Neuronale Netze

Pattern Recognition I&II

Alex Waibel

8+15.11.2011

Organisation

- Literatur:
  - Introduction to The Theory of Neural Computation
    Hertz, Krogh, Palmer, Santa Fe Institute
  - Neural Network Architectures – An Introduction,
    Judith Dayhoff, VNR Publishers
  - Andere Publikationen, TBD
  - Papers On-Line

- Sprechstunde:
  - Dienstag, 12:00, oder Anmeldung Sekretariat

  - Link im Studienportal, und
    http://isl.anthropomatik.kit.edu
    lectures -> neuronale netze

Neural Nets

- Terminology:
  - Artificial Neural Networks
  - Connectionist Models
  - Parallel Distributed Processing (PDP)
  - Massive Parallel Processing
  - Multi-layer Perceptron (MLP)

The brain is:

- ~ 1000 supercomputers
- ~ 1/10 pocket calculator
- The brain is very good for some problems: vision, speech, language, motor-control.
- It is very poor at others: arithmetic

Von Neumann Computer - Neural Computation

- Processing: Sequential - Parallel
- Processors: One - Many
- Interaction: None - A lot
- Communication: Poor - Rich
- Processors: Fast, Accurate - Slow, Sloppy
- Knowledge: Local - Distributed
- Hardware: General Purpose - Dedicated
- Design: Programed - Learned

Why Neural Networks

- Massive parallelism.
- Massive constraint satisfaction for ill-defined input.
- Simple computing units.
- Many processing units, many interconnections.
- Uniformity (-> sensor fusion)
- Non-linear classifiers/ mapping (-> good performance)
- Learning/ adapting
- Brain like ??

History

- McCollough & Pitts, 1943
- Perceptrons, Rosenblatt ~1960
- Minsky, 1967
- PDP, Neuro Rennaissance ~1985
- Since then
  - Bio, Neuro, Brain-Science
  - Machine Learning, Statistics
  - Applications, Part of the Toolbox

Using Neural Nets

- Classification
- Prediction
- Function Approximation
- Continuous Mapping
- Pattern Completion
- Coding

Design Criteria

- Recognition Error Rate
- Training Time
- Recognition Time
- Memory Requirements
- Training Complexity
- Ease of Implementation
- Ease of Adaptation

Neural Models

- Back-Propagation
- Boltzman Machines
- Decision Tree Classifiers
- Feature Map Classifiers
- Learning Vector Quantizer (LVQ, LVQ2)
- High Order Networks
- Modified Nearest Neighbor
- Radial Basis Functions
- Kernels
- SVM
- etc.

Network Specification

- What parameters are typically chosen by the network designer:
  - Net Topology
  - Node Characteristics
  - Learning Rule
  - Objective Function
  - (Initial) Weights
  - Learning Parameters

Applications

- Space Robot*
- Autonomous Navigation*
- Speech Recognition and Understanding*
- Natural Language Processing*
- Music*
- Gesture Recognition
- Lip Reading
- Face Recognition
- Household Robots
- Signal Processing
- Banking, Bond Rating,...
- Sonar
- etc....

Advanced Neural Models

- Time-Delay Neural Networks (Waibel)
- Recurrent Nets (Elman, Jordan)
- Higher Order Nets
- Modular System Construction
- Adaptive Architectures
- Hybrid Neural/Non-Neural Architectures

Neural Nets - Design Problems

- Local Minima
- Speed of Learning
- Architecture must be selected
- Choice of Feature Representation
- Scaling
- Systems, Modularity
- Treatment of Temporal Features and Sequences

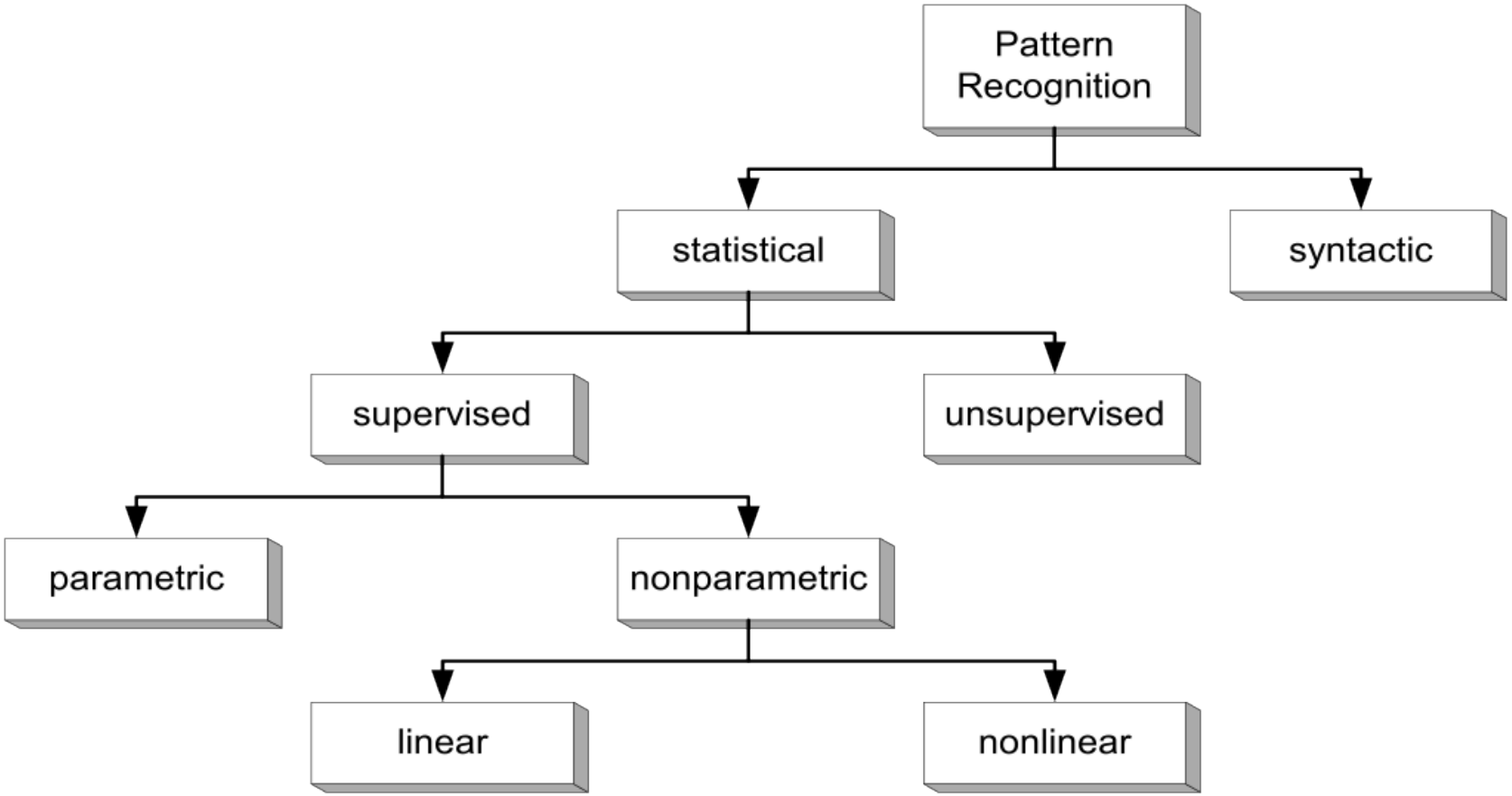
Neural Nets - Modeling

- Neural Nets are
  - Non-Linear Classifiers
  - Approximate Posterior Probabilities
  - Non-Parametric Training
- The Problem with Time
  - How to Consider Context
  - How to Operate Shift-Invariantly
  - How to Process Pattern Sequences

Pattern Recognition Overview

- Static Patterns, no dependence on Time or Sequential Order
- Important Notions
  - Supervised - Unsupervised Classifiers
  - Parametric - Non-Parametric Classifiers
  - Linear - Non-linear Classifiers
- Classical Methods
  - Bayes Classifier
  - K-Nearest Neighbor
- Connectionist Methods
  - Perceptron
  - Multilayer Perceptrons

Pattern Recognition



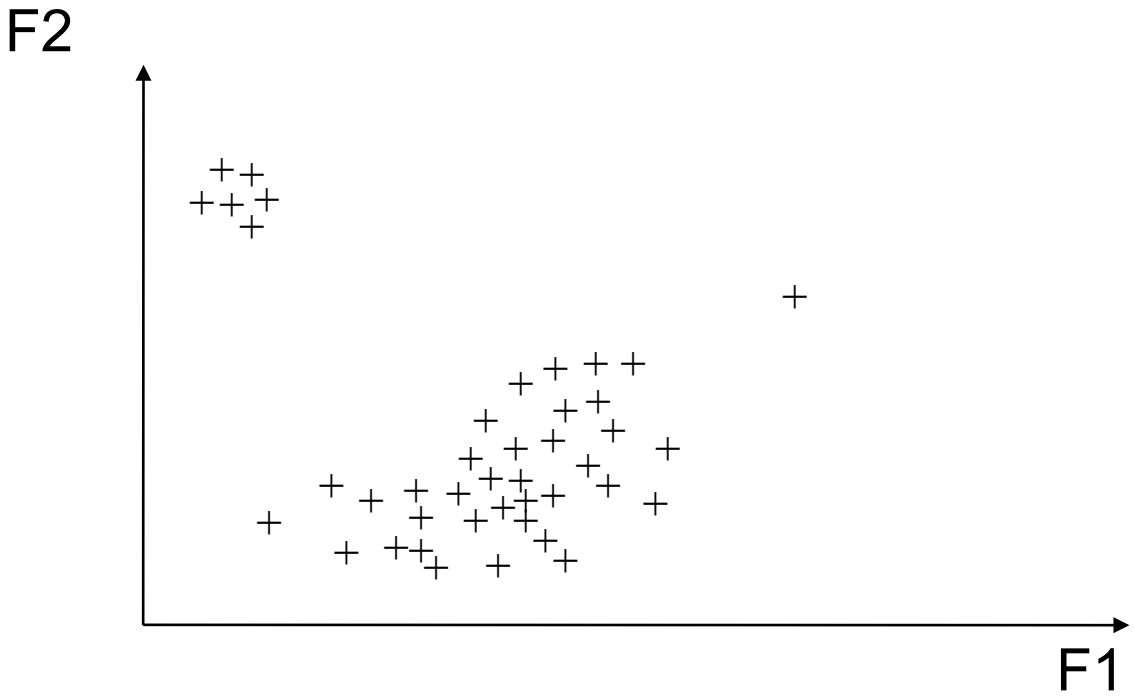
Supervised - Unsupervised

- Supervised training:
  Class to be recognized is known for each sample in training data. Requires a priori knowledge of useful features and knowledge/labeling of each training token (cost!).

- Unsupervised training:
  Class is not known and structure is to be discovered automatically.
  Feature-space-reduction

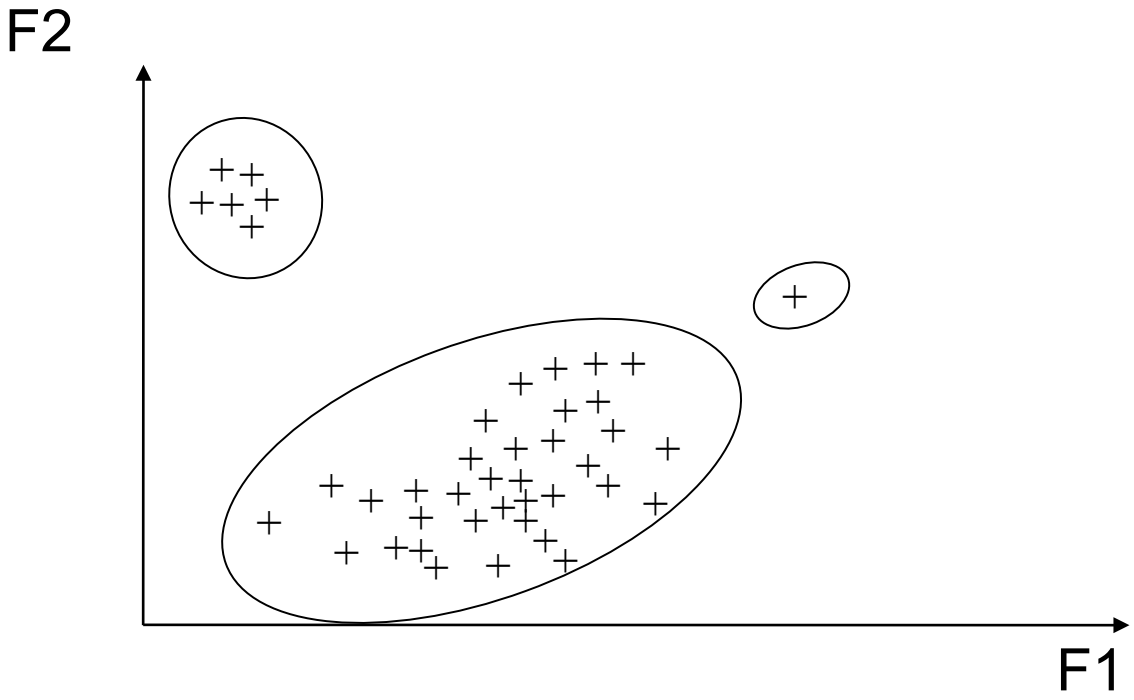
  example: clustering, autoassociative nets

Unsupervised Classification



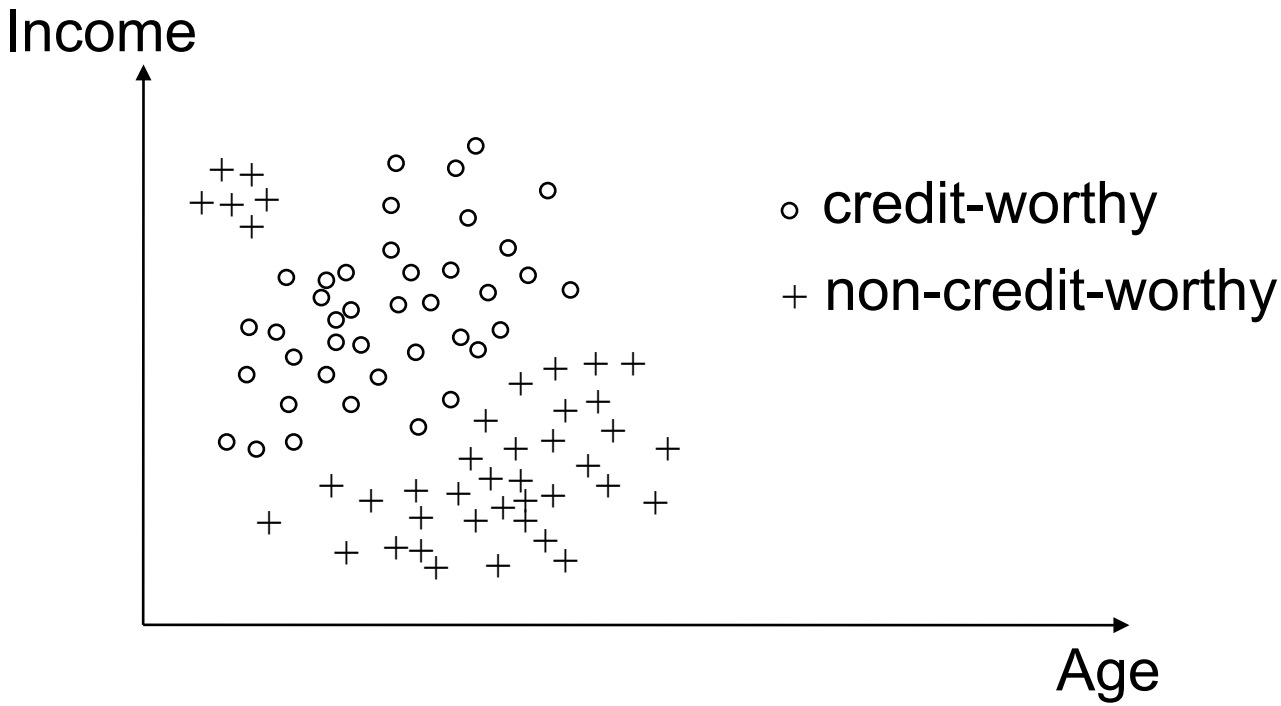
- Classification:
  - Classes Not Known: Find Structure

Unsupervised Classification



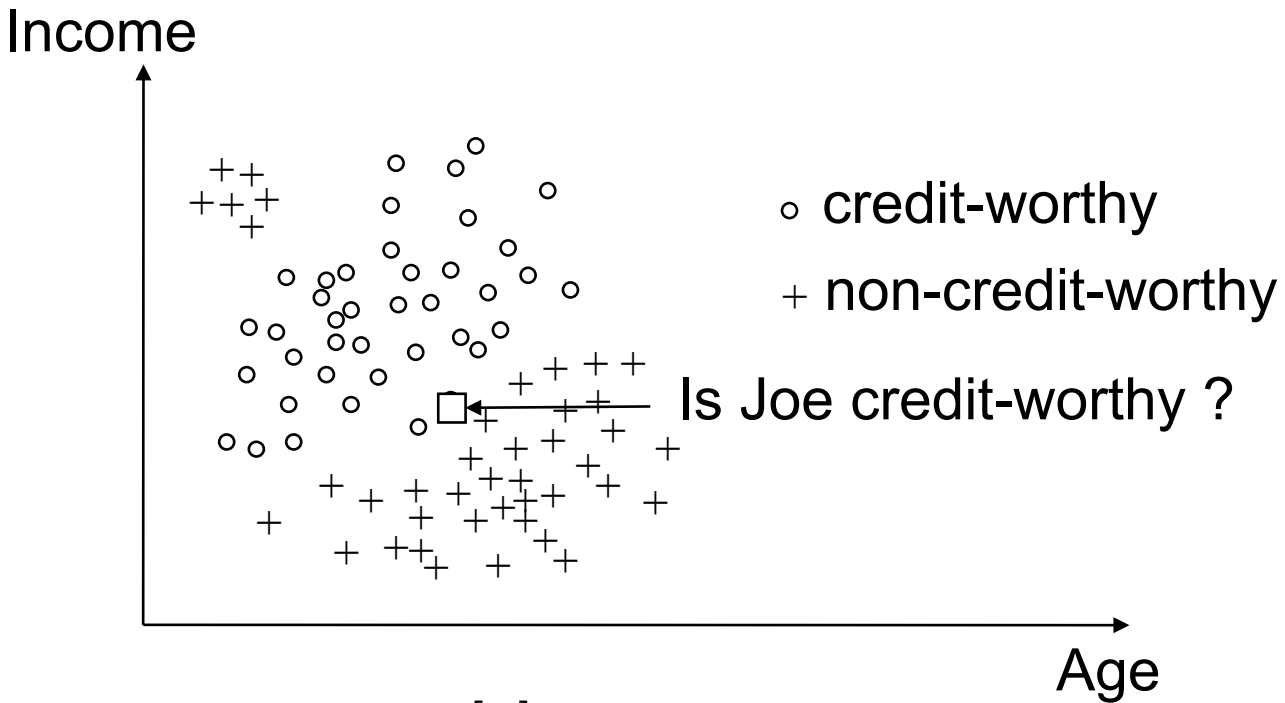
- Classification:
  - Classes Not Known: Find Structure
  - Clustering
  - How? How many?

Supervised Classification



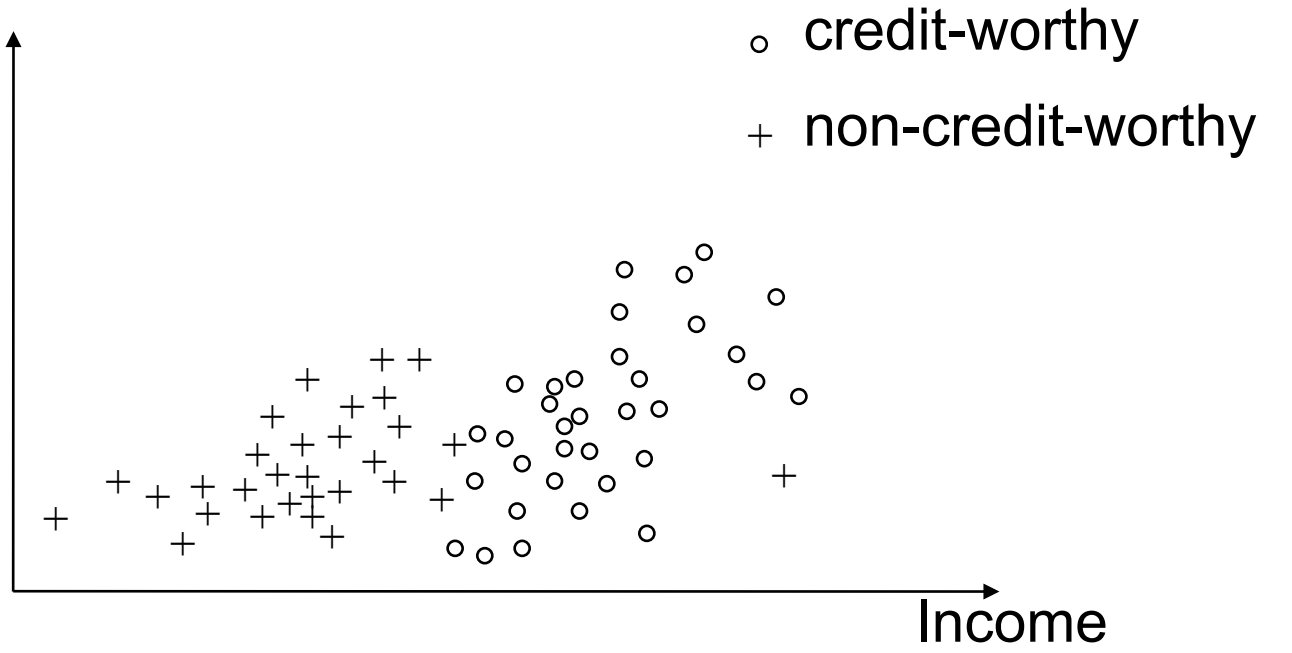
- Classification:
  - Classes Known: Creditworthiness: Yes-No
  - Features: Income, Age
  - Classifiers

Classification Problem

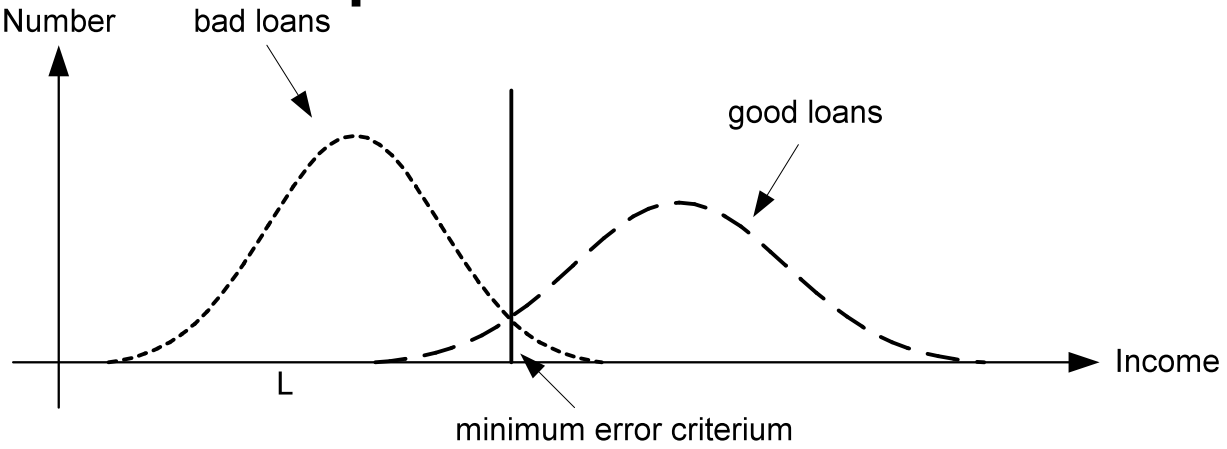


- Features: age, income $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
- Classes: creditworthy, non-creditworthy
- Problem: Given Joe's income and age, should a loan be made?
- Other Classification Problems: Fraud Detection, Customer Selection...

Classification Problem



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 citerion directly from training data.
  - Examples: Parzen Window, k-nearest neighbor, perceptron...

Bayes Decision Theory

Bayes Rule: $P(\omega_j / x) = \frac{p(x / \omega_j)P(\omega_j)}{p(x)}$

where $p(x) = \sum_j p(x / \omega_j)P(\omega_j)$

A priori probability $P(\omega_j)$

$\downarrow$ observation of x

A posteriori probability $P(\omega_j / x)$

Class-conditional Probability Density $p(x / \omega_j)$

Two classes case:

$$P(\text{error} / x) = \begin{cases} P(\omega_1 / x) & \text{if we decide } \omega_2 \\ P(\omega_2 / x) & \text{else} \end{cases}$$

Error is minimized, if we:

Decide ω_1 if $P(\omega_1 / x) > P(\omega_2 / x)$;
ω_2 otherwise

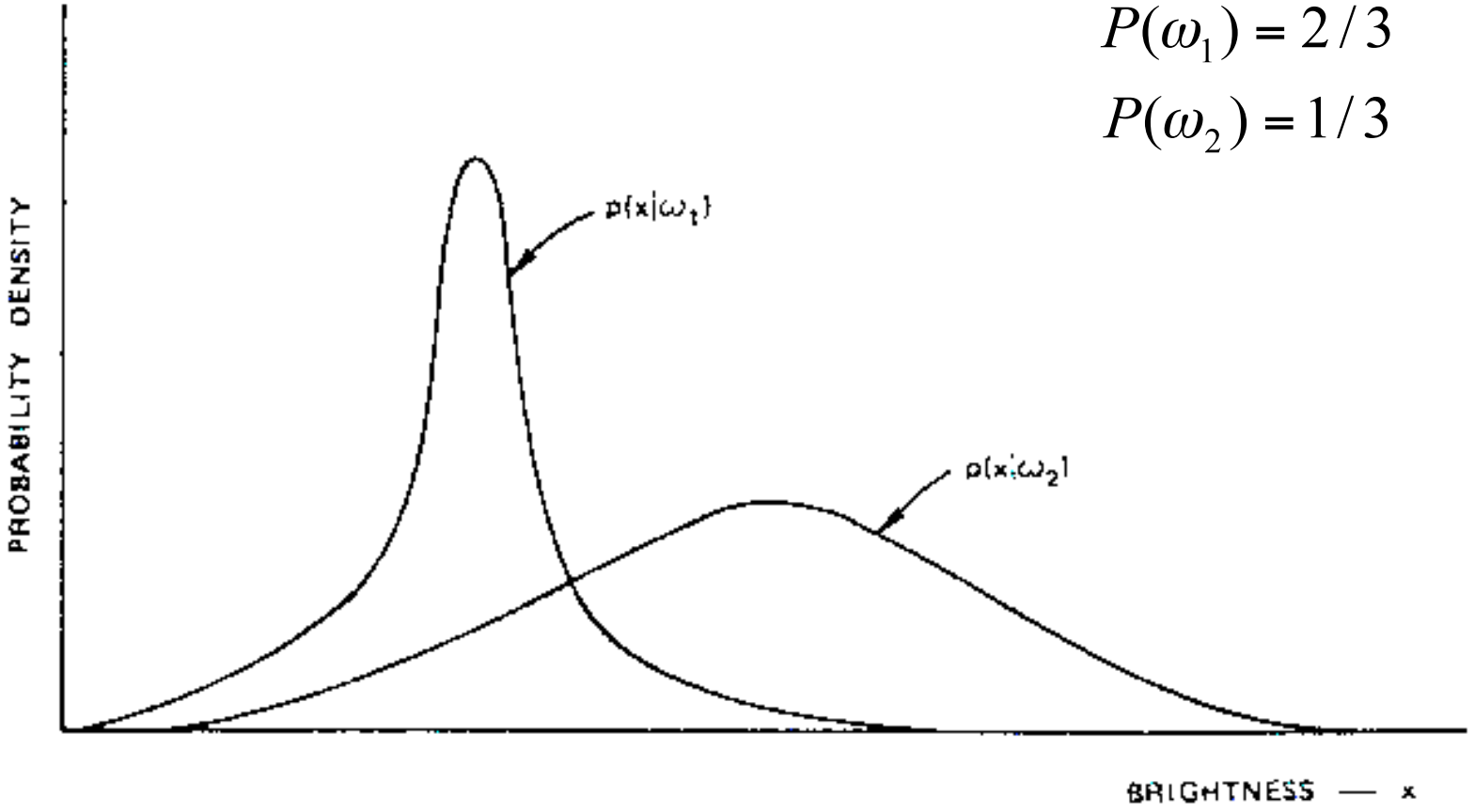
Decide ω_1 if $p(x / \omega_1)P(\omega_1) > p(x / \omega_2)P(\omega_2)$;
ω_2 otherwise

For the multicategory case:

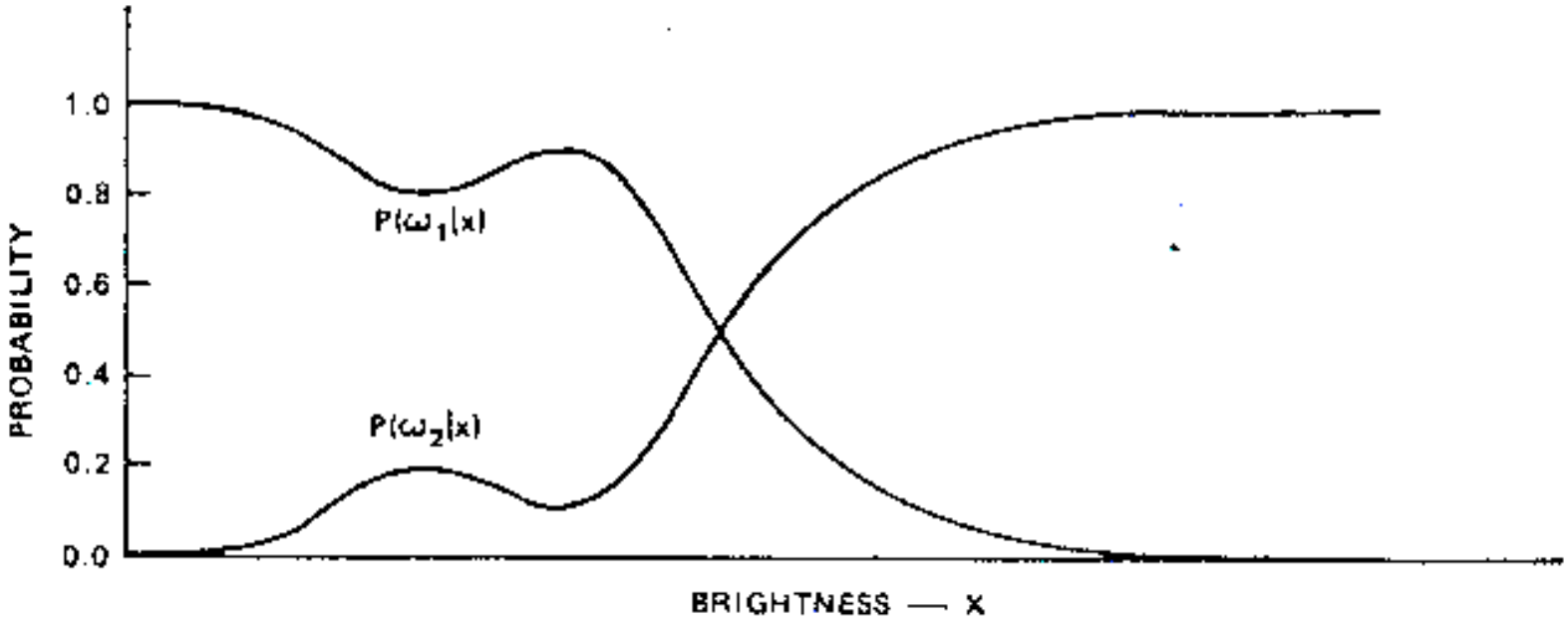
Decide ω_i if $P(\omega_i / x) > P(\omega_j / x)$ for all $j \neq i$

Hypothetical class-conditional probability density function

$$P(\omega_1) = 2/3$$
$$P(\omega_2) = 1/3$$



A posteriori probabilities



Classifier Discriminant Functions

$$g_i(x), i = 1, \ldots, c$$

Assign x to class ω_i , if $g_i(x) > g_j(x)$ for all $j \neq i$

$$g_i(x) = P(\omega_i / x)$$

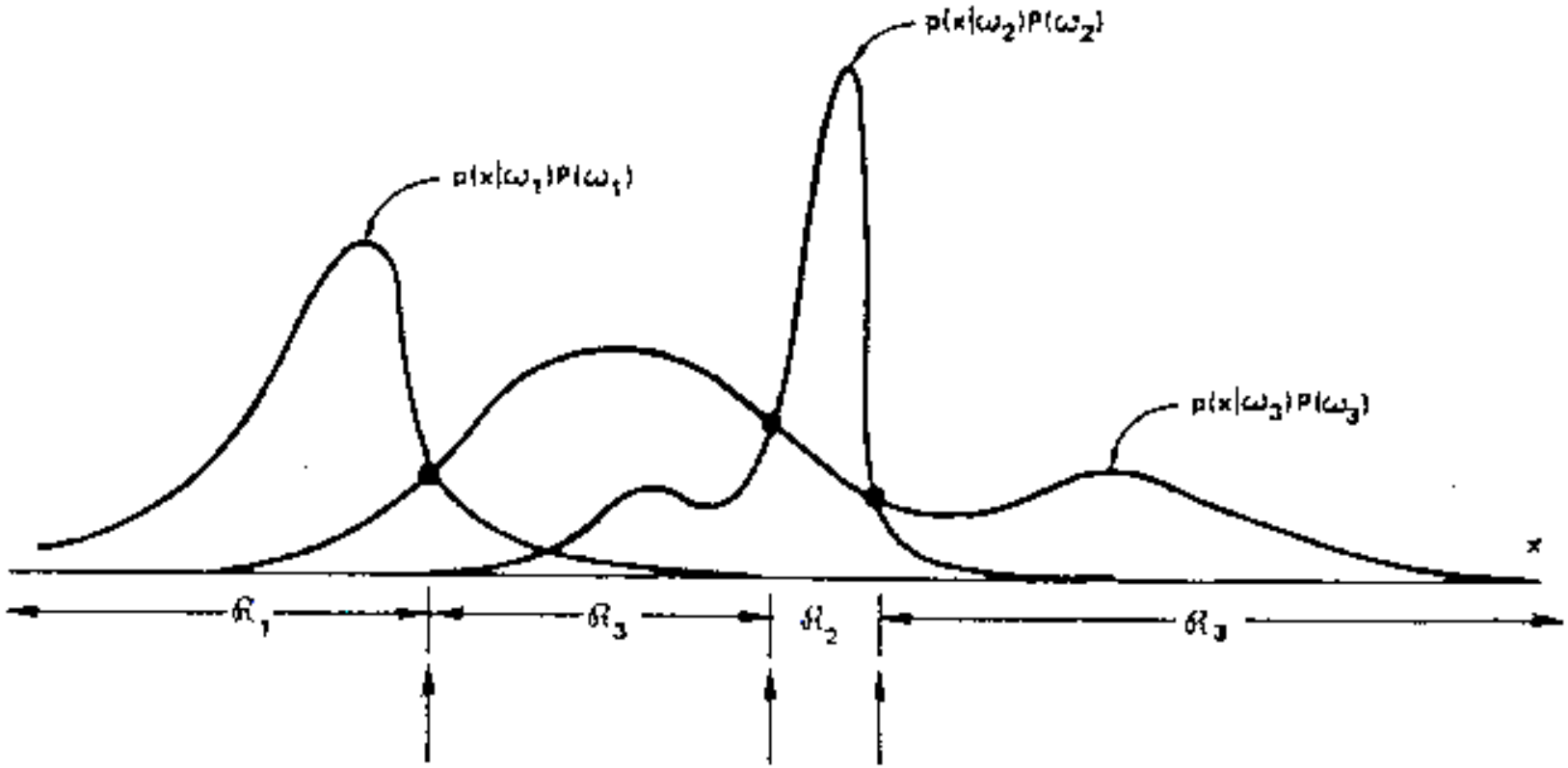
$$= \frac{p(x / \omega_i)P(\omega_i)}{\sum_{j=1}^{c} p(x / \omega_j)(\omega_j)} \longleftarrow \text{independent of class i}$$

$$g_i(x) = p(x / \omega_i)P(\omega_i)$$

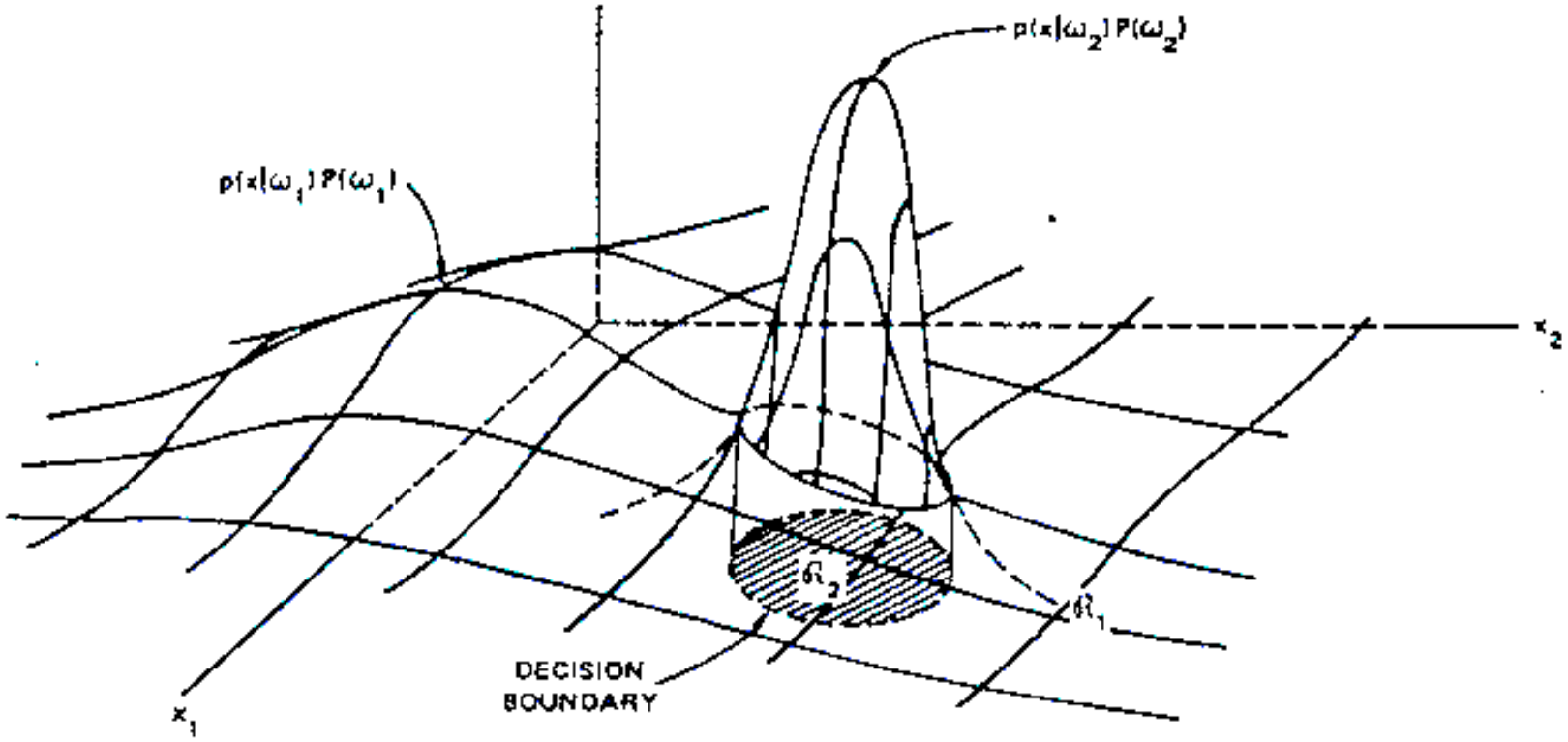
$$g_i(x) = \log(p(x / \omega_i)) + \log(P(\omega_i))$$

class conditional probability density function A priori probability

Examples Decision boundaries I



Examples Decision boundaries II



Classifier Design in Practice

- Need a priori probability $P(\omega_i)$ (not too bad)
- Need class conditional PDF $p(x / \omega_i)$
- Problems:
  - limited training data
  - limited computation
  - class-labelling potentially costly and errorful
  - classes may not be known
  - good features not known
- Parametric Solution:
  - Assume that $p(x / \omega_i)$ has a particular prametric form
  - Most common representative: multivariate normal density

GaussianClassifier

Univariate Normal Density: $$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma} \right)^2 \right]$$

$$\sim N(\mu, \sigma^2)$$

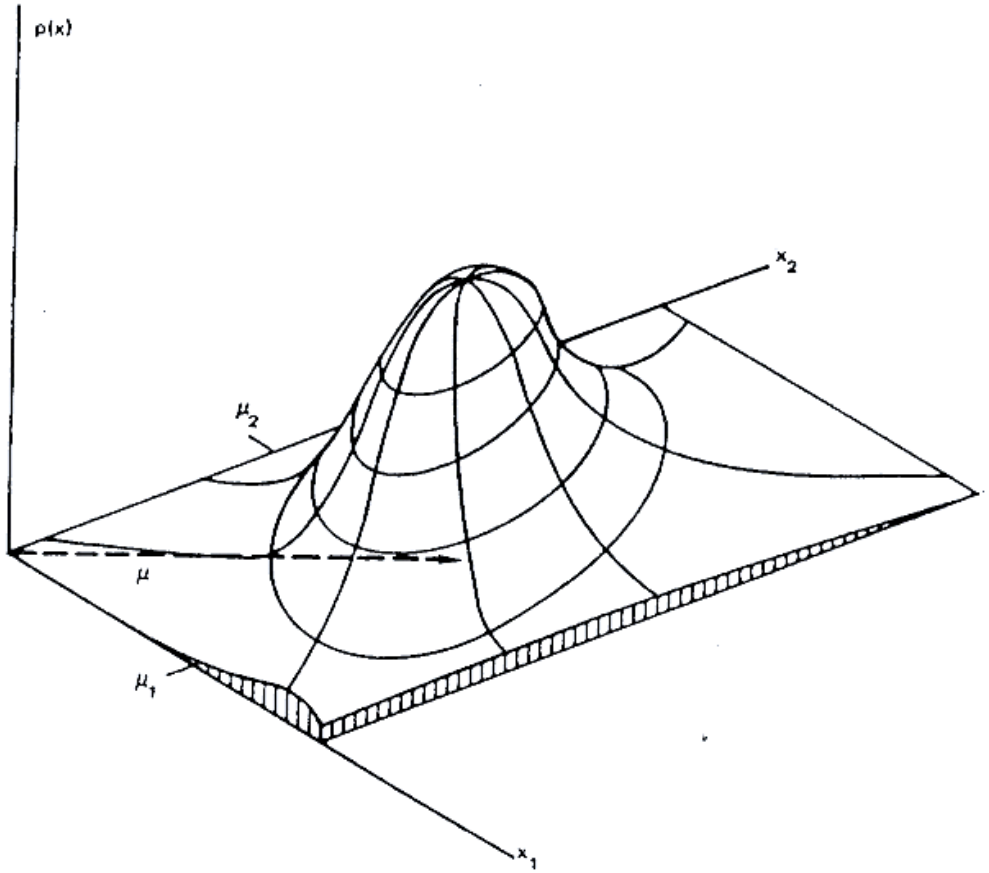
Multivariate Density: $$p(\vec{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu})^t \Sigma^{-1} (\vec{x}-\vec{\mu}) \right]$$

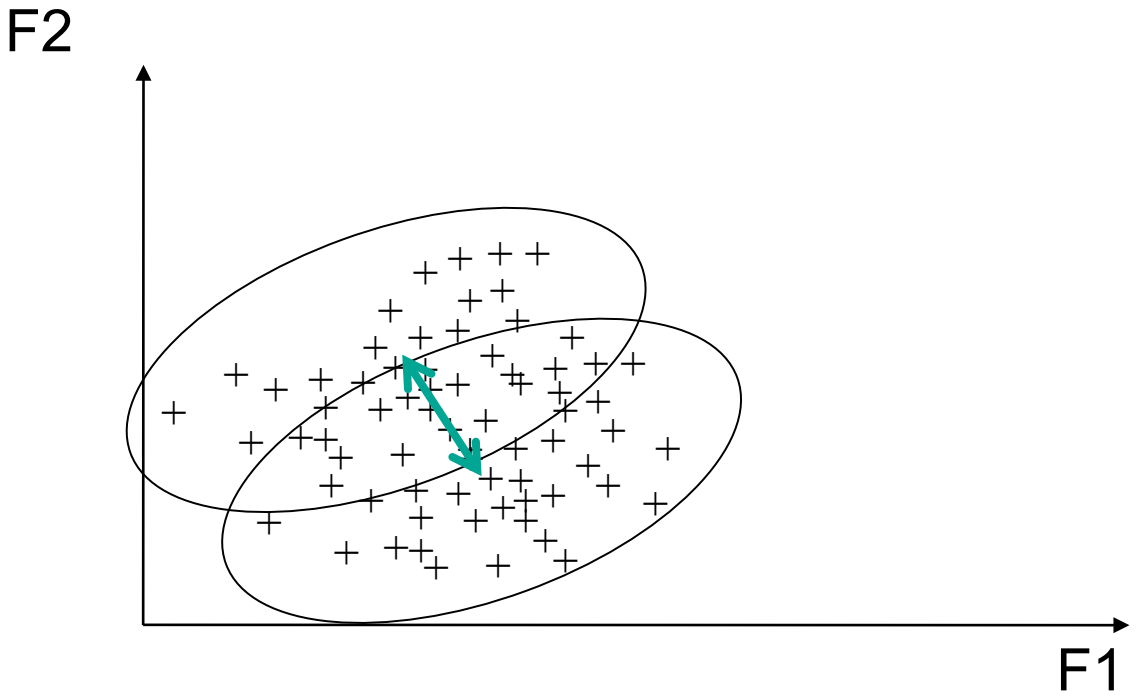
$$\sim N(\vec{\mu}, \Sigma)$$

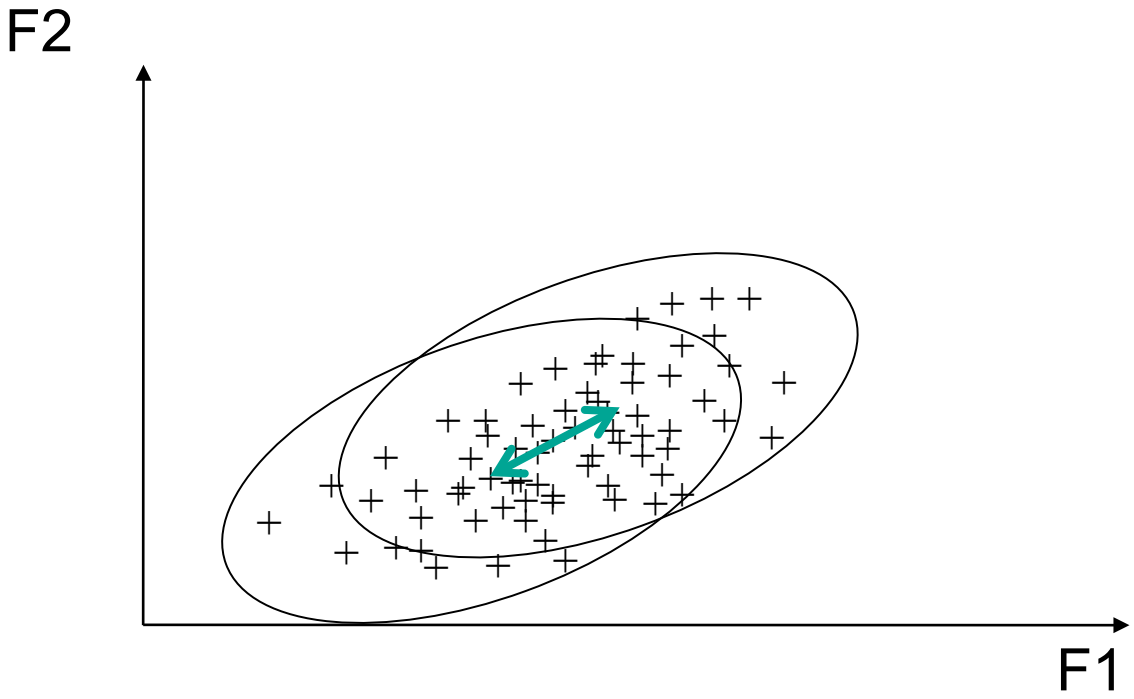
$$g_i(\vec{x}) = -\frac{1}{2}(\vec{x}-\vec{\mu}_i)^t \sum_i^{-1} (\vec{x}-\vec{\mu}_i) - \frac{d}{2}\log(2\pi)$$

$$-\frac{1}{2}\log|\Sigma_i| + \log P(\omega_i)$$

Bivariate Normal Density




F2
F1


F2
F1

Gaussian Classifier

- For each class i, need to estimate from trainingdata:
- covariance matrix Σ_i
- mean vector $\vec{\mu}_i$

Estimation of Parameters

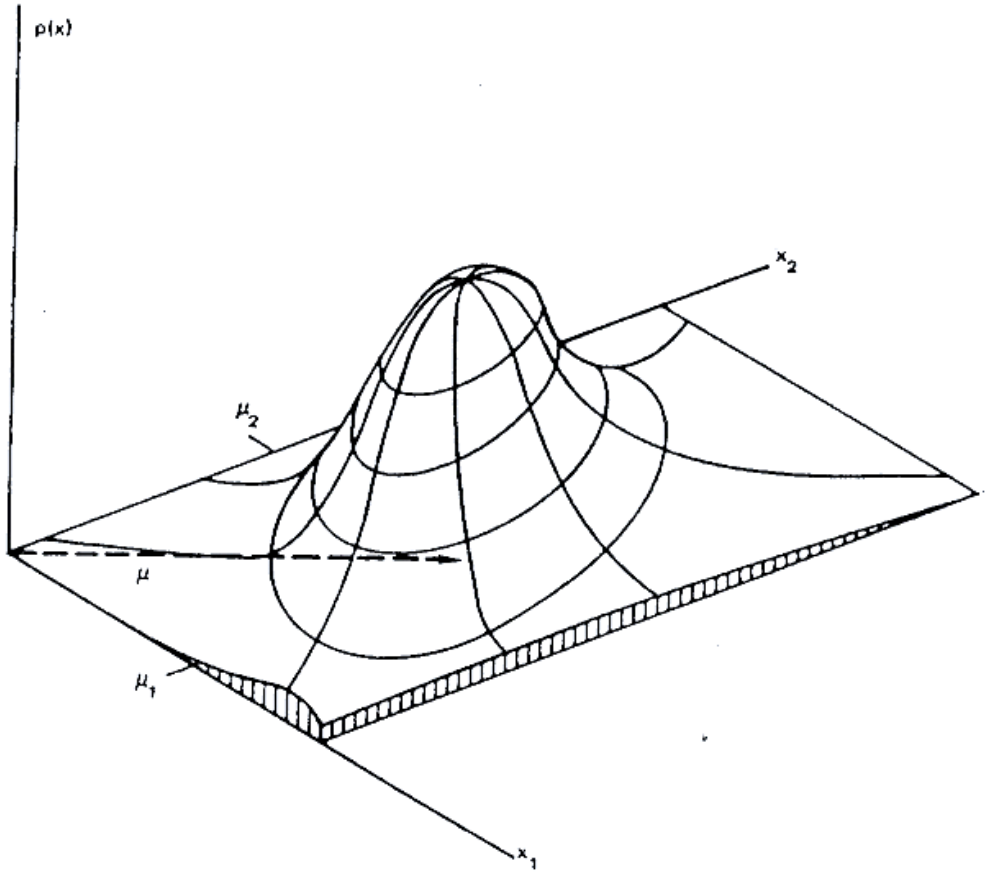
- MLE, Maximum Likelihood Estimation

- For Multivariate Case:

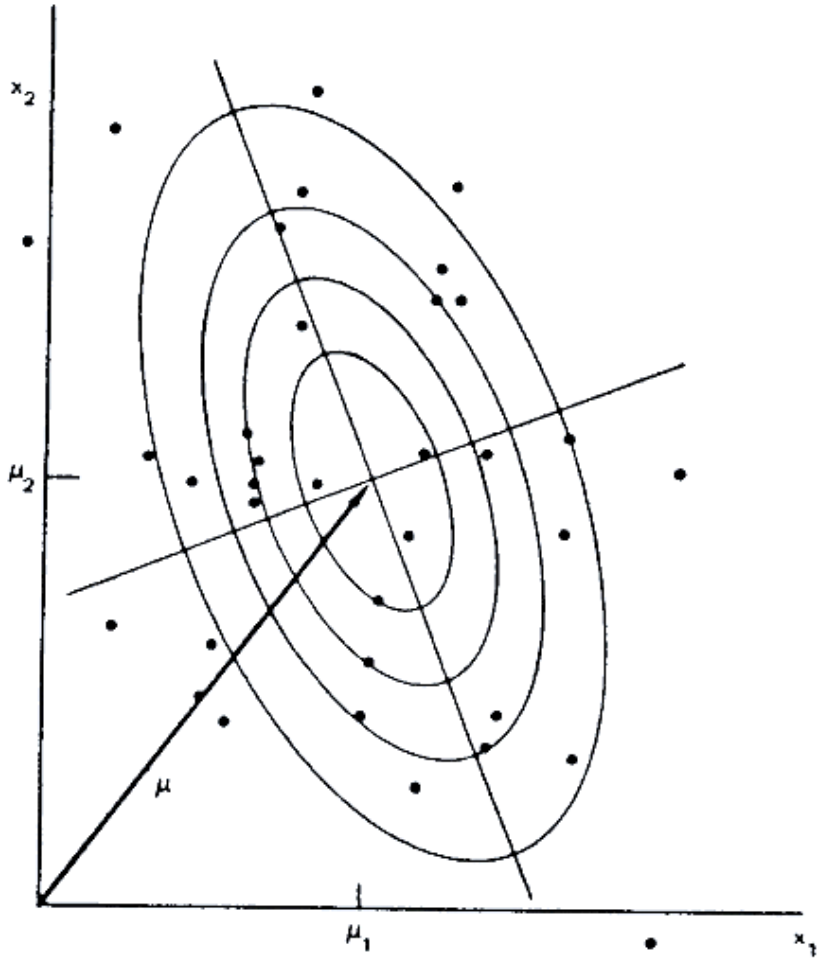
$$\vec{\mu} = \frac{1}{N}\sum_{k=1}^{N}\vec{x}_k$$

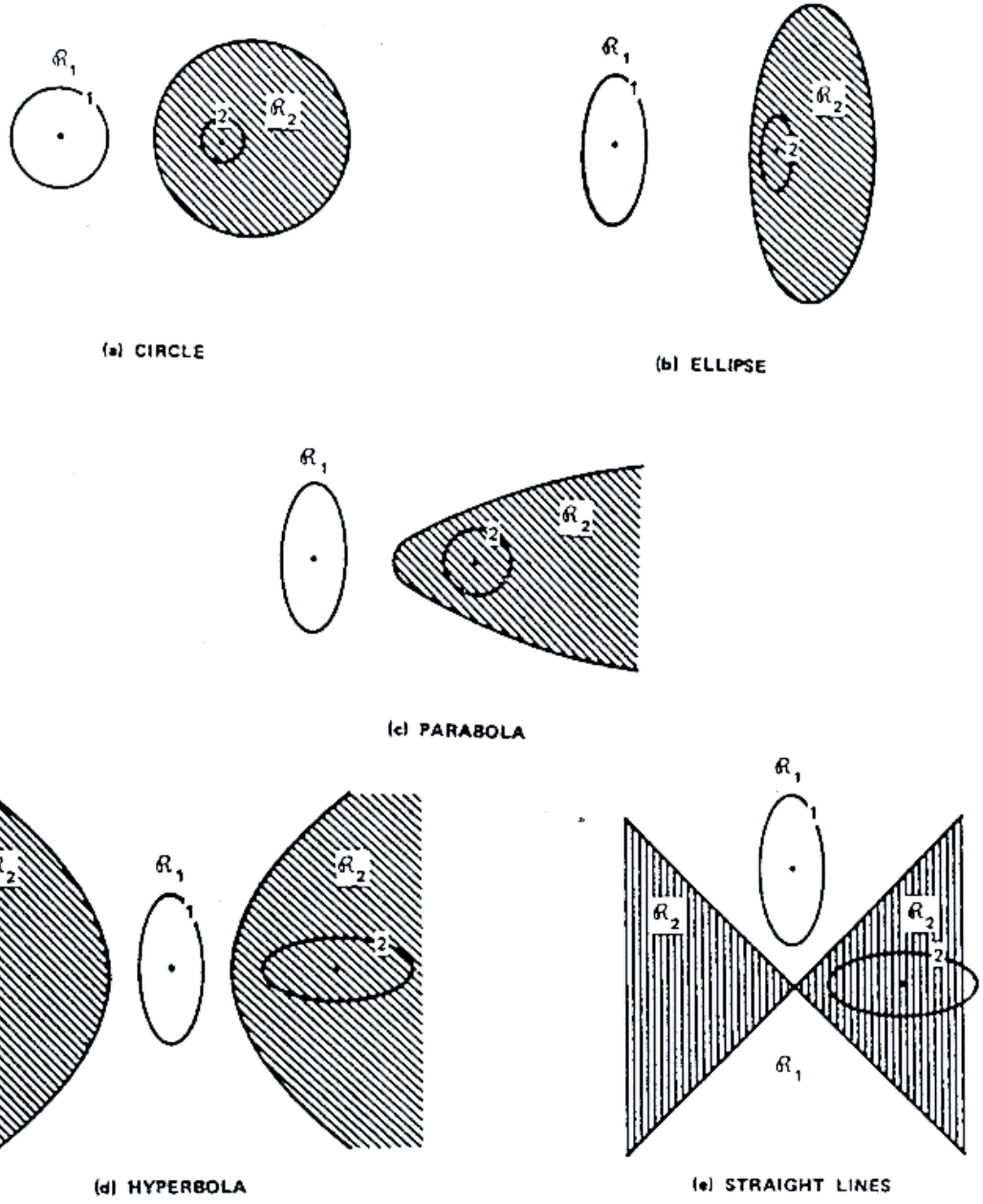
$$\Sigma = \frac{1}{N}\sum_{k=1}^{N}(\vec{x}_k - \vec{\mu})(\vec{x}_k - \vec{\mu})^T$$

Bivariate Normal Density



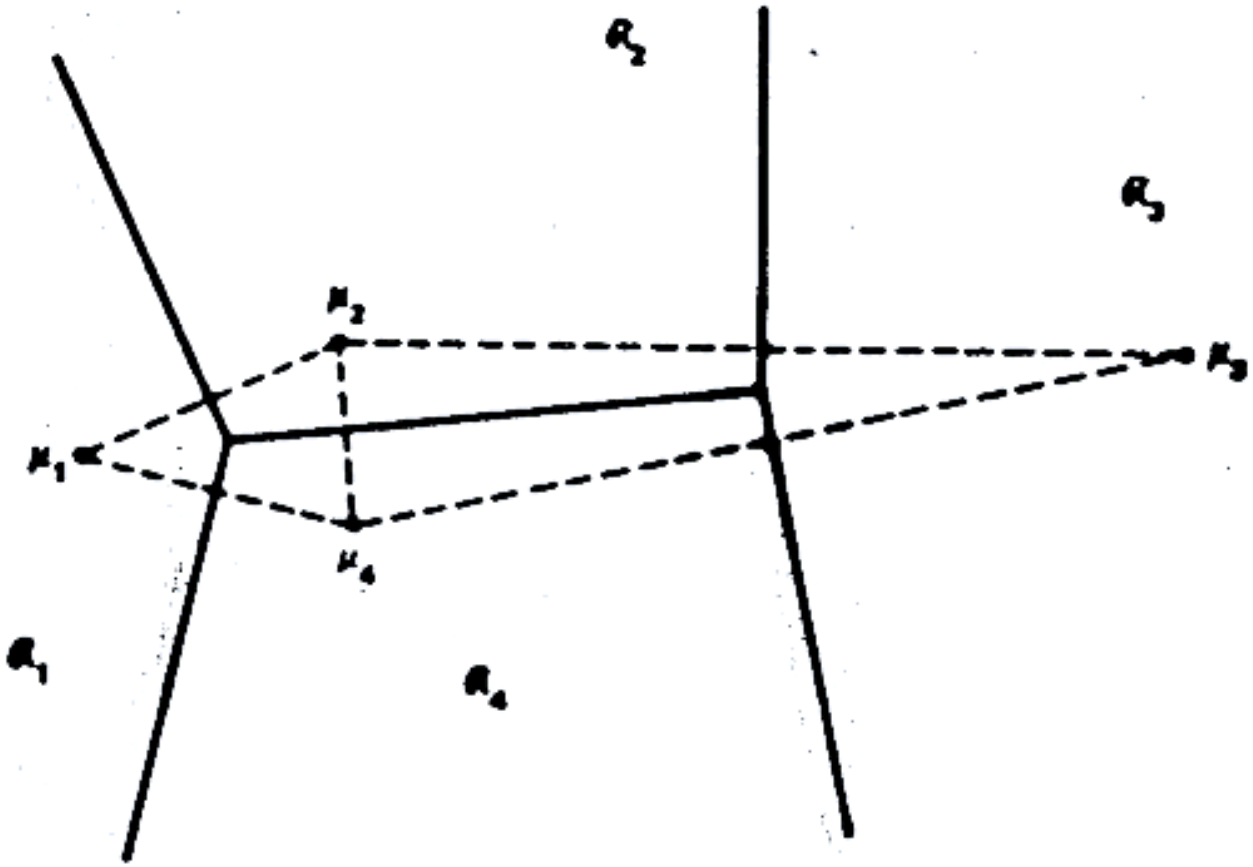
Scatter Diagram





Forms for decision boundaries for the general bivariate normal case

Decision Boundaries for a Minimum-Distance Classifier



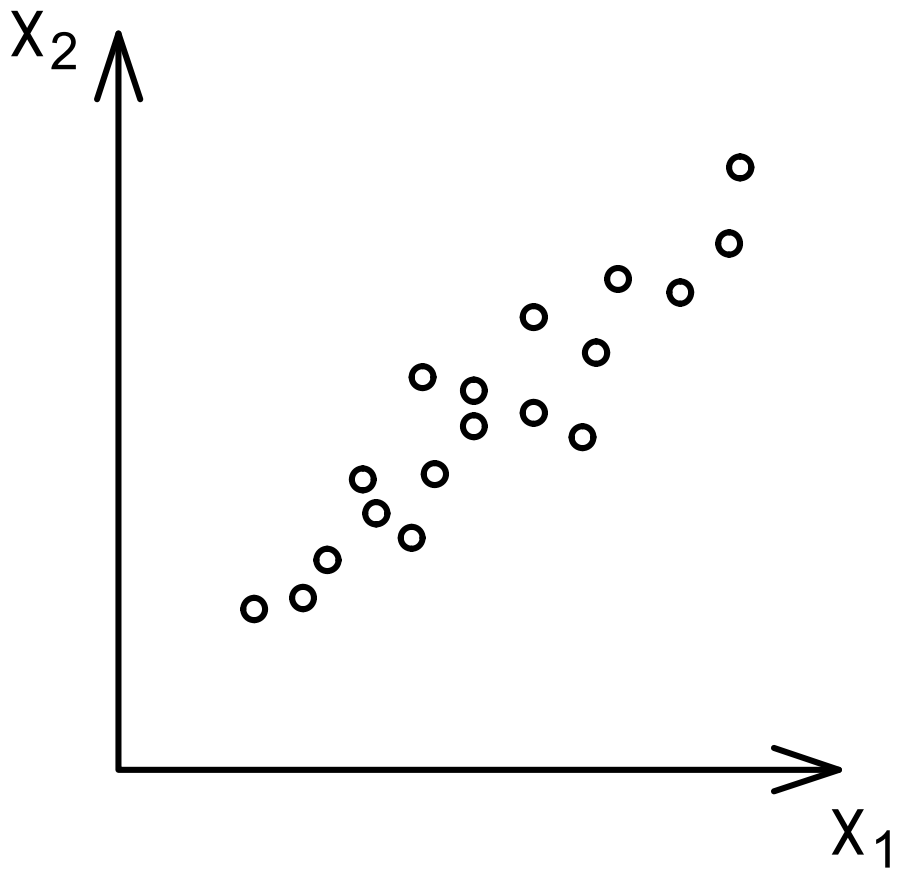
Problems of Classifier Design

- Features:
  - What and how many features should be selected?
  - Any features?
  - The more the better?
  - If additional features not useful (same mean and covariance), classifier will automatically ignore them?

Curse of Dimensionality

- Generally, adding more features indiscriminantly leads to worse performance!
- Reason:
  - Training Data vs. Number of Parameters
  - Limited training data.
- Solution:
  - select features carefully
  - reduce dimensionality
  - Principle Component Analysis

Principal Component Analysis (PCA)



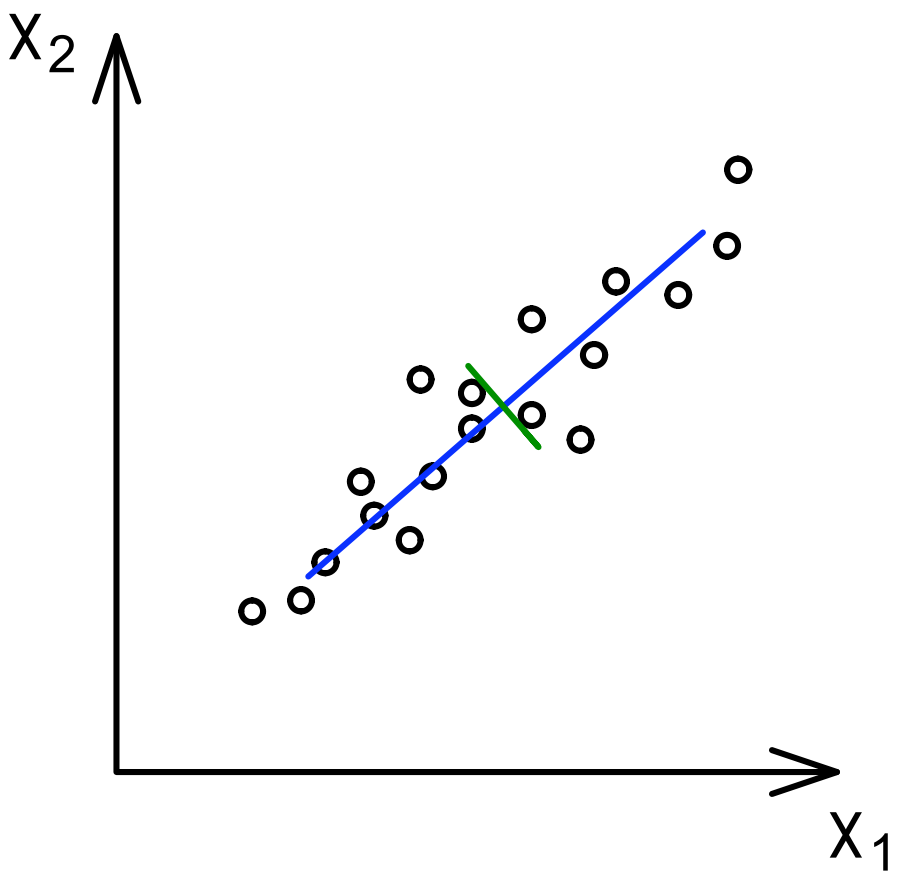
Assumption:

Single dimensions are correlated

Aim:

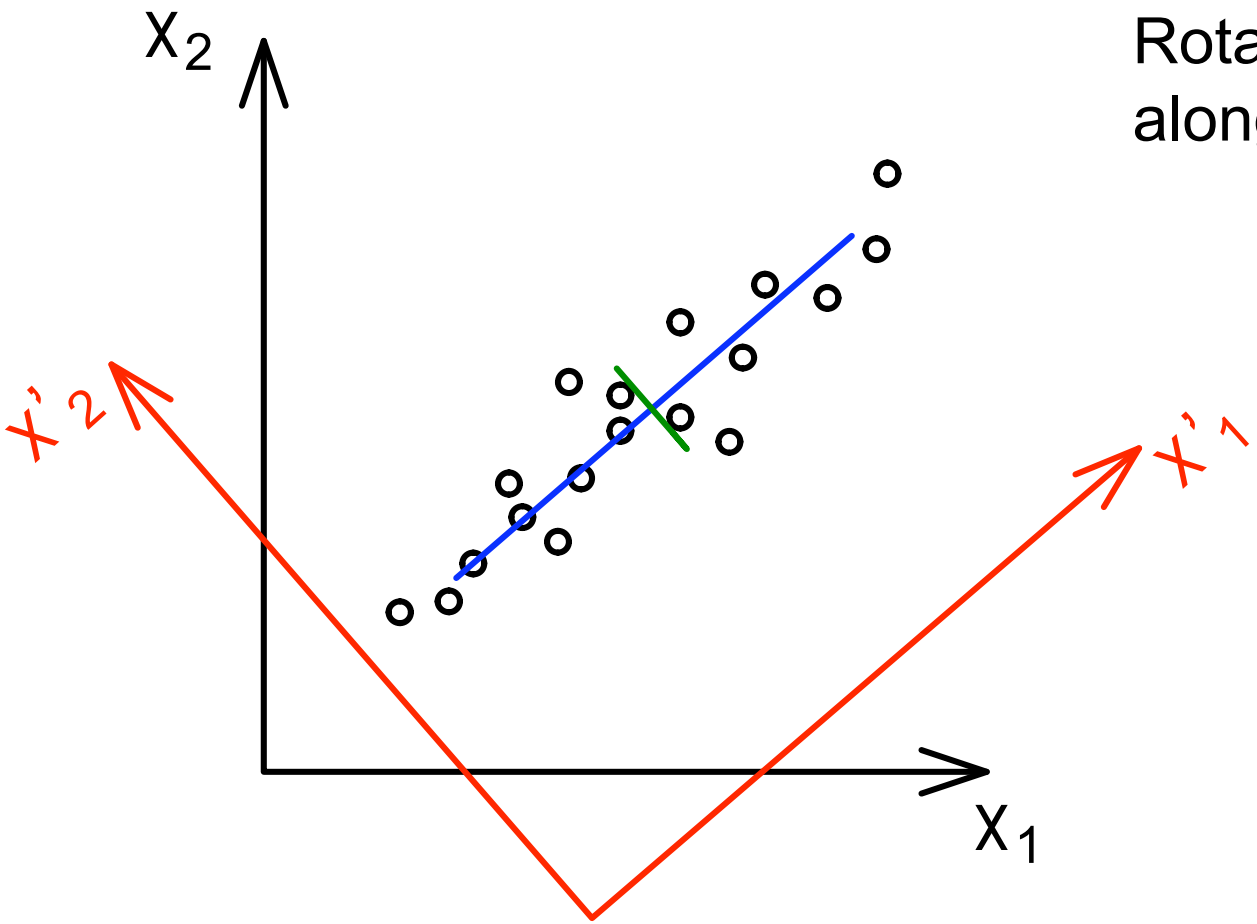
Reduce number of dimensions with minimum loss of information

Principal Component Analysis (PCA)



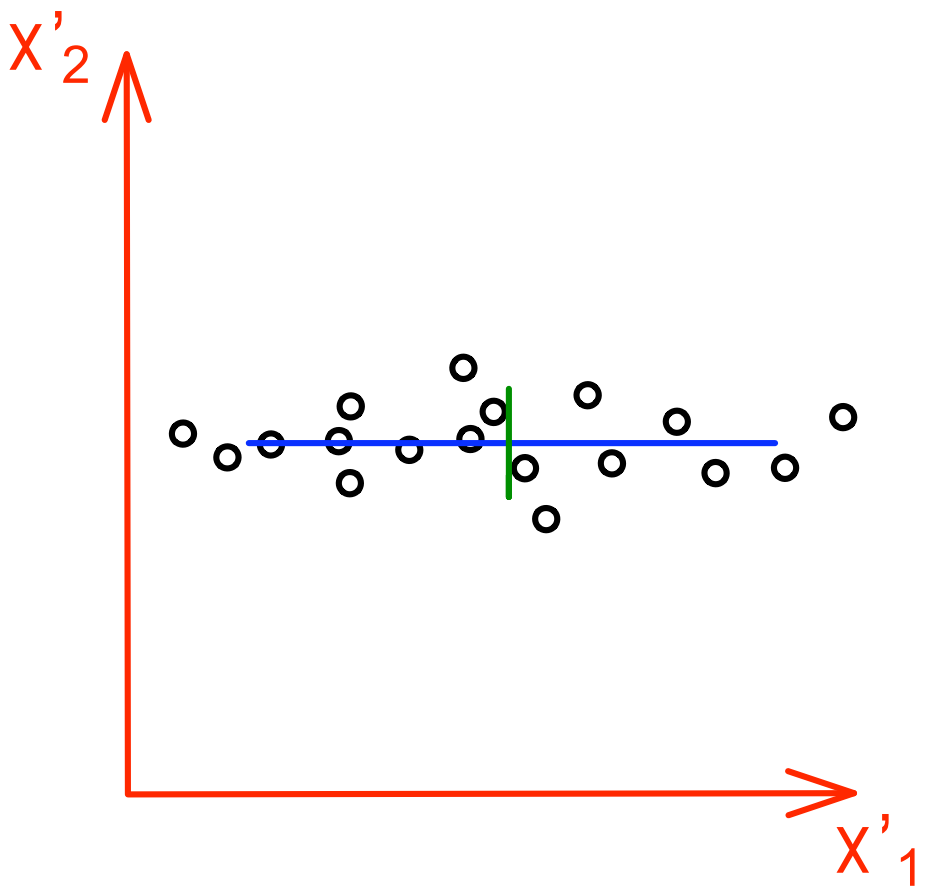
Find the axis along the highest variance

Principal Component Analysis (PCA)



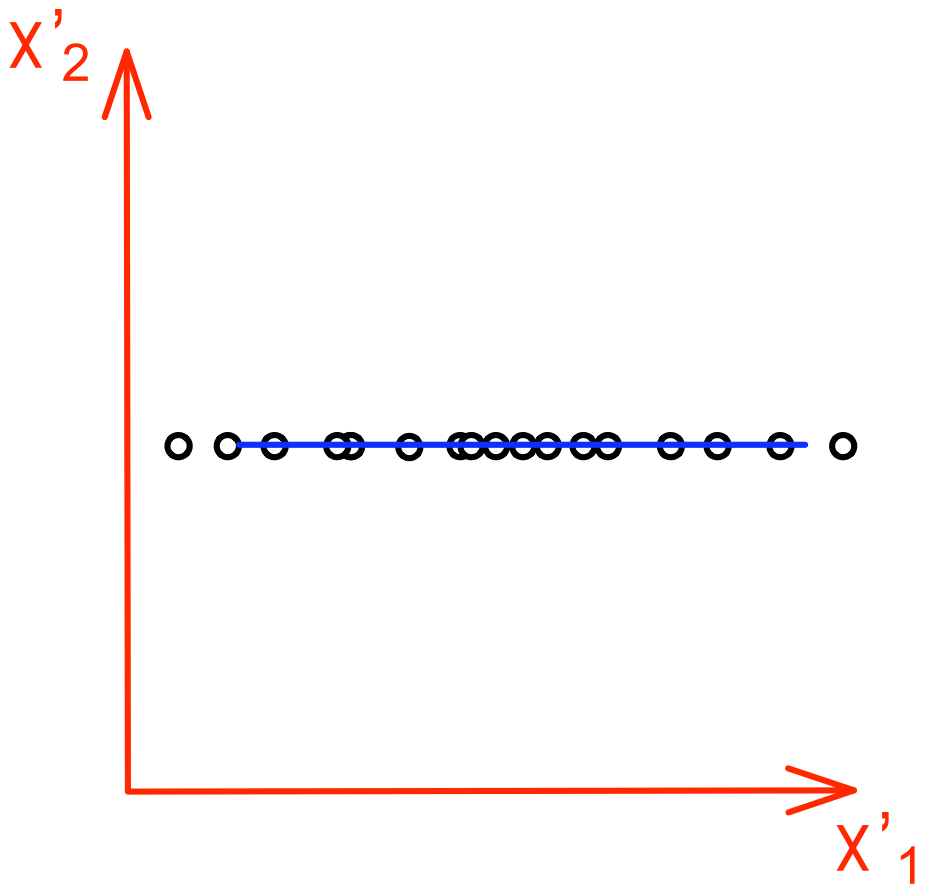
Rotate the space along the axis

Principal Component Analysis (PCA)



Dimensions are uncorrelated now

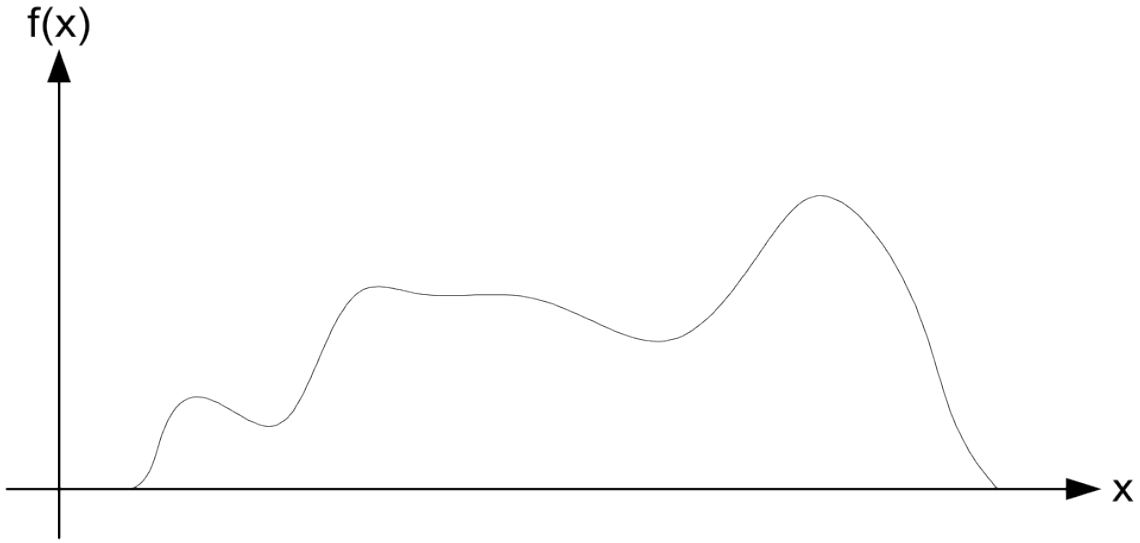
Principal Component Analysis (PCA)



Remove dimensions with low variance

=> Reduction of dimensionality with minimum loss of information

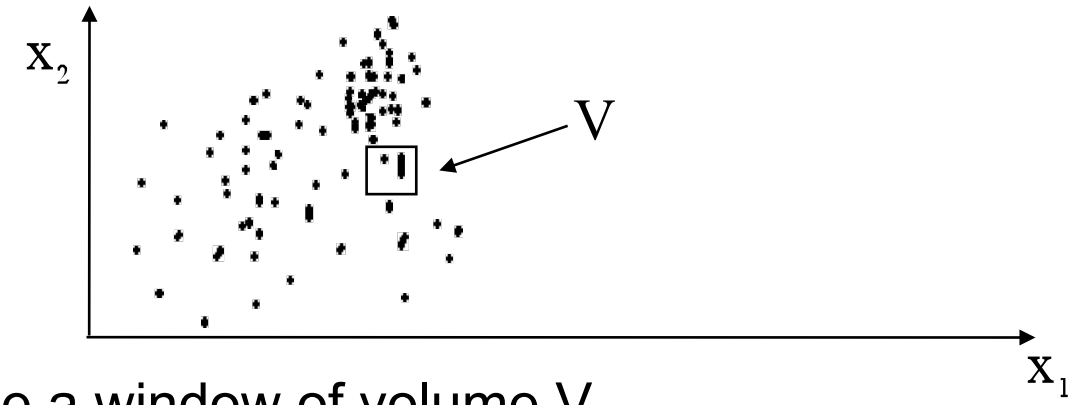
Problems



- Normal distribution does not model this situation well.
- other densities may be mathematically intractable.
  –>non-parametric techniques

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 -> loose resolution
  Volume too small -> erratic, poor estimate

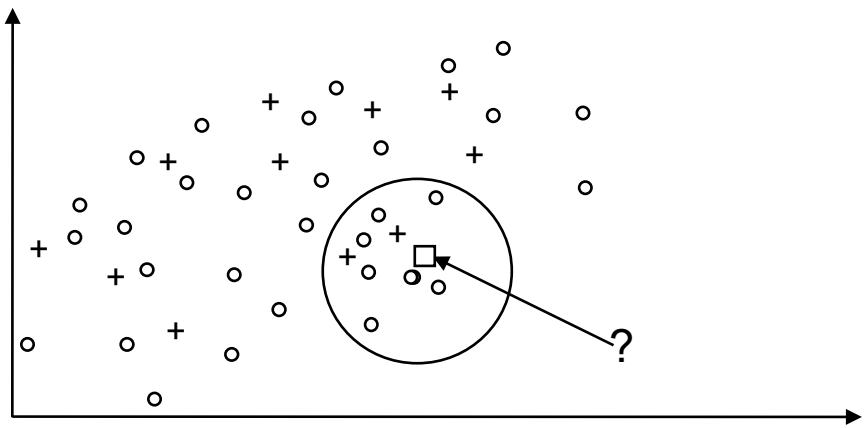
- set $V_n = \frac{1}{\sqrt{n}}$

$$\left.\begin{aligned} \lim_{n \to \infty} V_n &= 0 \\ \lim_{n \to \infty} k_n &= \infty \\ \lim_{n \to \infty} k_n / n &= 0 \end{aligned}\right\} p_{guess}(x) \to p(x)$$

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



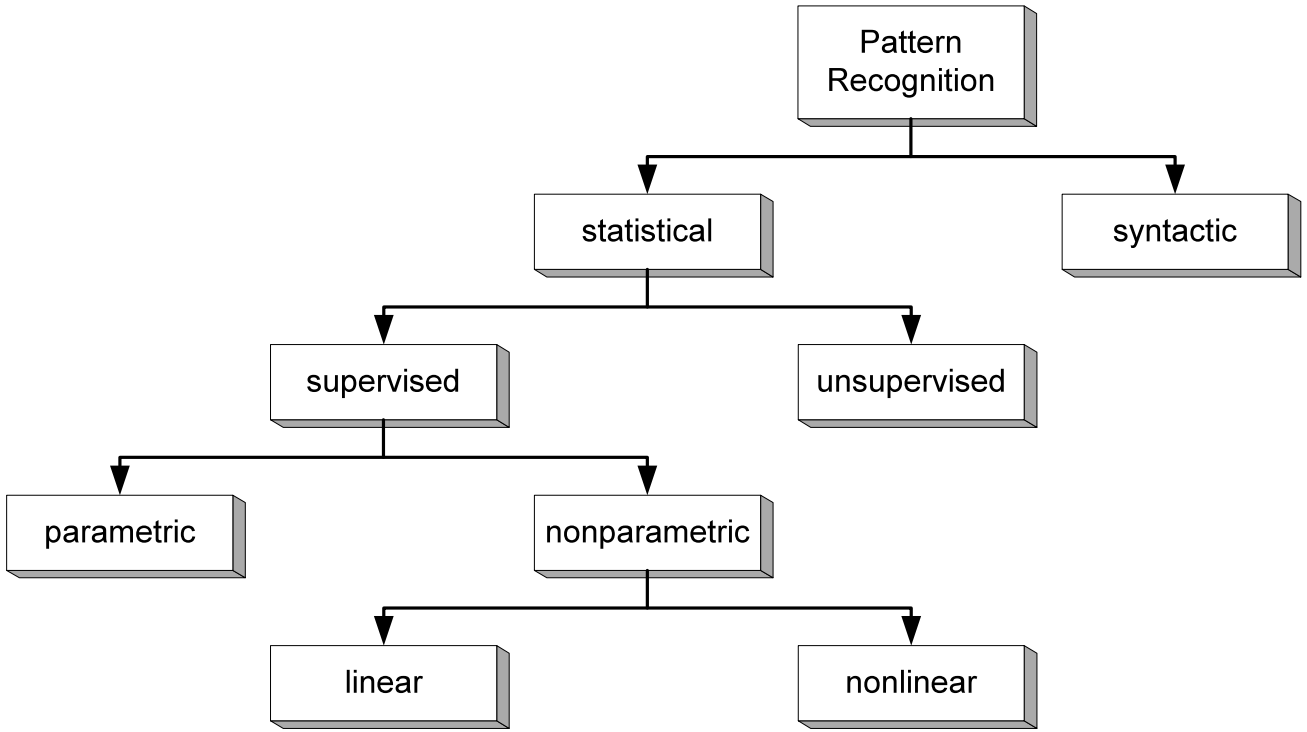
KNN-Classifier: Problem

- 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.

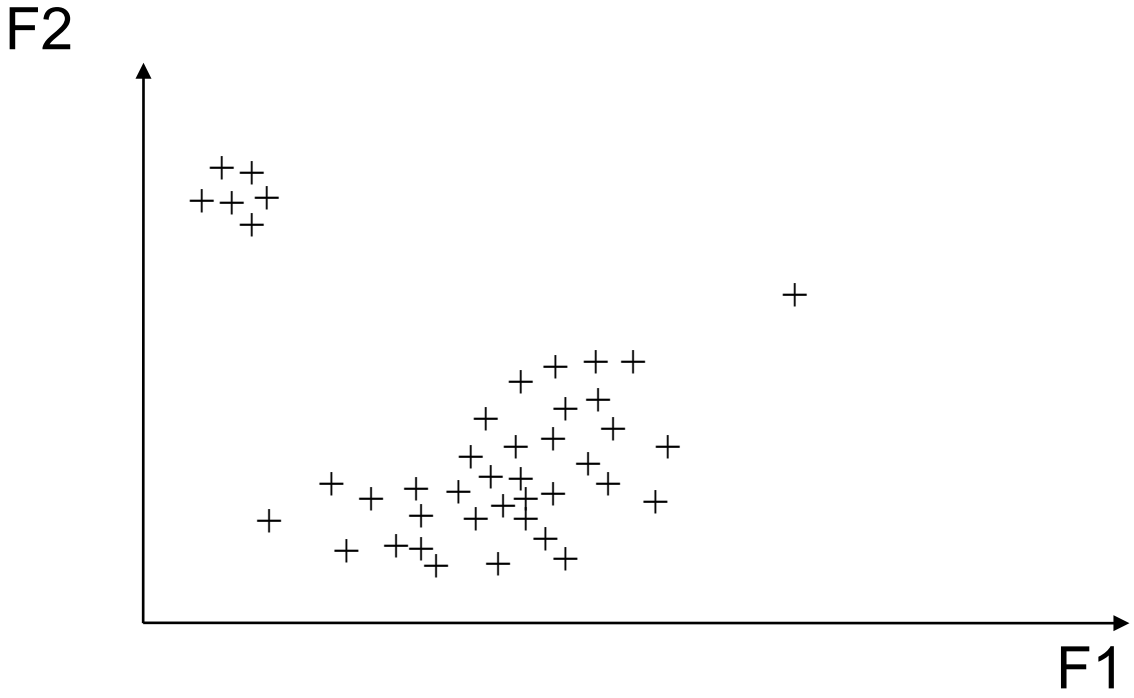
- Need training database to be larger.

  "There is no data like more data."

Pattern Recognition

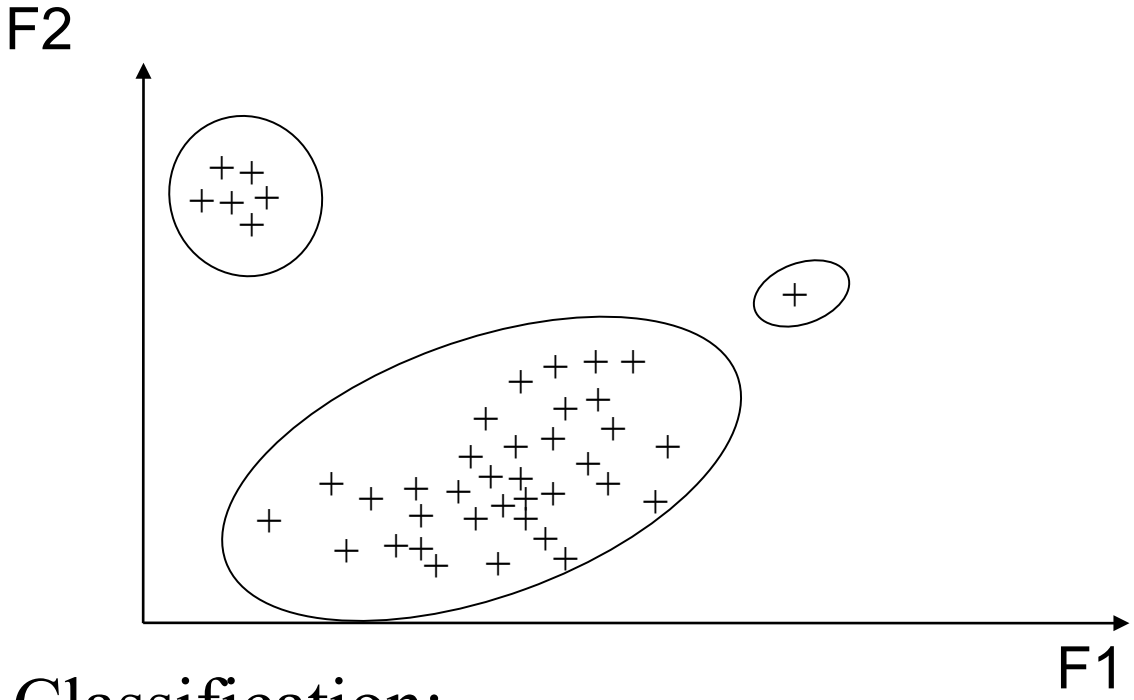


Unsupervised Classification



- Classification:
  - Classes Not Known: Find Structure

Unsupervised Classification



- Classification:
  - Classes Not Known: Find Structure
  - Clustering
  - How? How many?

Unsupervised Learning

- Data collection and labeling costly and time consuming
- Characteristics of patterns can change over time
- May not have insight into nature or structure of data

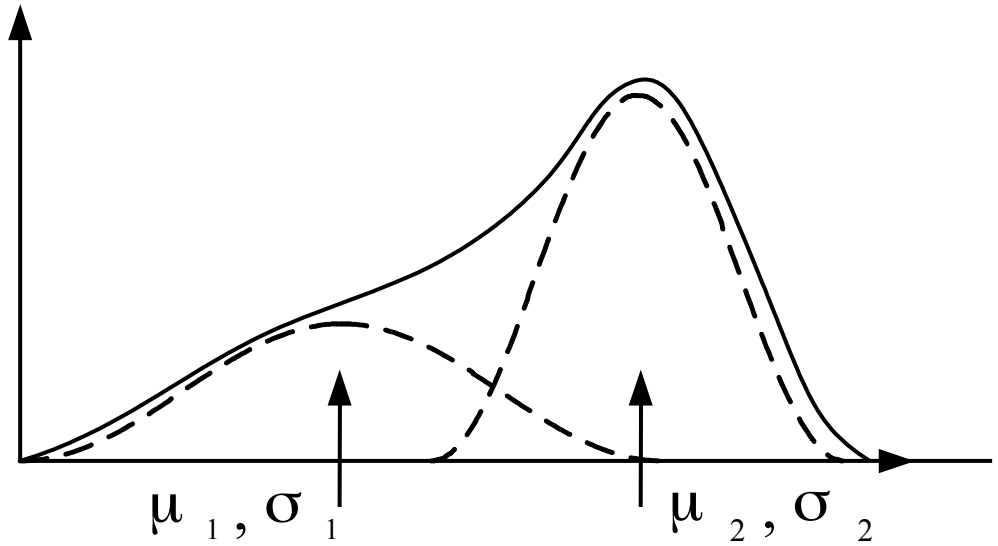
→ **Classes NOT known**

Mixture Densities

- Samples come from c classes
- A priori probability $P(\omega_j)$
- Assume: The forms for the class-conditional

PDF's $P(X / \omega_j, \Theta_j)$ are known (usually normal)

- Unknown parameter vector $\Theta_1 \ldots \Theta_c$



Mixture Densities

Problem: Hairy Math

Simplification/ Approximation:

Look only for means -> Isodata

1. Choose initial $\mu_1 \ldots \mu_c$
2. Classify n samples to closest mean
3. Recompute means from samples in class
4. Means changed? Goto step 2, else stop

Mixture Densities

Isodata, problems:

- Choosing initial means μ
- Knowing number of classes
- Assuming distribution
- What is "closest"?

Clustering

- Similarity
- Criterion function
- Samples in same class should extremize criterion function, that measures cluster quality
- Example: sum of error criterion

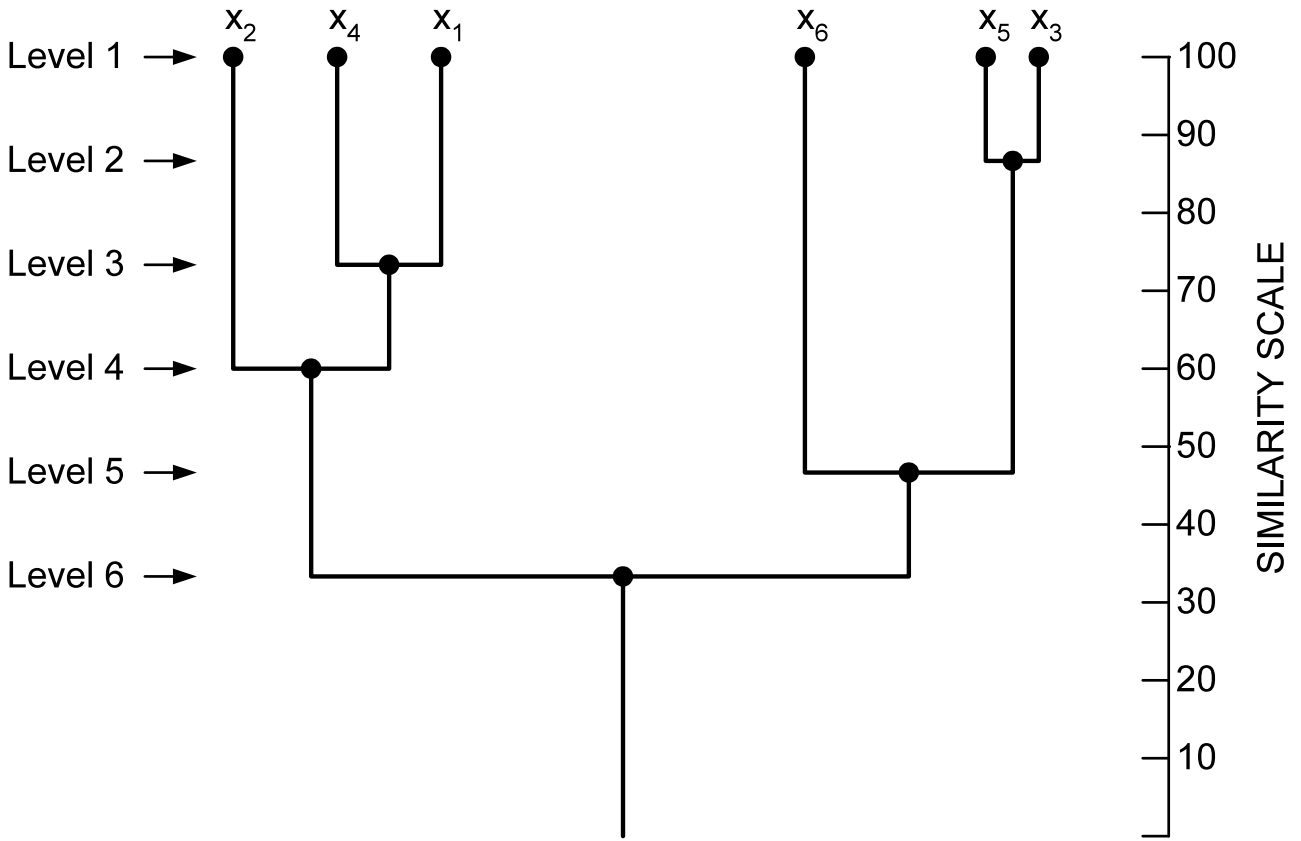
$$J = \sum_{i=1}^{c} \sum_{\max} ||x - m_i||^2$$

Hierarchical Clustering

- Need not determine c
- Need not guess initial means

1. Initialize c := n
2. Find nearest pair of distinct clusters X_i and X_j
3. Merge them and decrement c
4. If $c \leq c_{stop}$ stop, otherwise goto step 2

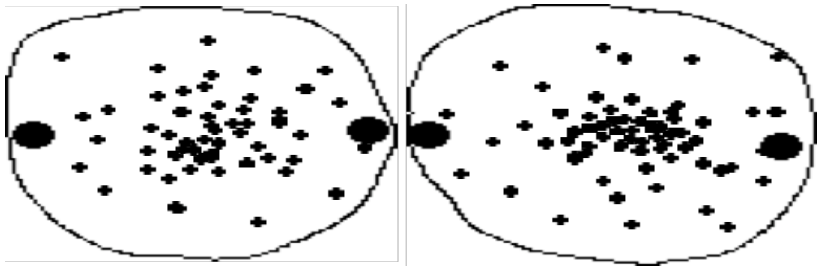
Dendrogram for hierarchical clustering



Similarity

What constitutes "nearest cluster"?

- D_{min}
- D_{ave}
- D_{max}



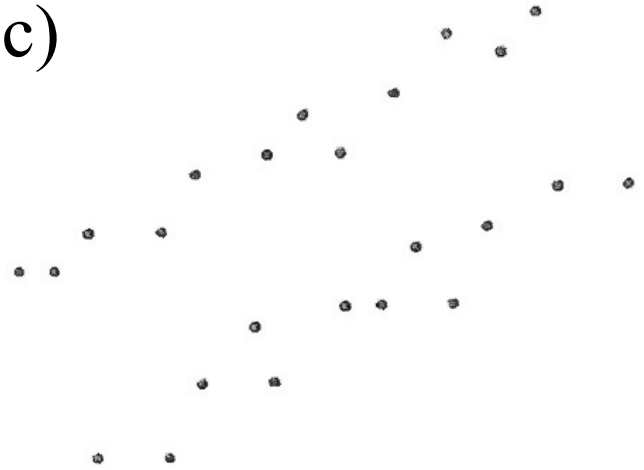
X_i X_j

Three illustrative examples

a)

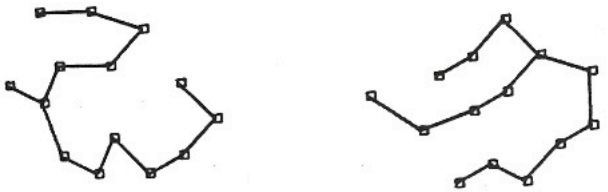
b)

c)



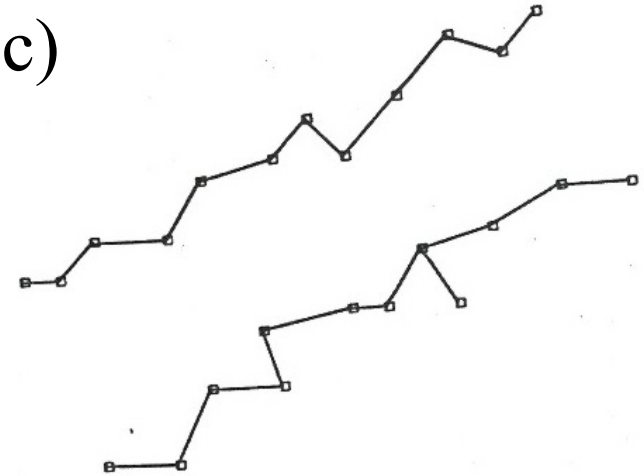
Results of the nearest-neighbor algorithm

a)



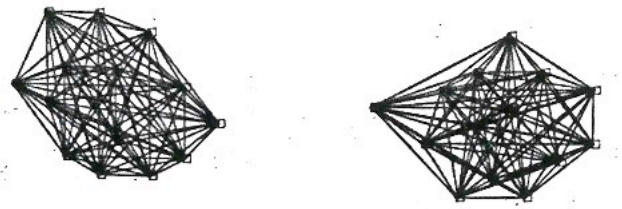
b)

c)

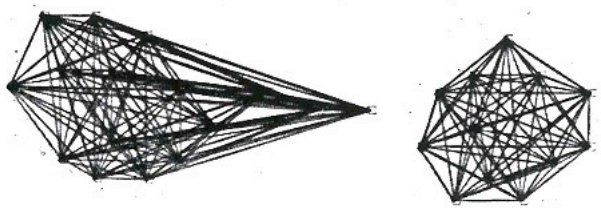


Results of the furthest-neighbor algorithm

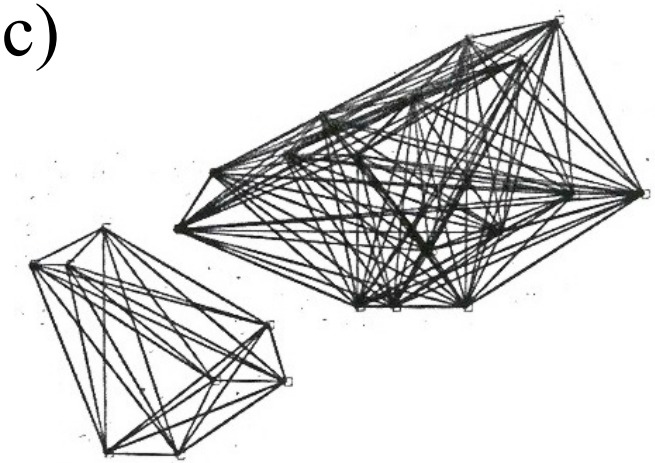
a)



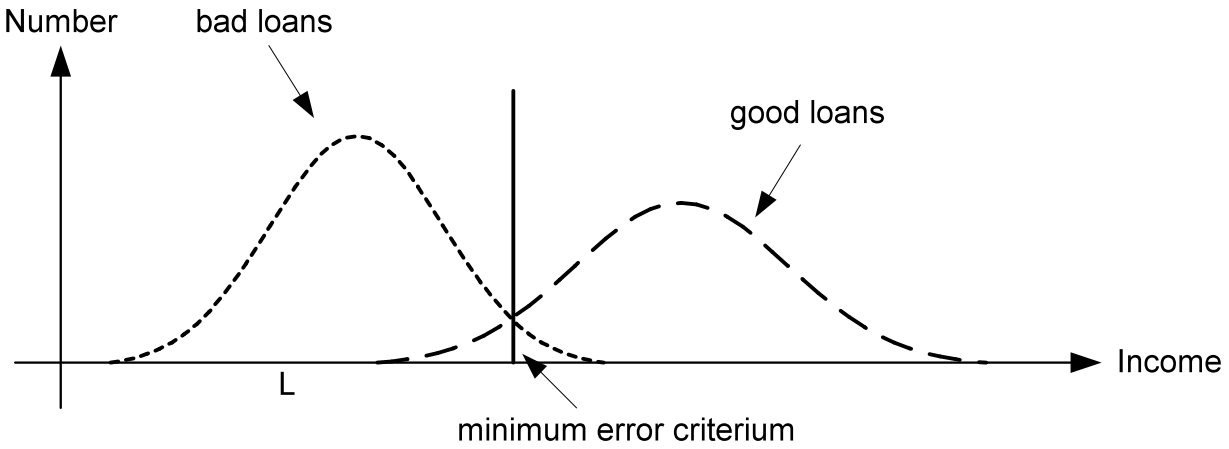
b)



c)



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, perceptron...

Decision Function g(x)

$g(\vec{x}) > 0 \quad \Rightarrow$ Class A

$g(\vec{x}) < 0 \quad \Rightarrow$ Not class A

$g(\vec{x}) = 0 \quad \Rightarrow$ No decision

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

$\vec{x} = (x_1, \ldots, x_n)^T$ Feature vector

$\vec{w} = (w_1, \ldots, w_n)^T$ Weight vector

w_0 Threshold weight

Linear Discriminant Functions

- No assumption about distributions (non-parametric)
- Linear Decision Surfaces
- Begin by supervised training (given classes of training data)
- Discriminant function:

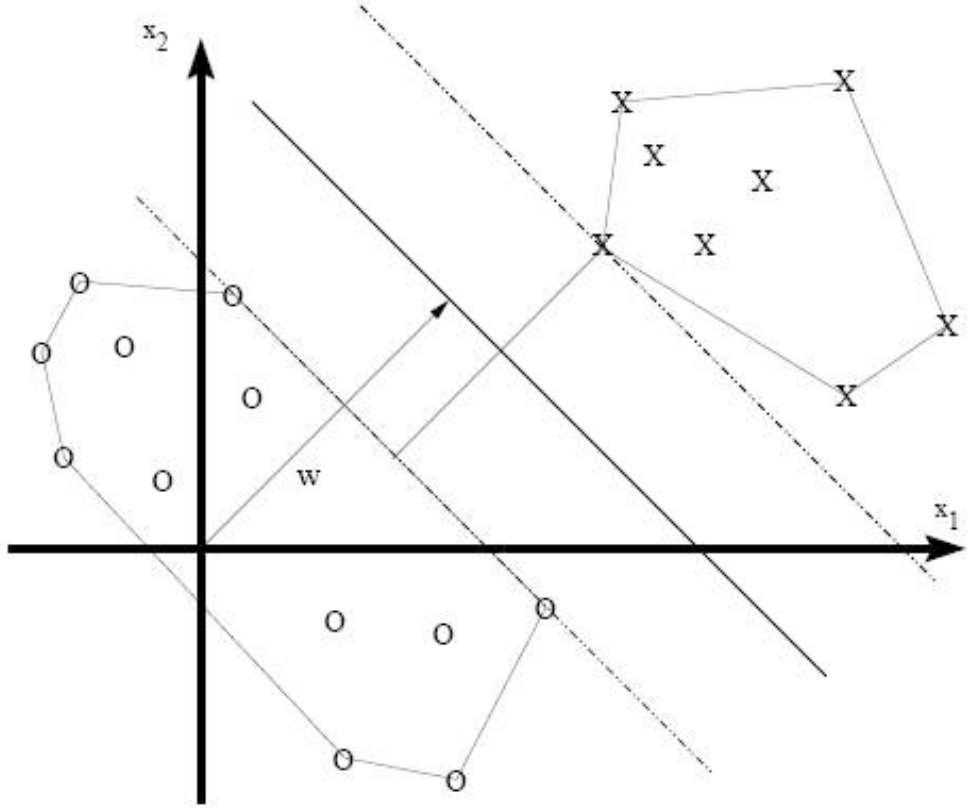
$$g(x) = w_0 + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i; \qquad x_0 = 1$$

- g(x) gives distance from decision surface
- Two category case:

$$g_1(x) > 0 \quad \Rightarrow \quad \text{class } 1$$

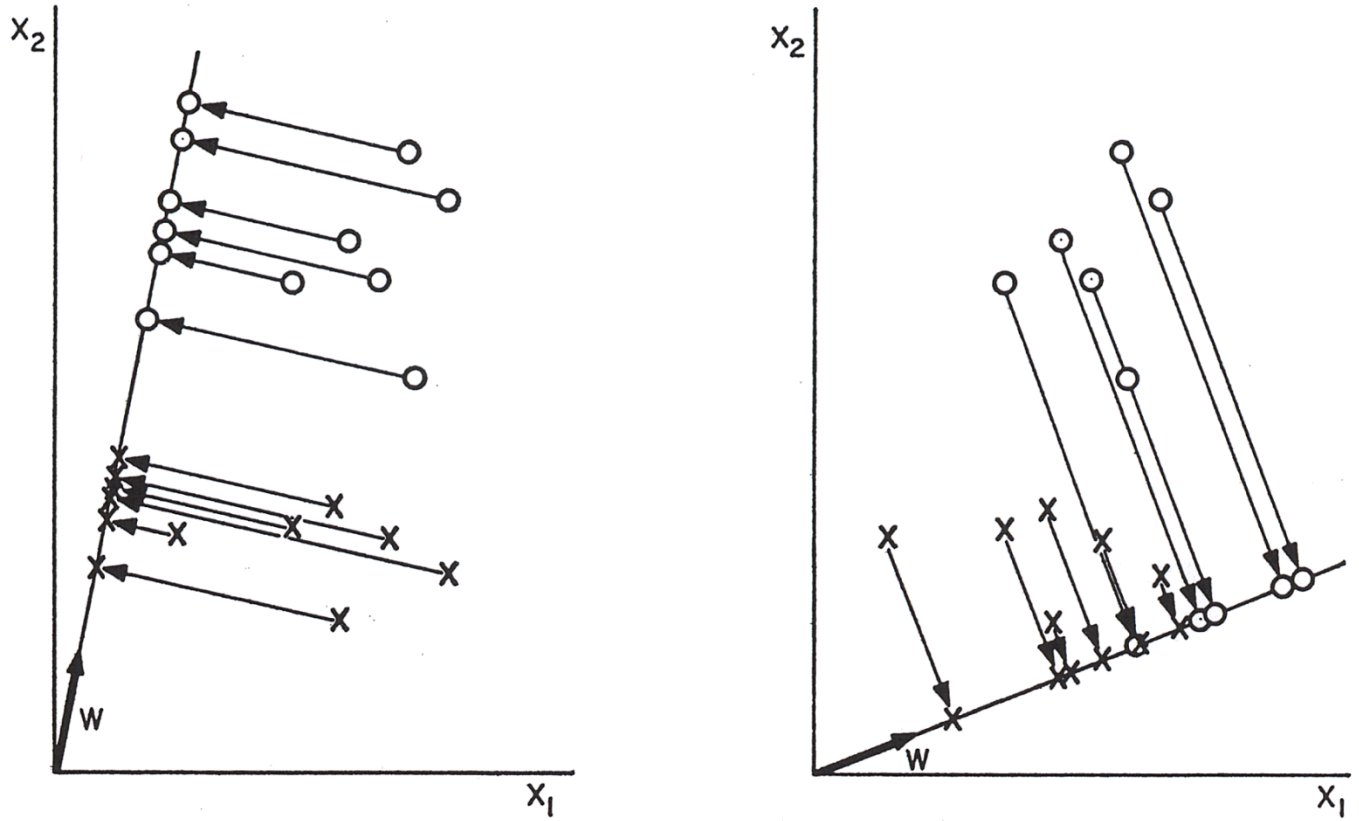
$$g_1(x) < 0 \quad \Rightarrow \quad \text{class } 2$$

Linear Separable



- Each class can be surounded by a convex polygon
- Maximum „safety“ area is half of the distance of the polygons

Discriminant Analysis



Goal: find an orientation of the line, were the projected samples are well separated.

Fisher-Linear Discriminant

- Dimensionality Reduction;
- Project a set of multidimensional points onto a line $y = \vec{w}\vec{x}$

- Fisher Discriminant is function that maximizes criterion

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

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

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

Fisher Linear Discriminant

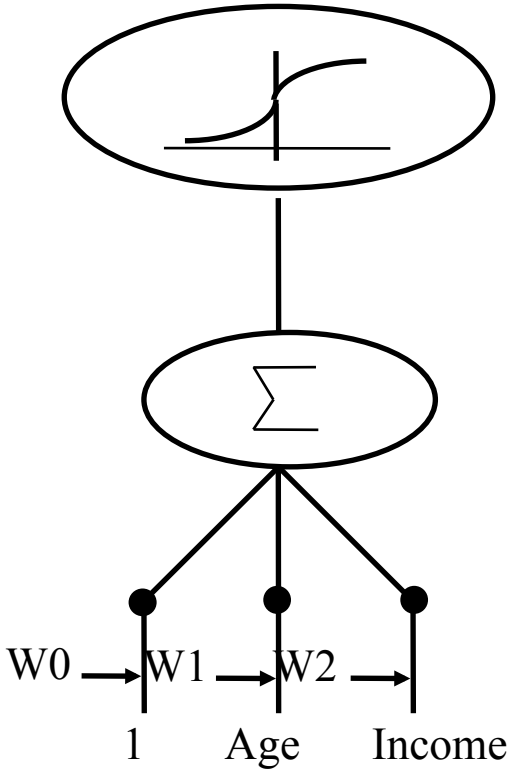
Fisher's linear discriminant:

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

$$s_w = s_1 + s_2 \qquad \text{(within-class scatter matrix)}$$

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

The Perceptron



Decision Function g(x)

$g(\vec{x}) > 0 \quad \Rightarrow$ Class A

$g(\vec{x}) < 0 \quad \Rightarrow$ Not class A

$g(\vec{x}) = 0 \quad \Rightarrow$ No decision

$$g(\vec{x}) = \sum_{i=1}^{n} w_1 x_1 + w_0 = \vec{w}^T \vec{x} + w_0 = \vec{w} \cdot \vec{x} + w_0$$

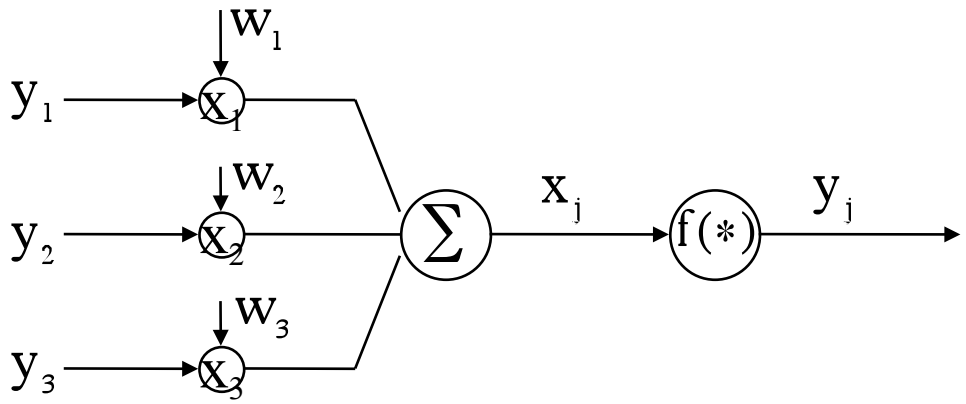
$\vec{x} = (x_1, \ldots, x_n)^T$ Feature vector

$\vec{w} = (w_1, \ldots, w_n)^T$ Weight vector

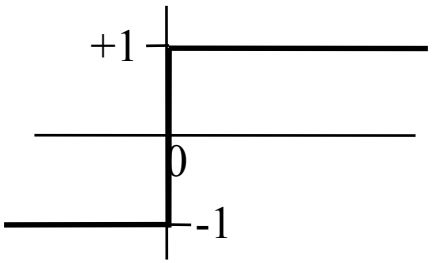
w_0 Threshold weight

$\cdot$ denotes scalar product

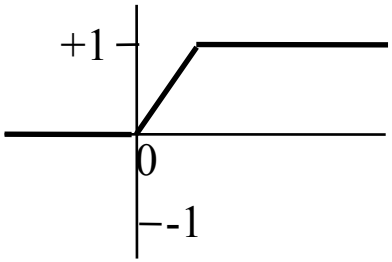
Perceptrons - Nonlinearities



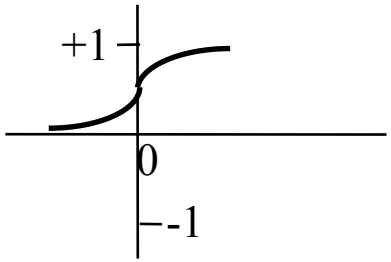
Three common non-linearities f(*):



Hard Limiter



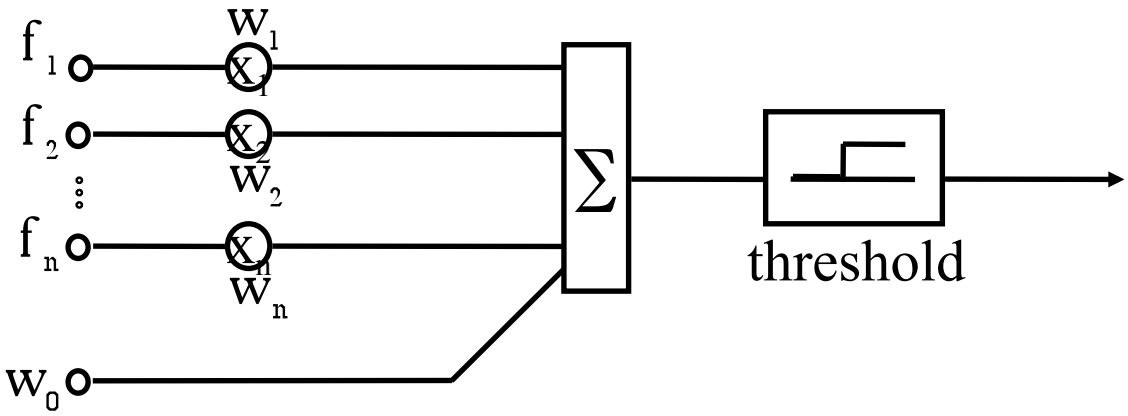
Threshold Logic



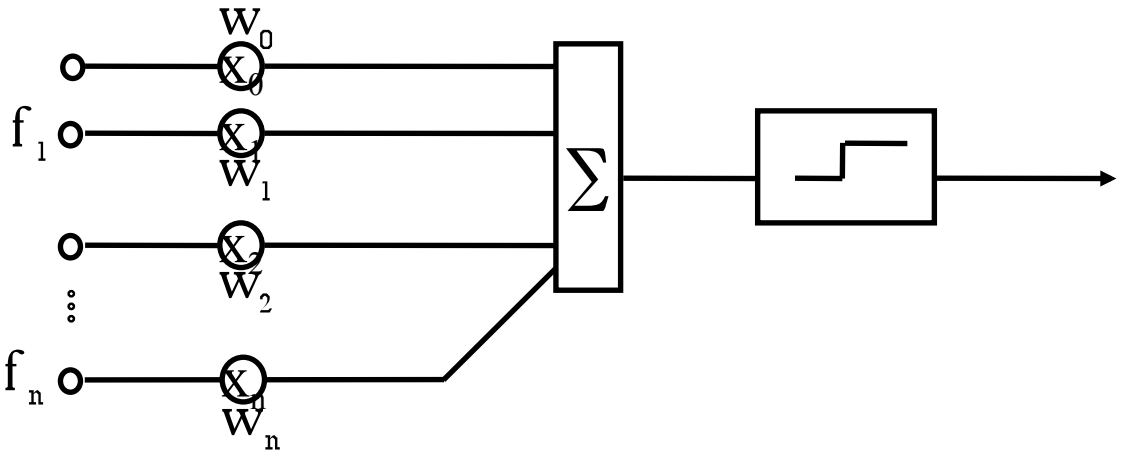
sigmoid $y_j = \frac{1}{1+e^{-x_j}}$

The Perceptron

Linear Decision Element:



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



$$g(x) = \sum_{i=0}^{n} w_i x_i ;$$

$$x_0 = 1$$

Classifier Discriminant Functions

$$g_i(x), i = 1, \ldots, c$$

Assign x to class ω_i, if

$$g_i(x) > g_j(x) \quad \text{for all } j \neq i$$

$$g_i(x) = P(\omega_i / x)$$

$$= \frac{p(x / \omega_i)P(\omega_i)}{\sum_{j=1}^{c} p(x / \omega_j)(\omega_j)} \leftarrow \text{independent of class i}$$

$$g_i(x) = p(x / \omega_i)P(\omega_i)$$

$$g_i(x) = \log(p(x / \omega_i)) + \log(P(\omega_i))$$

class conditional probability density function

A priori probability

Linear Discriminant Functions

- No assumption about distributions (non-parametric)
- Linear Decision Surfaces
- Begin by supervised training (given classes of training data)
- Discriminant function:

$$g(x) = w_0 + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i; \qquad x_0 = 1$$

- g(x) gives distance from decision surface
- Two category case:

$$g_1(x) > 0 \quad \Rightarrow \quad \text{class 1}$$

$$g_1(x) < 0 \quad \Rightarrow \quad \text{class 2}$$

Perceptron

$$g(x) = \sum_{i=0}^{n} w_i x_i ; \quad x_0 = 1$$

find $\vec{w}$

All vectors $\vec{x}_j$ are labelled correctly, if for all j

$\vec{w} \cdot \vec{x}_j > 0$ $\vec{x}_j$ if labelled ω_1

$\vec{w} \cdot \vec{x}_j < 0$ $\vec{x}_j$ if labelled ω_2

Now we set all samples belonging to ω_2 to their negative $(-\vec{x}_j)$.

Then all vectors are classified correctly, if $\vec{w} \cdot \vec{x}_j > 0$ for all j

Perceptron Criterion Function

$$J_p(\vec{w}) = \sum_{\vec{x} \in X} (-\vec{w} \cdot \vec{x})$$

X is the set of misclassified tokens

Since $\vec{w} \cdot \vec{x}$ is negation for misclassified tokens, $J_p(\vec{w})$ is positive

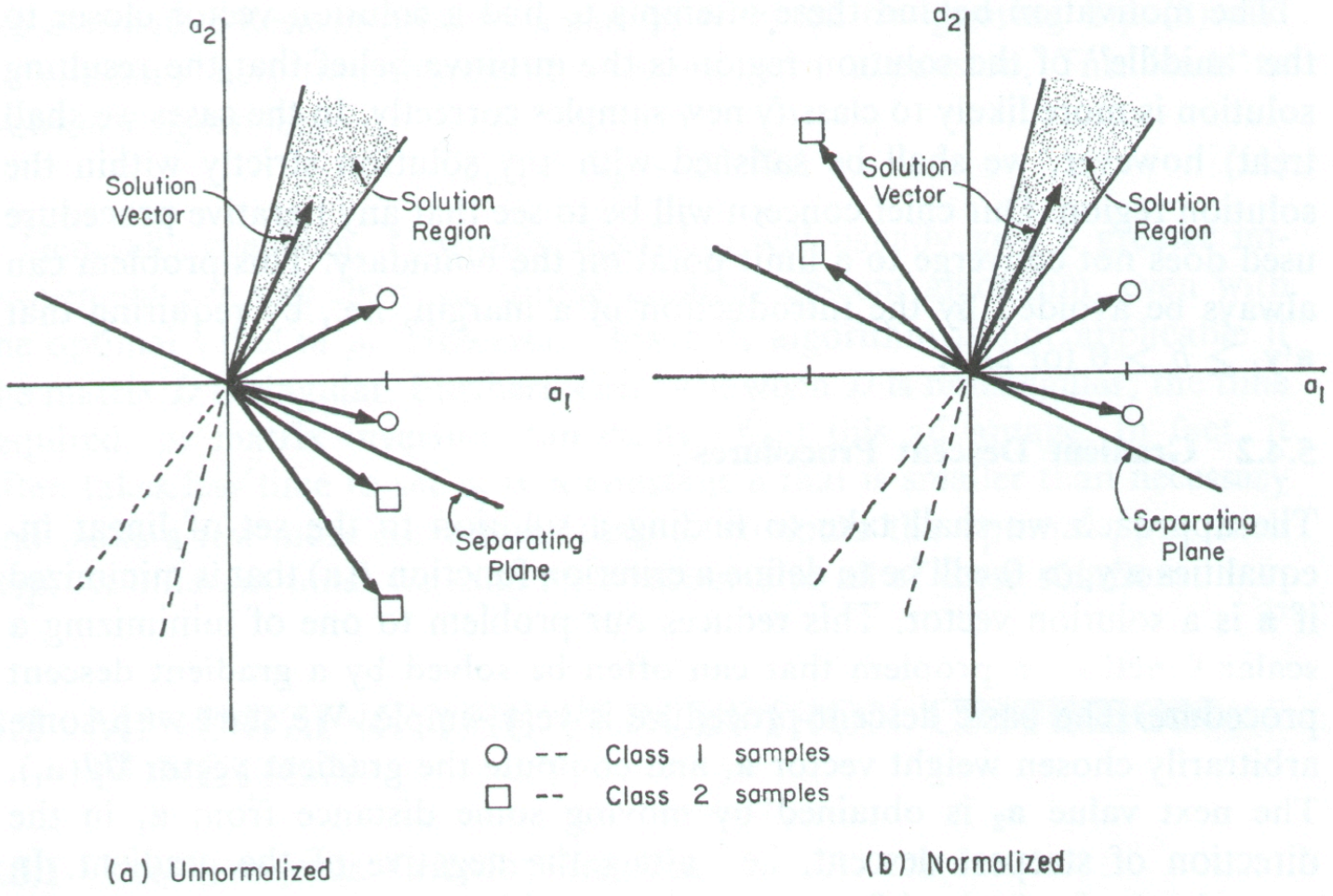
When J_p is zero a solution vector $\vec{w}$ is found.

J_p is proportional to the sum of distances of misclassified samples to decision boundary

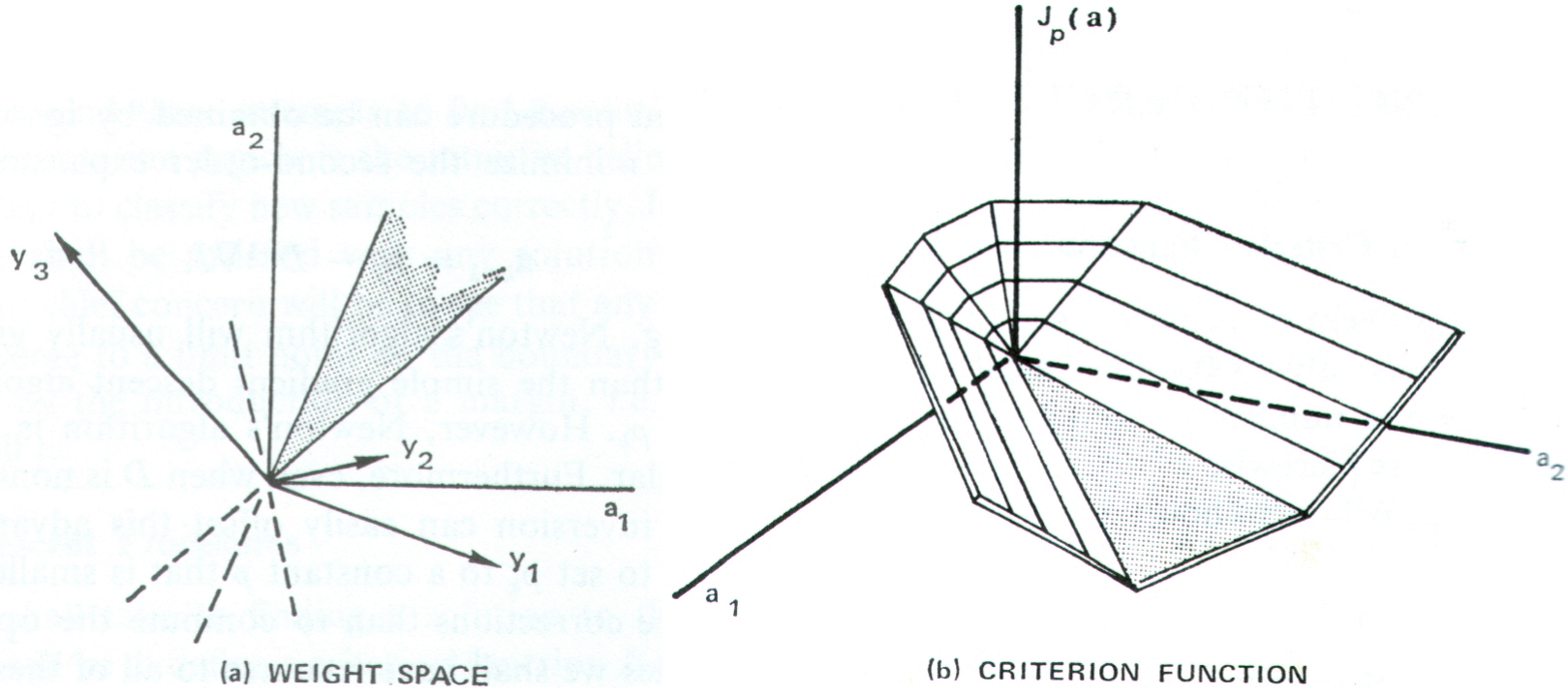
$$\nabla J_p = \sum_{\vec{x} \in X} (-\vec{x})$$

$$\vec{w}_{k+1} = \vec{w}_k + \varsigma_k \sum_{\vec{x} \in X} \vec{x}$$

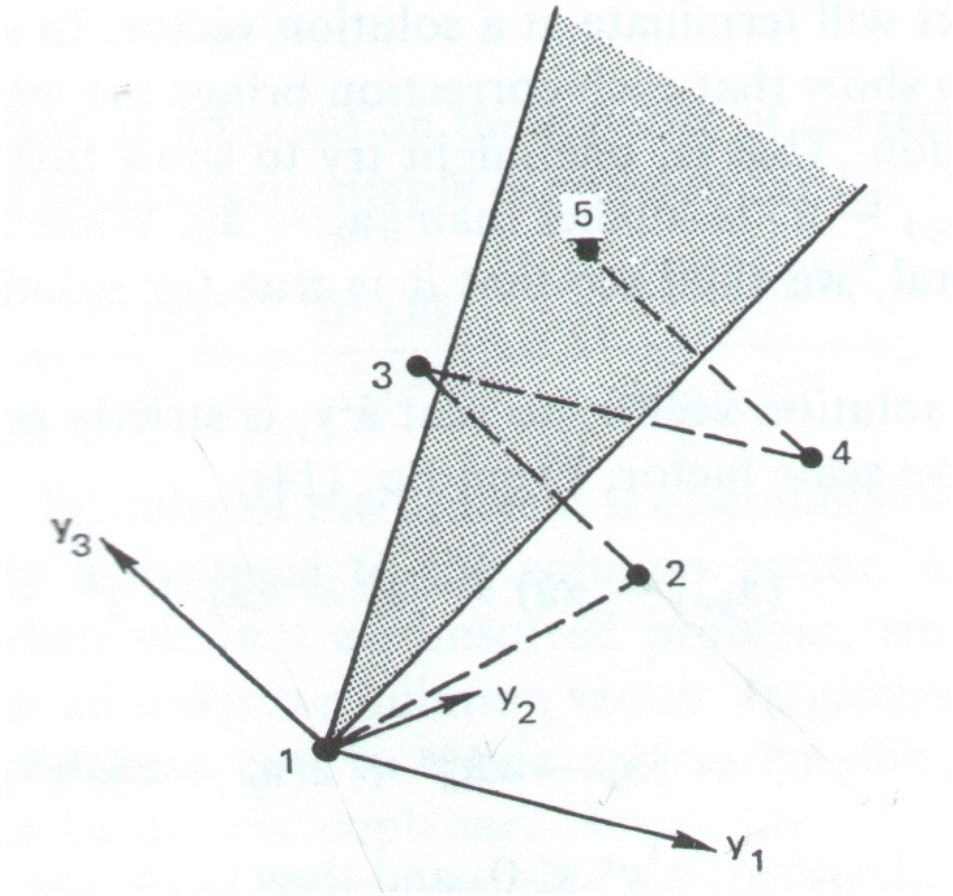
Linearly Separable Samples and the Solution Region in Weight Space



The Perceptron Criterion Function



Finding a Solution Region by Gradient Search



Perceptron Learning

- Issues:
  - How to set the Learning Rate
  - How to set initial weights
- Problems:
  - Non-Separable Data
  - Separable Data, but Which Decision Surface
  - Non-Linearly Separable

Variations

Relaxation Procedure

$$J_q(\vec{w}) = \sum_{\vec{x} \in X} (\vec{w} \cdot \vec{x})^2$$

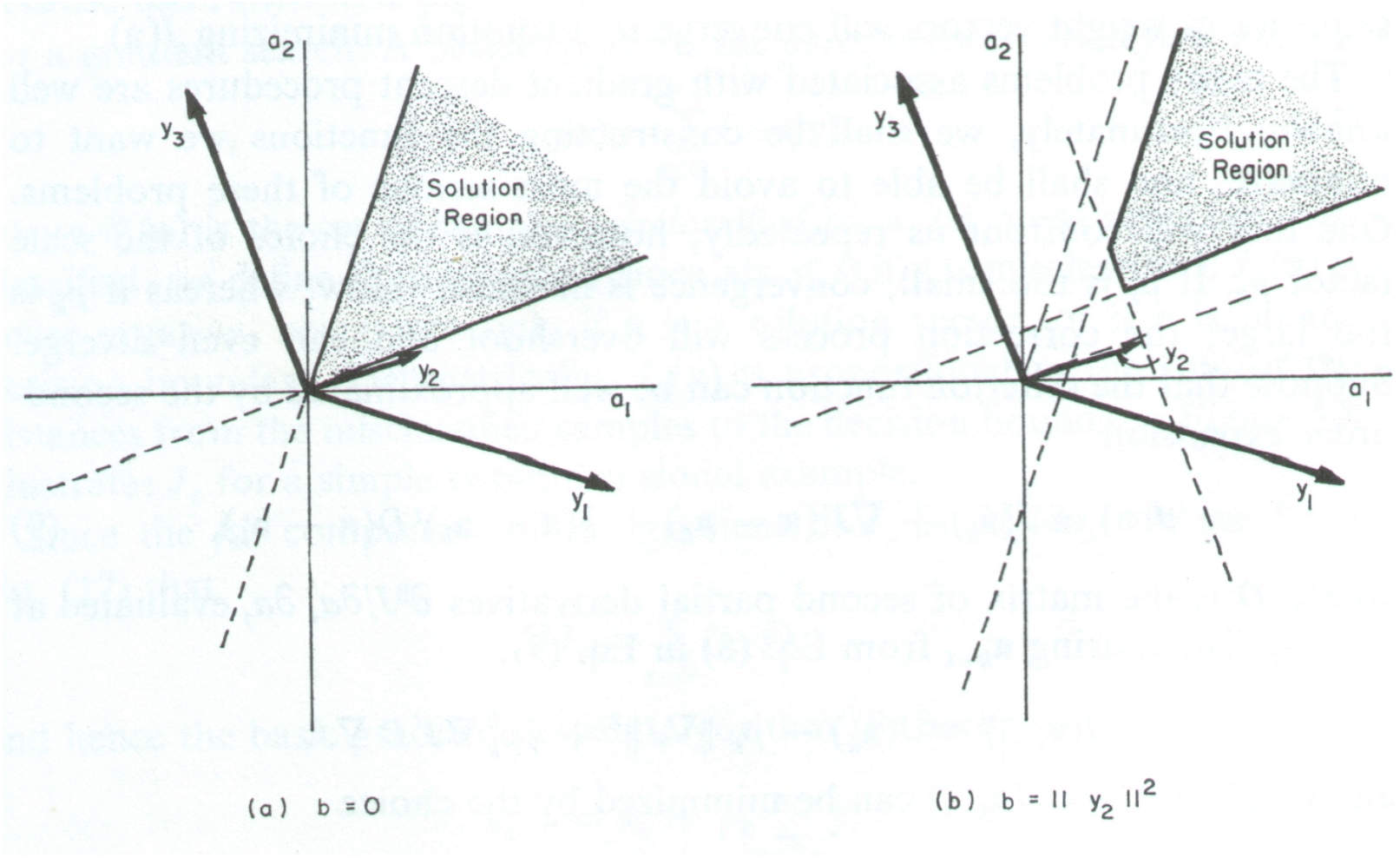
Gradient is continuous -> smoother surface

Margin

$$J_r(\vec{w}) = \frac{1}{2} \sum_{\vec{x} \in X} \frac{(\vec{x} \cdot \vec{w} - b)^2}{\|\vec{x}\|^2}$$

margin b

Effect of the Margin on the Solution Region



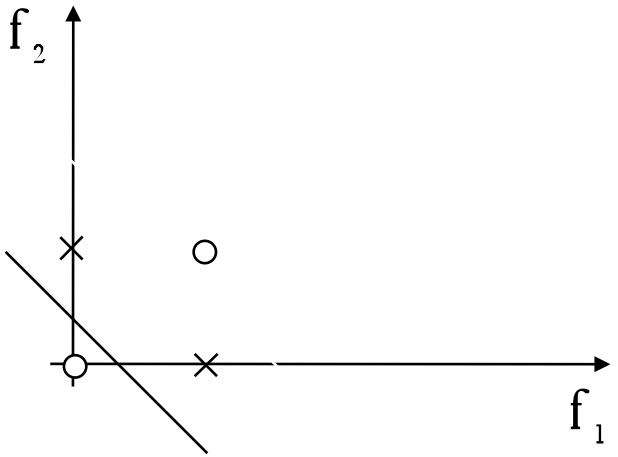
Nonseparable behavior

- Minimize MSE

$$J_s(\vec{w}) = \sum_{\vec{x}_j \in X} (\vec{w} \cdot \vec{x}_j - b)^2$$

The Perceptron:

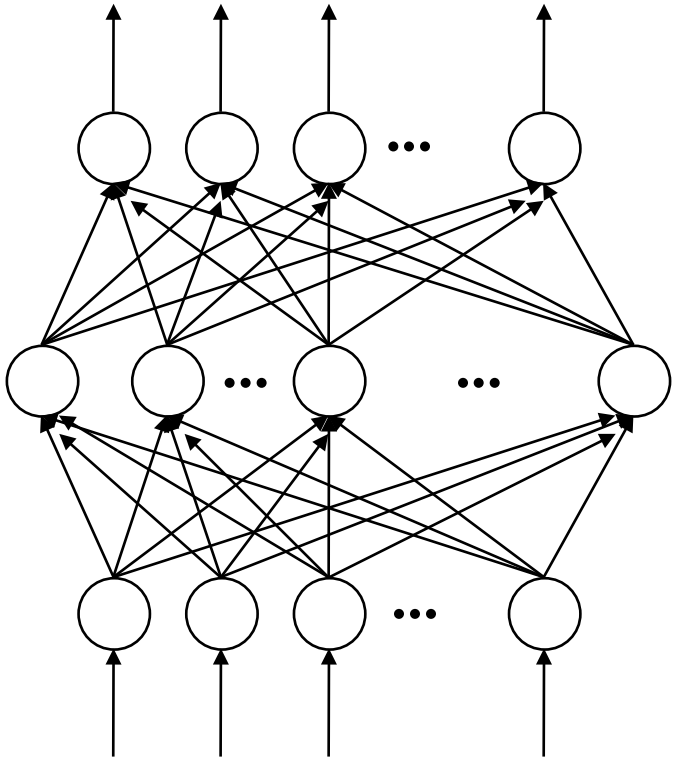
- The good news:
  - Simple computing units
  - Learning algorithm
- The bad news:
  - Single units generate linear decision surfaces (The infamous XOR problem).



  - Multilayered machines could not be learned. Local optimization.

Networks of Neurons/ Multi-Layer Perceptron

- Many interconnected simple processing elements:

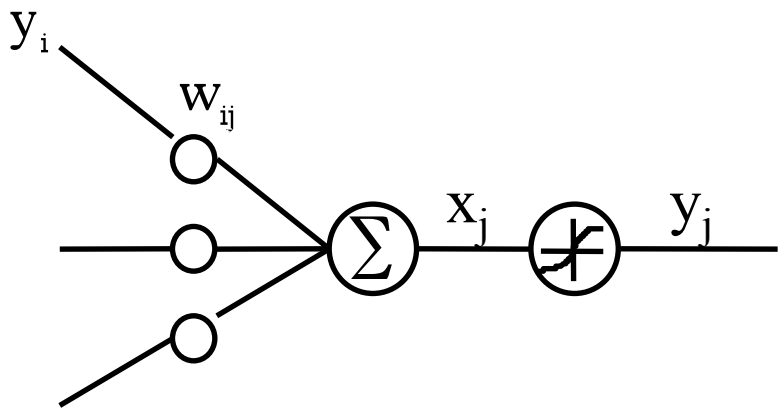


Output patterns

Internal Representation Units

Input Patterns

Connectionist Units



$$x_j = \sum_i y_i w_{ij}$$

$$y_j = \frac{1}{1 + e^{-x_j}}$$

Backpropagation of error:

$$E = \frac{1}{2} \sum_j (y_j - d_j)^2$$

Training the MLP by Error Back-Propagation

- Choose random initial weights
- Apply input, get some output
- Compare output to *desired* output and compute error
- Back-propagate error through net and compute $\partial E / \partial w_{ij}$, the contribution of each weight to the overall error.
- Adjust weights slightly to reduce error.

Backpropagation of Error

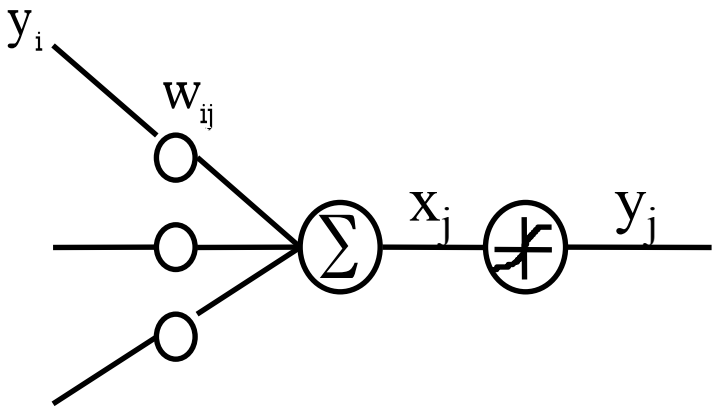
$$E = \frac{1}{2}\sum_j (y_j - d_j)^2 \qquad y_j = \frac{1}{1+e^{-x_j}} \qquad x_j = \sum_i y_i w_{ij}$$

1). $$\frac{\partial E}{\partial y_j} = y_j - d_j$$

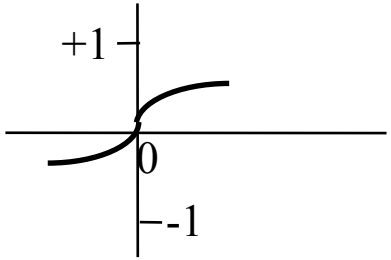
2). $$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} = \frac{\partial E}{\partial y_j} y_j [1 - y_j]$$

3). $$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} \cdot y_i$$

4). $$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \cdot w_{ij}$$



Derivative dy/dx



$$\text{sigmoid } y_j = \frac{1}{1+e^{-x_j}}$$

$$\frac{\partial y_j}{\partial x_j} = (-1)(1+e^{-x_j})^{-2} \cdot (-1)e^{-x_j}$$

$$= \frac{e^{-x_j} \cdot 1}{(1+e^{-x_j}) \cdot (1+e^{-x_j})}$$

$$= \frac{e^{-x_j}}{1+e^{-x_j}} \cdot y_j$$

$$= y_j(1-y_j)$$

Statistical Interpretation of MLP's

- What is the output of an MLP?
- Output represents a Posteriori Probabilities P(w|x)
- Assumptions:
  - Training Targets: 1, 0
  - Output Error Function: Mean Squared Error
  - Other Functions Possible

Statistical Interpretation of MLP's

- What is the output of an MLP?
- It can be shown, that the output units represent the a posteriori probability
  - Provided certain objective functions (e.g. MSE, ..)
  - Sufficiently large training database

What does the sigmoid do?

$$y_j = \frac{1}{1 + e^{-x_j}}$$

$$e^{-x_j} = \frac{1}{y_j} - 1$$

$$e^{x_j} = \frac{y_j}{1 - y_j}$$

$$x_j = \log\left[\frac{y_j}{1 - y_j}\right]$$

What does the sigmoid do?

If $y_j = p(q / x)$ **then**

$$x_j = \log\left[\frac{p(q / x)}{p(\bar{q} / x)}\right]$$

$$= \underbrace{\log\left[\frac{p(x / q)}{p(x / \bar{q})}\right]}_{\text{inputs}} + \underbrace{\log\left[\frac{p(q)}{p(\bar{q})}\right]}_{\text{Bias}}$$

log. likelihood ratio or "odds"

Statistical Interpretation of MLP's

- What is the output of an MLP?
- Using MSE, for two class problem, $c_1 \triangleq 1,\ c_2 \triangleq 0$

$$E = \frac{1}{N}\left[\sum_{x \in c_1}[f(x)-1]^2 + \sum_{x \in c_2}[f(x)]^2\right]$$

if N large and reflects prior distribution

$$E \approx \int (f(x)-1)^2\, p(x, c_1)dx + \int (f(x))^2\, p(x, c_2)dx$$

Statistical Interpretation of MLP's

$$E = \int f^2[p(x_1 c_1) + p(x_1 c_2)]dx - 2\int f(x)p(x_1 c_1)dx + 1 \cdot \int p(x_1 c_1)dx$$

$$= \int f^2(x)p(x)dx - 2\int f(x)p(x_1 c_1)dx + \int p(x_1 c_1)dx$$

$$= \int [f^2(x)p(x) - 2f(x)p(x_1 c_1) + p(x_1 c_1)]dx$$

$$= \int [f^2(x) - 2f(x)p(c_1 / x) + p(c_1 / x)]p(x)dx$$

$$E = \int \underbrace{[f(x) - p(c_1 / x)]^2}_{\text{net approx. posterior}} p(x)dx - \int p^2(c_1 / x)p(x)dx + P(c_1)$$