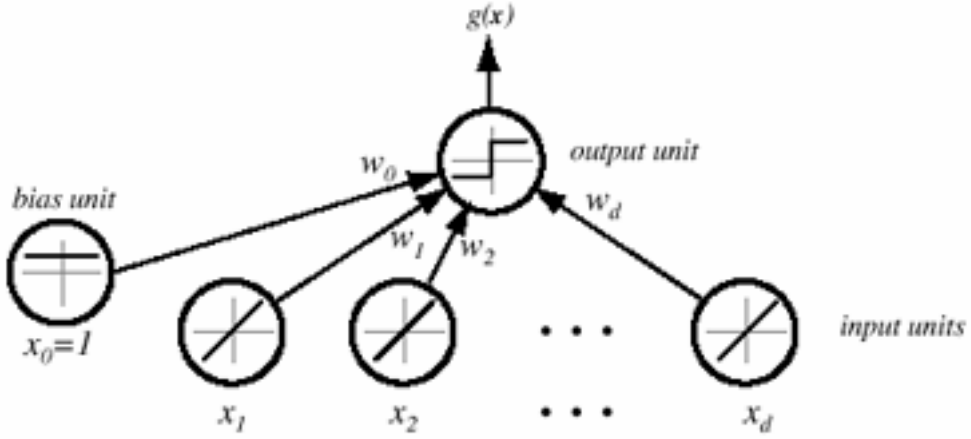


FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show in each unit its effective input-output function. Thus each of the d input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit unit always emits the constant value 1.0. The single output unit emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a -1 otherwise. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

$g(x) = 0$ … decision surface

if $g(x)$ is linear ➔ hyperplane

If x_1 and x_2 are on the same decision surface, then

$$w^T x_1 + w_0 = w^T x_2 + w_0$$

or

$$w^T (x_1 - x_2) = 0$$

➔ w (vector) is normal to any vector lying in the hyperplane

Hyperplane H divides feature space in 2 half-spaces

$g(x)$ gives a measure of distance from x to hyperplane

$$x = x_p + r \left(w / \|w\| \right)$$

where

x_p is the normal projection of x onto H
r is the distance

Of course $g(x_p) = 0$ ➔

$$g(x) = w^T x + w_0 = r \|w\|$$

$$r = g(x) / \|w\|$$

Distance from origin to H: $w_0 / \|w\|$

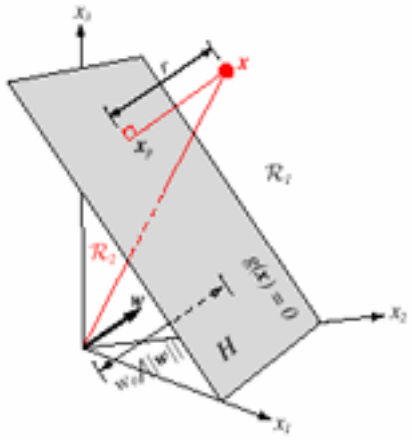


FIGURE 5.2. The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Multicategory case:

many ways how to do it wrong …

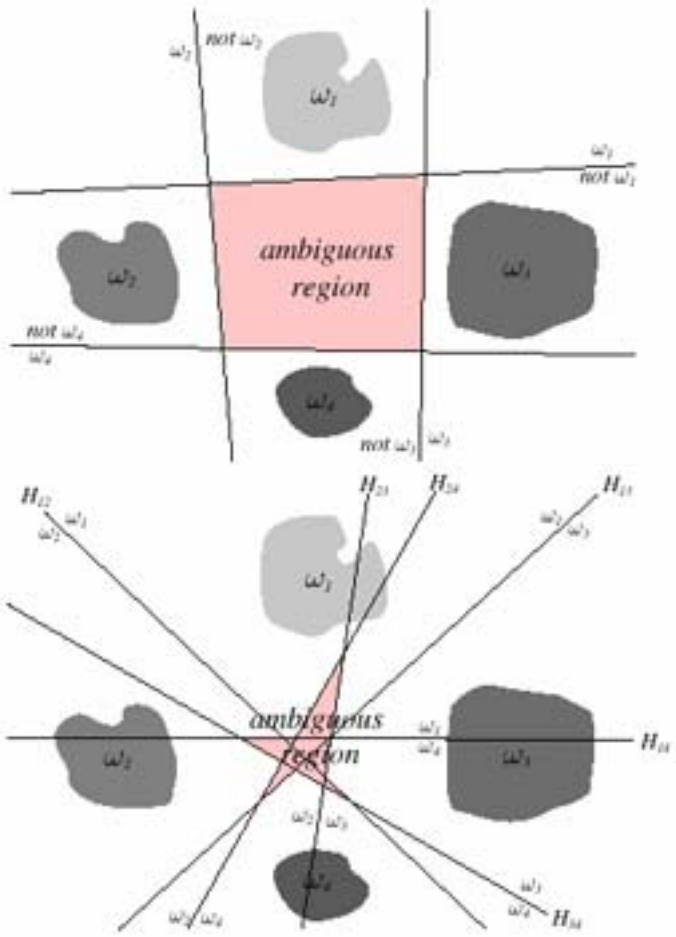


FIGURE 5.3. Linear decision boundaries for a four-class problem. The top figure shows ω_i/not ω_i dichotomies while the bottom figure shows ω_i/ω_j dichotomies and the corresponding decision boundaries H_{ij}. The pink regions have ambiguous category assignments. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

better solution… several discriminant functions …
"c" functions for c classes

$$g_i(x) = w^T x_i + w_{i0}$$

assign x to ω_i if $g_i(x) > g_j(x)$ for all j except j=i

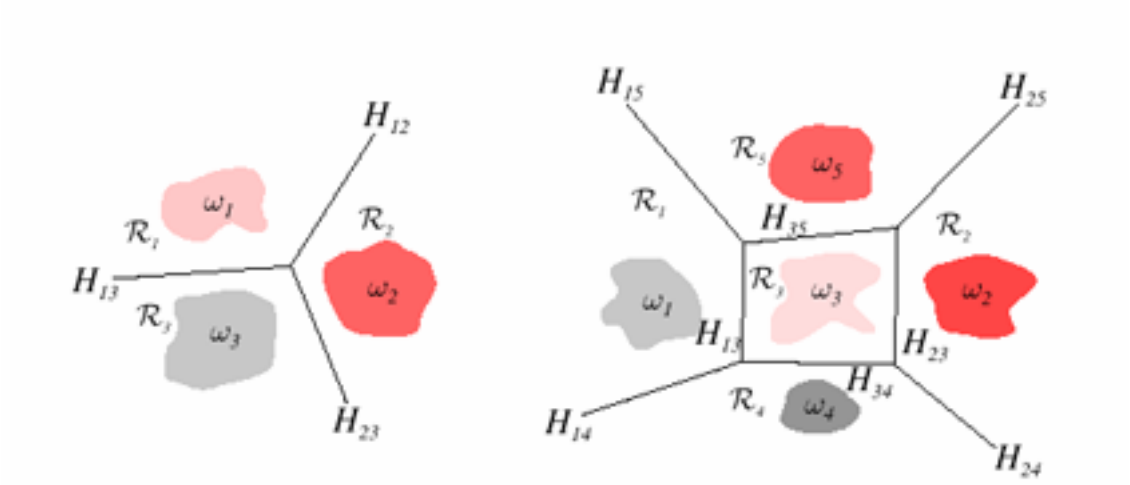


FIGURE 5.4. Decision boundaries produced by a linear machine for a three-class problem and a five-class problem. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Hyperplanes H_{ij} $g_i(x) = g_j(x)$

or

$$(w_i - w_j)^T x + (w_{io} - w_{j0}) = 0$$

➔ $(w_i - w_j)$ is normal to H_{ij}

distance from x to H_{ij} is $(g_i - g_j)/||w_i - w_j||$

➔ weight vectors are not important,
their differences are!

Decision regions for linear classification machines are convex.

Linear machines most suitable for unimodal distributions - but counterexamples exist.

Generalized linear discriminant functions

$$g(x) = w_o + \Sigma_{i=1}^{d} w_i x_i$$

w_i … components of the weight vector w

Adding products of pairs:

Quadratic discriminant functions

$$g(x) = w_o + \Sigma^{d}_{i=1} w_i x_i + \Sigma^{d}_{i=1} \Sigma^{d}_{j=1} w_{ij} x_i x_j$$

$g(x) = 0$ … hyperquadrics

Higher order ... hyperellipsoids

Generalized linear functions and Phi function

$g(x) = \Sigma^{d\wedge}_{i=1}\ a_i\ y_i(x)$

or

$g(x) = a^T\ y$ …

y(x) e.g., maps x from a d-dim space to d^-dim. space

Single-dimensional case example - quadratic discr. function:

$g(x) = a_1 + a_2\ x + a_3\ x^2$

$y(x) = (1, x, x^2)^T$

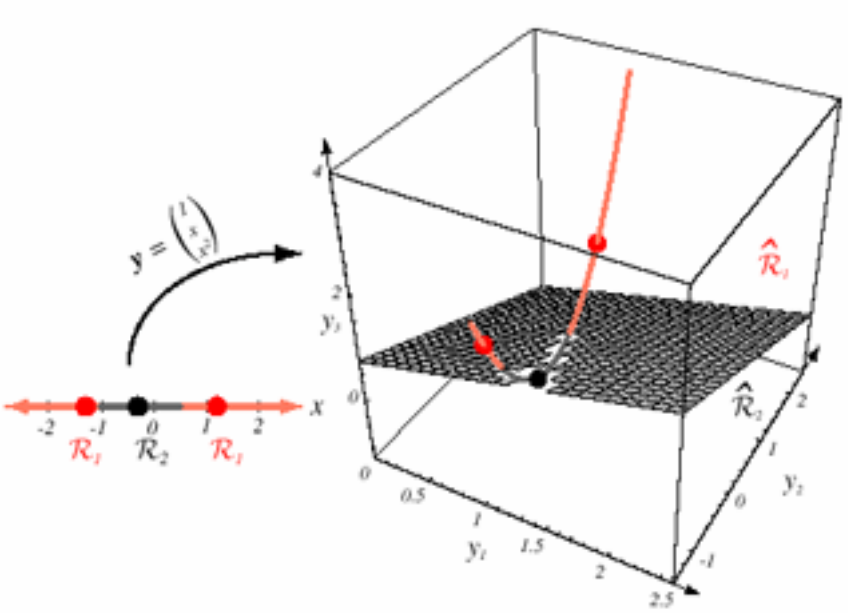


FIGURE 5.5. The mapping $\mathbf{y} = (1, x, x^2)^t$ takes a line and transforms it to a parabola in three dimensions. A plane splits the resulting **y**-space into regions corresponding to two categories, and this in turn gives a nonsimply connected decision region in the one-dimensional *x*-space. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Effectively, we are increasing dimensionality of the problem → increased flexibility of partitioning space.

Curse of dimensionality → may lead to unrealistic requirements for computation and data (training set sizes)

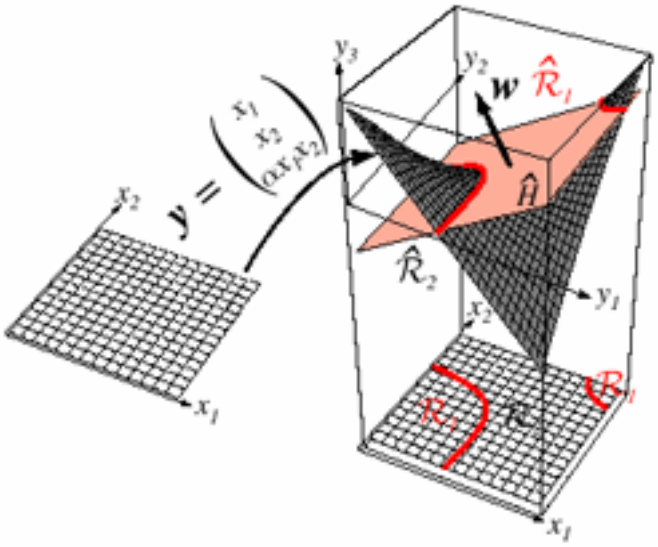


FIGURE 5.6. The two-dimensional input space **x** is mapped through a polynomial function f to **y**. Here the mapping is $y_1 = x_1$, $y_2 = x_2$ and $y_3 \propto x_1 x_2$. A linear discriminant in this transformed space is a hyperplane, which cuts the surface. Points to the positive side of the hyperplane $\hat{H}$ correspond to category ω_1, and those beneath it correspond to category ω_2. Here, in terms of the **x** space, $\mathcal{R}_1$ is a not simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Two-category linearly separable case

- linear separability

- normalization → all ω_2 samples replaced with their negatives

→ we look for a vector "a"

$$a^T y_i > 0$$

for all samples

"a" is called … separating vector

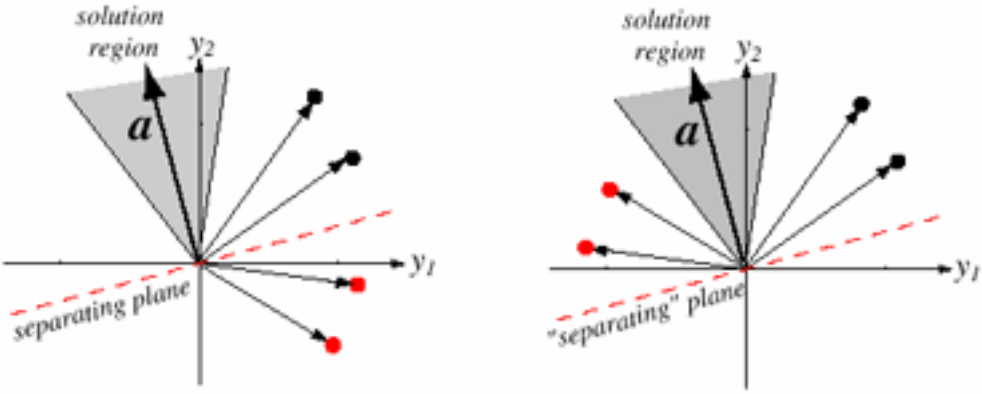


FIGURE 5.8. Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized"—that is, changed in sign. Now the solution vector leads to a plane that places all "normalized" points on the same side. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

→ Solution region

Margin …

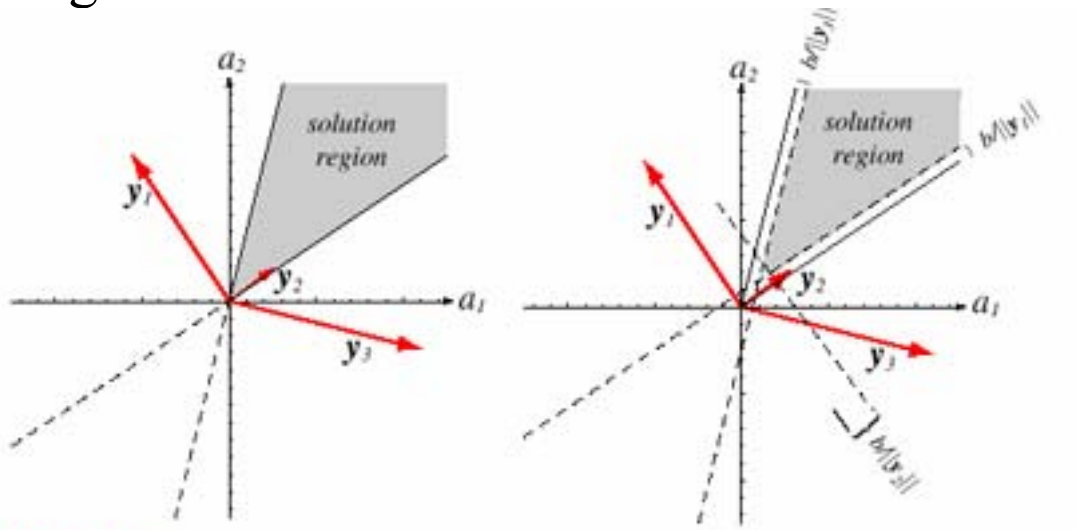


FIGURE 5.9. The effect of the margin on the solution region. At the left is the case of no margin ($b = 0$) equivalent to a case such as shown at the left in Fig. 5.8. At the right is the case $b > 0$, shrinking the solution region by margins $b/\|\mathbf{y}_i\|$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Gradient descent procedures

- solution to a set of linear inequalities

$$a^T y_i > 0$$

➔ define criterion function
J(a) that is minimized
if **a** is a solution vector

➔ minimizing a scalar function - solvable by gradient descent approaches

$$a(k+1) = a(k) - \eta(k) \nabla J(a(k))$$

where η is a positive scale factor … *learning rate*

Algorithm: Basic gradient descent

initialize a, threshold θ, $\eta(.)$, k=0
  do k := k+1
    a:= a - $\eta(k) \nabla J(a)$
  until $|\eta(k) \nabla J(a)| < \theta$
return a
end

Choice of $\eta(k)$ …

if $\eta(k)$ too small … slow convergence
if $\eta(k)$ too large … overshooting, even divergence

Assuming that criterion function can be approximated by 2nd order expansion around value a(k)

$$J(a) \approx J(a(k)) + \nabla J^T(a-a(k)) + 1/2\ (a-a(k))^T\ H\ (a-a(k))$$

H … Hessian matrix of 2nd partial derivatives

$$\partial^2 J / \partial a_i\ \partial a_j$$

evaluated at a(k).

As above, $a(k+1) = a(k) - \eta(k)\ \nabla J(a(k))$

thus:

$$J(a(k+1)) \approx J(a(k)) - \eta(k)\|\nabla J\|^2 + 1/2\ \eta^2(k)\ \nabla J^T H \nabla J$$

Therefore, J(a(k+1)) can be minimized by choosing

$$\eta(k) = \|\nabla J\|^2 / \nabla J^T H \nabla J$$

where H depends on a and thus on k.

(If J is a quadratic function, H is constant and thus η is constant independent of k)

Newton Descent:

identical, but

$$a(k+1) = a(k) - H^{-1} \nabla J$$

Newton's algorithm gives greater improvement per step then simple gradient descent, but cannot work for singular H.

Newton algorithm is also more computationally demanding, and thus simple gradient descent is frequently faster overall (more iterations, but faster executed).

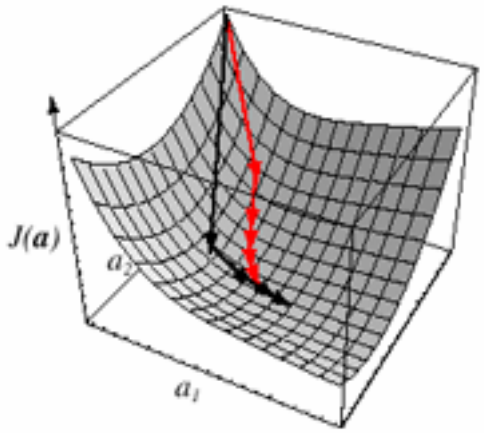


FIGURE 5.10. The sequence of weight vectors given by a simple gradient descent method (red) and by Newton's (second order) algorithm (black). Newton's method typically leads to greater improvement per step, even when using optimal learning rates for both methods. However the added computational burden of inverting the Hessian matrix used in Newton's method is not always justified, and simple gradient descent may suffice. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification.* Copyright © 2001 by John Wiley & Sons, Inc.

Perceptron Criterion Function

criterion function for solving linear inequalities

$$a^T y_i > 0$$

Obvious choice of J(a, y) is the number of samples misclassified by a … but is piecewise constant, not good for optimization.

Perceptron criterion function:

$$J_p(a) = \Sigma_{y \in Y} (-a^T y)$$

where Y is a set of samples misclassified by a.

(Y is empty for perfect classification)

$a^T y <= 0$ for misclassified y

➔ $J_p(a)$ is non-negative

➔ $J_p(a) = 0$ for a on the decision boundary

→ $J_p(a)$ is proportional to sum of distances from misclassified samples to decision boundaries - see figure below:

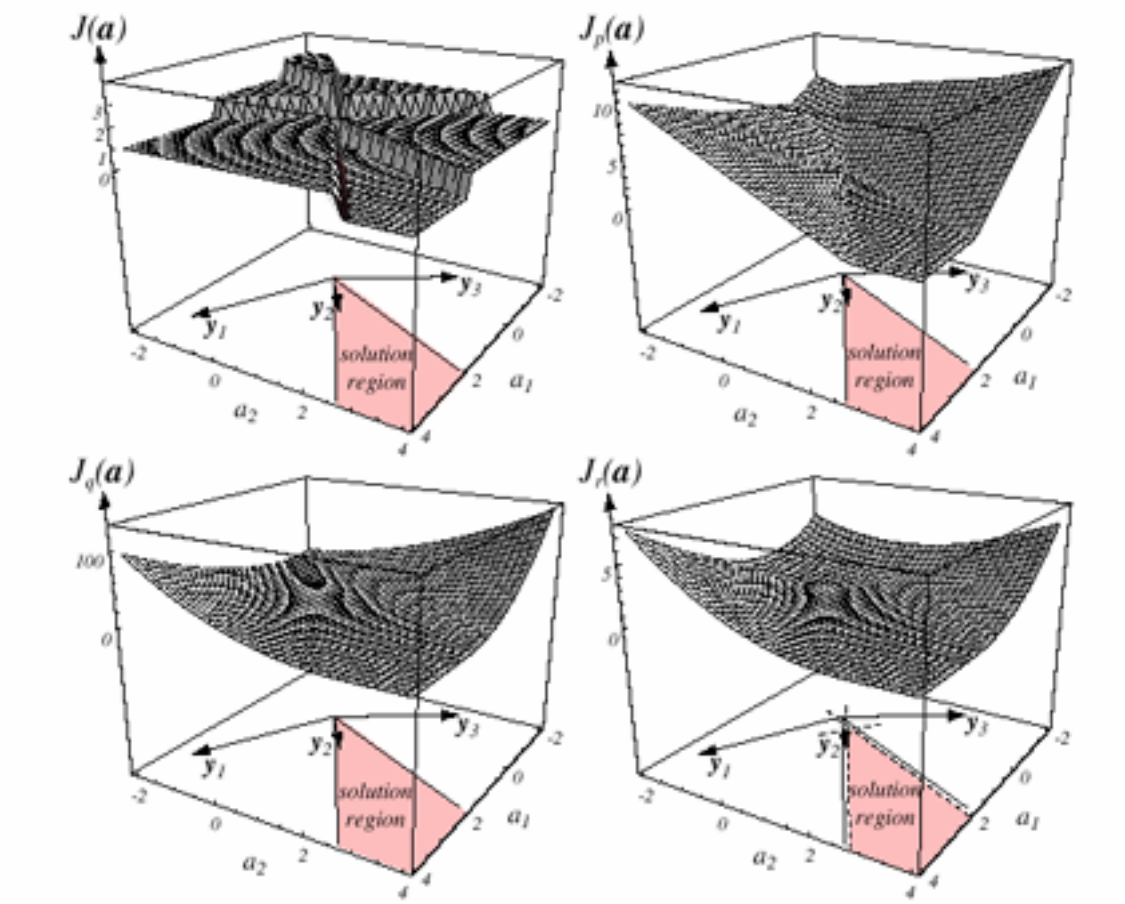


FIGURE 5.11. Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the Perceptron criterion (Eq. 16), which is piecewise linear and acceptable for gradient descent. The lower left is squared error (Eq. 32), which has nice analytic properties and is useful even when the patterns are not linearly separable. The lower right is the square error with margin (Eq. 33). A designer may adjust the margin b in order to force the solution vector to lie toward the middle of the $b = 0$ solution region in hopes of improving generalization of the resulting classifier. From: Richard O. Duda, Peter E. Hart, and

j-th component of gradient of J_p is $\partial J_p / \partial a_j$

$$\nabla J_p = \Sigma_{y \in Y} (-y)$$

and

$$a(k+1) = a(k) + \eta(k) \Sigma_{y \in Y_k} (y)$$

Perceptron algorithm (batch):

initialize a, threshold η(.), criterion θ, k=0
do k := k+1
a:= a + η $\Sigma_{y \in Yk}$ (y)
until |η(k) $\Sigma_{y \in Yk}$ (y)| < θ
return a
end

➔ The next vector is obtained by adding some multiple of the sum of the misclassified samples to the present weight vector.

Example for a(1) = 0, η(k) = 1 :

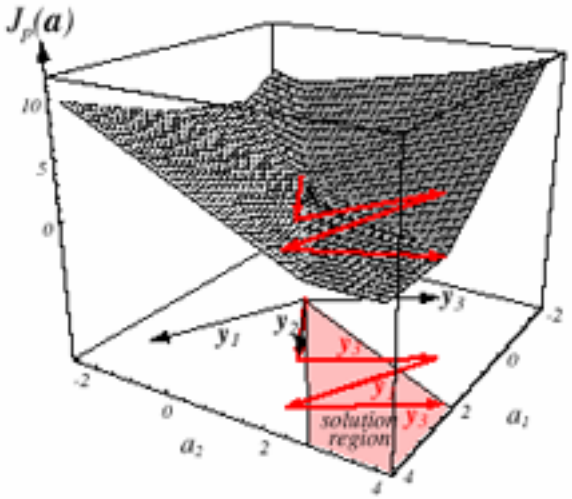


FIGURE 5.12. The Perceptron criterion, $J_p(\mathbf{a})$, is plotted as a function of the weights a_1 and a_2 for a three-pattern problem. The weight vector begins at **0**, and the algorithm sequentially adds to it vectors equal to the "normalized" misclassified patterns themselves. In the example shown, this sequence is $\mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_1, \mathbf{y}_3$, at which time the vector lies in the solution region and iteration terminates. Note that the second update (by $\mathbf{y}_3$) takes the candidate vector *farther* from the solution region than after the first update (cf. Theorem 5.1). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Single-sample fixed increment perceptron

- samples considered cyclically, misclassified samples are used for modification of function a.

… a(1) .. arbitrary
… $a(k+1) = a(k) + y^k$ for $k>0$

where y^k is one of the n samples $y_1, \ldots, y_n$ that is misclassified at the current stage

again, $a^T(k)\, y^k >= 0$ for all k

Perceptron algorithm (single sample, fixed increment):

```
initialize a, k=0
   do k := (k+1) mod n
        if y^k is misclassified by a
             then a := a + y^k
   until all patterns are correctly classified
return a
end
```

Algorithm only converges for linearly separable classes.

Variations on single-sample, fixed increment:

e.g., variable increment with margin

$a(1)$ .. arbitrary
$a(k+1) = a(k) + \eta(k)\, y^k$ for $k>0$

modification whenever $a^T(k)\, y^k$ fails to exceed margin b

Perceptron algorithm (single sample, variable increment):

initialize a, , threshold θ, $\eta(.)$, k=0
  do k := (k+1) mod n
    if $a^T(k)\, y^k <= b$
      then $a := a + \eta(k)\, y^k$
  until $a^T(k)\, y^k > b$ for all k
return a
end

All the approaches above are "error-correcting" approaches, they keep modifying the weight vector until no errors are present.

Non-separable behavior

- above-mentioned approaches require linearly separable classes
- even if this is the case for training set, the set-up does not guarantee good performance in reality

➔ how will these approaches behave in linearly non-separable classes?

Minimum Squared-Error Procedures

- let's involve ALL samples in the criterion function.

let's now try to make

$$a^T y_i = b_i$$

where b_i is some arbitrary positive constant

It is reasonable to hope that by minimizing squared-error criterion function, a useful discriminant function will be obtained for separable and non-separable cases.

$a^T y_i = b_i$ … set of linear equations

➔ find a vector a satisfying

$$Y\, a = b$$

Y … matrix, i-th row is vector y_i^T

for non-singular Y … $a = Y^{-1} b$ … but Y is usually not square, typically more rows than columns ➔ overdetermined set of equations

→ minimization of error

$$e = YA - b$$

e.g., minimizing square of error vector

$$J_s(a) = ||Y\, a - b\,||^2 = \Sigma^n_{i=1} (a^T y_i - b_i)^2$$

$$\nabla J_s = \Sigma^n_{i=1} 2(a^T y_i - b_i)\, y_i \;=\; 2\, Y^T (Ya-b)$$

➔solution …necessary condition $Y^T Y\, a = Y^T b$

$Y^T Y$ is a square matrix

➔ if regular $a = (Y^T Y)^{-1} Y^T b$

$$Y^{+} = (Y^{T} Y)^{-1} Y^{T}$$

called <u>pseudoinverse</u> of Y … always exists

$Y^{+} = Y^{-1}$ for square regular matrix Y

$$Y^{+} Y = I$$

but

$$Y Y^{+} \neq I$$

➔ $a = Y^{+} b$

is the MSE solution to $Ya = b$