Machine Learning for Signal Processing

Presentation · February 2016

DOI: 10.13140/RG.2.1.1399.4641

CITATIONS
0

READS
1,957

1 author:

Enrique V. Carrera
Universidad de las Fuerzas Armadas-ESPE
113 PUBLICATIONS **2,268** CITATIONS

SEE PROFILE

Machine Learning for Signal Processing

Enrique V. Carrera

Department of Electrical Engineering
Ecuadorian Armed Forces University

2016

Objectives

- This course introduces mathematical and empirical background required for analysis and processing of signals through machine learning techniques
- At the end of the course, the student should be able to:
  - Understand the basic concepts associated to machine learning
  - Characterize the main machine learning algorithms used for signal processing
  - Propose alternatives for developing signal processing and machine learning applications

Outline

Bibliography

- S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th Ed., Academic Press, ISBN 978-1-59749-272-0, 2009
- S. Theodoridis and K. Koutroumbas, *Introduction to Pattern Recognition: A Matlab Approach*, Academic Press, ISBN 978-0-12-374486-9, 2010
- K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, The MIT Press, ISBN 978-0-262-01802-9, 2012
- F. Hu and Q. Hao, *Intelligent Sensor Networks: The Integration of Sensor Networks, Signal Processing and Machine Learning*, CRC Press, ISBN 978-1-4398-9282-4, 2013

Grading

Labs	$\rightarrow$	35%
Exam	$\rightarrow$	15%
Project	$\rightarrow$	50%

More information at `http://vinicio.url.ph/MLSP/`

Outline

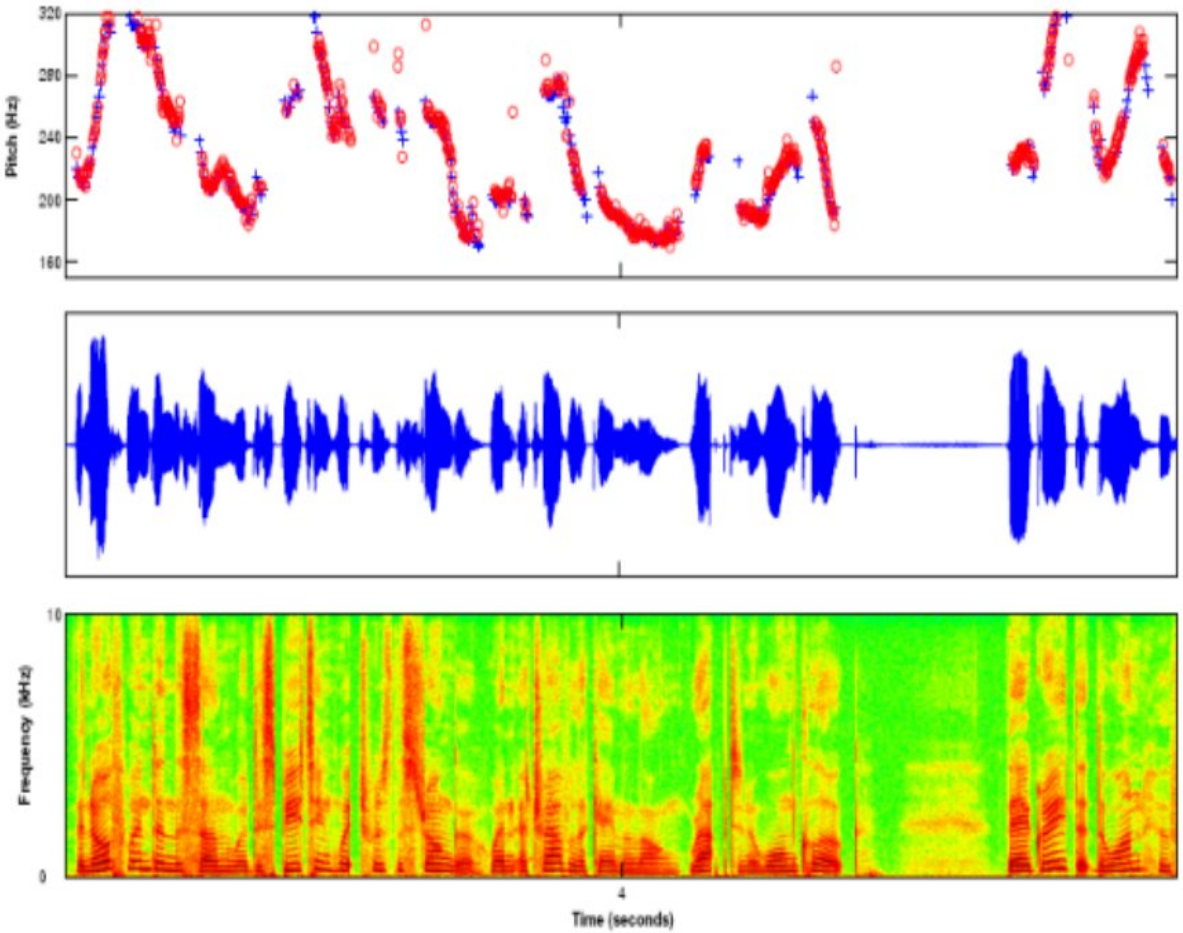
Introduction

- Nowadays, various sensors are used to collect the data of our environment
- A typical sensor converts the physical energy of the object under examination into electrical signals to deduce the information of interest
- The whole sensing procedure can be performed at three levels:
  - Data level: Each sample represents a measure of target energy within a certain temporal-spatial volume
  - Information level: The information acquired by a sensor is represented by a probabilistic belief over random variables
  - Knowledge level: The knowledge acquired by a sensor is represented by a statistical model describing relations among random variables

Introduction

- The goal of machine learning is to design and develop algorithms that allow systems to use empirical data, experience, and training to evolve and adapt to changes that occur in their environment
- A major focus of machine learning research is to automatically induce models, such as rules and patterns, from the training data it analyzes
- A number of machine learning algorithms have been employed in a wide range of sensor network applications, including activity recognition, health care, education, security, fault and event detection, and for improving the efficiency of complex systems

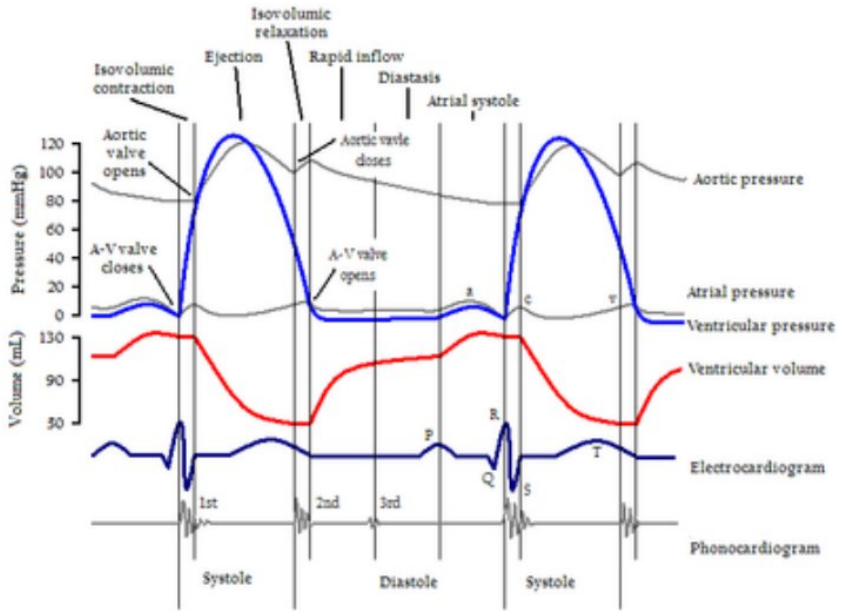
Application Examples



Speech recognition

Application Examples

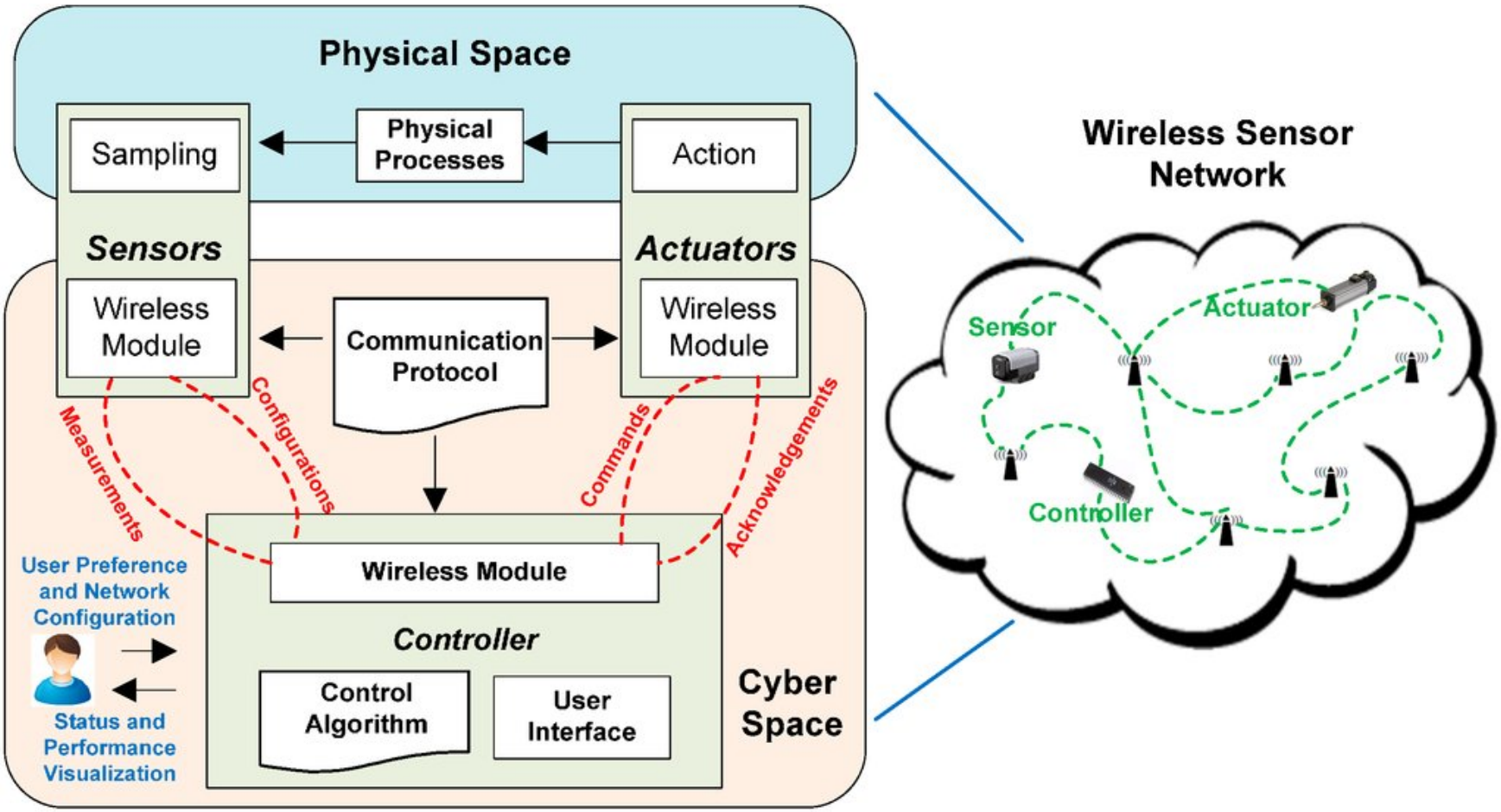
- Biomedical signal processing



Heartbeat processing

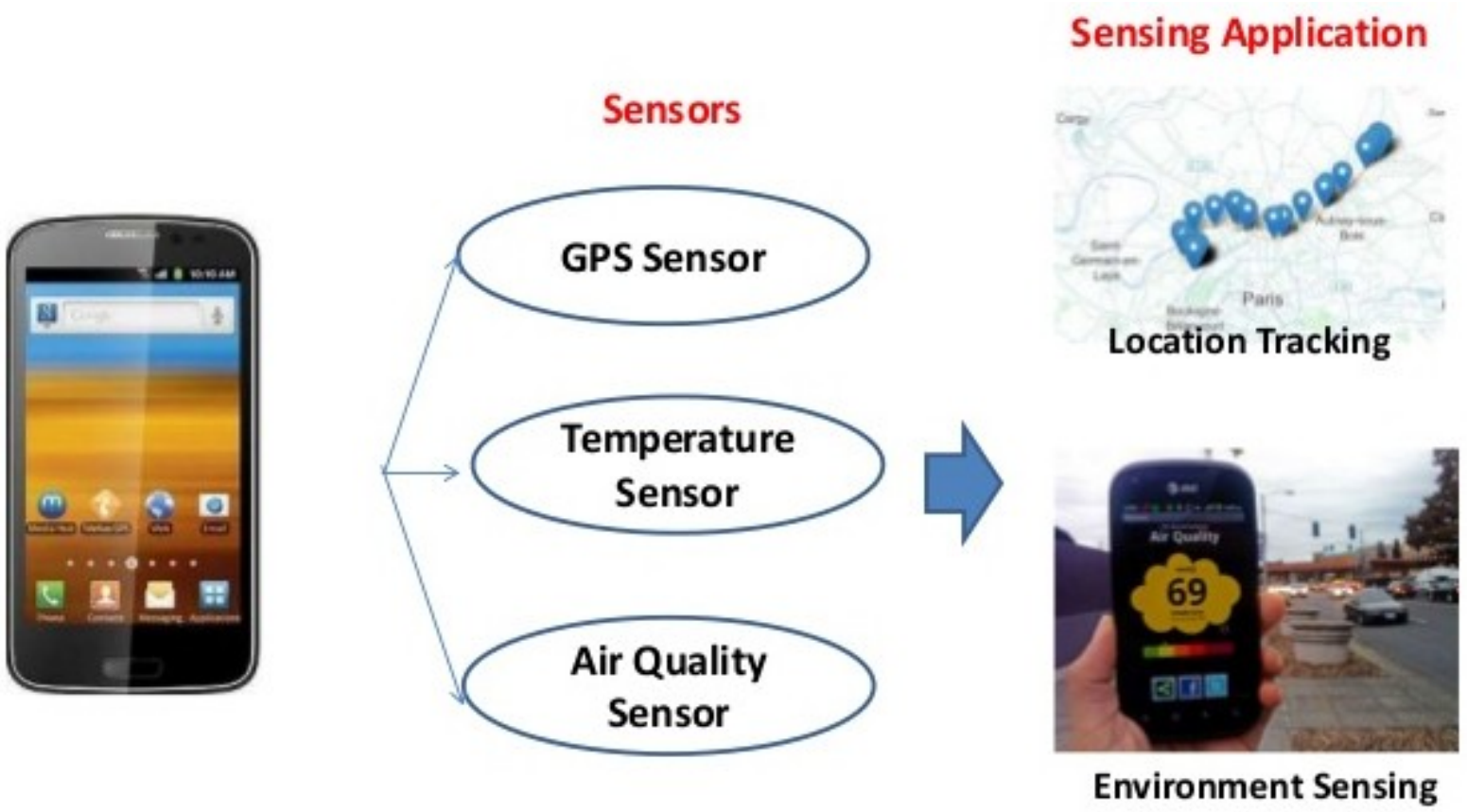
Brain-machine interfaces
BMI

Application Examples



Cyber-physical systems – CPS

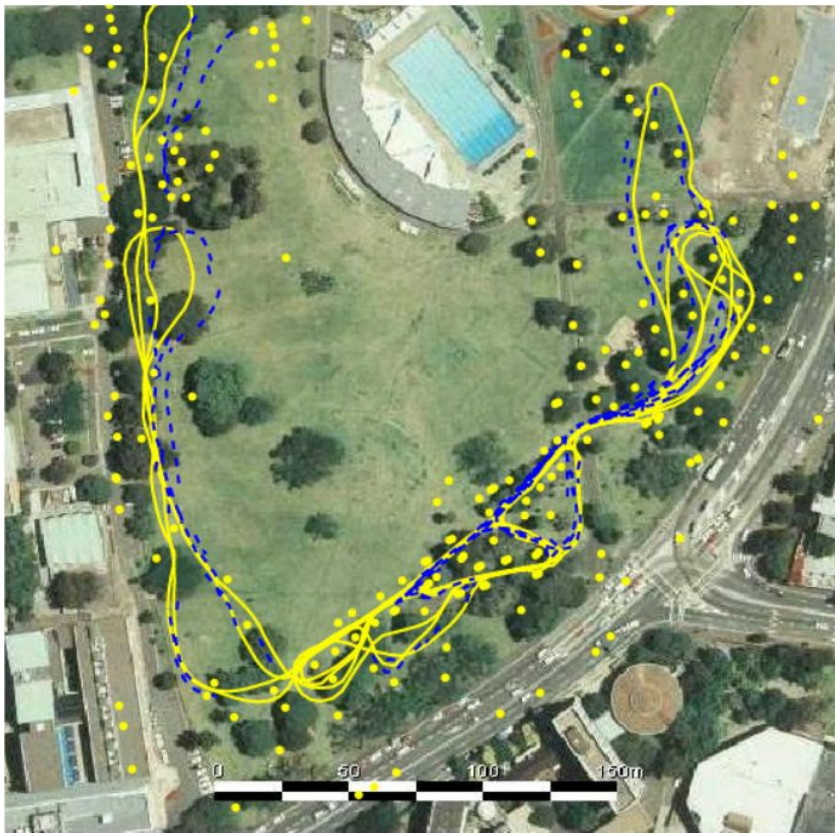
Application Examples



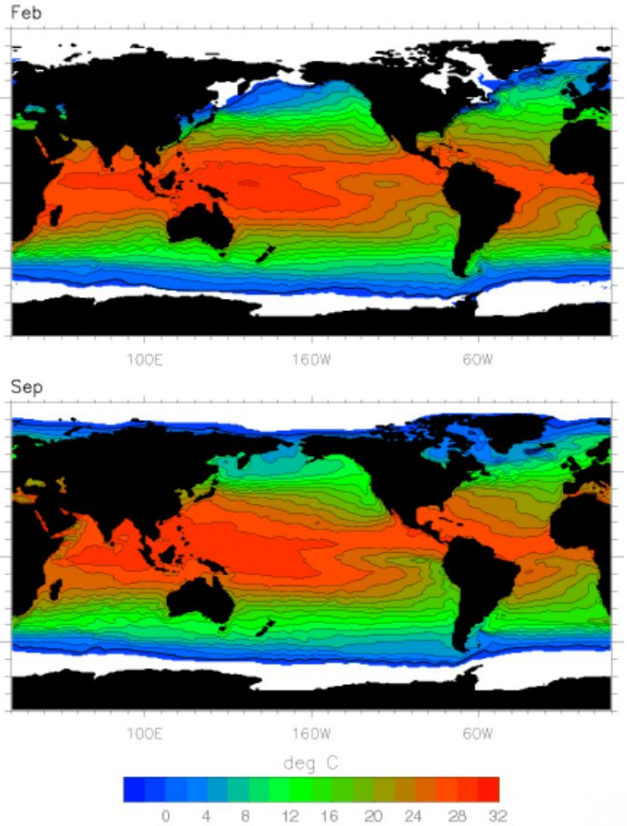
Mobile crowd sensing – MCS

Application Examples

- Intelligent sensing



Simultaneous localization and mapping – SLAM



Climate modeling

Application Examples

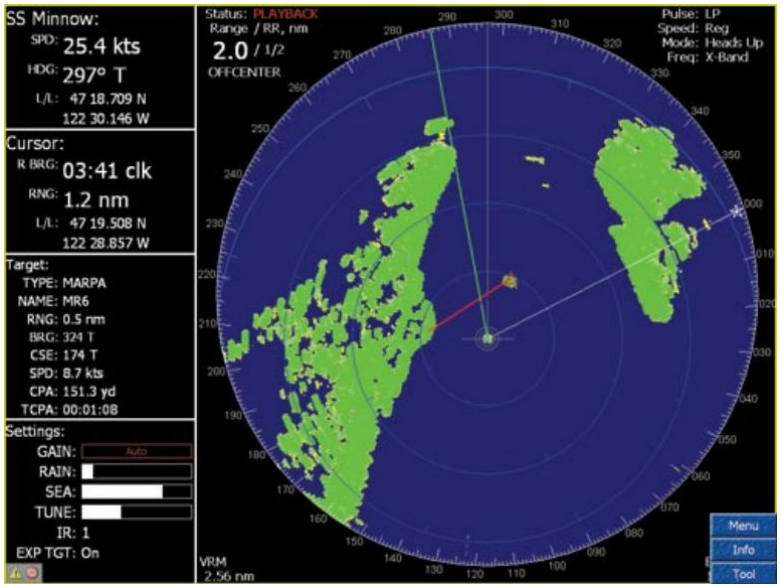
- Advanced surveillance

Pedestrian detection

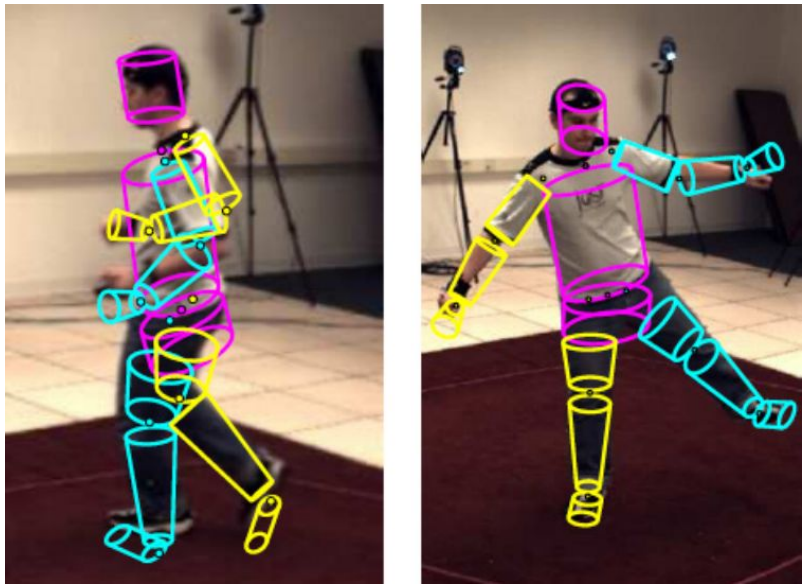
Gunshot detection

Application Examples

- Target tracking

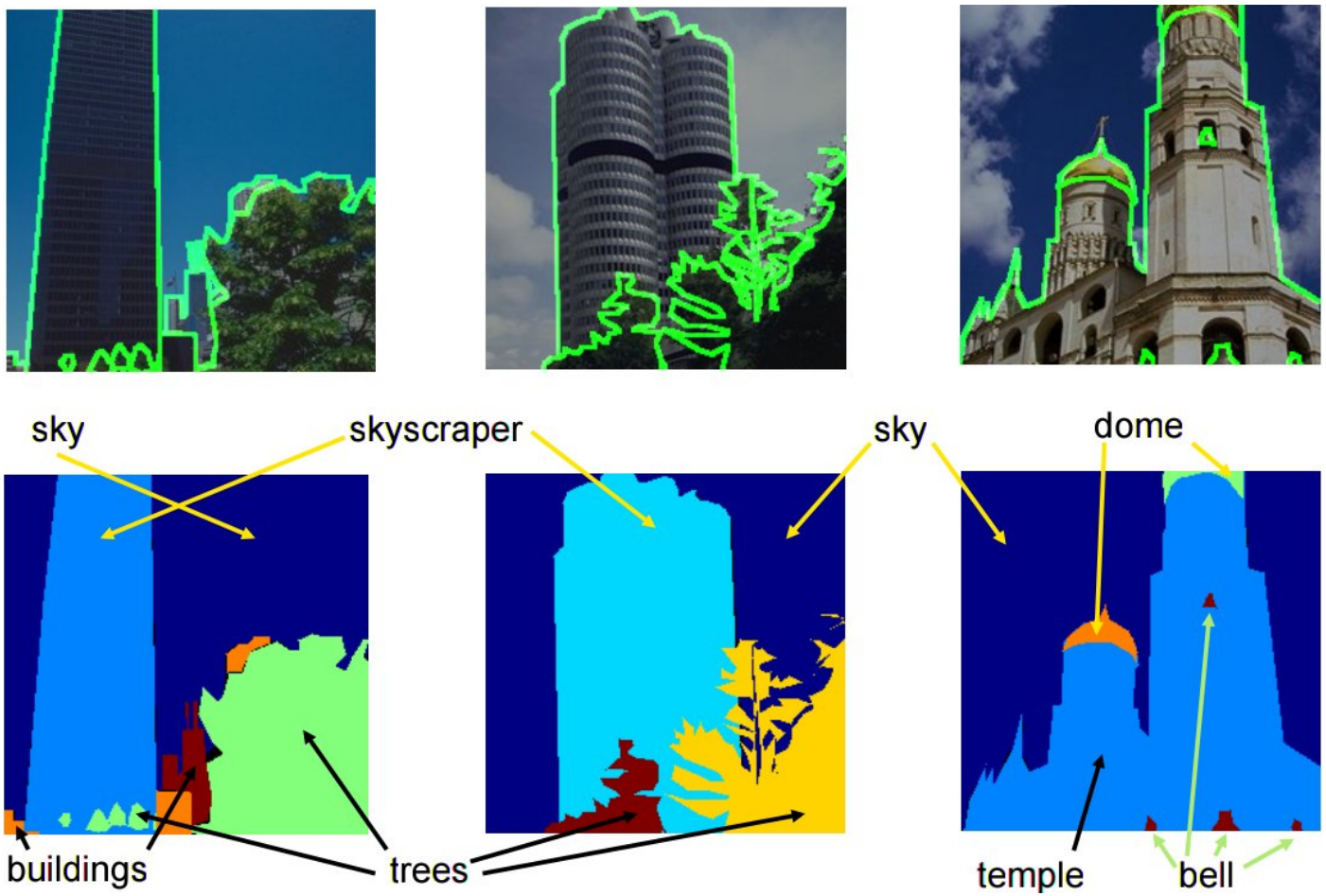


Radar-based tracking of multiple targets



Visual tracking of objects

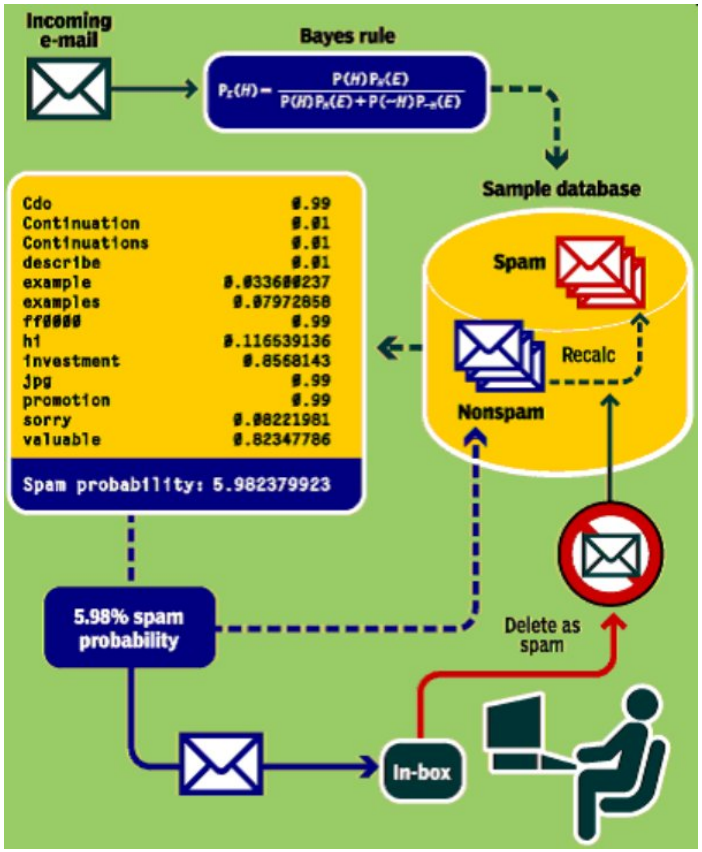
Application Examples



Visual object recognition

Application Examples

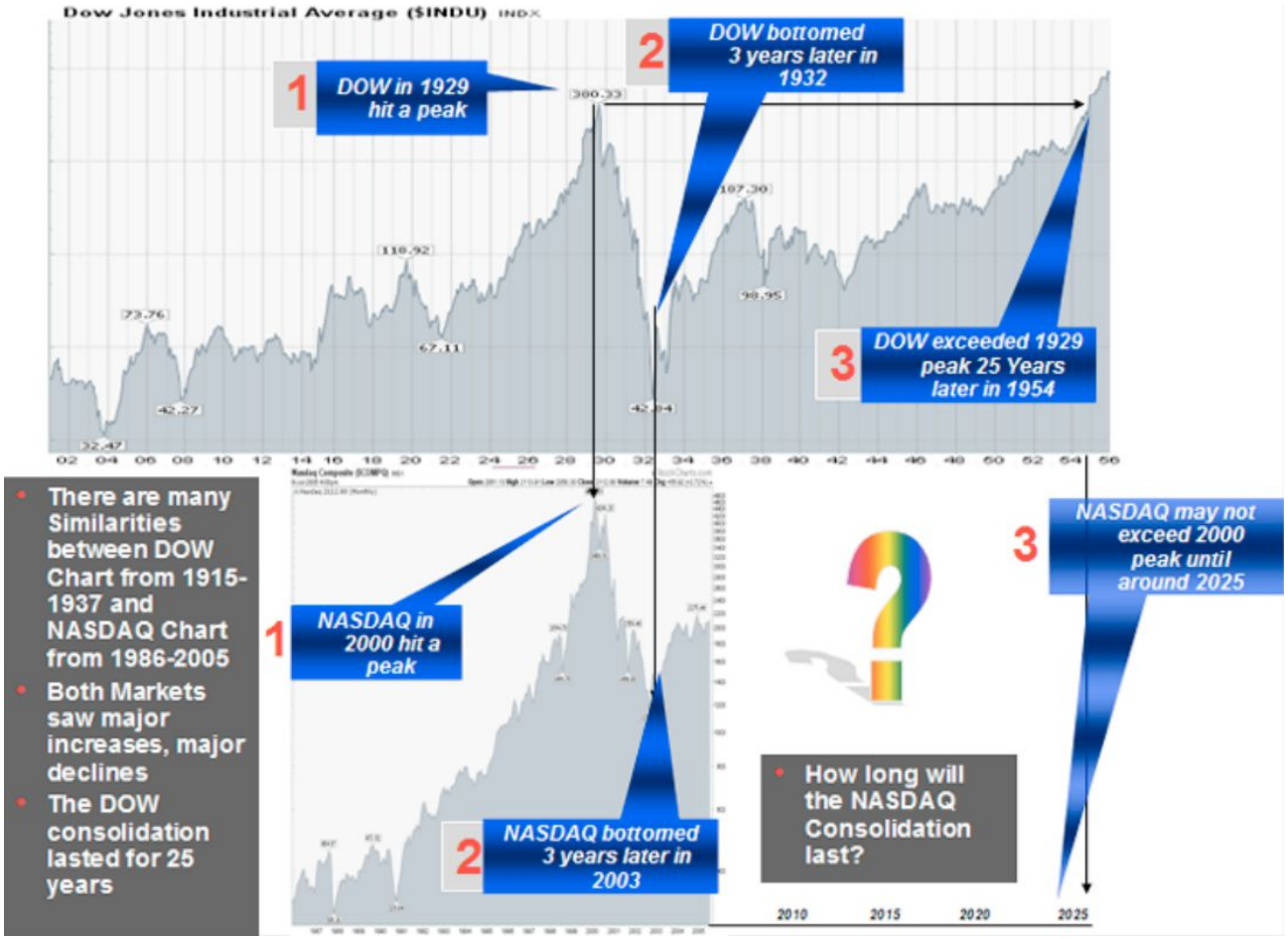
- Filtering



Spam filtering

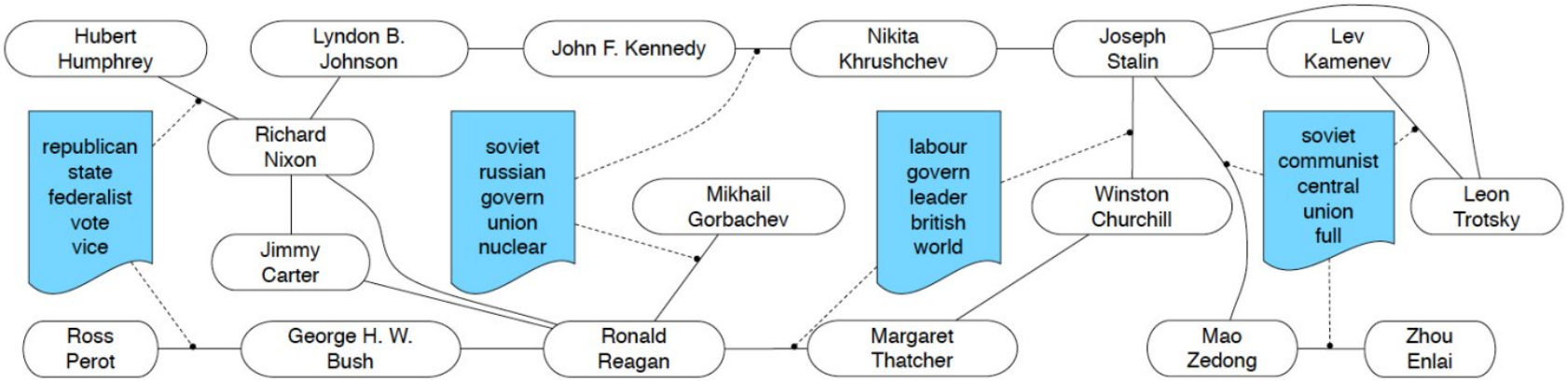
Collaborative filtering

Application Examples



Financial forecasting

Application Examples



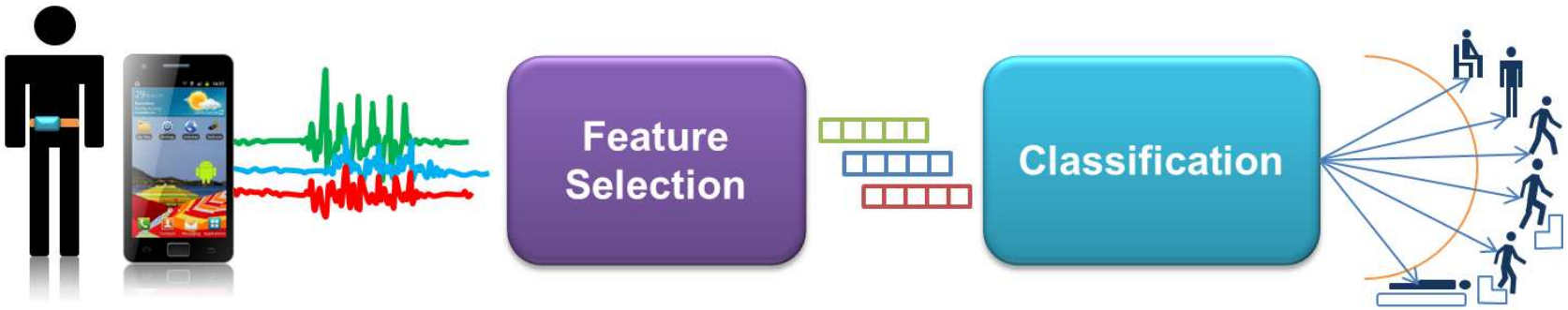
Social network analysis

- Other examples:
  - Structural health monitoring – SHM
  - Music information retrieval – MIR
  - Data mining and knowledge discovery in databases – DM & KDD

Our Application

- This is our example to be studied along the course:

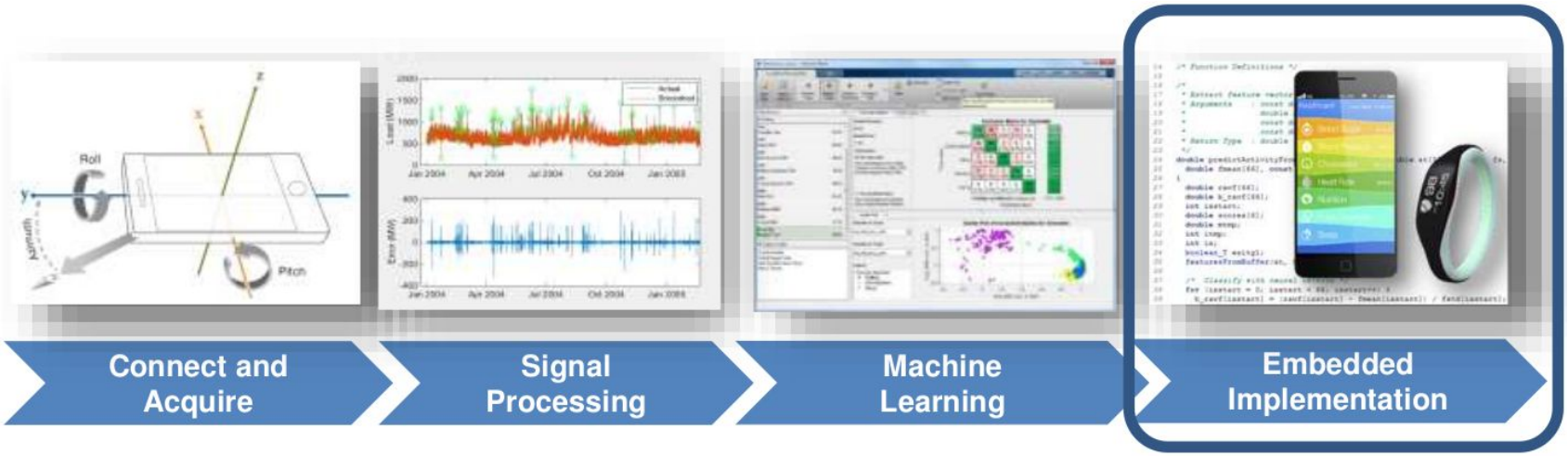
  Human activity recognition on smartphones
  D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. Reyes-Ortiz



Dataset available at `http://tinyurl.com/qbua5rs`

Our Application

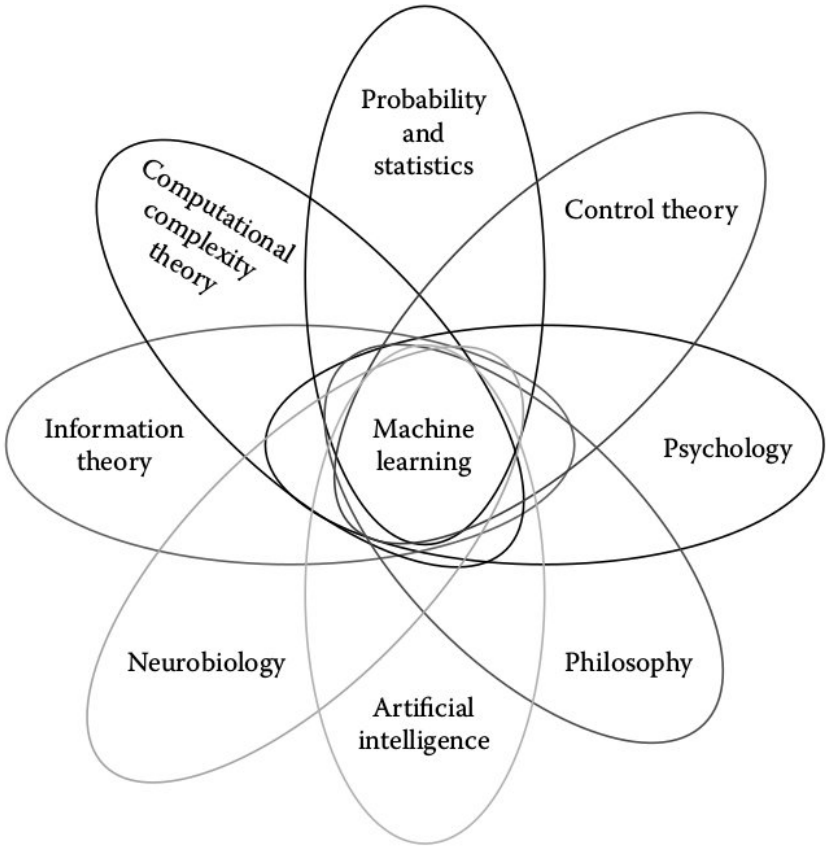
- Sensor data analytics workflow



Review 'Android sensor support from Matlab'

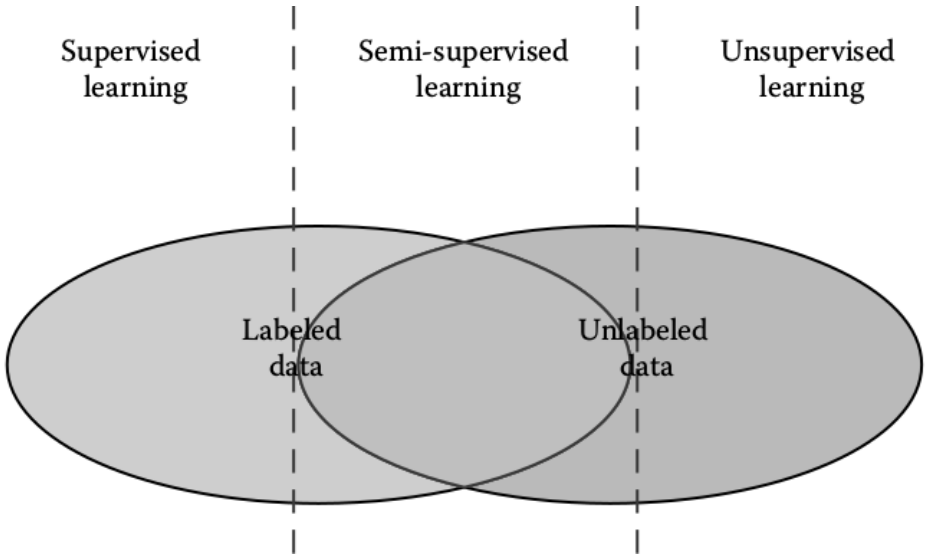
Machine learning

- Machine learning is a broad discipline, combining approaches from many different areas

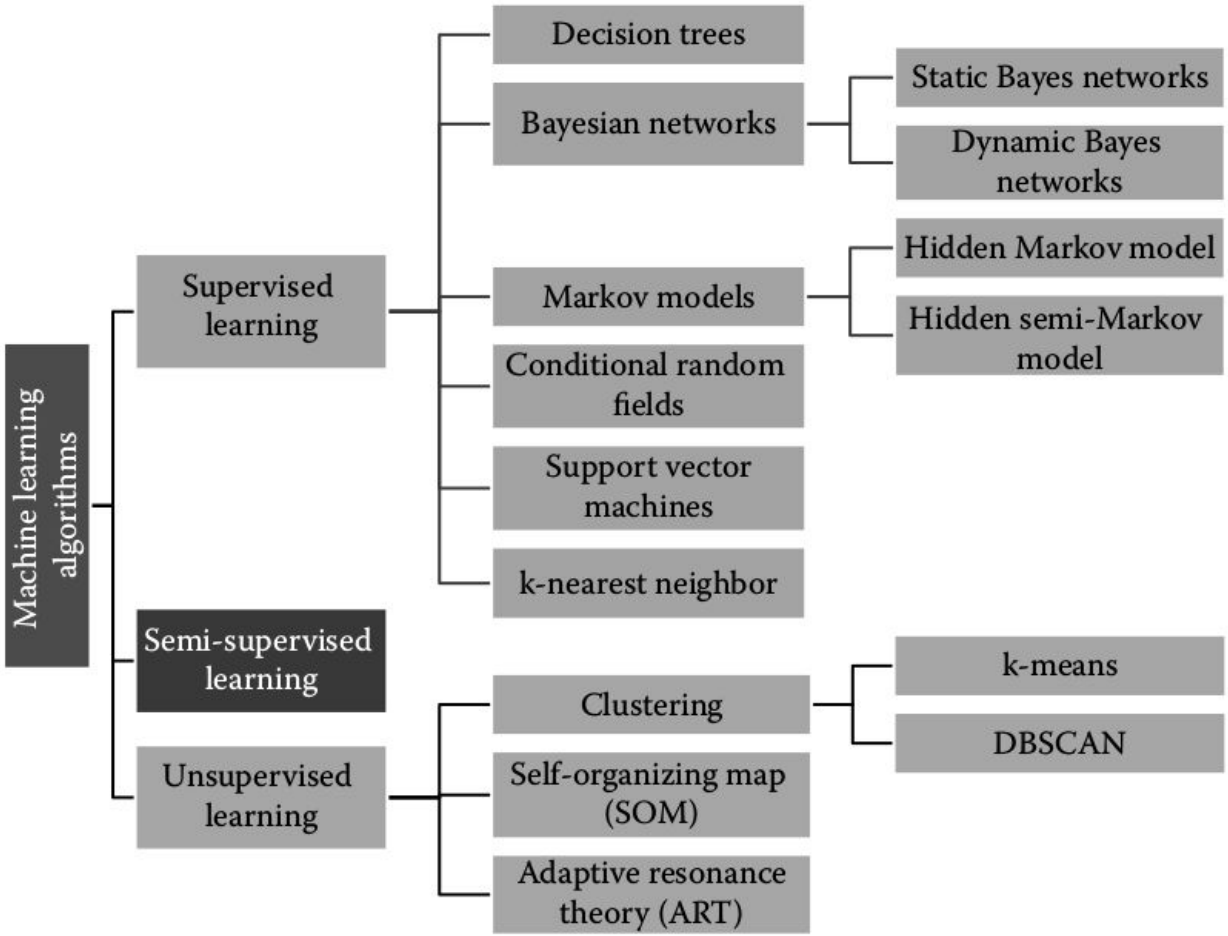


Machine learning

- Machine learning algorithms are divided into supervised learning, which used labeled training data, and unsupervised learning, where labeled training data is not available
- A third class of machine learning technique, semi-supervised learning, makes use of both labeled and unlabeled training data



Machine learning



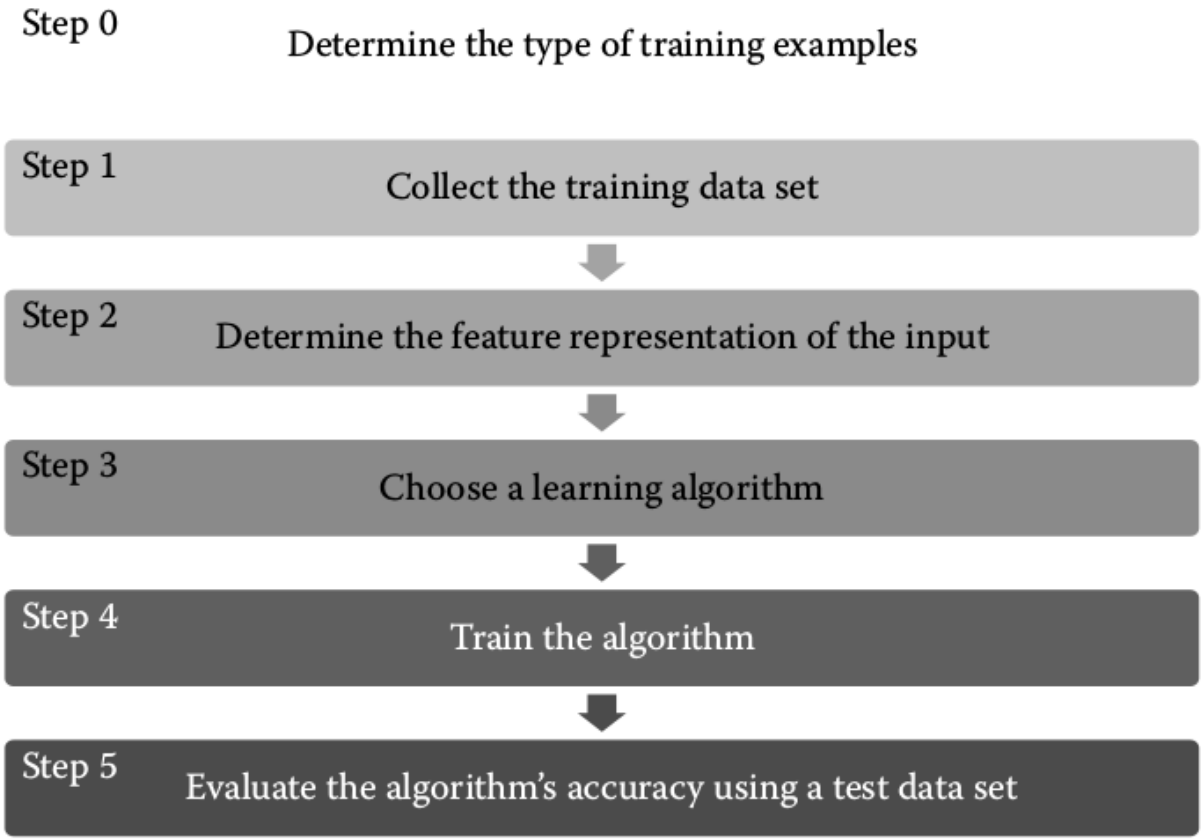
Most widely used machine learning algorithms

Supervised learning

- In supervised learning, the learner is provided with labeled input data
- This data contains a sequence of input/output pairs of the form $\langle x_i, y_i \rangle$, where x_i is a possible input and y_i is the correctly labeled output associated with it
- The aim of the learner in supervised learning is to learn the mapping from inputs to outputs
- The learning program is expected to learn a function f that accounts for the input/output pairs seen so far, $f(x_i) = y_i$, for all i
- This function f is called a *classifier* if the output is discrete and a regression function if the output is continuous

Supervised learning

- The stages of supervised machine learning

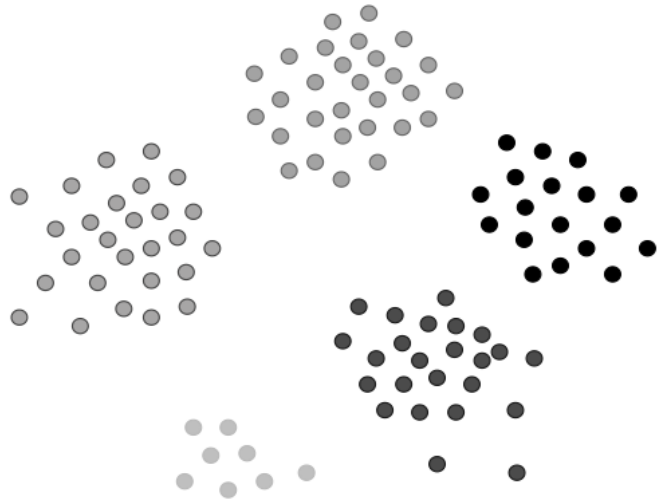


Unsupervised learning

- Collecting labeled data is resource and time consuming, and accurate labeling is often hard to achieve
- In unsupervised learning, the learner is provided with input data, which has not been labeled
- The aim of the learner is to find the inherent patterns in the data that can be used to determine the correct output value for new data instances
- The assumption here is that there is a structure to the input space, such that certain patterns occur more often than others, and we want to see what generally happens and what does not
- In statistics, this is called *density estimation*

Unsupervised learning

- Unsupervised learning algorithms are very useful for sensor network applications for the following reasons:
  - Collecting labeled data is resource and time-consuming
  - Accurate labeling is hard to achieve
  - Sensor networks applications are often deployed in unpredictable and constantly changing environments

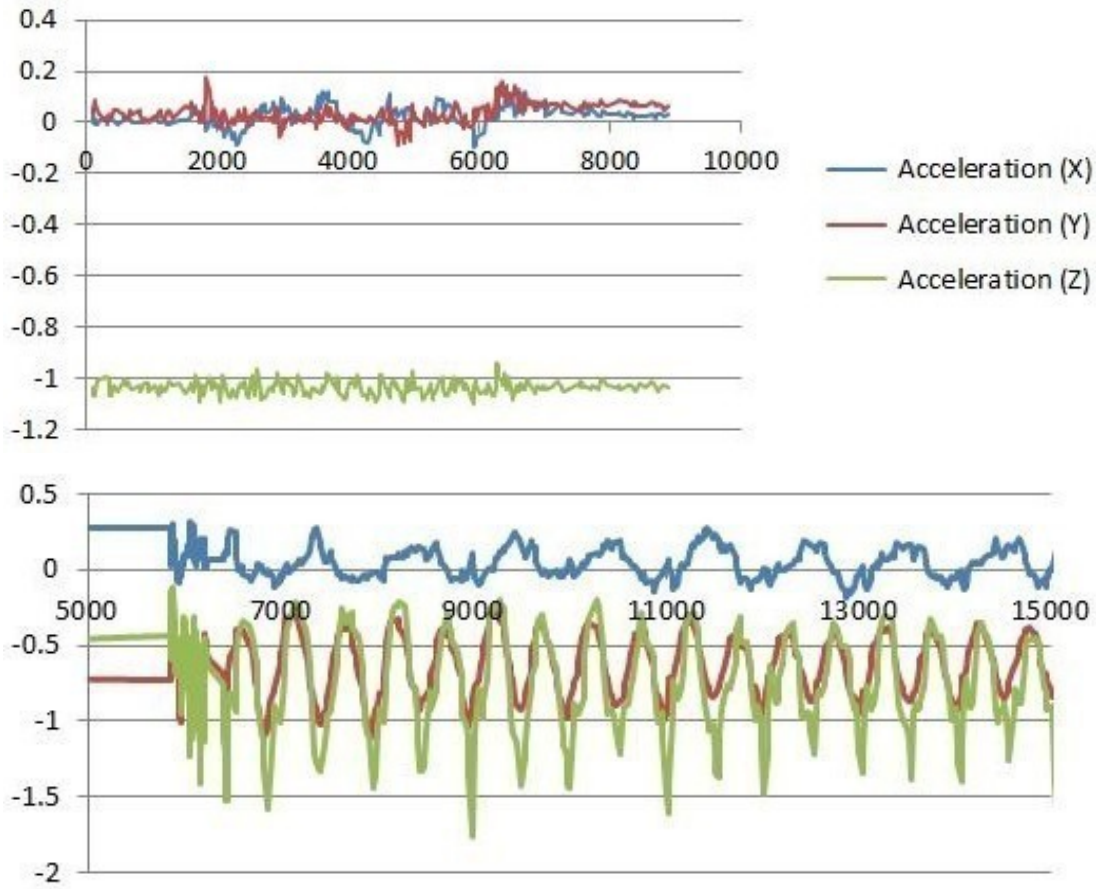


Semi-supervised learning

- Semi-supervised learning algorithms use both labeled and unlabeled data for training
- The labeled data is typically a small percentage of the training dataset
- The goal of semi-supervised learning is to:
  1. Understand how combining labeled and unlabeled data may change the learning behavior
  2. Design algorithms that take advantage of such a combination
- Semi-supervised learning is a very promising approach since it can use readily available unlabeled data to improve supervised learning tasks when the labeled data is scarce or expensive

Features and classifiers

- Figure shows two accelerometer signals corresponding to different activities: resting vs. walking

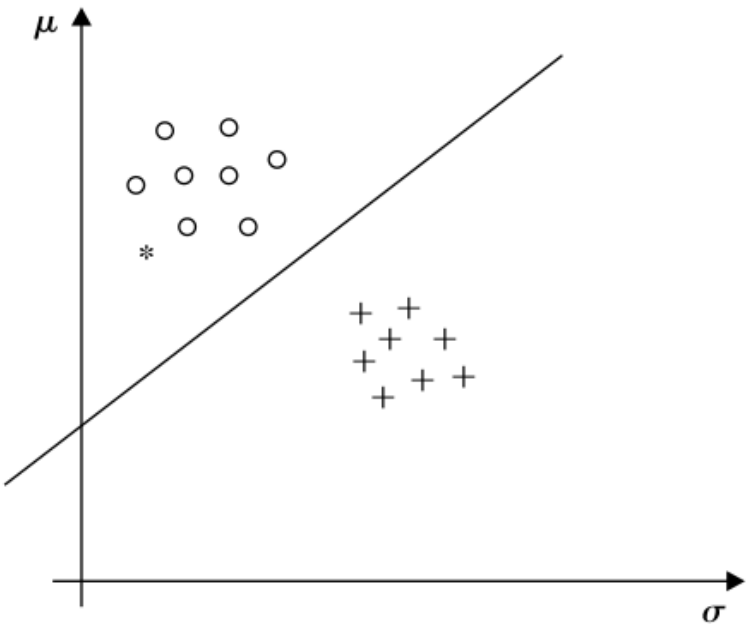


Features and classifiers

- The two signals are also themselves visually different
- We will further assume that these are not the only patterns (signals) that are available to us, but we have access to a signal database with a number of patterns, some of which are known to originate from resting and some from walking
- The first step is to identify the measurable quantities that make these two regions distinct from each other
- We can plot of the mean value of the intensity in each region of interest versus the corresponding standard deviation around this mean
- Each point will correspond to a different signal/region from the available database

Features and classifiers

- It turns out that resting patterns ($\circ$) tend to spread in a different area from waking patterns ($+$)
- The straight line seems to be a good candidate for separating the two classes ($\circ$ and $+$)



Features and classifiers

- The preceding artificial classification task has outlined the rationale behind a large class of pattern recognition problems
- The measurements used for the classification, the mean value and the standard deviation in this case, are known as *features*
- In the more general case l features x_i, $i = 1, 2, \ldots, l$, are used, and they form the *feature vector*

$$\mathbf{x} = [x_1, x_2, \ldots, x_l]^T$$

- Each of the feature vectors identifies uniquely a single pattern
- Throughout this course features and feature vectors will be treated as *random variables* and *vectors*, respectively

Features and classifiers

- Measurements resulting from different patterns exhibit a random variation
- This is due partly to the measurement noise of the measuring devices and partly to the distinct characteristics of each pattern
- This is the reason for the scattering of the points in each class shown in previous figure
- The straight line in that figure is known as the *decision* line, and it constitutes the classifier whose role is to divide the feature space into regions that correspond to either class $\circ$ or class $+$

Features and classifiers

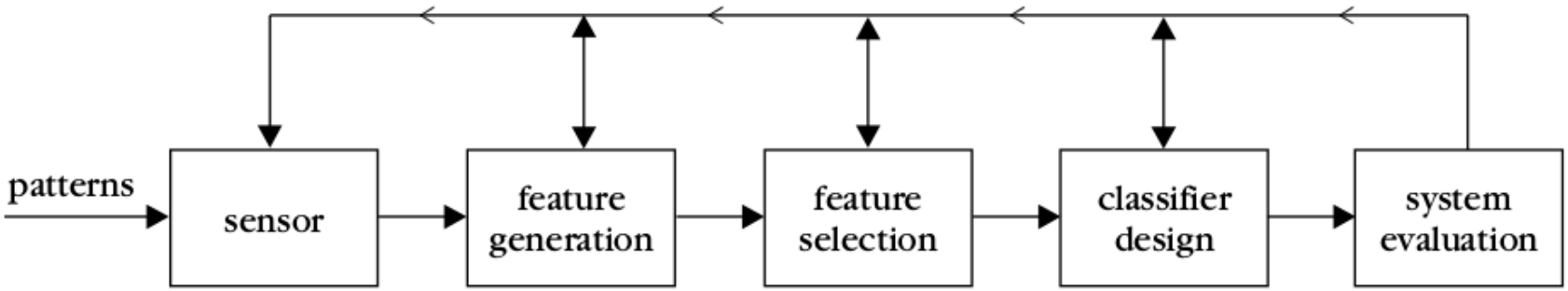
- If a feature vector $\mathbf{x}$, corresponding to an unknown pattern, falls in the class $\circ$ region, it is classified as class $\circ$, otherwise as class $+$
- This does not necessarily mean that the decision is correct
- If it is not correct, a *misclassification* has occurred
- In order to draw the straight line in previous figure we exploited the fact that we knew the labels (class $\circ$ or $+$) for each point of the figure
- The patterns (feature vectors) whose true class is known and which are used for the design of the classifier are known as *training patterns* (training feature vectors)

Features and classifiers

- Having outlined the definitions and the rationale, let us point out the basic questions arising in a classification task:
  - How are the features generated?
  - What is the best number l of features to use?
  - Having adopted the appropriate features, how does one design the classifier?
    (*optimality criterion*)
  - Once the classifier has been designed, how can one assess the performance of the designed classifier?
    (*classification error rate*)

Features and classifiers

- The basic stages involved in the design of a classification system are:



- As is apparent from the feedback arrows, these stages are not independent

Outline

Linear classifiers

- The major advantage of linear classifiers is their simplicity and computational attractiveness
- We start assuming that *all* feature vectors from the available classes can be classified correctly using a linear classifier, and we will develop techniques for the computation of the corresponding linear functions
- Next, we will focus on a more general problem, in which a linear classifier cannot correctly classify all vectors, yet we will seek ways to design an *optimal linear* classifier by adopting an appropriate optimality criterion

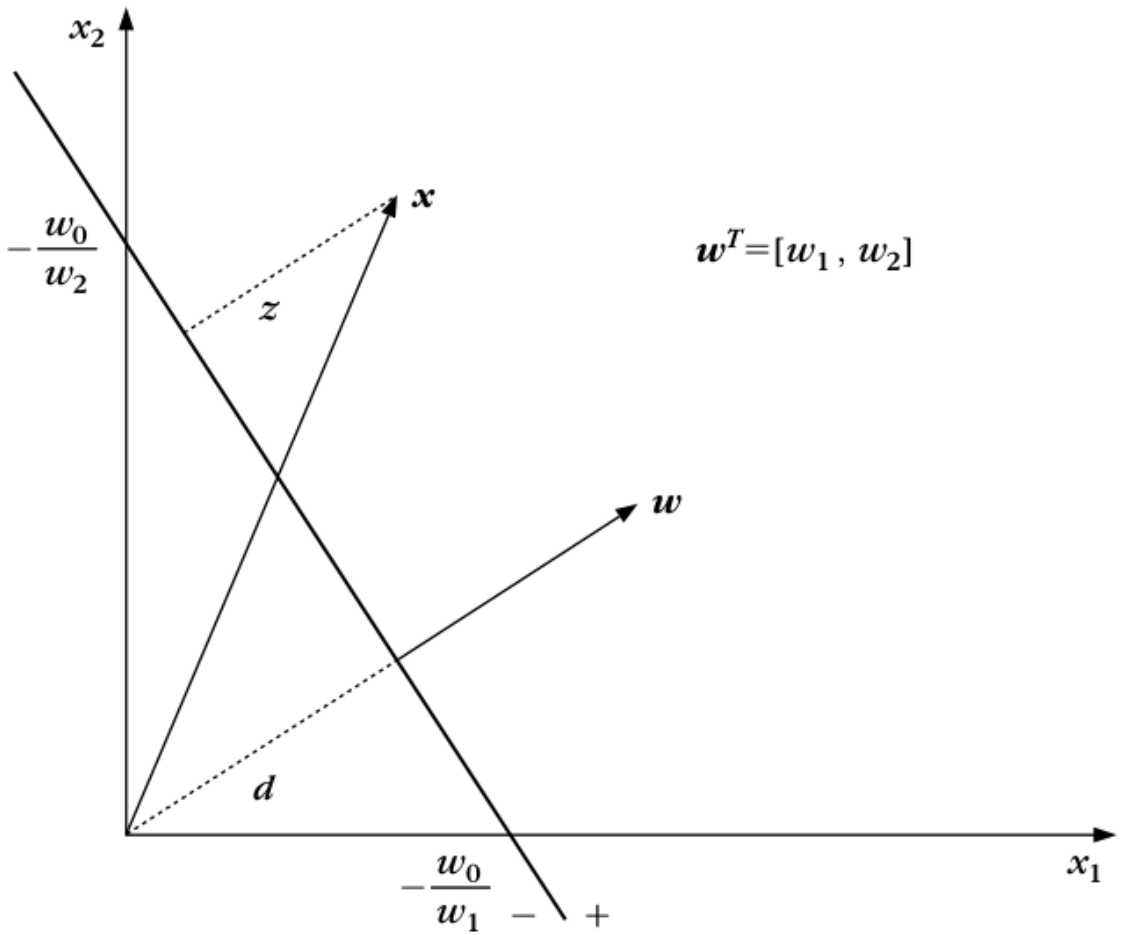
Decision hyperplanes

- Let us once more focus on the two-class case and consider linear discriminant functions
- Then the respective decision hypersurface in the l-dimensional feature space is a hyperplane, that is

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$$

  where $\mathbf{w} = [w_1, w_2, \ldots, w_l]^T$ is known as the *weight vector* and w_0 as the *threshold*
- It can be shown that the vector $\mathbf{w}$ is *orthogonal* to the decision hyperplane

Decision hyperplanes



Geometry for the decision line (for $w_1 > 0, w_2 > 0, w_0 < 0$)

Decision hyperplanes

- It is easy to see that

$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}$$

and

$$z = \frac{|g(\mathbf{x})|}{\sqrt{w_1^2 + w_2^2}}$$

- In other words, $|g(\mathbf{x})|$ is a measure of the Euclidean distance of the point $\mathbf{x}$ from the decision hyperplane
- On one side of the plane $g(\mathbf{x})$ takes positive values and on the other negative

The Perceptron algorithm

- Our major concern now is to compute the unknown parameters $w_i, i = 0, \ldots, l$, defining the decision hyperplane
- In this section, we assume that the two classes ω_1, ω_2 are linearly separable
- In other words, we assume that there exists a hyperplane, defined by $\mathbf{w}^T\mathbf{x} = 0$, such that

$$\mathbf{w}^T\mathbf{x} > 0 \quad \forall \mathbf{x} \in \omega_1$$
$$\mathbf{w}^T\mathbf{x} < 0 \quad \forall \mathbf{x} \in \omega_2$$

- This formulation also covers the case of a hyperplane not crossing the origin, since this can be brought into the previous formulation by defining the extended $(l+1)$-dimensional vectors $\mathbf{x}' \equiv [\mathbf{x}^T, 1]^T$, $\mathbf{w}' \equiv [\mathbf{w}^T, w_0]^T$

The Perceptron algorithm

- We will approach the problem as a typical optimization task
- Thus we need to adopt (a) an appropriate cost function and (b) an algorithmic scheme to optimize it
- To this end, we choose the perceptron cost defined as

$$J(\mathbf{w}) = \sum_{\mathbf{x} \in Y} (\delta_x \mathbf{w}^T \mathbf{x})$$

  where Y is the subset of the training vectors, which are misclassified by the hyperplane defined by the weight vector $\mathbf{w}$
- The variable δ_x is chosen so that $\delta_x = -1$ if $\mathbf{x} \in \omega_1$ and $\delta_x = +1$ if $\mathbf{x} \in \omega_2$
- This cost function is *continuous and piecewise linear*

The Perceptron algorithm

- Indeed, if we change the weight vector smoothly, the cost $J(\mathbf{w})$ changes linearly until the point at which there is a change in the number of misclassified vectors
- At these points the gradient is not defined, and the gradient function is discontinuous
- To derive the algorithm for the iterative minimization of the cost function, we will adopt an iterative scheme in the spirit of the gradient descent method, that is

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \rho_t \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}(t)}$$

where $\mathbf{w}(t)$ is the weight vector estimate at the tth iteration step and ρ_t is a sequence of positive real numbers

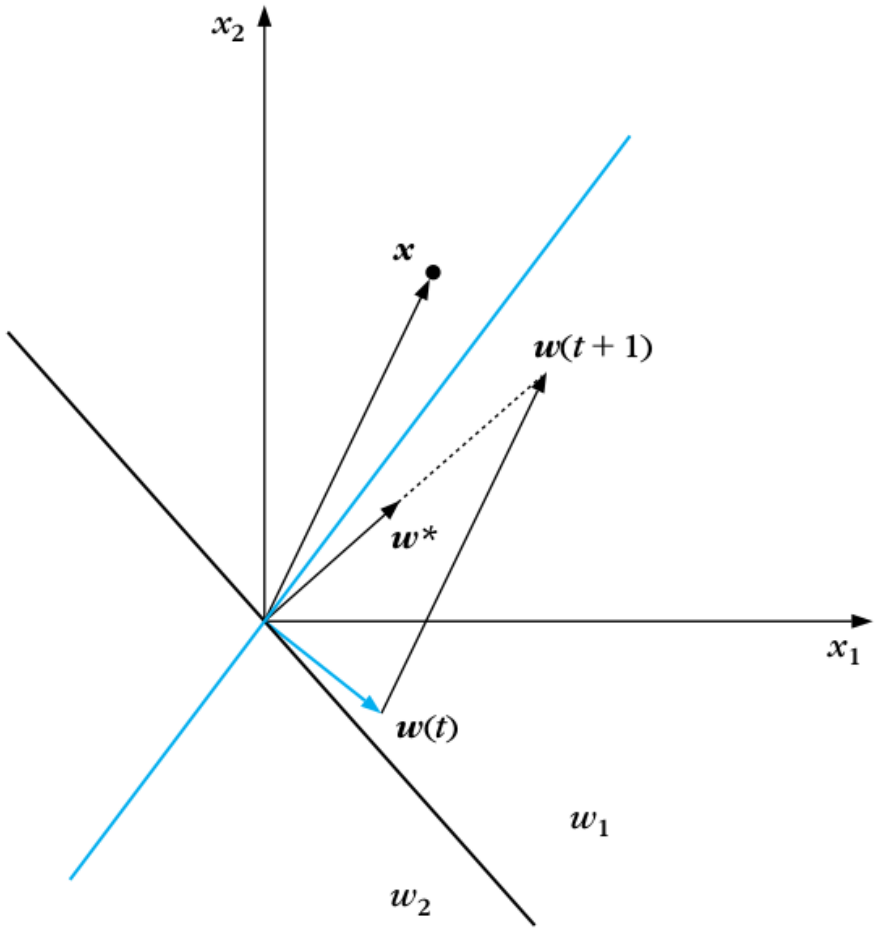
The Perceptron algorithm

- Substituting the cost function $J(\mathbf{w})$, we obtain

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \rho_t \sum_{\mathbf{x} \in Y} \delta_x \mathbf{x}$$

- The algorithm is known as the perceptron algorithm and is quite simple in its structure
- The algorithm is initialized from an arbitrary weight vector $\mathbf{w}(0)$, and the correction vector is formed using the misclassified features
- The weight vector is then corrected according to the preceding equation
- This is repeated until the algorithm converges to a solution, that is, all features are correctly classified

The Perceptron algorithm



Geometric interpretation of the perceptron algorithm

The Perceptron algorithm

- A pseudocode for the perceptron algorithm

Choose $\mathbf{w}(0)$ randomly
Choose ρ_0
$t = 0$
Repeat until $Y = \emptyset$
    $Y = \emptyset$
    For $i = 1$ to N
        If $\delta_{x_i} \mathbf{w}(t)^T \mathbf{x}_i \geq 0$ then $Y = Y \cup \mathbf{x}_i$
    $\mathbf{w}(t+1) = \mathbf{w}(t) - \rho_t \sum_{\mathbf{x} \in Y} \delta_x \mathbf{x}$
    Adjust ρ_t
    $t = t + 1$

The Perceptron algorithm

- No doubt, the sequence ρ_t is critical for the convergence
- It can be shown that the perceptron algorithm converges to a solution in a *finite number of iteration steps*, provided that the sequence ρ_t is properly chosen
  - An example of a properly chosen sequence is $\rho_t = c/t$, where c is a constant
  - The algorithm also converges for constant $\rho_t = \rho$, provided ρ is properly bounded
- In practice, the proper choice of the sequence ρ_t is vital for the convergence speed of the algorithm
- The solution is not unique, because there are more than one hyperplanes separating two linearly separable classes

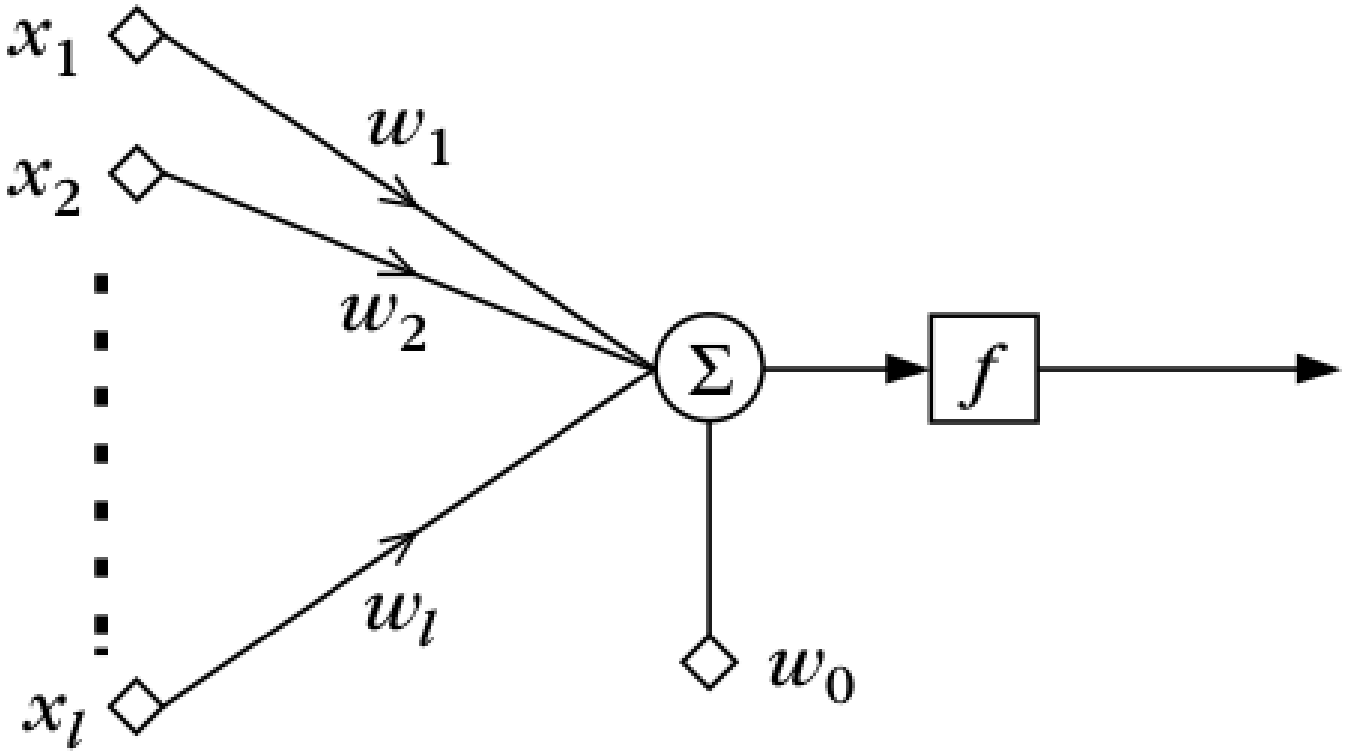
The Perceptron algorithm

- The perceptron algorithm was originally proposed by Rosenblatt in the late 1950s
- The algorithm was developed for training the perceptron, the basic unit used for modeling neurons of the brain
- This was considered central in developing powerful models for machine learning
- Once the perceptron algorithm has converged to a weight vector $\mathbf{w}$ and a threshold w_0, our next goal is the classification of an unknown feature vector to either of the two classes
- Classification is achieved via the simple rule

$$\text{If } \mathbf{w}^T\mathbf{x} + w_0 > 0 \text{ assign } \mathbf{x} \text{ to } \omega_1$$

$$\text{If } \mathbf{w}^T\mathbf{x} + w_0 < 0 \text{ assign } \mathbf{x} \text{ to } \omega_2$$

The Perceptron algorithm



The basic perceptron model

The Perceptron algorithm

- This basic network is known as a *perceptron* or *neuron*
- Perceptrons are simple examples of the so-called *learning machines*—that is, structures whose free parameters are updated by a *learning algorithm*, such as the perceptron algorithm, in order to 'learn' a specific task, based on a set of training data
- A basic requirement for the convergence of the perceptron algorithm is the linear separability of the classes; if this is not true, as is usually the case in practice, the perceptron algorithm does not converge
  - Algorithms that find reasonably good solutions when the classes are not linearly separable are the pocket algorithm, the thermal perceptron algorithm, the loss minimization algorithm, and the barycentric correction procedure

The Perceptron algorithm

- So far we have dealt with the two-class case
- The generalization to an M-class task is straightforward
- A linear discriminant function $\mathbf{w}_i, i = 1, 2, \ldots, M$, is defined for each of the classes
- A feature vector $\mathbf{x}$ (in the $[l+1]$-dimensional space to account for the threshold) is classified in class C_i if

$$\mathbf{w}_i^T \mathbf{x} > \mathbf{w}_j^T \mathbf{x}, \quad \forall j \neq i$$

- This condition leads to the so-called Kesler's construction
- The perceptron algorithm will have no difficulty in solving this problem for us, provided that such a solution is possible

Least squares methods

- Although we know that the classes are not linearly separable, we still wish to adopt a linear classifier in many cases, despite the fact that this will lead to suboptimal performance from the classification error probability point of view
- Remember, the goal now is to compute the corresponding weight vector under a suitable optimality criterion
- So, in this section we will attempt to design a linear classifier so that its desired output is again ± 1, depending on the class ownership of the input vector
- However, we will have to live with errors; that is, the true output will not always be equal to the desired one

Least squares methods

- Given a vector $\mathbf{x}$, the output of the classifier will be $\mathbf{w}^T\mathbf{x}$
- The desired output will be denoted as $y(\mathbf{x} \equiv y = \pm 1$
- The weight vector will be computed so as to minimize the mean square error (MSE) between the desired and true outputs, that is

$$J(\mathbf{w}) = E[|y - \mathbf{x}^T\mathbf{w}|^2]$$

$$\hat{\mathbf{w}} = \min_{\mathbf{w}} J(\mathbf{w})$$

- Since

$$J(\mathbf{w}) = P(\omega_1)\int(1 - \mathbf{x}^T\mathbf{w})^2 P(\mathbf{x}|\omega_1)d\mathbf{x} \quad + \quad P(\omega_2)\int(1 - \mathbf{x}^T\mathbf{w})^2 P(\mathbf{x}|\omega_2)d\mathbf{x}$$

Least squares methods

- Minimizing $J(\mathbf{w})$ easily results in

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 2E[\mathbf{x}(y - \mathbf{x}^T\mathbf{w})] = 0$$

- Then

$$\hat{\mathbf{w}} = R_x^{-1}E[\mathbf{x}y]$$

where

$$R_x \equiv E[\mathbf{x}\mathbf{x}^T] = \begin{bmatrix} E[x_1x_1] & \cdots & E[x_1x_l] \\ E[x_2x_1] & \cdots & E[x_2x_l] \\ \vdots & \ddots & \vdots \\ E[x_lx_1] & \cdots & E[x_lx_l] \end{bmatrix}$$

is known as the *autocorrelation* matrix and is equal to the *covariance* matrix if the respective mean values are zero

Least squares methods

- The vector

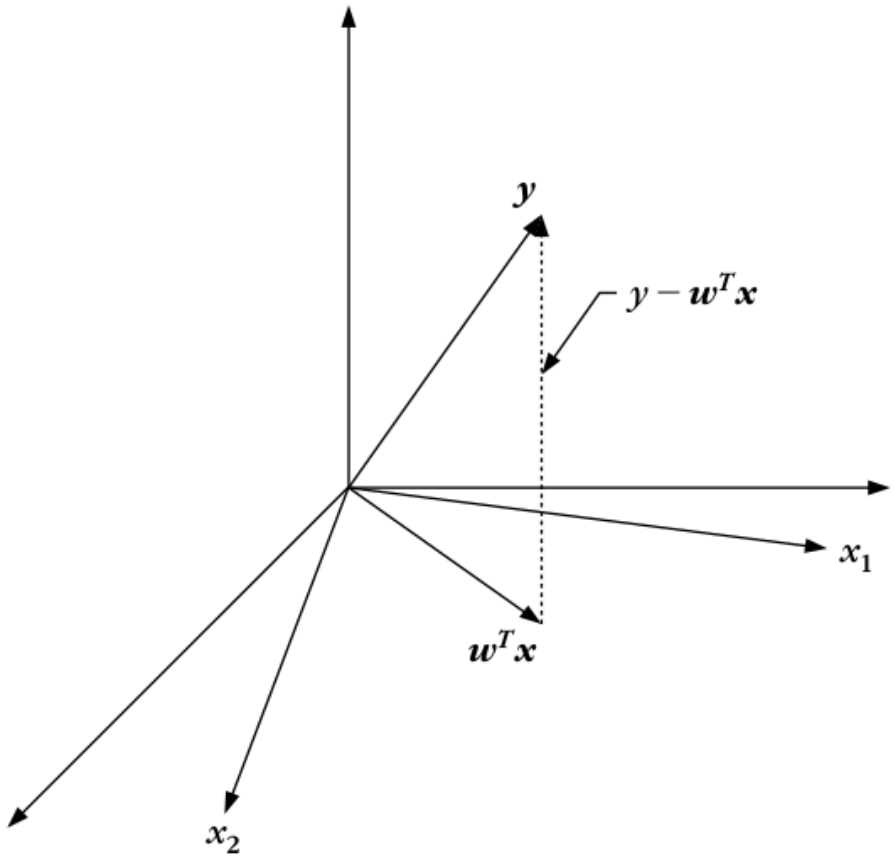
$$E[\mathbf{x}y] = E\left[\begin{bmatrix} x_1 y \\ x_2 y \\ \vdots \\ x_l y \end{bmatrix}\right]$$

  is known as the *cross-correlation* between the desired output and the (input) feature vectors

- Thus, the mean square optimal weight vector results as the solution of a linear set of equations, provided, of course, that the correlation matrix is invertible

- It is interesting to point out that there is a geometrical interpretation of this solution

Least squares methods



Interpretation of the MSE estimate as an orthogonal projection on the input vector elements' subspace

Least squares methods

- A criterion closely related to the MSE is the *sum of error squares* or simply the *least squares* (LS) criterion defined as

$$J(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \mathbf{w})^2 = \sum_{i=1} e_i^2$$

- In other words, the errors between the desired output of the classifier and the true output are summed up over all the available training feature vectors, instead of averaging them out
- In this way, we overcome the need for explicit knowledge of the underlying PDFs

Least squares methods

- Minimizing with respect to $\mathbf{w}$ results in

$$\sum_{i=1}^{N} \mathbf{x}_i (y_i - \mathbf{x}_i^T \hat{\mathbf{w}}) = 0 \quad \Rightarrow \quad \left(\sum_{i=1} \mathbf{x}_i \mathbf{x}_i^T \right) \hat{\mathbf{w}} = \sum_{i=1} \mathbf{x}_i y_i$$

- For the sake of mathematical formulation let us define

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1I} \\ x_{21} & x_{22} & \cdots & x_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NI} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Least squares methods

- Hence

$$(X^T X)\hat{\mathbf{w}} = X^T \mathbf{y} \quad \Rightarrow \quad \hat{\mathbf{w}} = (X^T X)^{-1} X^T \mathbf{y}$$

- Thus, the optimal weight vector is again provided as the solution of a linear set of equations
- Matrix $X^T X$ is known as the *sample correlation matrix*
- Matrix $X^+ \equiv (X^T X)^{-1} X^T$ is known as the *pseudoinverse* of X, and it is meaningful only if $X^T X$ is invertible, that is, X is of rank l
- X^+ is a generalization of the inverse of an invertible square matrix

Mean square estimation

- Let $\mathbf{y}$, $\mathbf{x}$ be two random vector variables of dimensions $M \times 1$ and $l \times 1$, respectively, and assume that they are described by the joint pdf $p(\mathbf{y}, \mathbf{x})$
- The task of interest is to estimate the value of $\mathbf{y}$, given the value of $\mathbf{x}$ that is obtained from an experiment
- No doubt the classification task falls under this more general formulation
- In a more general setting, the values of $\mathbf{y}$ may not be discrete
- Take, as an example, the case where $y \in \mathcal{R}$ is generated by an unknown rule

$$y = f(\mathbf{x}) + \epsilon$$

Mean square estimation

- The task now is to estimate (predict) the value of y, given the value of $\mathbf{x}$
- Once more, this is a problem of designing a function $g(\mathbf{x})$, based on a set of training data points $(y_i, \mathbf{x}_i), i = 1, 2, \ldots, N$, so that the predicted value

$$\hat{y} = g(\mathbf{x})$$

  to be as close as possible to the true value y in some optimal sense
- This type of problem is known as a *regression* task
- One of the most popular optimality criteria for regression is the mean square error (MSE)

Mean square estimation

- The mean square estimate $\hat{\mathbf{y}}$ of the random vector $\mathbf{y}$, given the value $\mathbf{x}$, is defined as

$$\hat{\mathbf{y}} = \min_{\tilde{\mathbf{y}}} E[|\mathbf{y} - \tilde{\mathbf{y}}|^2]$$

- Note that the mean value here is with respect to the conditional pdf $p(\mathbf{y}|\mathbf{x})$
- It can be shown that the optimal estimate is the mean value of $\mathbf{y}$, that is

$$\hat{\mathbf{y}} = E[\mathbf{y}|\mathbf{x}] \equiv \int_{-\infty}^{\infty} \mathbf{y} p(\mathbf{y}|\mathbf{x}) d\mathbf{y}$$

Mean square estimation

- Least Mean Squares (LMS) algorithm
  - The best choice of $\mathbf{w}$ is the one that minimizes our cost function
$$J(\mathbf{w}) = \frac{1}{2}|\mathbf{y} - \mathbf{x}^T\mathbf{w}|^2$$
  - In this sense, the LMS algorithm performs the following update for each new $\mathbf{w}$
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \rho_t \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \rho_t \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T\mathbf{w})\mathbf{x}_i$$

Mean square estimation

- Regularization
  - Minimize

$$J(\mathbf{w}) = \frac{1}{2}|\mathbf{y} - \mathbf{x}^T\mathbf{w}|^2 + \epsilon|\mathbf{w}|^2$$

    with ϵ 'small' by solving

$$(X^T X + \epsilon\, I)\, \hat{\mathbf{w}} = X^T \mathbf{y}$$

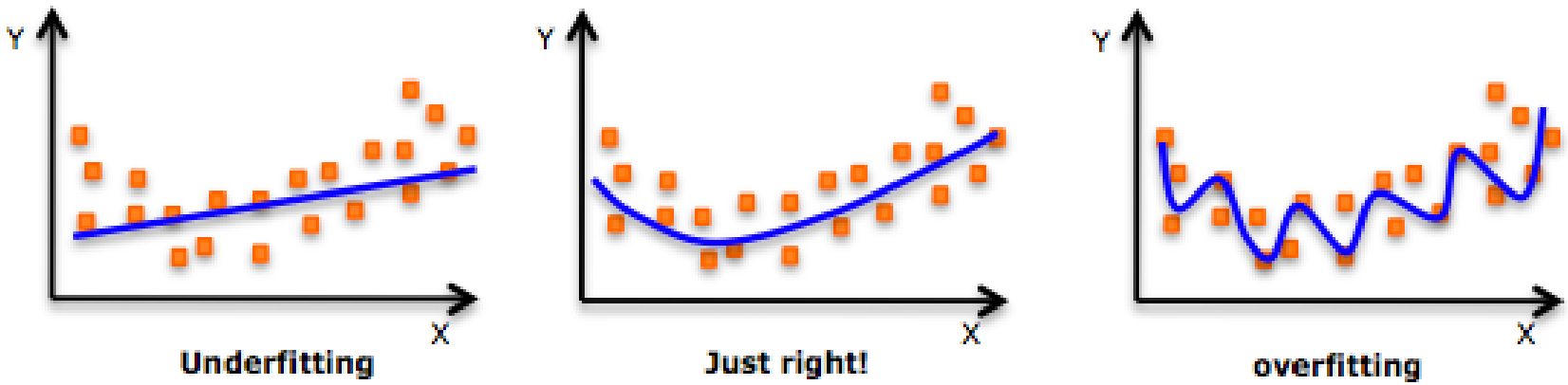
- Beyond hyperplanes
  - Linear models become powerful function approximators when we consider non-linear feature transformations

$$\mathbf{x}_i = [1 \quad x_i \quad x_i^2] \quad \Rightarrow \quad \hat{y}_i = w_0 + w_1 x_i + w_2 x_i^2$$

  - Predictions are still linear in $\mathbf{x}$ (all the math is the same!)

Mean square estimation

- Overfitting
  - So the more features the better?
  - Carefully selected features can improve model accuracy
  - But adding too many can lead to overfitting
  - Feature selection will be discussed in a separate lecture



Support vector machines

- Let $x_i, i = 1, 2, \ldots, N$, be the feature vectors of the training set, X
- These belong to either of two classes, ω_1 and ω_2, which are assumed to be linearly separable
- The goal, once more, is to design a hyperplane

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$$

  that classifies correctly all the training vectors
- As we have already discussed, such a hyperplane is not unique
- The perceptron algorithm may converge to any one of the possible solutions

Support vector machines

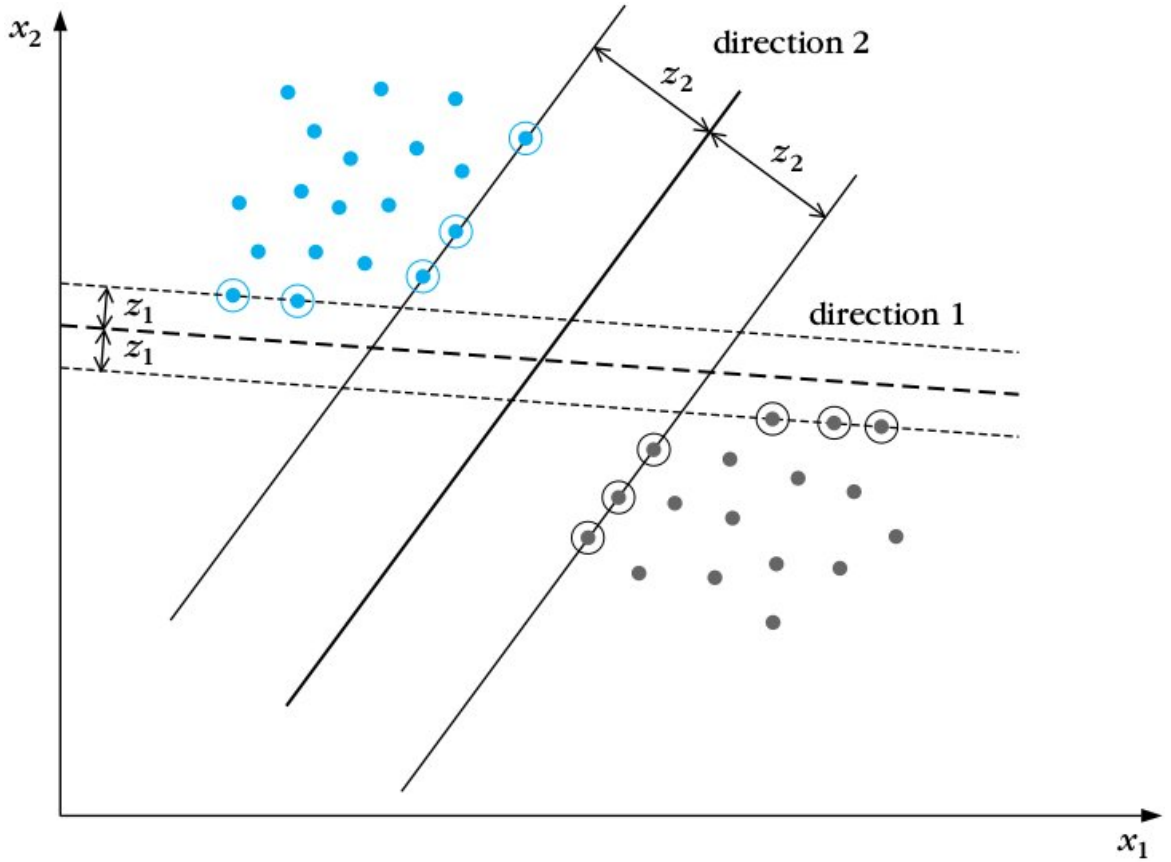
- A very sensible choice for the hyperplane classifier would be the one that leaves the maximum margin from both classes
- Here we have touched a very important issue in the classifier design stage: it is known as the *generalization performance of the classifier*
- This refers to the capability of the classifier, designed using the training data set, to operate satisfactorily with data outside this set
- Let us now quantify the term *margin* that a hyperplane leaves from both classes
- Every hyperplane is characterized by its direction (determined by $\mathbf{w}$) and its exact position in space (determined by w_0)

Support vector machines

- Since we want to give no preference to either of the classes, then it is reasonable for each direction to select that hyperplane which has the same distance from the respective nearest points in ω_1 and ω_2
- So, our goal is to search for the direction that gives the maximum possible margin
- Recall that the distance of a point from a hyperplane is given by

$$z = \frac{|g(\mathbf{x})|}{|\mathbf{w}|}$$

Support vector machines



An example of a linearly separable two-class problem with two possible linear classifiers

Support vector machines

- Our task can now be summarized as: Compute the parameters $\mathbf{w}, w_0$ of the hyperplane so that

$$\text{minimize } J(\mathbf{w}, w_0) \equiv \frac{1}{2}|\mathbf{w}|^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1, i = 1, 2, \ldots, N$$

- This is a nonlinear (quadratic) optimization task subject to a set of linear inequality constraints
- Combining the Karush-Kuhn-Tucker (KKT) conditions that the minimizer has to satisfy and the Lagrange multipliers

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \quad \text{and} \quad \sum_{i=1}^{N} \lambda_i y_i = 0$$

Support vector machines

- The Lagrange multipliers can be either zero or positive
- Thus, the vector parameter $\mathbf{w}$ of the optimal solution is a linear combination of $N_s \leq N$ feature vectors that are associated with $\lambda_i \neq 0$
- These are known as *support vectors* and the optimum hyperplane classifier as a *support vector machine* (SVM)
- Lagrange multiplier corresponds to a so called active constraint
- Hence, as the set of constraints suggests for $\lambda_i \neq 0$, the support vectors lie on either of the two hyperplanes

$$\mathbf{w}^T\mathbf{x} + w_0 = \pm 1$$

Support vector machines

- In other words, they are the training vectors that are closest to the linear classifier,and they constitute the *critical elements of the training set*
- The resulting hyperplane classifier is insensitive to the number and position of feature vectors corresponding to $\lambda_i = 0$
- The cost function $J(\mathbf{w})$ is a strict convex one, a property that is guaranteed by the fact that the corresponding Hessian matrix is positive definite
- These conditions guarantee that any local minimum is also global and unique: The optimal hyperplane classifier of a support vector machine is unique

Support vector machines

- Now, we need to compute the involved parameters
- It belongs to the *convex programming* family of problems, since the cost function is convex and the constraints are linear and define a convex set of feasible solutions
- The final optimization task is

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right)$$

$$\text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0, \quad \boldsymbol{\lambda} \geq 0$$

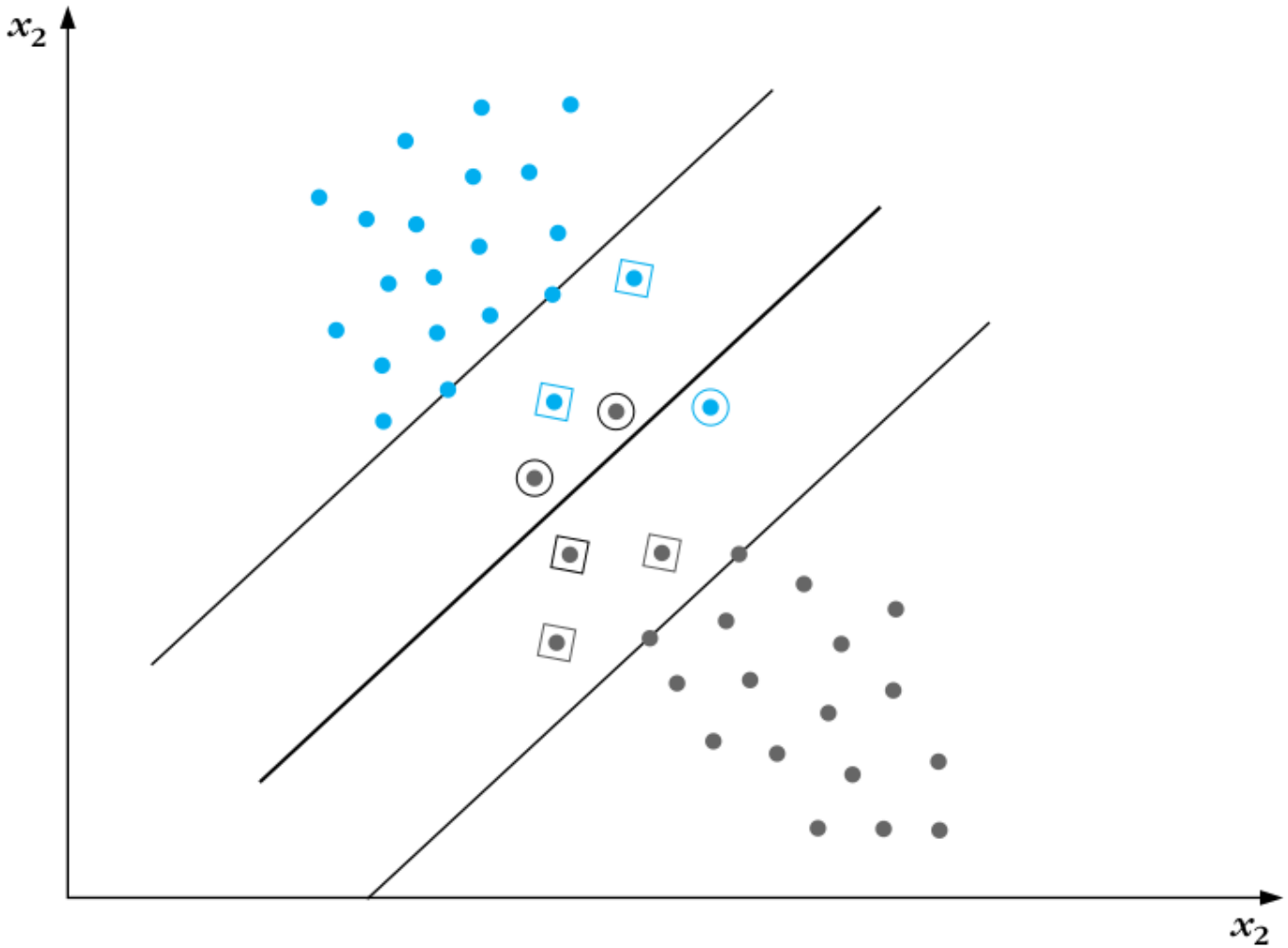
Support vector machines

- Once the optimal Lagrange multipliers have been computed, the optimal hyperplane is obtained via

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

- The cost function does not depend explicitly on the dimensionality of the input space!
- Although the resulting optimal hyperplane is unique, there is no guarantee about the uniqueness of the associated Lagrange multipliers λ_i
- When the classes are not separable, the above setup is no longer valid

Support vector machines



Support vector machines

- All cases can be treated under a single type of constraints by introducing a new set of variables, namely

$$y_i[\mathbf{w}^T\mathbf{x} + w_0] \geq 1 - \xi_i$$

- Vectors that fall outside the band and are correctly classified corresponds to $\xi_i = 0$, vectors falling inside the band and are correctly classified corresponds to $0 < \xi_i \leq 1$, and vectors that are misclassified corresponds to $\xi_i > 1$
- The variables ξ_i are known as *slack variables*
- The goal now is to make the margin as large as possible but at the same time to keep the number of points with $\xi > 0$ as small as possible

Support vector machines

- This is equivalent to minimize the cost function

$$J(\mathbf{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}|\mathbf{w}|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

where

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases}$$

- The parameter C is a positive constant that controls the relative influence of the two competing terms
- The optimization requires the Karush-Kuhn-Tucker conditions and the associated Wolfe dual representation

Support vector machines

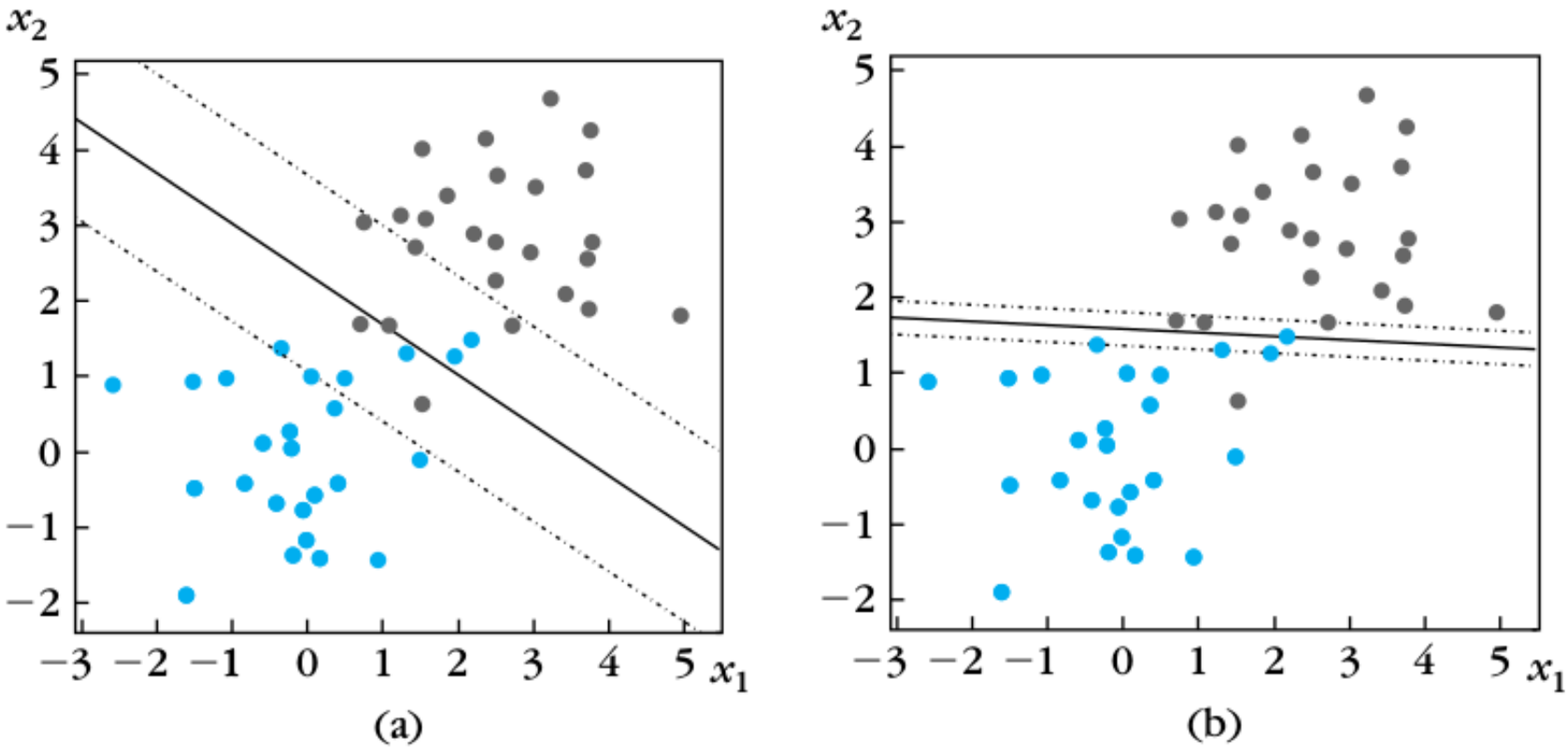
- We end up with

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \right)$$

$$\text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0, \quad 0 \leq \lambda_i \leq C, \; i = 1, 2, \ldots, N$$

- Note that the Lagrange multipliers corresponding to the points residing either within the margin or on the wrong side of the classifier

Support vector machines



Two non-separable classes and the resulting SVM linear classifier with the associated margin for *(a)* $C = 0.2$ and *(b)* $C = 1000$

Support vector machines

- In an M-class problem, a straightforward extension is to consider it as a set of M two-class problems (*one-against-all*)
- For each one of the classes, we seek to design an optimal discriminant function, $g_i(\mathbf{x}),\ i = 1, 2, \ldots, M$, so that $g_i(\mathbf{x}) > g_j(\mathbf{x}),\ \forall j \neq i$, if $\mathbf{x} \in \omega_i$
- Classification is then achieved according to the following rule

$$\text{assign } \mathbf{x} \text{ in } \omega_i \text{ if } i = \max_k \{g_k(\mathbf{x})\}$$

- This technique, however, may lead to indeterminate regions, where more than one $g_i(\mathbf{x})$ is positive
- Another drawback of the technique is that each binary classifier deals with a rather asymmetric problem

Support vector machines

- An alternative technique is the *one-against-one*
- In this case, $M(M-1)/2$ binary classifiers are trained and each classifier separates a pair of classes
- The decision is made on the basis of a majority vote
- The obvious disadvantage of the technique is that a relatively large number of binary classifiers has to be trained
- A different and very interesting rationale has been adopted: the multi-class task is treated in the context of error correcting coding, inspired by the coding schemes used in communications
- Thus, the pattern is classified to the class corresponding to the smallest Hamming distance

Support vector machines

- Remarks
  - The only difference between the linearly separable and non-separable cases lies in the fact that for the latter one the Lagrange multipliers need to be bounded above by C
  - A major limitation of support vector machines is the high computational burden required, both during training and in the test phase
  - *Sequential Minimal Optimization* algorithm is proposed where the idea of decomposition is pushed to its extreme and each working set consists of only two points

Outline

Bayes classifier

- Now, we will design classifiers that classify an unknown pattern in the most probable of the classes
- Thus, our task now becomes that of defining what 'most probable' means
- Given a classification task of M classes, $\omega_1, \omega_2, \ldots, \omega_M$, and an unknown pattern, which is represented by a feature vector $\mathbf{x}$, we form the M conditional probabilities

$$P(\omega_i|\mathbf{x}),\ i = 1, 2, \ldots, M$$

- Sometimes, these are also referred to as *a posteriori probabilities*

Bayes classifier

- Who could then argue that these conditional probabilities are not sensible choices to quantify the term *most probable*?
- Indeed, the classifiers to be considered in this chapter compute either the maximum of these M values or, equivalently, the maximum of an appropriately defined function of them
- The unknown pattern is then assigned to the class corresponding to this maximum
- We will initially focus on the two-class case: Let ω_1, ω_2 be the two classes in which our patterns belong

Bayes decision theory

- We assume that the a priori probabilities $P(\omega_1), P(\omega_2)$ are known
- If N is the total number of available training patterns, and N_1, N_2 of them belong to ω_1 and ω_2, respectively, then $P(\omega_1) \approx N_1/N$ and $P(\omega_2) \approx N_2/N$
- The other statistical quantities assumed to be known are the class-conditional probability density functions $p(\mathbf{x}|\omega_i)$, $i = 1, 2$, describing the distribution of the feature vectors in each of the classes
- If these are not known, they can also be estimated from the available training data

Bayes decision theory

- The pdf $p(\mathbf{x}|\omega_i)$ is sometimes referred to as the *likelihood function* of ω_i with respect to $\mathbf{x}$
- In the case that feature vectors can take only discrete values, density functions $p(\mathbf{x}|\omega_i)$ become probabilities and will be denoted by $P(\mathbf{x}|\omega_i)$
- We now have all the ingredients to compute our conditional probabilities using the Bayes rule

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

where

$$p(\mathbf{x}) = \sum_{i=1}^{2} p(\mathbf{x}|\omega_i)P(\omega_i)$$

Bayes decision theory

- The Bayes classification rule can now be stated as

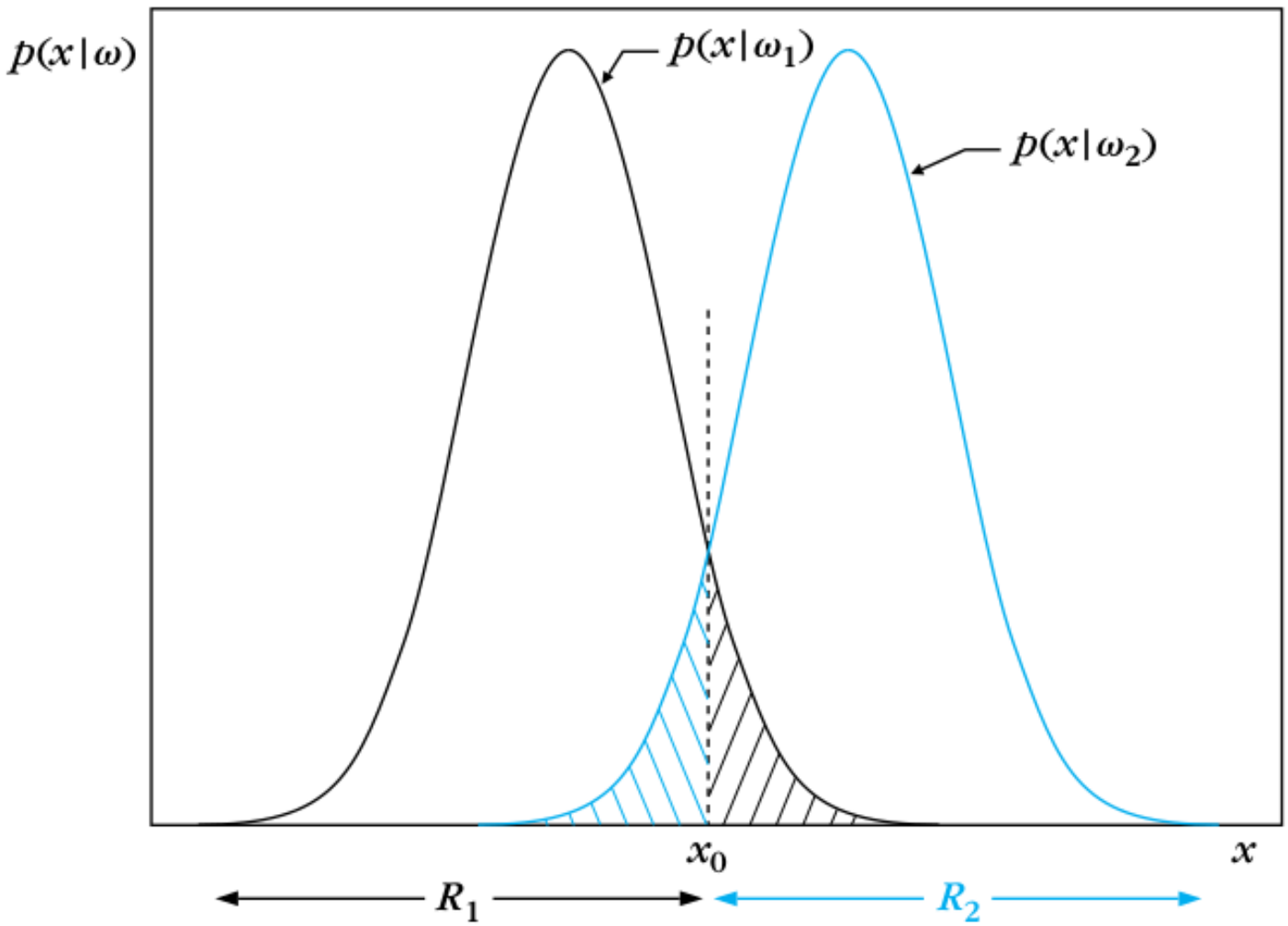
$$p(\mathbf{x}|\omega_1)P(\omega_1) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} p(\mathbf{x}|\omega_2)P(\omega_2)$$

- Furthermore, if the a priori probabilities are equal, that is, $P(\omega_1) = P(\omega_2) = 0.5$, then

$$p(\mathbf{x}|\omega_1) \underset{\omega_2}{\overset{\omega_1}{\gtrless}} p(\mathbf{x}|\omega_2)$$

- Thus, the search for the maximum now rests on the values of the conditional pdfs evaluated at $\mathbf{x}$
- It is obvious from the next figure that decision errors are unavoidable

Bayes decision theory



Bayes decision theory

- The total probability of committing a decision error for the case of 2 equiprobable classes is given by

$$P_e = \frac{1}{2}\int_{-\infty}^{x_0} p(\mathbf{x}|\omega_2)dx + \frac{1}{2}\int_{x_0}^{\infty} p(\mathbf{x}|\omega_1)dx$$

- We can show that the Bayesian classifier is optimal with respect to minimizing the classification error probability
- However, the classification error probability is not always the best criterion to be adopted for minimization
  - This is because it assigns the same importance to all errors
  - There are cases in which some wrong decisions may have more serious implications than others

Bayes decision theory

- In such cases it is more appropriate to assign a penalty term to weigh each error
- Thus, we will minimize a modified version of it, that is

$$r = \lambda_{12} P(\omega_1) \int_{R_2} p(\mathbf{x}|\omega_1) dx + \lambda_{21} P(\omega_2) \int_{R_1} p(\mathbf{x}|\omega_2) dx$$

  where each of the two terms that contributes to the overall error probability is weighted according to its significance
- For instance, let us denote by ω_1 the class of malignant tumors and as ω_2 the class of the benign ones: a reasonable choice would be to have $\lambda_{12} > \lambda_{21}$

Bayes decision theory

- Let us now consider an M-class problem and let R_j, $j = 1, \ldots, M$, be the regions of the feature space assigned to classes ω_j, respectively
- A penalty term λ_{ki}, known as *loss*, is associated with a wrong decision
- The *risk* or *loss* associated with ω_k is defined as

$$r_k = \sum_{i=1}^{M} \lambda_{ki} \int_{R_i} p(\mathbf{x}|\omega_k) dx$$

- Then, the *average risk* is defined as

$$r = \sum_{k=1}^{M} r_k P(\omega_k)$$

Bayes decision theory

- Minimizing the average risk becomes equivalent to minimizing the classification error probability
- For the two-class case

$$(\lambda_{21} - \lambda_{22})p(\mathbf{x}|\omega_2)P(\omega_2) \underset{\omega_1}{\overset{\omega_2}{\gtrless}} (\lambda_{12} - \lambda_{11})p(\mathbf{x}|\omega_1)P(\omega_1)$$

- It is also natural to assume that $\lambda_{ij} > \lambda_{ii}$

$$\frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} \underset{\omega_2}{\overset{\omega_1}{\lessgtr}} \frac{P(\omega_2)(\lambda_{21} - \lambda_{22})}{P(\omega_1)(\lambda_{12} - \lambda_{11})}$$

- This ratio is known as the *likelihood ratio* and the preceding test as the *likelihood ratio test*

Bayes decision theory

- An alternative cost that sometimes is used for two class problems is the Neyman-Pearson criterion
- The error for one of the classes is now constrained to be fixed and equal to a chosen value
- Such a decision rule has been used, for example, in radar detection problems
- One type of error is the so-called *false alarm*–that is, to mistake the noise for a signal (target) present
- the other type of error is to miss the signal and to decide in favor of the noise (*missed detection*)

Discriminant functions and decision surfaces

- It is clear that minimizing either the risk or the error probability or the Neyman-Pearson criterion is equivalent to partitioning the feature space into M regions, for a task with M classes
- If regions R_i, R_j happen to be contiguous, then they are separated by a decision surface in the multidimensional feature space
- For the minimum error probability case, this is described by the equation

$$P(\omega_i|\mathbf{x}) - P(\omega_j|\mathbf{x}) = 0$$

- From the one side of the surface this difference is positive, and from the other it is negative

Discriminant functions and decision surfaces

- Sometimes, instead of working directly with probabilities (or risk functions), it may be more convenient, from a mathematical point of view, to work with an equivalent function of them, for example, $g_i(\mathbf{x}) \equiv f(P(\omega_i|\mathbf{x}))$, where $f(\cdot)$ is a monotonically increasing function
- $g_i(\mathbf{x})$ is known as a *discriminant function*
- The decision test is now stated as

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

- The decision surfaces, separating contiguous regions, are described by

$$g_{ij}(\mathbf{x}) \equiv g_i(\mathbf{x}) - g_j(\mathbf{x}) = 0, \quad i, j = 1, 2, \ldots, M, \quad i \neq j$$

Discriminant functions and decision surfaces

- We have approached the classification problem via Bayesian probabilistic arguments and the goal was to minimize the classification error probability or the risk
- However, not all problems are well suited to such approaches
- For example, in many cases the involved pdfs are complicated and their estimation is not an easy task
- In such cases, it may be preferable to compute decision surfaces directly by means of alternative costs
- Such approaches give rise to discriminant functions and decision surfaces, which are entities with no (necessary) relation to Bayesian classification, and they are, in general, suboptimal with respect to Bayesian classifiers

Bayesian classification for normal distributions

- The multivariate generalization of a Gaussian pdf in the l-dimensional space is given by

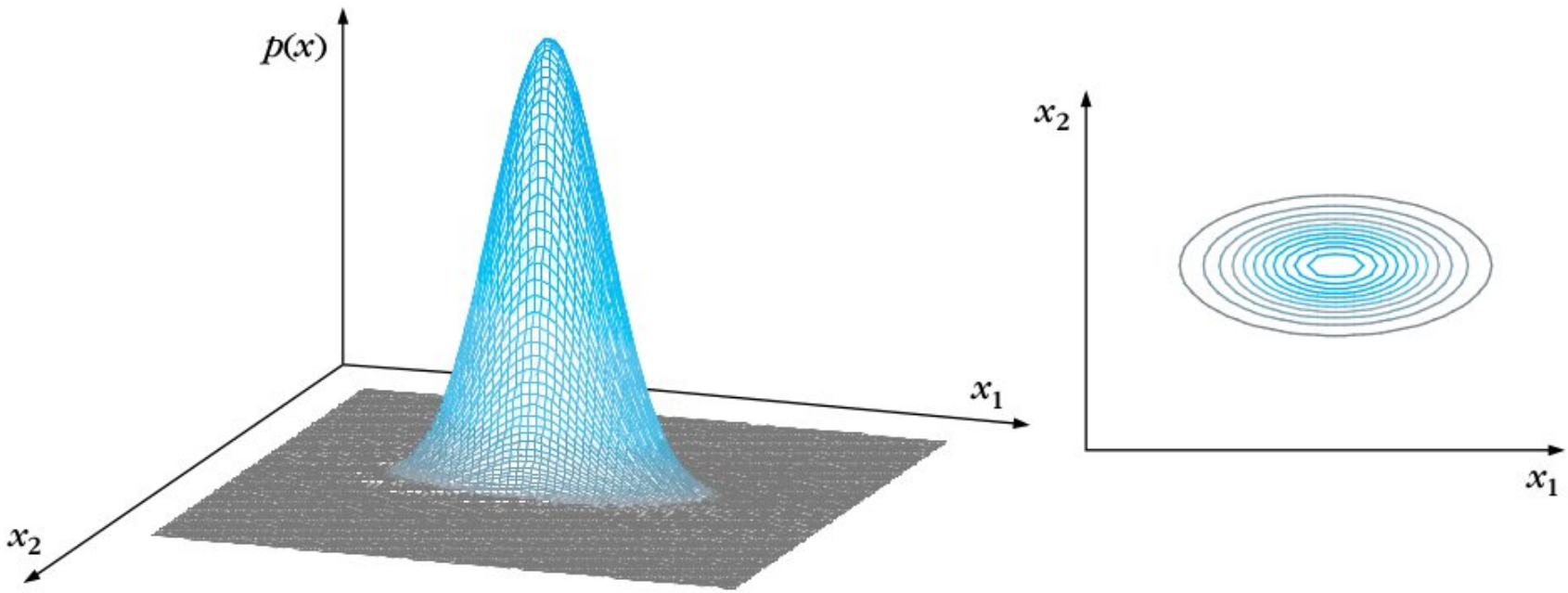
$$p(\mathbf{x}) = \frac{1}{(2\pi)^{l/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

where $\boldsymbol{\mu} = E[\mathbf{x}]$ is the mean value and Σ is the $l \times l$ *covariance matrix* defined as

$$\Sigma = E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T]$$

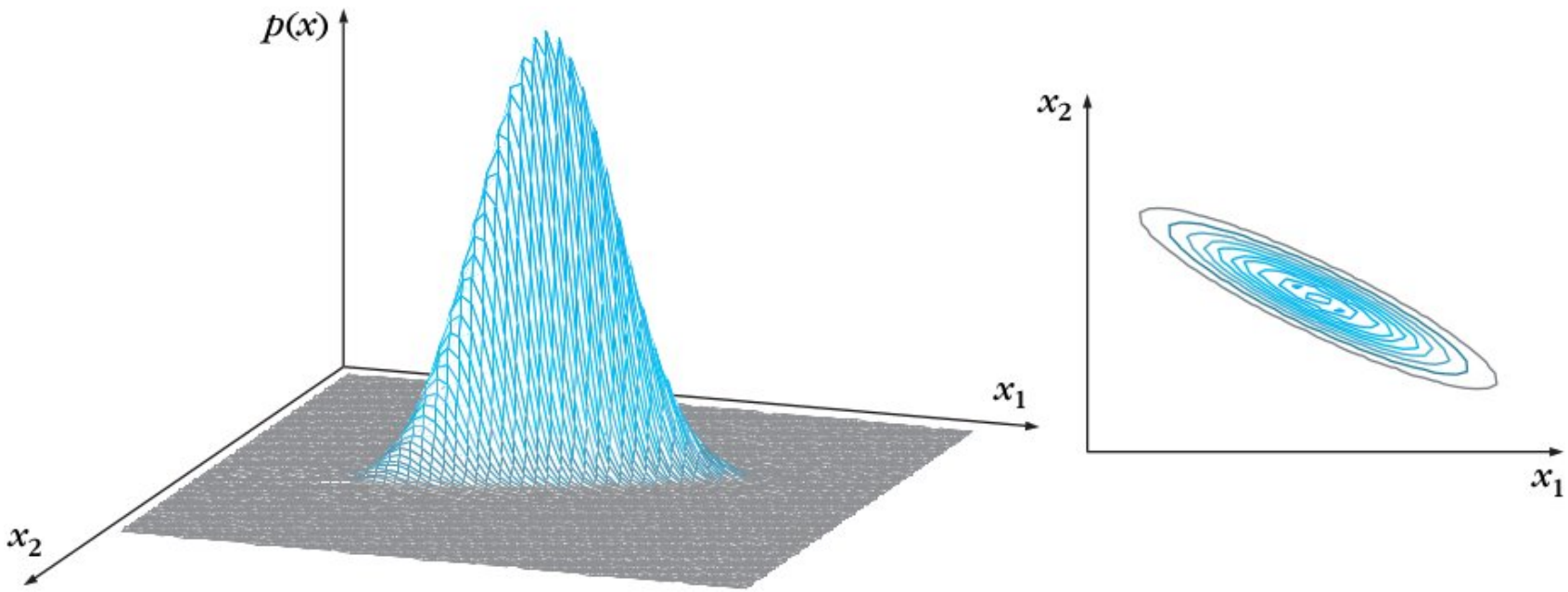
and $|\Sigma|$ denotes the determinant of Σ

Bayesian classification for normal distributions



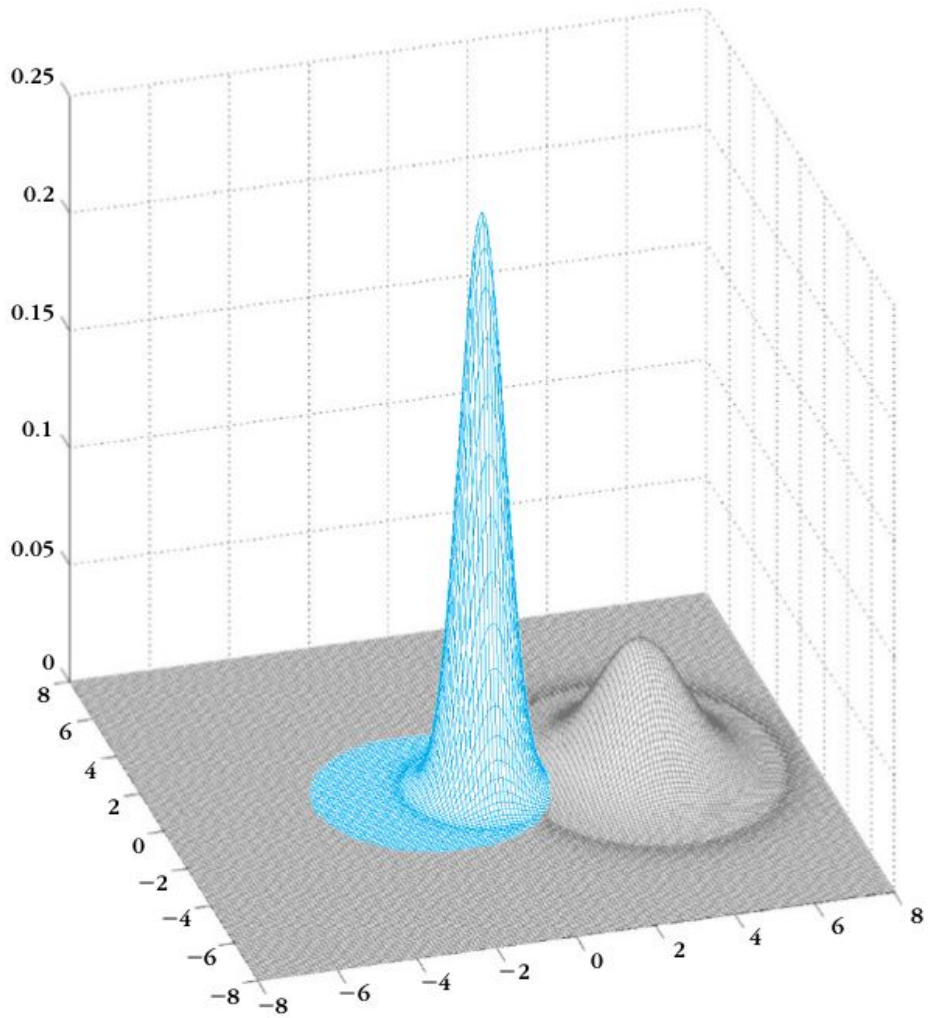
A two-dimensional Gaussian pdf and the corresponding isovalue curves for a diagonal Σ with $\sigma_1^2 < \sigma_2^2$

Bayesian classification for normal distributions



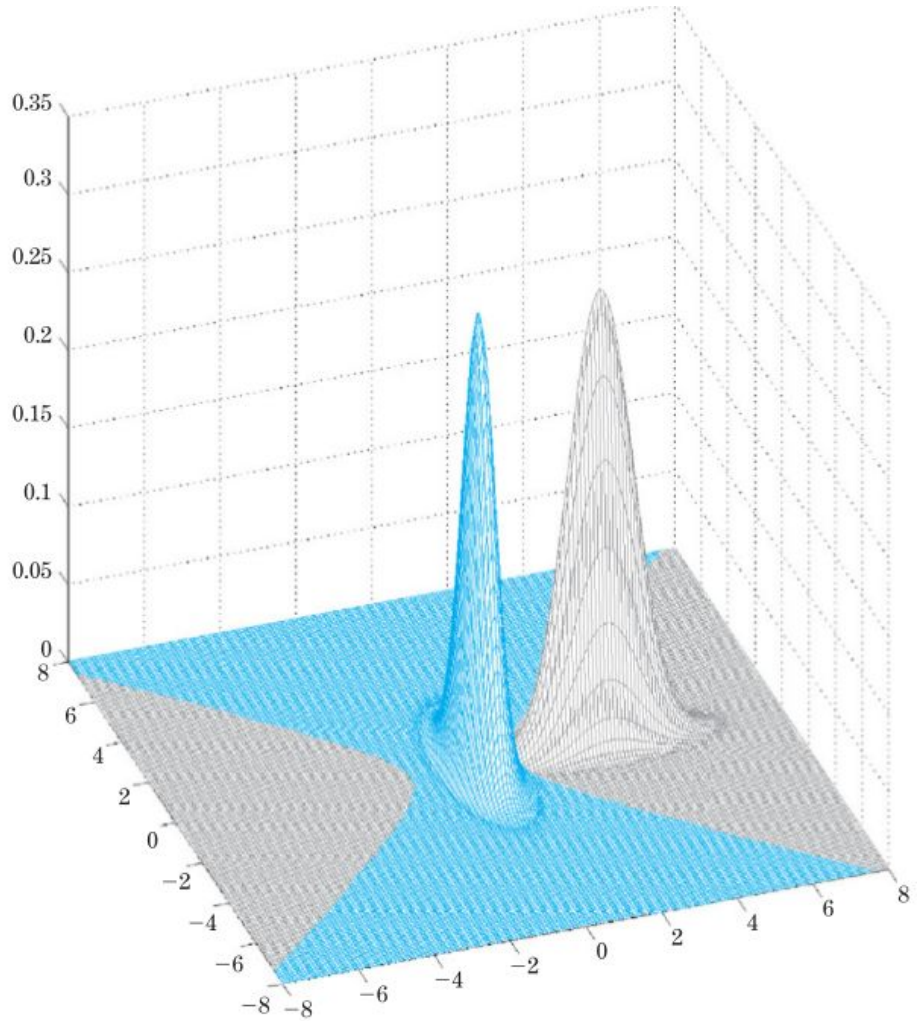
A two-dimensional Gaussian pdf and the corresponding isovalue curves for a case of a non-diagonal Σ

Bayesian classification for normal distributions



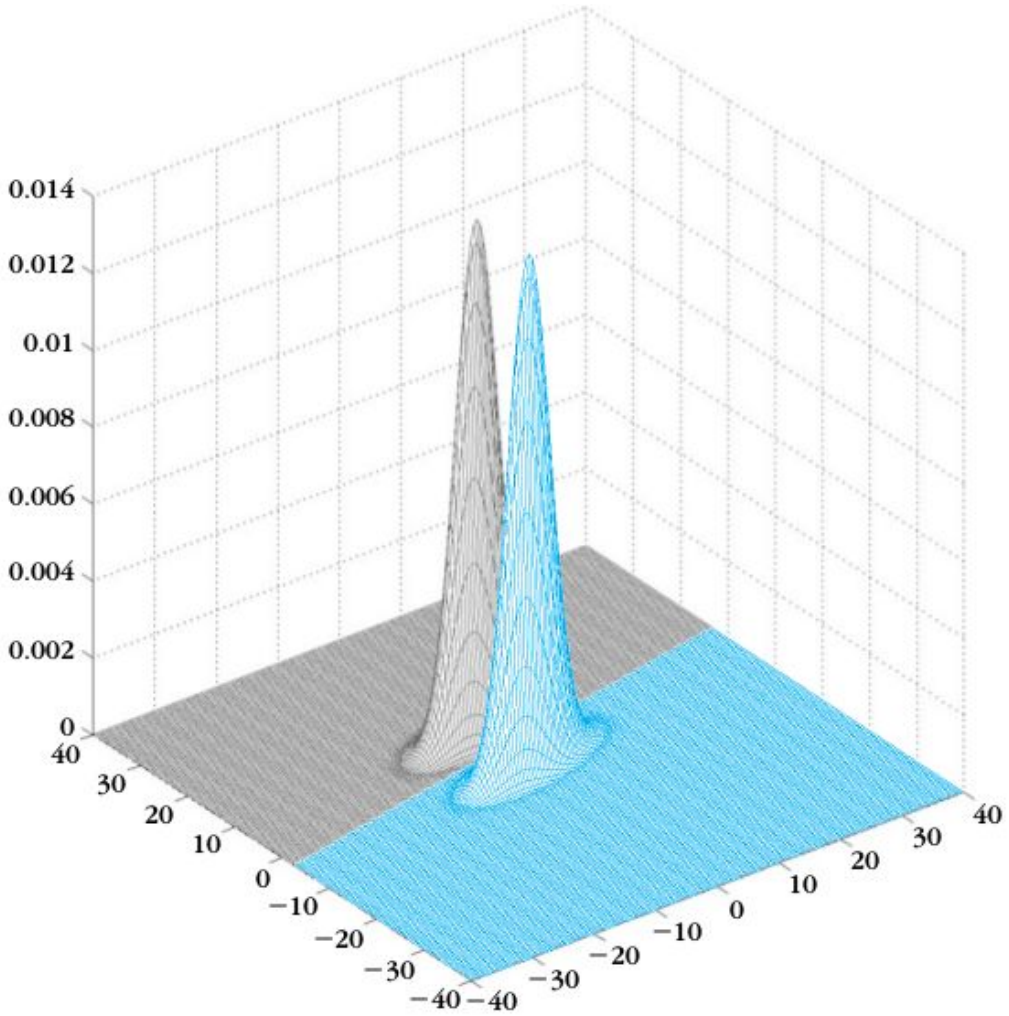
An example of the pdfs of two equiprobable classes in the two-dimensional space

Bayesian classification for normal distributions



An example of the pdfs of two equiprobable classes in the two-dimensional space

Bayesian classification for normal distributions



An example of the pdfs of two equiprobable classes in the two-dimensional space

Estimation of unknown probability density functions

- So far, we have assumed that the probability density functions are known
- However, this is not the most common case
- In many problems, the underlying pdf has to be estimated from the available data
- There are various ways to approach the problem
- Sometimes we may know the type of the pdf (*e.g.*, Gaussian, Rayleigh), but we do not know certain parameters, such as the mean values or the variances
- In contrast, in other cases we may not have information about the type of the pdf but we may know certain statistical parameters, such as the mean value and the variance

The naive-Bayes classifier

- In order to safeguard good estimates of the pdfs the number of training samples, N, must be large enough
- The demand for data increases exponentially fast with the dimension, l, of the feature space
- Accepting this reality, one has to make concessions about the degree of accuracy that is expected from the pdf estimates
- One widely used approach is to assume that individual features x_j, $j = 1, 2, \ldots, l$, are statistically independent
- Under this assumption, we can write

$$p(\mathbf{x}|\omega_i) = \prod_{j=1}^{l} p(x_j|\omega_i), \quad i = 1, 2, \ldots, M$$

The naive-Bayes classifier

- This leads to the so-called naive-Bayes classifier, which assigns an unknown sample $\mathbf{x} = [x_1, x_2, \ldots, x_l]^T$ to the class

$$\omega_m = \max_{\omega_i} \prod_{j=1}^{l} p(x_j|\omega_i), \quad i = 1, 2, \ldots, M$$

- It turns out that the naive-Bayes classifier can be very robust to violations of its independence assumption, and it has been reported to perform well for many real-world data sets
- Let see how this classifier can work even on nominal attributes

The naive-Bayes classifier

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

The naive-Bayes classifier

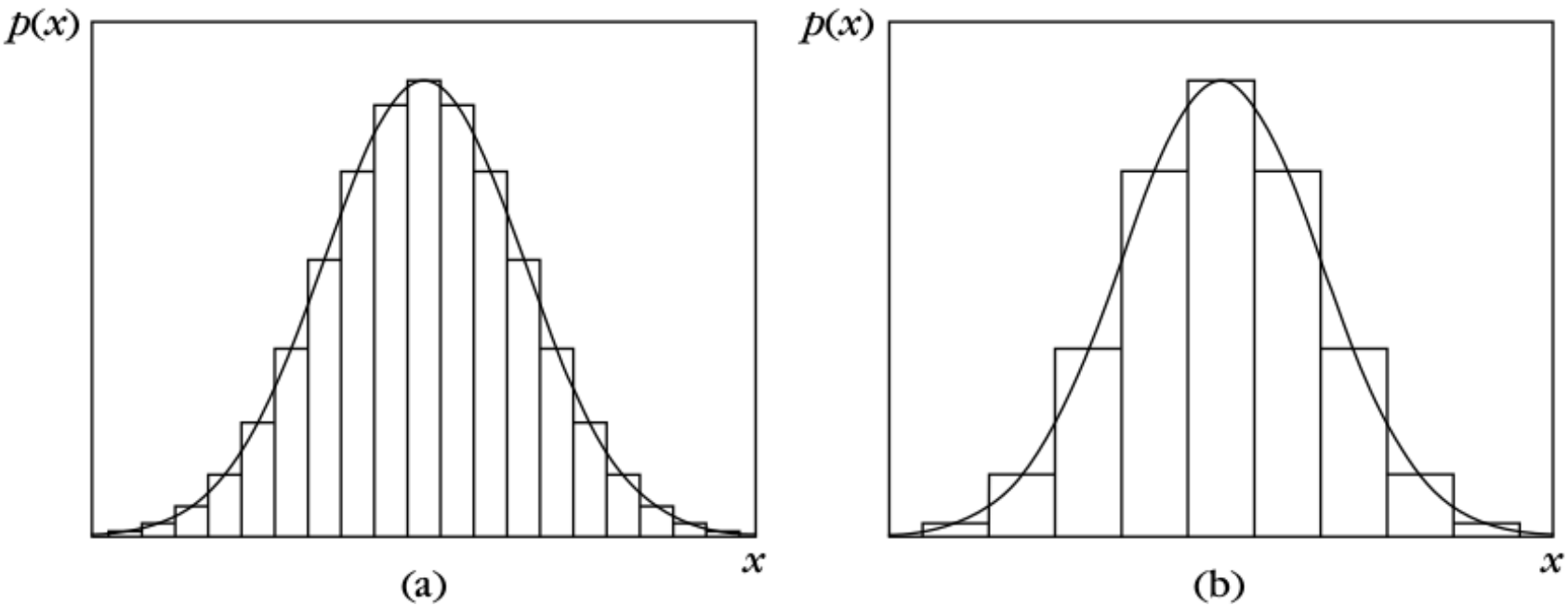
- The tuple we wish to classify is $\mathbf{x}$ = (age=youth, income=medium, student=yes, credit_rating=fair)
  - P(buys_computer=yes) = 9/14 = 0.643
    P(buys_computer=no) = 5/14 = 0.357
  - P(age=youth|buys_computer=yes) = 2/9 = 0.222
    P(age=youth|buys_computer=no) = 3/5 = 0.600
    P(income=medium|buys_computer=yes) = 4/9 = 0.444
    P(income=medium|buys_computer=no) = 2/5 = 0.400
    P(student=yes|buys_computer=yes) = 6/9 = 0.667
    P(student=yes|buys_computer=no) = 1/5 = 0.200
    P(credit_rating=fair|buys_computer=yes) = 6/9 = 0.667
    P(credit_rating=fair|buys_computer=no) = 2/5 = 0.400
  - P(buys_computer=yes|$\mathbf{x}$) = 0.028 $\Leftarrow$
    P(buys_computer=no|$\mathbf{x}$) = 0.007

Non-parametric Estimation

- Normally, a pdf parametric modeling is incorporated, in one way or another, and the associated unknown parameters are estimated
- The naive-Bayes classifier is already a non-parametric estimation: it estimates the probability density functions based on the available training set
- Other alternative is the probability density function approximation by the histogram method
- If N is the total number of samples and k_N of these are located inside a bin, the corresponding probability is approximated by the frequency ratio

$$P \approx \frac{k_N}{N}$$

Non-parametric Estimation

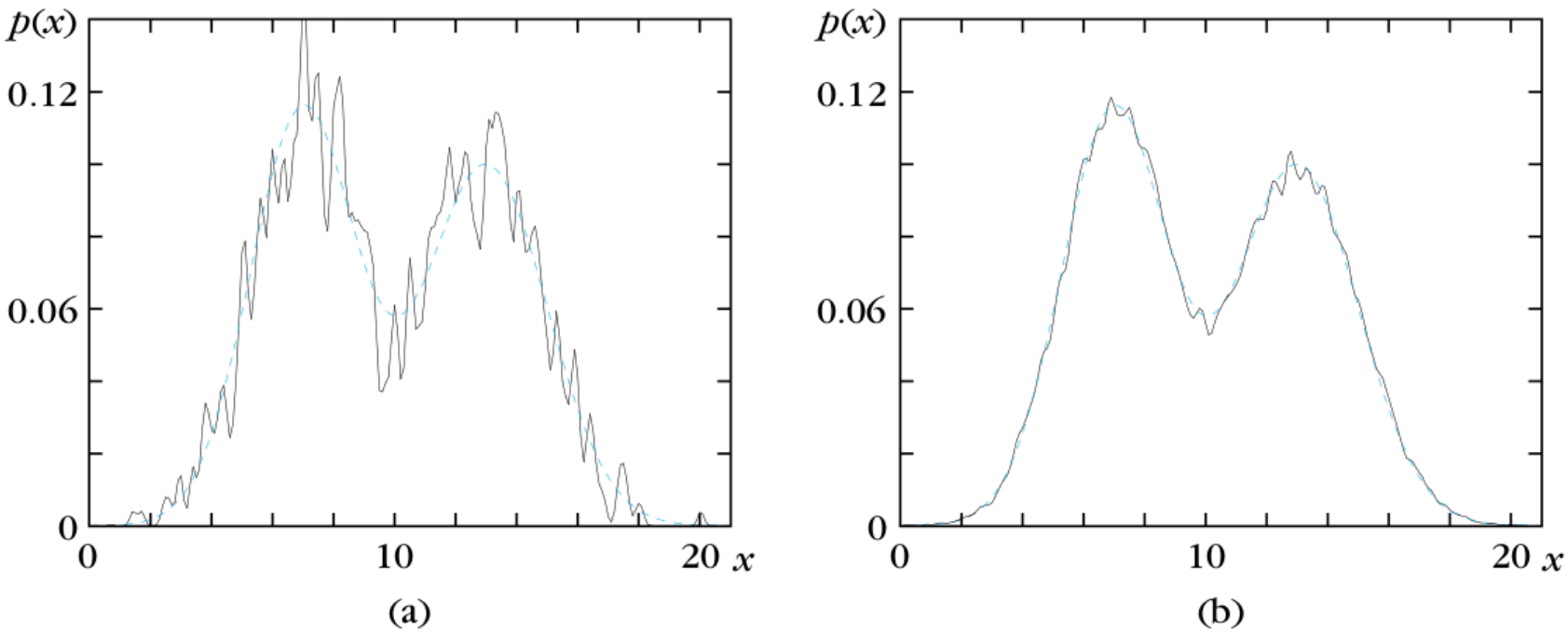


Probability density function approximation by the histogram method with *(a)* small and *(b)* large-size intervals (bins)

Non-parametric Estimation

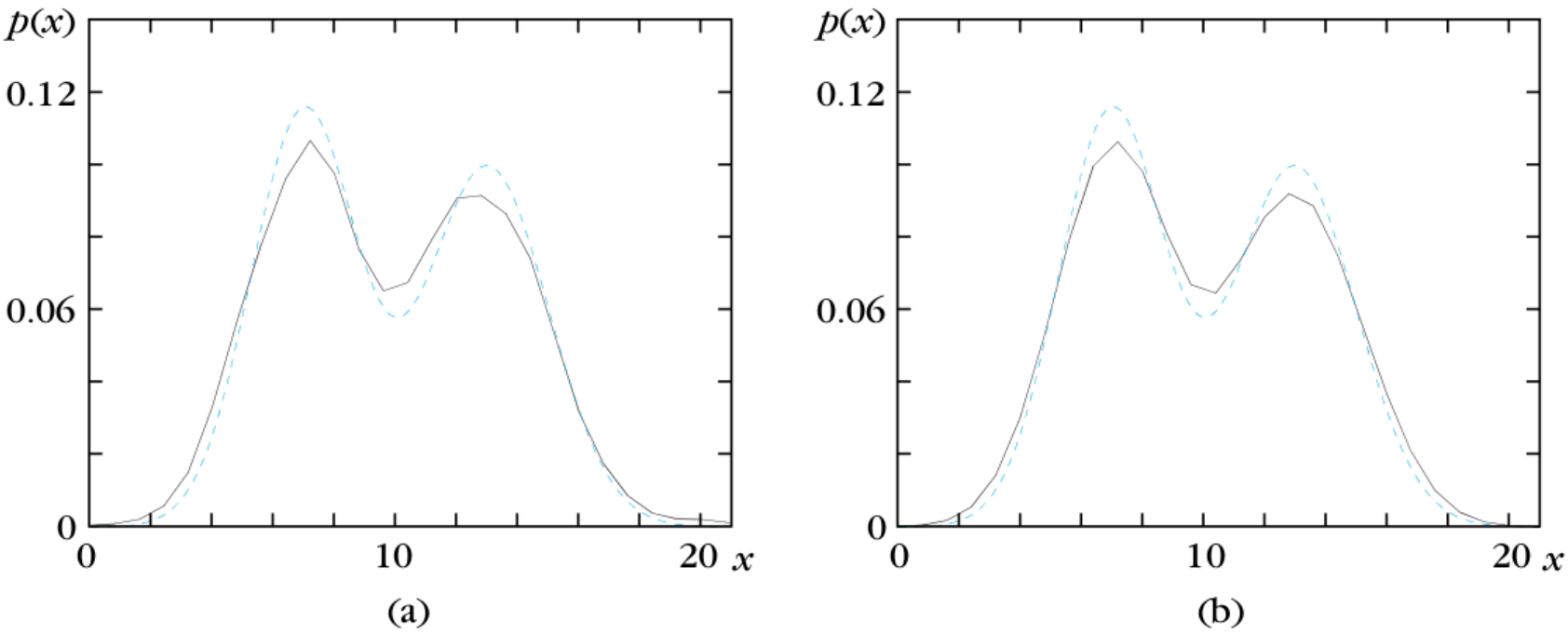
- In the multidimensional case, instead of bins of size h, the l-dimensional space is divided into hypercubes with length of side h and volume h^l
- In addition, we can try to approximate a continuous function $p(\mathbf{x})$ via an expansion in terms of discontinuous step functions $\phi(\cdot)$
- It can be shown that, provided an adequate smooth function, the resulting estimate is a legitimate pdf
- Such smooth functions are known as *kernels* or *potential functions* or *Parzen windows*
- A typical example is the Gaussian $N(0, 1)$ kernel

Non-parametric Estimation



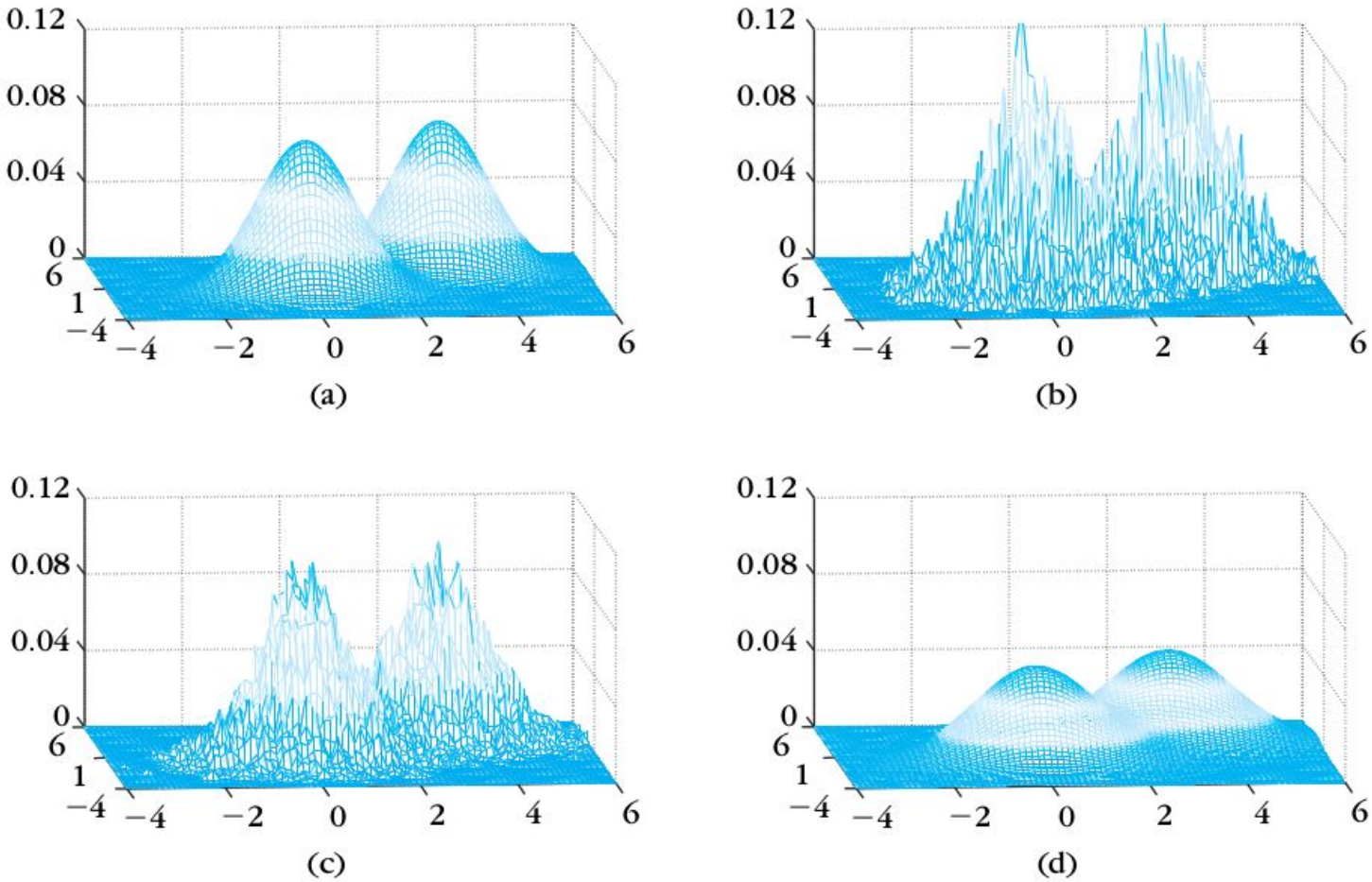
Approximation of a pdf via Parzen windows, using Gaussian kernels with *(a)* $h = 0.1$ and 1000 training samples and *(b)* $h = 0.1$ and 20000 samples

Non-parametric Estimation



Approximation of a pdf via Parzen windows, using Gaussian kernels with *(a)* $h = 0.8$ and 1000 training samples and *(b)* $h = 0.8$ and 20000 samples

Non-parametric Estimation



Approximation of *(a)* a two-dimensional pdf using Gaussian kernels with *(b)* $h = 0.05$ and 1000 training samples, *(c)* $h = 0.05$ and 20000 samples and *(d)* $h = 0.8$ and 20000 samples

Non-parametric Estimation

- On the reception of a feature vector $\mathbf{x}$ the likelihood text becomes

$$\frac{\frac{1}{N_1 h^l}\sum_{i=1}^{N_1}\phi\left(\frac{\mathbf{x}_i-\mathbf{x}}{h}\right)}{\frac{1}{N_2 h^l}\sum_{i=1}^{N_2}\phi\left(\frac{\mathbf{x}_i-\mathbf{x}}{h}\right)} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{P(\omega_2)}{P(\omega_1)}\frac{\lambda_{21}-\lambda_{22}}{\lambda_{12}-\lambda_{11}}$$

  where N_1, N_2 are the training vectors in class ω_1, ω_2, respectively

- The risk-related terms are ignored when the Bayesian minimum error probability classifier is used
- For large N_1, N_2 this computation is a very demanding job, in both processing time and memory requirements

k Nearest Neighbors

- In the Parzen estimation of the pdf, the volume around the points $\mathbf{x}$ was considered fixed (h^l) and the number of points k_N, falling inside the volume, was left to vary randomly from point to point
- Here we will reverse the roles:
  - The number of points $k_N = k$ will be fixed, and the size of the volume around $\mathbf{x}$ will be adjusted each time, to include k points
- Thus, in low-density areas the volume will be large and in high-density areas it will be small
- The estimator can now be written as

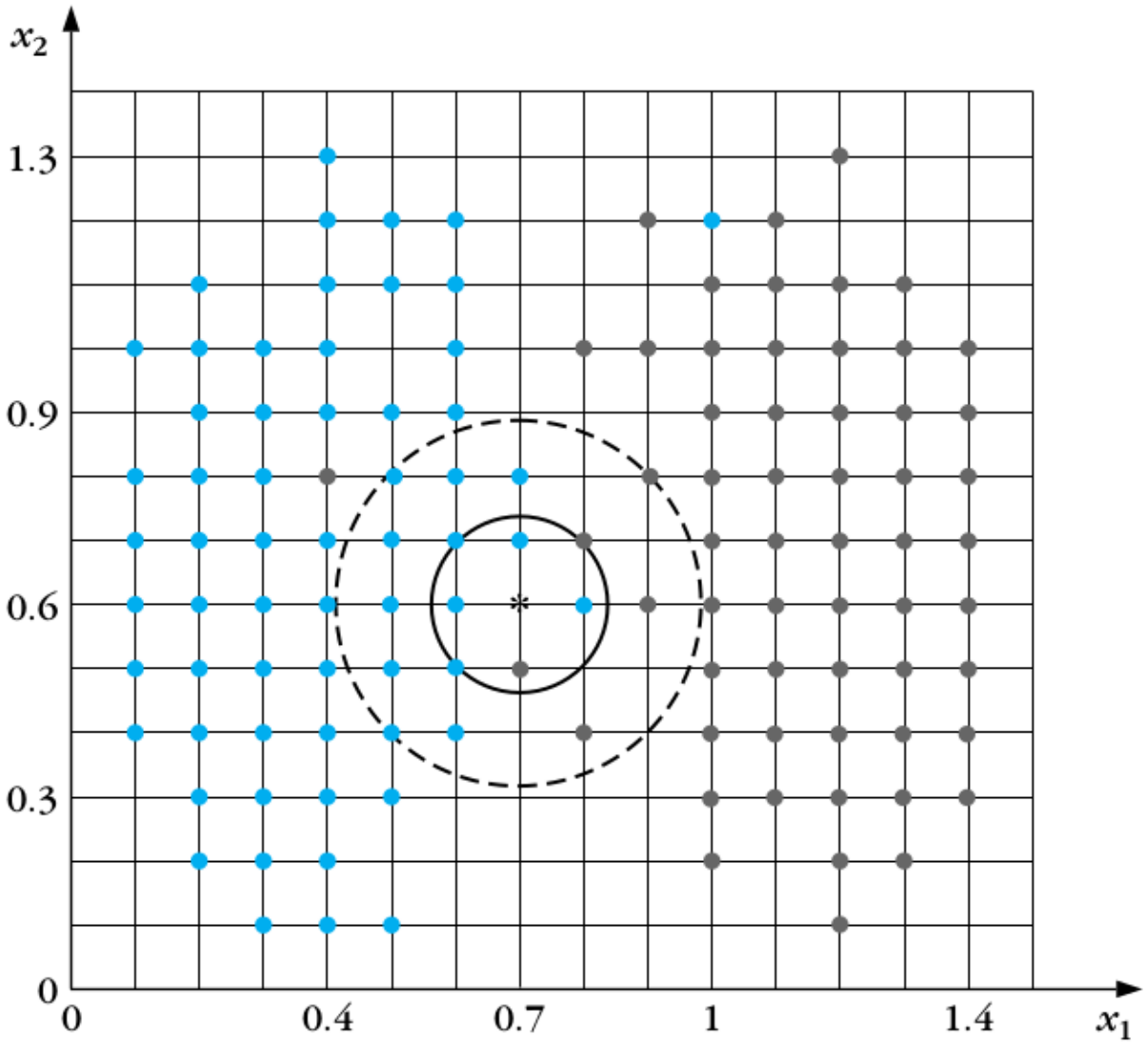
$$\hat{p}(\mathbf{x}) = \frac{k}{NV(\mathbf{x})}$$

k Nearest Neighbors

- From a practical point of view, at the reception of an unknown feature vector $\mathbf{x}$, we compute its distance d, for example, Euclidean, from all the training vectors of the various classes, for example, ω_1, ω_2
- Let r_1 be the radius of the hypersphere, centered at $\mathbf{x}$, that contains k points from ω_1 and r_2 the corresponding radius of the hypersphere containing k points from class ω_2 (k may not necessarily be the same for all classes)
- If we denote by V_1, V_2 the respective hypersphere volumes, the likelihood ratio test becomes

$$\frac{kN_2V_2}{kN_1V_1} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}}$$

k Nearest Neighbors



k Nearest Neighbors

- If the Mahalanobis distance is alternatively adopted, we will have hyper-ellipsoids in the place of the hyperspheres
- The volume of a hyper-ellipsoid, corresponding to Mahalanobis distance equal to r, is given by

$$V = V_0 |\Sigma|^{\frac{1}{2}} r^l$$

  where V_0 is the volume of the hypersphere of unit radius
- It is interesting to note that, although the performance of the methods, as density estimators, degrades in high-dimensional spaces due to the lack of sufficient data, their performance as classifiers may be sufficiently good
- After all, lack of enough training data points affects, in one way or another, all the methods

k Nearest Neighbors

- A variation of the kNN density estimation technique results in a suboptimal, yet popular in practice, nonlinear classifier
  - Although this does not fall in the Bayesian framework, it fits nicely at this point
- The algorithm for the so-called nearest neighbor rule is summarized as follows: Given an unknown feature vector $\mathbf{x}$ and a distance measure, then:
  - Out of the N training vectors, identify the k nearest neighbors, regardless of class label—k is chosen to be odd for a two class problem, and in general not to be a multiple of the number of classes M
  - Out of these k samples, identify the number of vectors, k_i, that belong to class ω_i, $i = 1, 2, \ldots, M$—Obviously, $\sum_i k_i = k$
  - Assign $\mathbf{x}$ to the class ω_i with the maximum number k_i of samples

k Nearest Neighbors

- Various distance measures can be used, including the Euclidean and Mahalanobis distance
- The simplest version of the algorithm is for $k = 1$, known as the nearest neighbor (NN) rule
- It can be shown that, as $N \to \infty$, the classification error probability, for the NN rule, P_{NN}, is bounded by

$$P_B \leq P_{NN} \leq 2P_B$$

  where P_B is the optimal Bayesian error
- The asymptotic performance of the kNN is better than that of the NN, and a number of interesting bounds have been derived
- These bounds suggest that as $k \to \infty$ the performance of the kNN tends to the optimal one

k Nearest Neighbors

- Thus,the kNN rule tends to the Bayesian classifier
  - Of course, all these are true asymptotically
- In the finite sample case there are even counterexamples where the kNN results in higher error probabilities than the NN
- However, in conclusion, it can be stated that the nearest neighbor techniques are among the serious candidates to be adopted as classifiers in a number of applications
- A serious drawback associated with (k)NN techniques is the complexity in search of the nearest neighbor(s) among the N available training samples
- Although, the kNN rule achieves good results when the data set is large, the performance of the classifier may degrade dramatically when the value of N is relatively small

k Nearest Neighbors

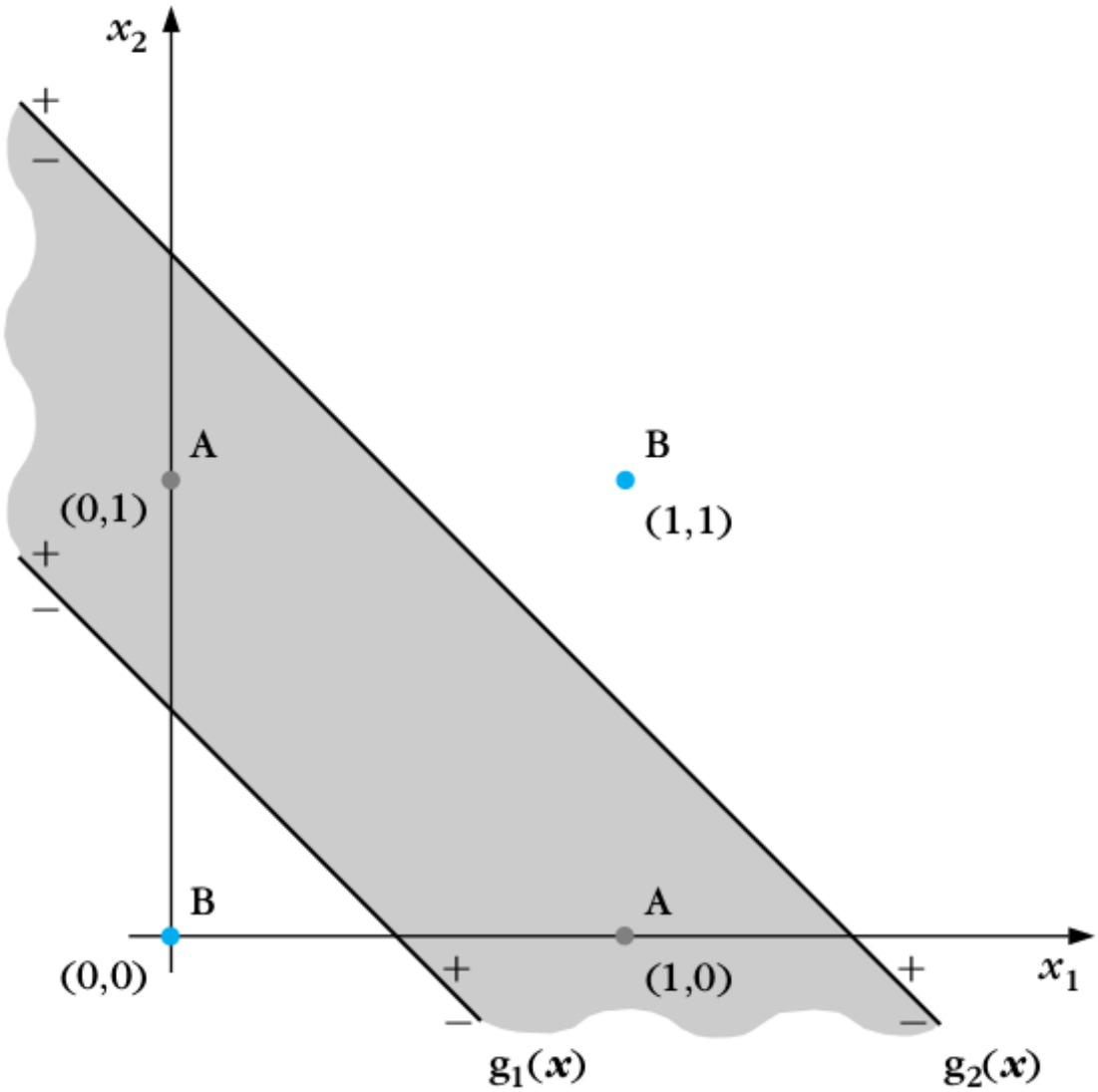
- A direction to cope with the performance degradation associated with small values of N is to employ distance measures that are optimized on the available training set
- The goal is to find a data-adaptive distance metric that leads to an optimal performance, according to an adopted cost
- Such trained metrics can be global ones (*i.e.*, the same at every point), class-dependent (*i.e.*, shared by all points of the same class), and/or locally dependent (*i.e.*, the metric varies according to the position in the feature space)
- See 'Data normalization' in the 'Feature selection' section

Outline

The 2-layer Perceptron

- To separate the two classes A and B in next figure, a first thought that comes to mind is to draw two, instead of one, straight lines
- So, what we need to do is to attack the problem in two successive phases
  - During the first phase, we calculate the position of a feature vector $\mathbf{x}$ with respect to each of the two decision lines
  - In the second phase, we combine the results of the previous phase and we find the position of $\mathbf{x}$ with respect to both lines, that is, outside or inside the shaded area
- Realization of the two decision lines (hyperplanes), $g_1(\cdot)$ and $g_2(\cdot)$, during the first phase of computations is achieved with the adoption of two perceptrons with inputs x_1, x_2 and appropriate synaptic weights

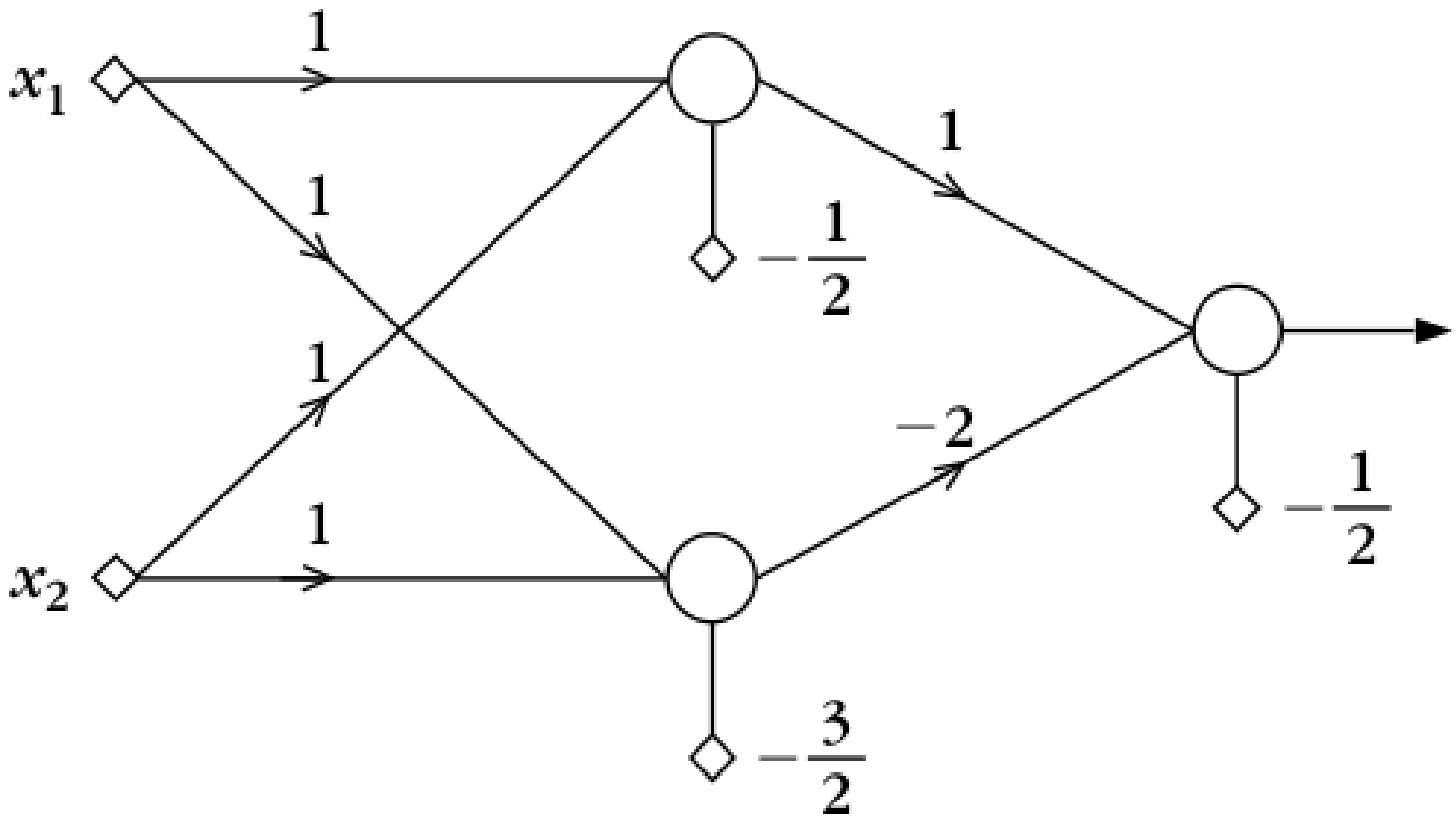
The 2-layer Perceptron



The 2-layer Perceptron

- The mapping of the first phase transforms the non-linearly separable problem to a linearly separable one
- The resulting multilayer architecture can be considered as a generalization of the perceptron, and it is known as a two-layer perceptron or a two-layer feedforward neural network
- The two neurons (nodes) of the first layer perform computations of the first phase and they constitute the so-called hidden layer
- The single neuron of the second layer performs the computing of the final phase and constitutes the output layer
- The input layer corresponds to the (nonprocessing) nodes where input data are applied

The 2-layer Perceptron



A two-layer perceptron solving the XOR problem

The 2-layer Perceptron

- For the more general case, we will consider input vectors in the l-dimensional space, that is, $\mathbf{x} \in \mathbb{R}^l$, and p neurons in the hidden layer
- Employing the step activation function, the mapping of the input space, performed by the hidden layer, is now onto the vertices of the hypercube of unit side length in the p-dimensional space, denoted by H_p
- The mapping of the input space onto the vertices of the hypercube is achieved via the creation of p hyperplanes
- The first layer of neurons divides the input l-dimensional space into polyhedra, 2 which are formed by hyperplane intersections
- All vectors located within one of these polyhedral regions are mapped onto a specific vertex of the unit H_p hypercube

The 2-layer Perceptron

- The output neuron subsequently realizes another hyperplane, which separates the hypercube into two parts, having some of its vertices on one and some on the other side
- This neuron provides the multilayer perceptron with the potential to classify vectors into classes consisting of unions of the polyhedral regions
- The inability of the two-layer perceptrons to separate classes resulting from any union of polyhedral regions springs from the fact that the output neuron can realize only a single hyperplane
- Solution: a three-layer perceptron architecture with two layers of hidden neurons and one output layer

The Backpropagation Algorithm

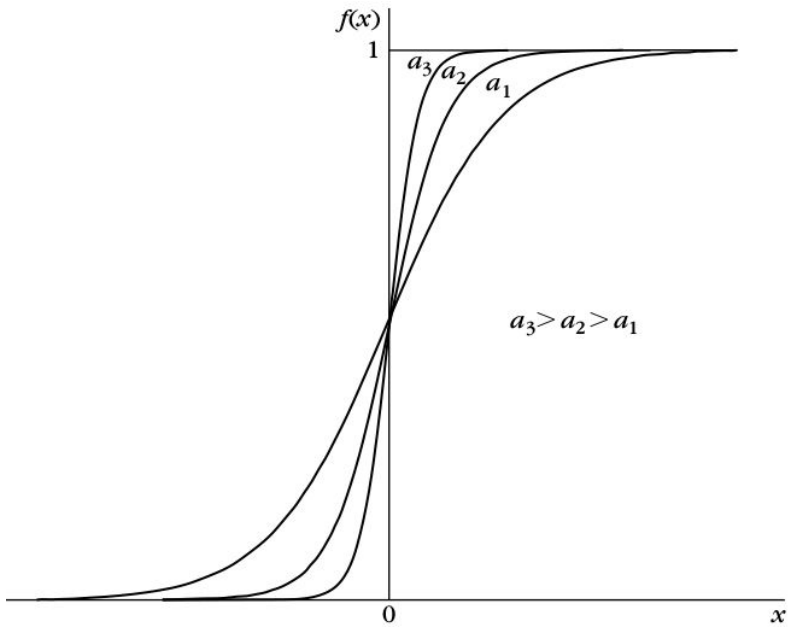
- Other alternative: Fix the architecture and compute its synaptic parameters so as to minimize an appropriate cost function of its output
- This is by far the most popular approach, which not only overcomes the drawback of the resulting large networks but also makes these networks powerful tools for a number of other applications, beyond pattern recognition
- But first, we need differentiable activation functions to minimize cost funtions
- A popular family of continuous differentiable functions,which approximate the step function, is the family of *sigmoid functions*

The Backpropagation Algorithm

- A typical representative is the *logistic function*

$$f(x) = \frac{1}{1 + e^{-ax}}$$

where a is a slope parameter



The Backpropagation Algorithm

- Let $\mathbf{w}_j^r$ be the weight vector (including the threshold) of the jth neuron in the rth layer, which is a vector of dimension $k_{r-1}+1$
- The basic iteration step will be of the form

$$\mathbf{w}_j^r(\text{new}) = \mathbf{w}_j^r(\text{old}) + \Delta\mathbf{w}_j^r$$

with

$$\Delta\mathbf{w}_j^r = -\mu\frac{\partial J}{\partial \mathbf{w}_j^r}$$

- The cost function has the form

$$J = \sum_{i=1}^{N} \xi(i)$$

where N training pairs $(y(i), x(i))$ have been used

The Backpropagation Algorithm

- An example very used for a cost function is

$$\xi(i) = \frac{1}{2}\sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L}(y_m(i) - \hat{y}_m(i))^2$$

- So, the update weight function becomes

$$\Delta \mathbf{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) y^{r-1}(i)$$

where

$$\delta_j^r(i) \equiv \frac{\partial \xi(i)}{\partial v_j^r(i)}$$

being $v_j^r(i)$ the argument of the activation function $f(\cdot)$ of the latter neuron

The Backpropagation Algorithm

- In other words

$$\delta_j^r(i) = e_j^r(i) f'(v_j^r(i))$$

and

$$e_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r$$

- In addition, for the logistic function we have

$$f'(x) = af(x)(1 - f(x))$$

- A number of criteria have been suggested for terminating the iterations
- The convergence speed of the backpropagation scheme depends on the value of the learning constant μ

The Backpropagation Algorithm

1. *Initialization*: Initialize all the weights with small random values
2. *Forward computations*: For each of the training feature vectors $x(i),\ i = 1, 2, \ldots, N$, compute all the $v_j^r(i)$, $y_j^r(i) = f(v_j^r(i))$, $j = 1, 2, \ldots k_r,\ r = 1, 2, \ldots, L$
   Compute the cost function for the current estimate of weights
3. *Backward computations*: For each $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, k_L$ compute $\delta_j^L(i)$ and in the sequel compute $\delta_j^{r-1}(i)$ for $r = L, L-1, \ldots, 2$, and $j = 1, 2, \ldots, k_r$
4. *Update the weights*: For $r = 1, 2, \ldots, L$ and $j = 1, 2, \ldots, k_r$ compute $\mathbf{w}_j^r(\text{new})$
5. Go to (2)

The Backpropagation Algorithm

- The cost function minimization for a multilayer perceptron is a nonlinear minimization task
- Hence, the backpropagation algorithm runs the risk of being trapped in a local minimum
- The algorithm described in this section updates the weights once all the training (input-desired output) pairs have appeared in the network
  - A variation of this approach is to update the weights for each of the training pairs
- Once training of the network has been achieved, the values to which the synapses and thresholds have converged are frozen, and the network is ready for classification

The Backpropagation Algorithm

- The backpropagation scheme inherits the disadvantage of all methods built on the gradient descent approach: their convergence to the cost function minimum is slow
- One way to overcome this problem is to use a *momentum term* that smoothes out the oscillatory behavior and speeds up the convergence

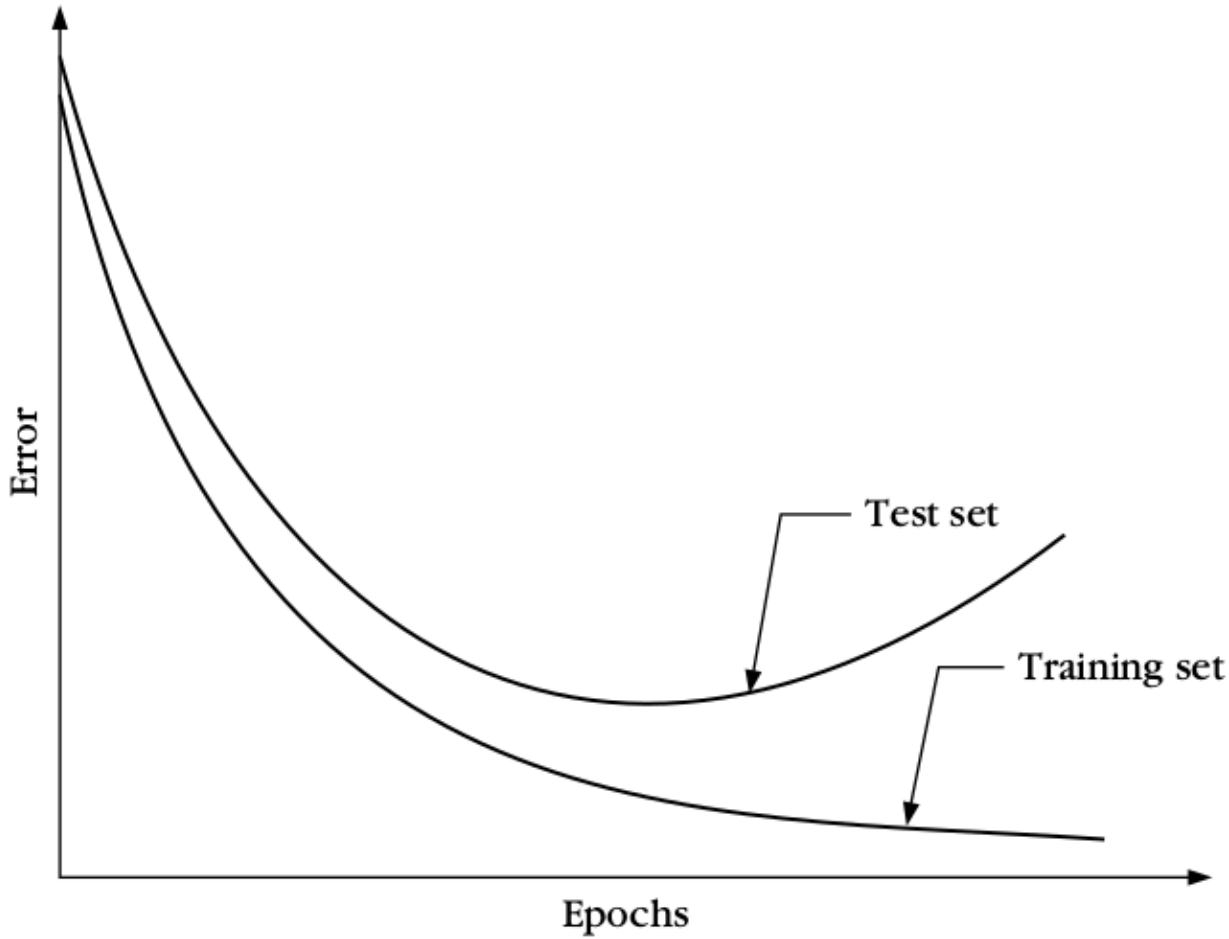
$$\Delta \mathbf{w}_j^r(new) = \alpha \Delta \mathbf{w}_j^r(old) - \mu \sum_i^N \delta_j^r(i) y^{r-1}(i)$$

- The constant α is called the *momentum factor* and in practice is chosen between 0.1 and 0.8

Choice of the Network Size

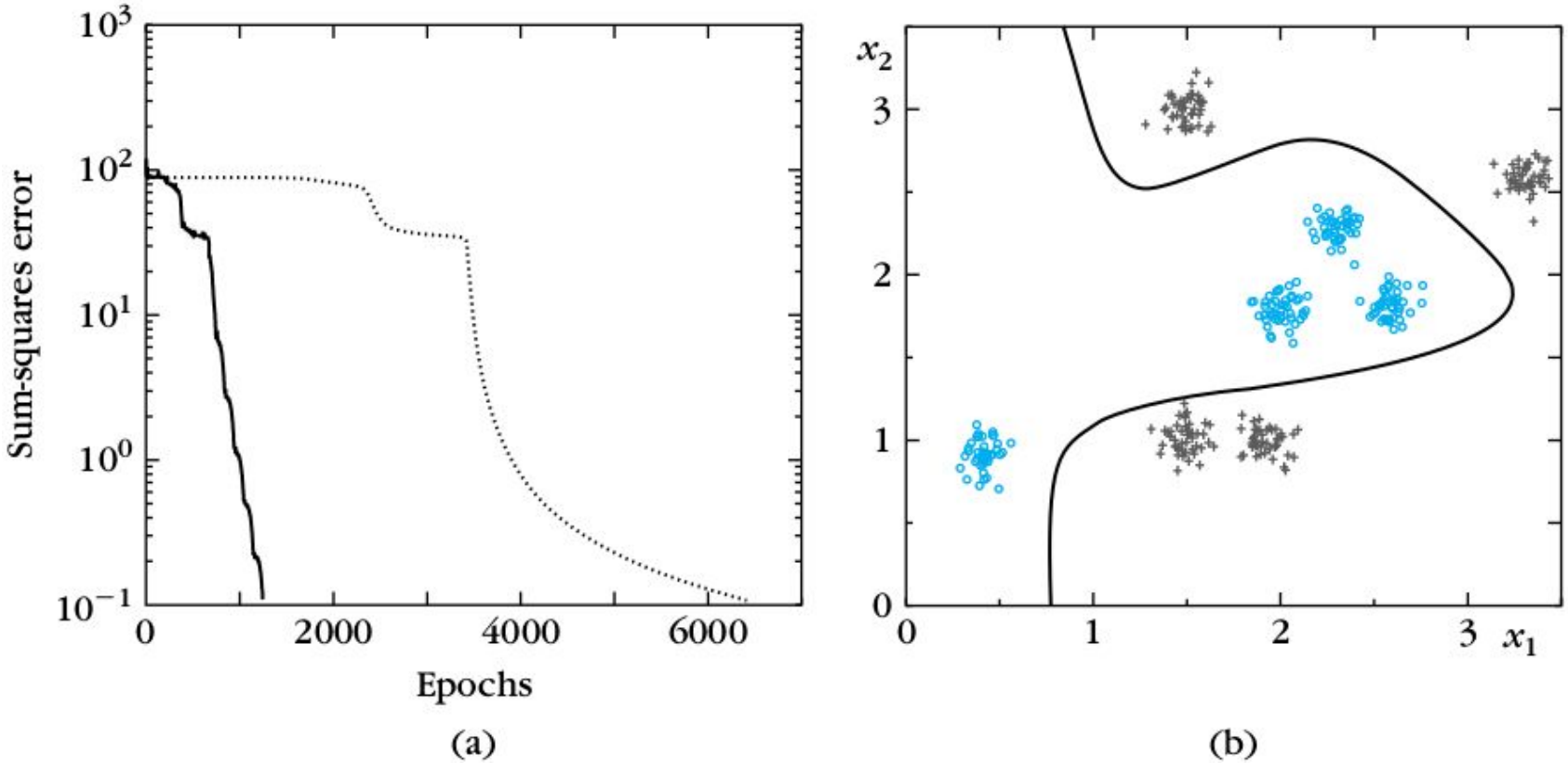
- One answer to this problem could be to choose the size of the network large enough and leave the training to decide about the weights
- However, besides the associated computational complexity problems, there is a major reason why the size of the network should be kept as small as possible
- This is imposed by the generalization capabilities that the network must possess
- Taking for granted the finite number N of training pairs, the number of free parameters (synaptic weights) to be estimated should be *(a)* large enough and *(b)* small enough

Choice of the Network Size



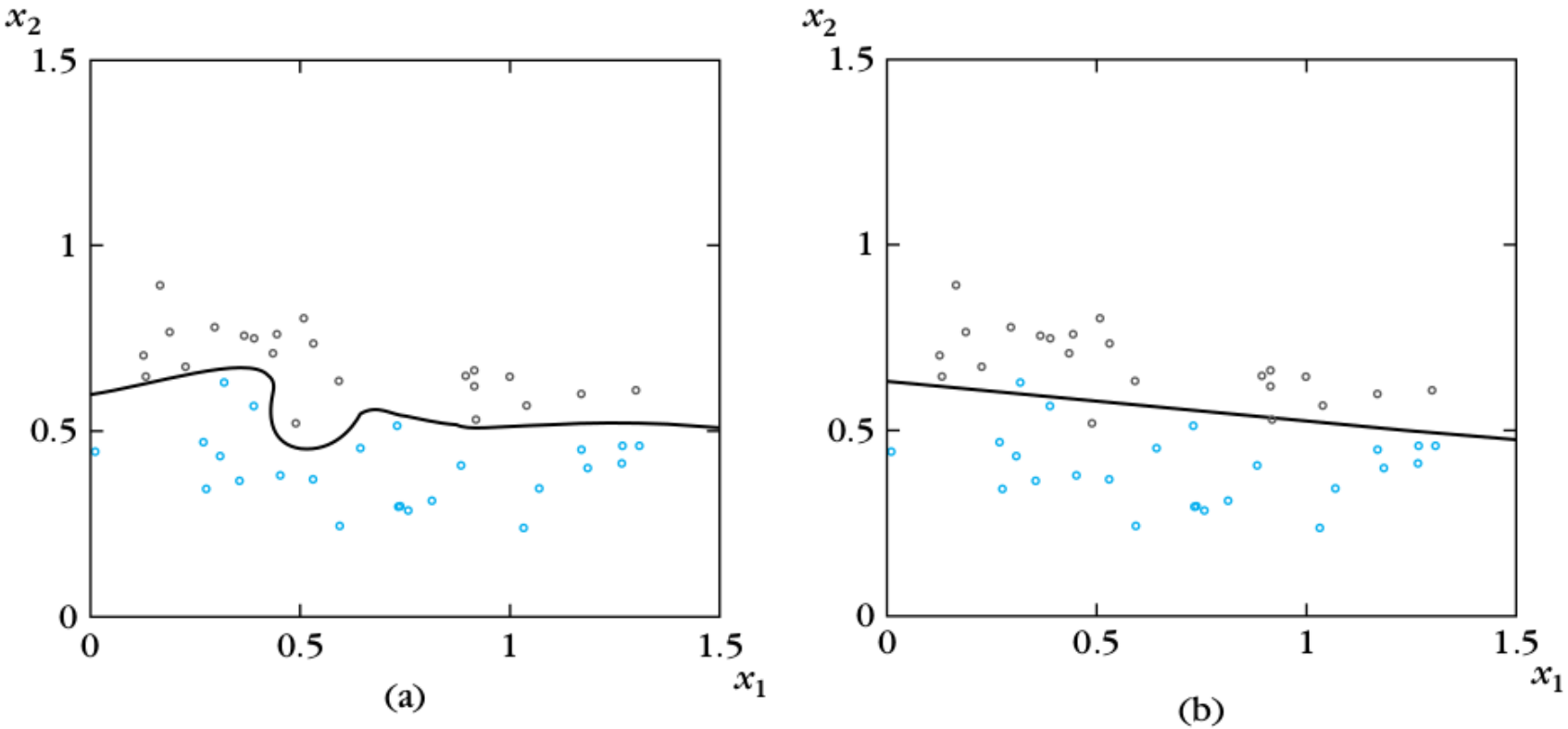
Trend of the output error versus the number of epochs illustrating overtraining of the training set

Examples of ANNs



(a) Error convergence curves for the adaptive momentum (dark line) and the momentum algorithms
(b) The decision curve formed by the multilayer perceptron

Examples of ANNs



Decision curve *(a)* before pruning and *(b)* after pruning

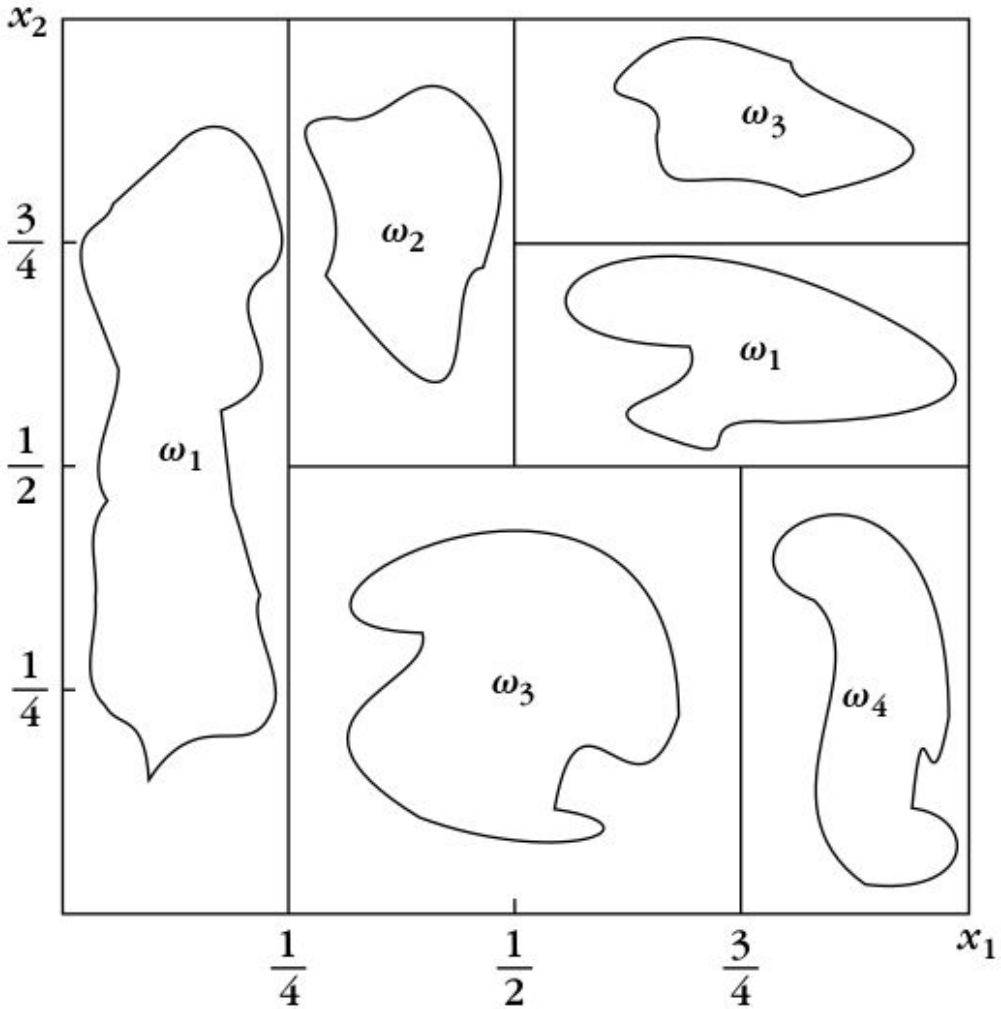
Decision Trees

- Another large class of nonlinear classifiers is known as *decision trees*
- These are *multistage* decision systems in which classes are sequentially rejected until we reach a finally accepted class
- To the end, the feature space is split into unique regions, corresponding to the classes, in a sequential manner
- Upon the arrival of a feature vector, the searching of the region to which the feature vector will be assigned is achieved via a sequence of decisions along a path of *nodes* of an appropriately constructed *tree*
- Such schemes offer advantages when a large number of classes are involved

Decision Trees

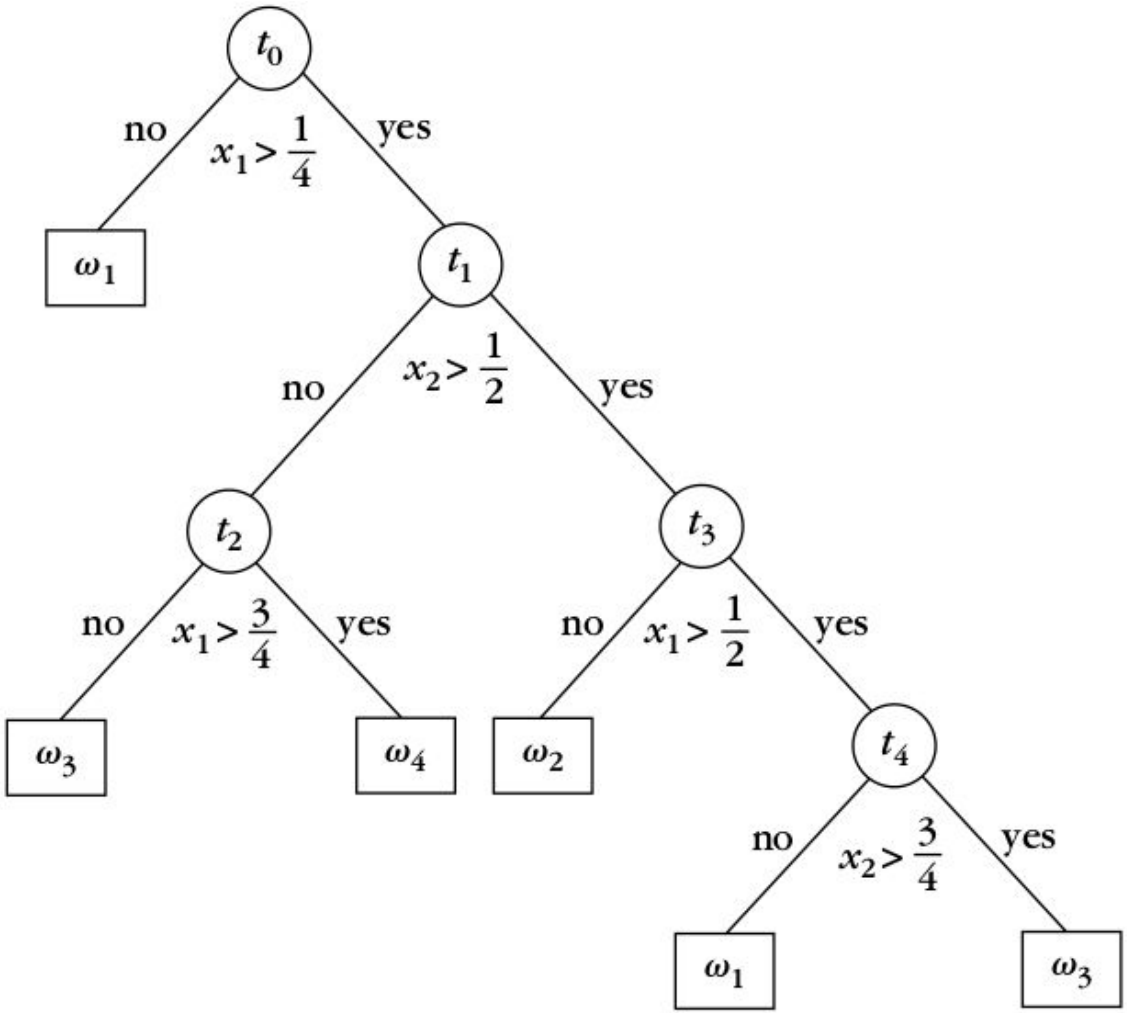
- The most popular decision trees are those that split the space into hyperrectangles with sides parallel to the axes
- The sequence of decisions is applied to individual features, and the questions to be answered are of the form "*is feature* $x_i \leq \alpha$" where α is a threshold value
- Such trees are known as *ordinary binary classification trees* (OBCTs)
- Other types of trees are also possible that split the space into convex polyhedral cells or into pieces of spheres
- The basic idea behind an OBCT is demonstrated via the following simplified example

Decision Trees



Decision tree partition of the space

Decision Trees



Decision tree classification

Decision Trees

- Note that it is possible to reach a decision without having tested all the available features
- In the general case, in order to develop a binary decision tree, the designer has to consider the following design elements in the training phase:
  - At each node, the set of candidate questions to be asked has to be decided. Each question corresponds to a specific binary split into two descendant nodes
  - A splitting criterion must be adopted according to which the best split from the set of candidate ones is chosen
  - A stop-splitting rule is required that controls the growth of the tree, and a node is declared as a terminal one (*leaf*)
  - A rule is required that assigns each leaf to a specific class

Set of Questions

- For the OBCT type of trees, the questions are of the form "Is $x_k \leq \alpha$?"
- For each feature, every possible value of the threshold α defines a specific split of the subset X_t
- Thus in theory, an infinite set of questions has to be asked if α varies in an interval $Y_\alpha \subseteq \mathbb{R}$
- In practice, only a finite set of questions can be considered
- For example, since the number, N, of training points in X is finite, any of the features x_k, $k = 1, \ldots, l$, can take at most $N_t \leq N$ different values, where N_t is the cardinality of the subset $X_t \subseteq X$

Set of Questions

- Thus, for feature x_k, one can use $\alpha_{kn},\ n = 1, 2, \ldots, N_{tk}$ ($N_{tk} \leq N_t$), where α_{kn} are taken halfway between consecutive distinct values of x_k in the training subset X_t
- The same has to be repeated for all features
- Thus in such a case, the total number of candidate questions is $\sum_{k=1}^{l} N_{tk}$
- However, only one of them has to be chosen to provide the binary split at the current node, t, of the tree
- This is selected to be the one that leads to the best split of the associated subset X_t
- The best split is decided according to a splitting criterion

Splitting Criterion

- Every binary split of a node, t, generates two descendant nodes
- Let us denote them by t_Y and t_N according to the 'Yes' or 'No' answer to the single question adopted for the node t, also referred as the ancestor node
- The descendant nodes are associated with two new subsets, that is, X_{t_Y} and X_{t_N}, respectively
- But every split must generate subsets that are more "class homogeneous" compared to the ancestor's subset X_t
- Therefore, the goal is to define a measure that quantifies node impurity and split the node so that the overall impurity of the descendant nodes is optimally decreased with respect to the ancestor node's impurity

Splitting Criterion

- A commonly used definition of node impurity, denoted as $I(t)$, is given by

$$I(t) = -\sum_{i=1}^{M} P(\omega_i|t) \log_2 P(\omega_i|t)$$

  where M is the number of classes

- This is nothing else than the entropy associated with the subset X_t, known from Shannon's Information Theory
- In practice, probabilities are estimated by the respective percentages, N_t^i/N_t, where N_t^i is the number of points in X_t that belong to class ω_i

Splitting Criterion

- Assume now that performing a split, N_{t_Y} points are sent into the 'Yes' node (X_{t_Y}) and N_{t_N} into the 'No' node (X_{t_N})
- The decrease in node impurity is defined as

$$\Delta I(t) = I(t) - \frac{N_{t_Y}}{N_t} I(t_Y) - \frac{N_{t_N}}{N_t} I(t_N)$$

  where $I(t_Y)$, $I(t_N)$ are the impurities of the t_Y and t_N nodes, respectively
- The goal now becomes to adopt, from the set of candidate questions, the one that performs the split leading to the highest decrease of impurity

Stop-Splitting Rule

- The natural question that now arises is when one decides to stop splitting a node and declares it as a leaf of the tree
- A possibility is to adopt a threshold T and stop splitting if the maximum value of $\Delta I(t)$, over all possible splits, is less than T
- Other alternatives are to stop splitting either if the cardinality of the subset X_t is small enough or if X_t is pure, in the sense that all points in it belong to a single class

Class Assignment Rule

- Once a node is declared to be a leaf, then it has to be given a class label
- A commonly used rule is the majority rule, that is, the leaf is labeled as ω_j where

$$j = \max_i P(\omega_i|t)$$

- In words, we assign a leaf, t, to that class to which the majority of the vectors in X_t belong
- Having discussed the major elements needed for the growth of a decision tree, we are now ready to summarize the basic algorithmic steps for constructing a binary decision tree

Basic Algorithm

- Begin with the root node, that is, $X_t = X$
- For each new node t
  1. For every feature $x_k,\ k = 1, 2, \ldots, l$
     - For every value $\alpha_{kn},\ n = 1, 2, \ldots, N_{tk}$

       Generate X_{t_Y} and X_{t_N} according to the answer in the question: is $x_k(i) \leq \alpha_{kn},\ i = 1, 2, \ldots, N_t$

       Compute the impurity decrease
     - Choose α_{kn_0} leading to the maximum decrease w.r.t. x_k
  2. Choose x_{k_0} and associated $\alpha_{k_0 n_0}$ leading to the overall maximum decrease of impurity
  3. If the stop-splitting rule is met, declare node t as a leaf and designate it with a class label
  4. If not, generate two descendant nodes t_Y and t_N with associated subsets X_{t_Y} and X_{t_N}, depending on the answer to the question: is $x_{k_0} \leq \alpha_{k_0 n_0}$

Basic Algorithm

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Basic Algorithm

- Expected information needed to classify a tuple in t

$$I(t) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$$

- Expected information needed to classify a tuple in t if the tuples are partitioned according to age

$$I(t_{age}) = \frac{5}{14}\left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) +$$

$$\frac{4}{14}\left(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right) +$$

$$\frac{5}{14}\left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) = 0.694$$

Basic Algorithm

- Hence, the gain in information from such a partitioning would be

$$\Delta I(t_{age}) = I(t) - I(t_{age}) = 0.246$$

- Similarly, we can compute

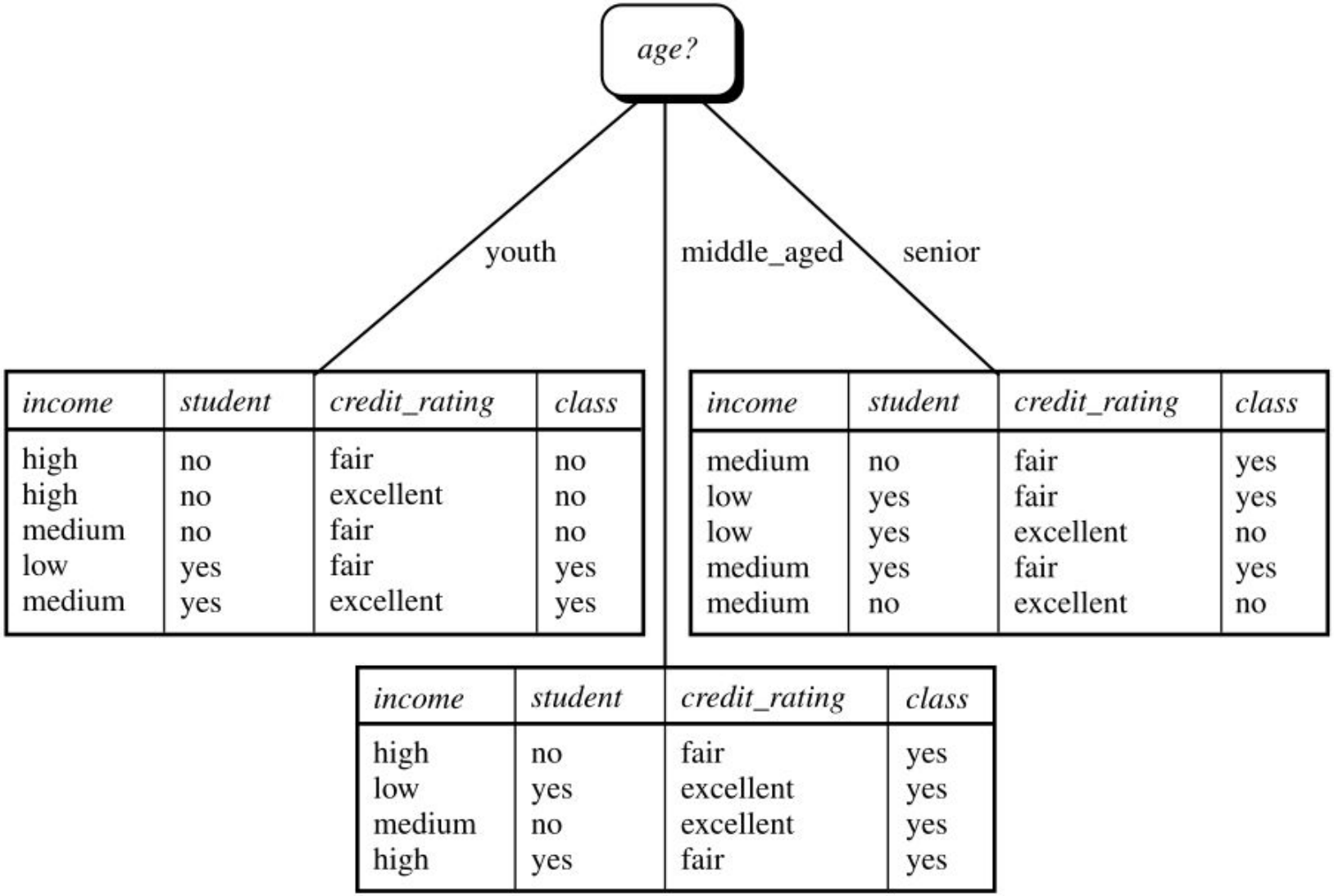
$$\Delta I(t_{income}) = 0.029$$

$$\Delta I(t_{student}) = 0.115$$

$$\Delta I(t_{cred\_rating}) = 0.115$$

- Because age has the highest information gain among the attributes, it is selected as the splitting attribute

Basic Algorithm

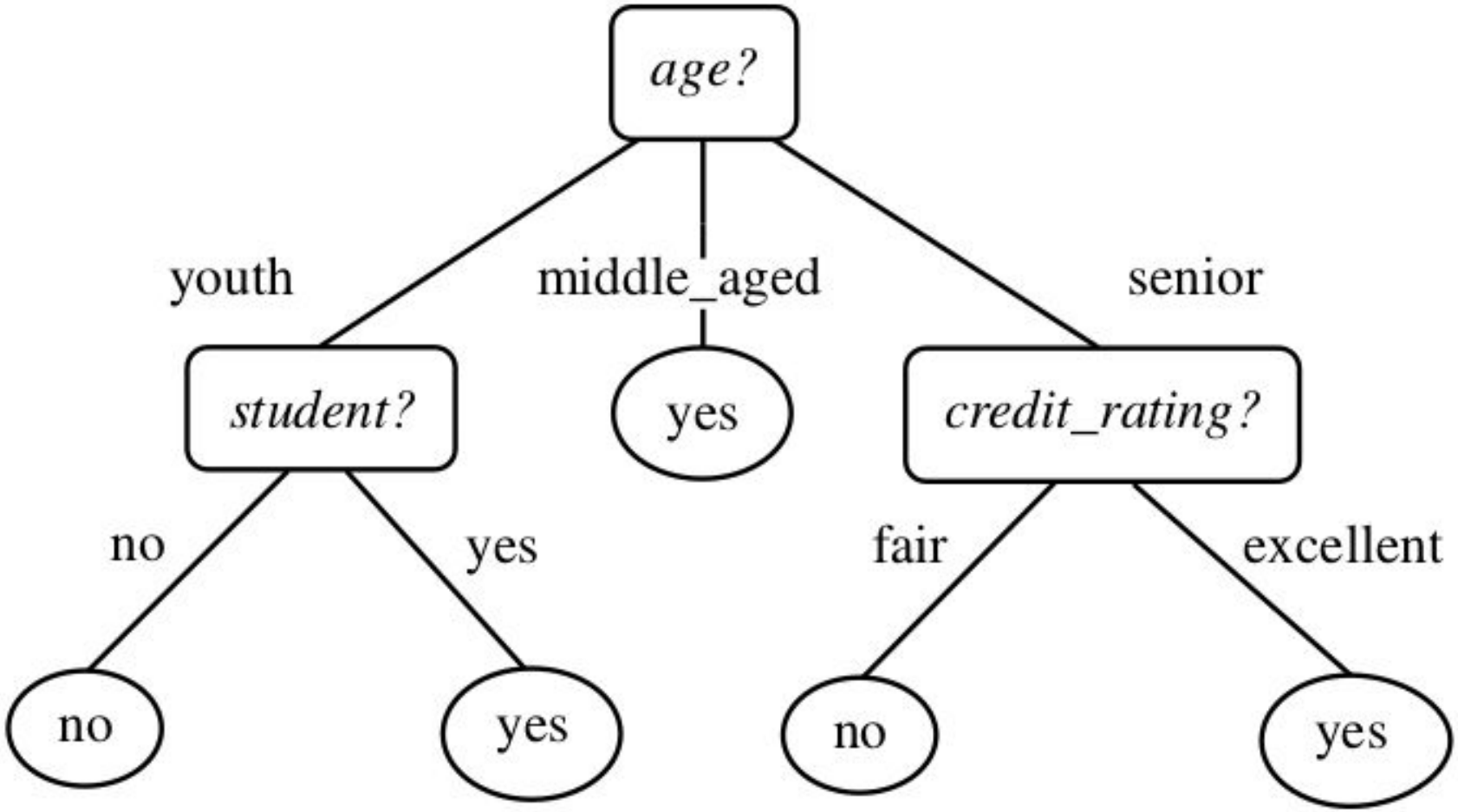


income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Basic Algorithm



A decision tree for the concept `buys_computer`

Remarks

- A variety of node impurity measures can be defined
- A critical factor in designing a decision tree is its size—the size of a tree must be large enough but not too large
- The most commonly used approach is to grow a tree up to a large size first and then prune nodes according to a pruning criterion
- A drawback associated with tree classifiers is their high variance—it is not uncommon for a small change in the training data set to result in a very different tree
- Comparative studies seem to give an advantage to the multilayer perceptrons with respect to the classification error, and an advantage to the decision trees with respect to the required training time

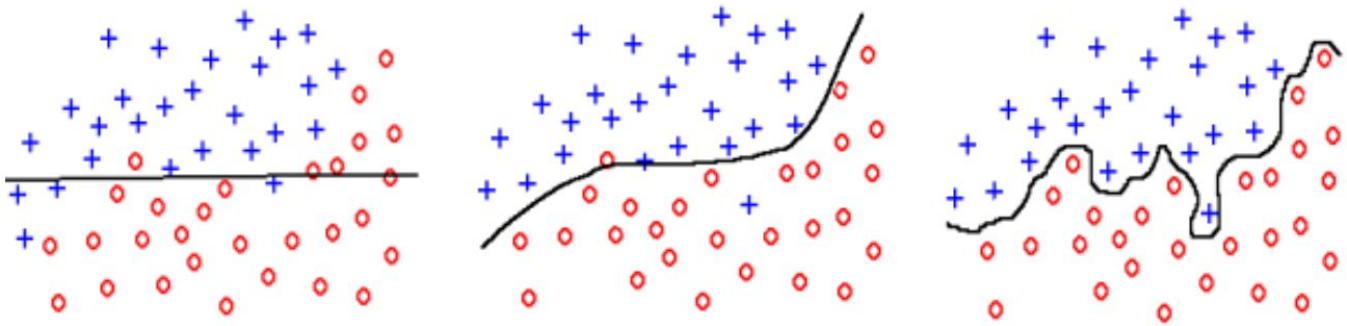
Outline

Experimental Evaluation

- Classifiers (both supervised and unsupervised) are learned (trained) on a finite training set
- A learned classifier has to be tested on a different test set experimentally, since the classifier performs on different data in the run mode that on which it has learned
- The experimental performance on the test data is a proxy for the performance on unseen data
- The main idea is to checks the classifier's generalization ability
- There is a need for a criterion function assessing the classifier performance experimentally (*e.g.*, its error rate, accuracy, expected Bayesian risk)
- In addition, there is a need for comparing classifiers experimentally

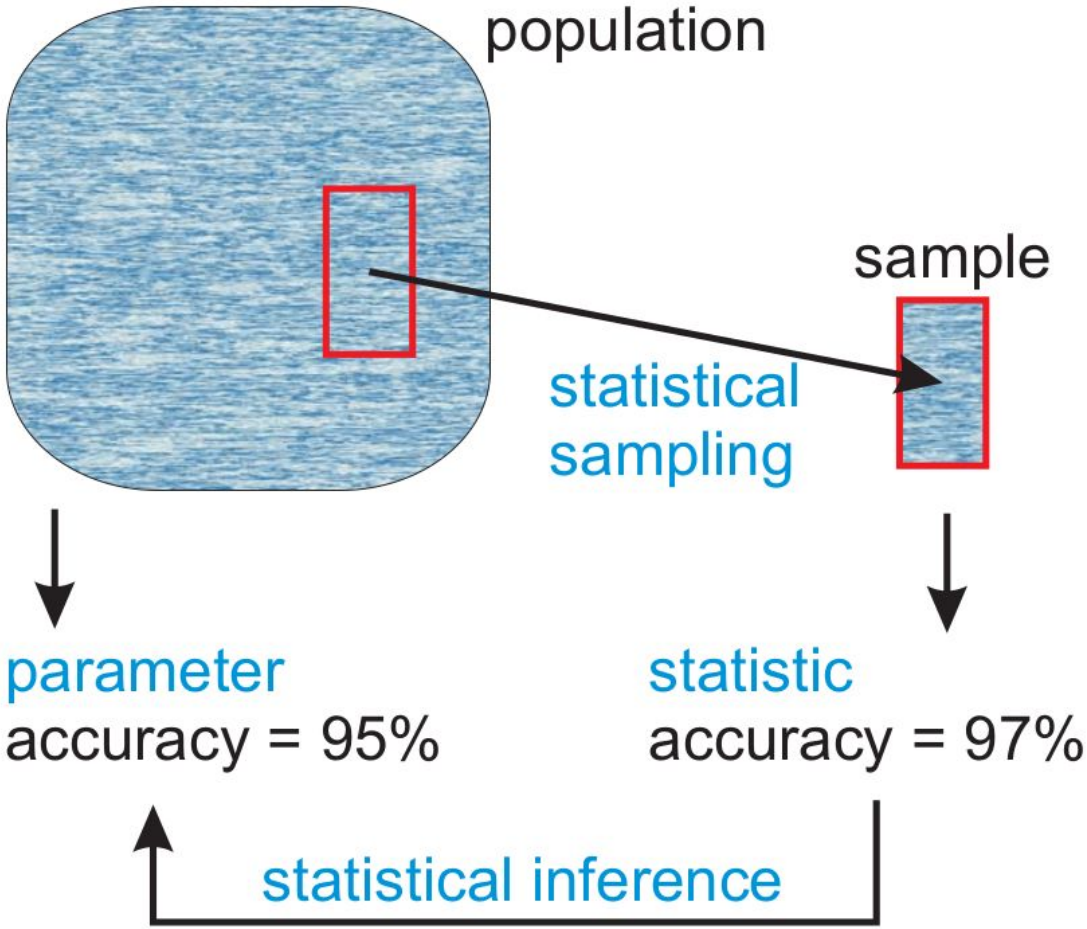
Experimental Evaluation

- Evaluation has to be treated as hypothesis testing in statistics
- The value of the population parameter has to be statistically inferred based on the sample statistics (*i.e.*, a training set in pattern recognition)
- Learning the training data too precisely usually leads to poor classification results on new data
- Classifier has to have the ability to generalize



underfit fit overfit

Experimental Evaluation



Experimental Evaluation

- Problem: Finite data are available only and have to be used both for training and testing
  - More training data gives better generalization
  - More test data gives better estimate for the classification error probability
- Never evaluate performance on training data—the conclusion would be optimistically biased
- Solution: Partitioning of available finite set of data to training/test sets
  - Hold out
  - Cross validation
  - Bootstrap
- Once evaluation is finished, all the available data can be used to train the final classifier

Experimental Evaluation

Hold out method

- Given data is randomly partitioned into two independent sets
- Training set (*e.g.*, 2/3 of data) for the statistical model construction (*i.e.*, learning the classifier)
- Test set (*e.g.*, 1/3 of data) is hold out for the accuracy estimation of the classifier
- Random sampling is a variation of the hold out method
  - Repeat the hold out k times, the accuracy is estimated as the average of the accuracies obtained
- It's not recommended to report performance results of just one hold out

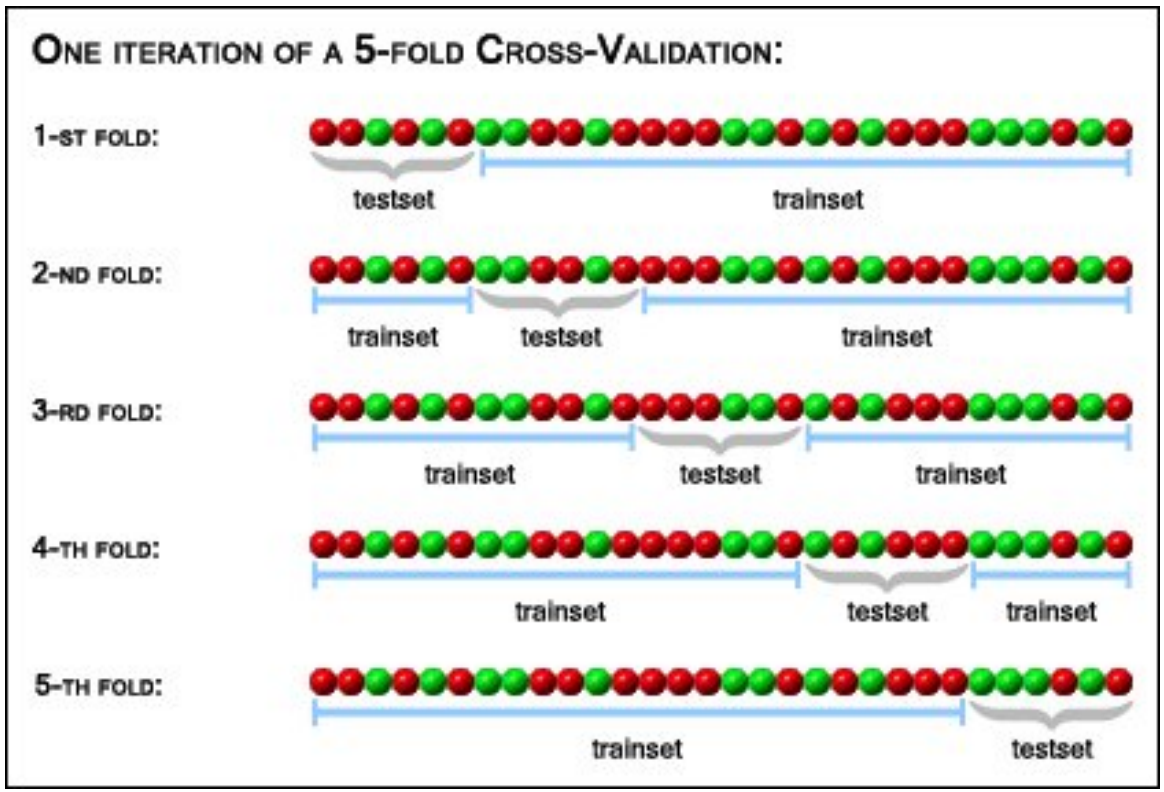
Experimental Evaluation

K-fold cross validation

- The training set is randomly divided into K disjoint sets of equal size where each part has roughly the same class distribution
- The classifier is trained K times, each time with a different set held out as a test set
- The estimated error is the mean of these K errors
- Recomended experimental validation procedure:
  - Use K-fold cross-validation ($K = 5$ or $K = 10$) for estimating performance estimates
  - Compute the mean value of performance estimate, standard deviation and 95% confidence intervals
  - Report mean values of performance estimates, and their standard deviations or confidence intervals

Experimental Evaluation

K-fold cross validation



Experimental Evaluation

Leave-one-out

- A special case of K-fold cross validation with $K = n$, where n is the total number of samples in the training set
- n experiments are performed using $n - 1$ samples for training and the remaining sample for testing
- It is rather computationally expensive
- Leave-one-out cross-validation does not guarantee the same class distribution in training and test data!
  - The extreme case: 50% class A, 50% class B—predict majority class label in the training data—true error 50%, leave-one-out error estimate 100%

Experimental Evaluation

Bootstrap

- The bootstrap uses sampling with replacement to form the training set
- Given the training set T consisting of n entries, bootstrap generates m new datasets T_i each of size $n' < n$ by sampling T uniformly with replacement
  - The consequence is that some entries can be repeated in T_i
- In a special case (called 632 boosting) when $n' = n$, for large n, T_i is expected to have $1 - \frac{1}{e} \approx 63.2\%$ of unique samples
  - The rest are duplicates

Experimental Evaluation

Bootstrap

- In this way, m statistical models (*e.g.*, classifiers, regressors) are learned using the above m bootstrap samples
- A special usage of the m statistical models is called *bootstrap aggregating* (it is also called *bagging*)
- In bagging, the statistical models are combined, *e.g.* by averaging the output (for regression) or by voting (for classification)
- Next section discusses the problem of combining several classifiers

Classifier Performance

- Criterion function to assess classifier performance
  - Accuracy or error rate
  - Other characteristics derived from the confusion matrix
  - Expected Bayesian risk
- Accuracy is the percent of correct classifications
- Error rate is the percent of incorrect classifications
- Accuracy $= 1 -$ Error rate
- Problems with the accuracy:
  - Assumes equal costs for misclassification
  - Assumes relatively uniform class distribution

Classifier Performance

- The confusion matrix is also called the contingency table
- The confusion matrix for two classes only is shown in next figure
- Performance measures calculated from the confusion matrix entries
  - Accuracy $= (a+d)/(a+b+c+d) = (TN+TP)/total$
  - True positive rate, recall, sensitivity $= d/(c+d) = TP/AP$
  - Specificity, true negative rate $= a/(a+b) = TN/AN$
  - Precision, predicted positive value $= d/(b+d) = TP/PP$
  - False positive rate, false alarm $= b/(a+b) = FP/AN =$ $1-$ specificity
  - False negative rate $= c/(c+d) = FN/AP$

Classifier Performance

		predicted	
		negative	positive
actual examples	negative	a TN - True Negative correct rejections	b FP - False Positive false alarms type I error
	positive	c FN - False Negative misses, type II error overlooked danger	d TP - True Positive hits

Confusion matrix

Classifier Performance

- A confusion matrix for more than two classes (*e.g.*, numeral OCR)

	class j predicted by a classifier										
true class i	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'	'R'
'0'	97	0	0	0	0	0	1	0	0	1	1
'1'	0	98	0	0	1	0	0	1	0	0	0
'2'	0	0	96	1	0	1	0	1	0	0	1
'3'	0	0	2	95	0	1	0	0	1	0	1
'4'	0	0	0	0	98	0	0	0	0	2	0
'5'	0	0	0	1	0	97	0	0	0	0	2
'6'	1	0	0	0	0	1	98	0	0	0	0
'7'	0	0	1	0	0	0	0	98	0	0	1
'8'	0	0	0	1	0	0	1	0	96	1	1
'9'	1	0	0	0	3	1	0	0	0	95	0

Classifier Performance

- Unequal costs of decisions
  - Medical diagnosis: The cost of falsely indicated breast cancer in population screening is smaller than the cost of missing a true disease
  - Defense against ballistic missiles: The cost of missing a real attack is much higher than the cost of false alarm
- Ideally, we want a measure that combines precision and recall, in addition to accuracy
  - F_Score $= 2PR/(P + R)$
- Bayesian risk is also able to represent unequal costs of decisions

Classifier Performance

- The problem of Bayesian risk is when we have unknown class distribution and costs
- The class distribution and the costs of each error determine the goodness of classifiers
- Additional problems:
  - In many circumstances, until the application time, we do not know the class distribution and/or it is difficult to estimate the cost matrix (*e.g.*, an email spam filter)
  - Statistical models have to be learned before
- Possible solution: Incremental learning

Classifier Performance

- As we mentioned, classes have often unequal frequency:
  - Medical diagnosis: 95% healthy, 5% disease
  - e-Commerce: 99% do not buy, 1% buy
  - Security: 99.999% of citizens are not terrorists
- Similar situation for multiclass classifiers
- Majority class classifier can be 99% correct but useless
- How should we train classifiers and evaluated them for unbalanced problems?

Classifier Performance

- Two class problems:
  - Build a balanced training set, use it for classifier training
    Randomly select desired number of minority class instances
    Add equal number of randomly selected majority class instances
  - Build a balanced test set (different from training set, of course) and test the classifier using it
- Multiclass problems:
  - Generalize 'balancing' to multiple classes
  - Ensure that each class is represented with approximately equal proportions in training and test datasets

Classifier Performance

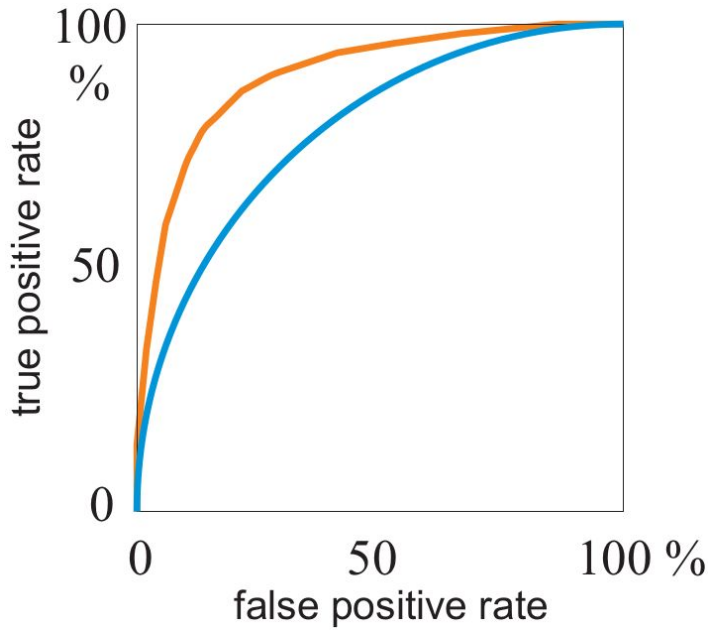
- Scalar characteristics are not good for evaluating performance
- Scalar characteristics as the accuracy, expected cost, area under ROC curve (AUC, to be explained soon) do not provide enough information
- We are interested in:
  - How are errors distributed across the classes?
  - How will each classifier perform in different testing conditions (costs or class ratios other than those measured in the experiment)?
- Two numbers–true positive rate and false positive rate–are much more informative than the single number
- These two numbers are better visualized by a curve, *e.g.*, by a Receiver Operating Characteristic (ROC)

Classifier Performance

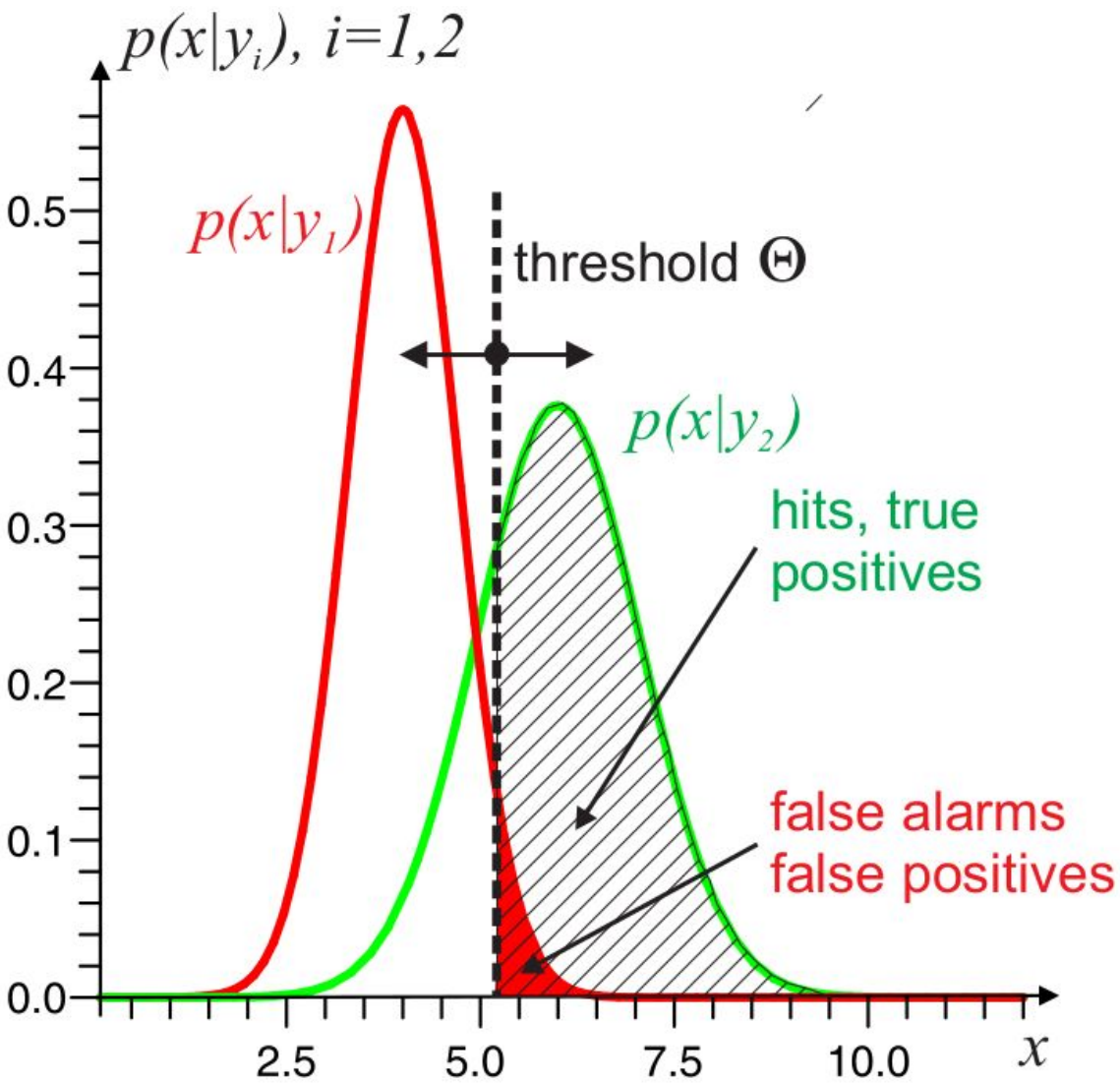
- The ROC informs about:
  - Performance for all possible misclassification costs
  - Performance for all possible class ratios
  - Under what conditions the classifier C_1 outperforms the classifier C_2?
- The ROC curve originates in WWII processing of radar signals
- It is useful for the evaluation of dichotomic classifiers performance
- Characterizes degree of overlap of classes for a single feature
- Decision is based on a single threshold Θ (called also operating point)

Classifier Performance

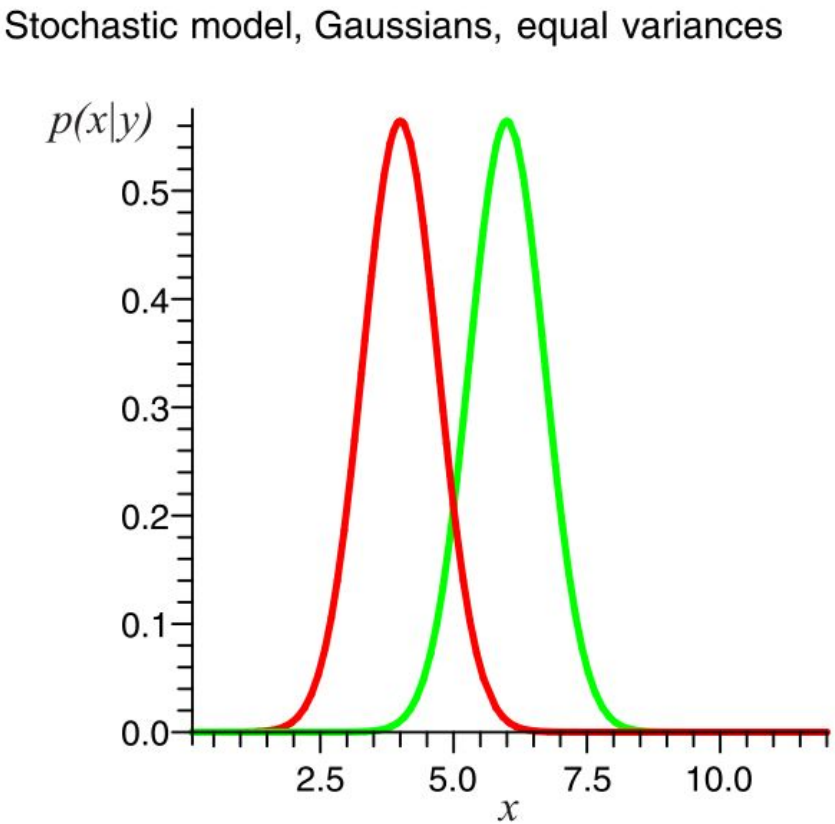
- Generally, false alarms go up with attempts to detect higher percentages of true objects
- Different ROC curves correspond to different classifiers
- The single curve is the result of changing threshold Θ

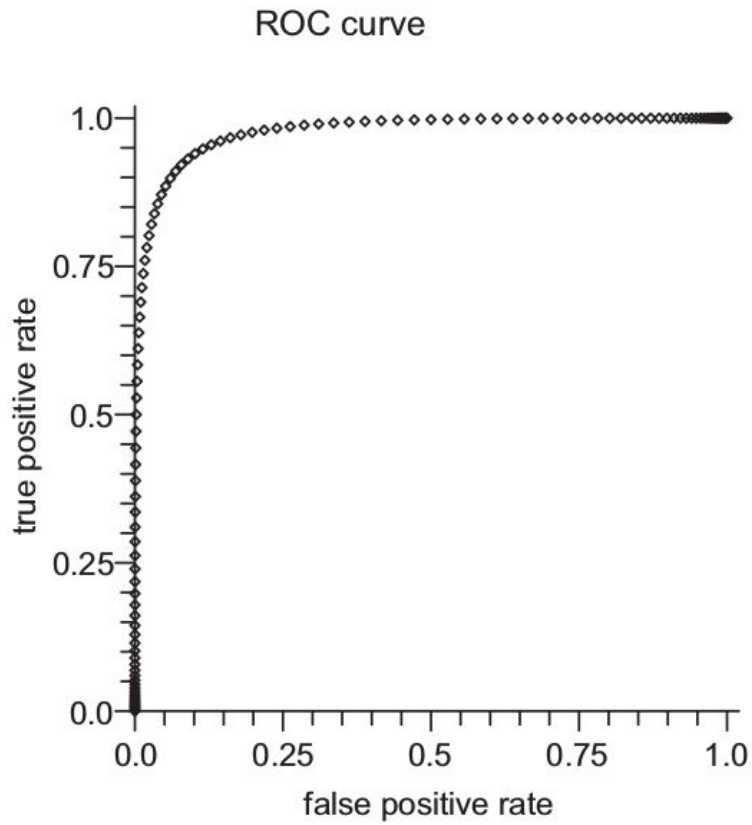


Classifier Performance



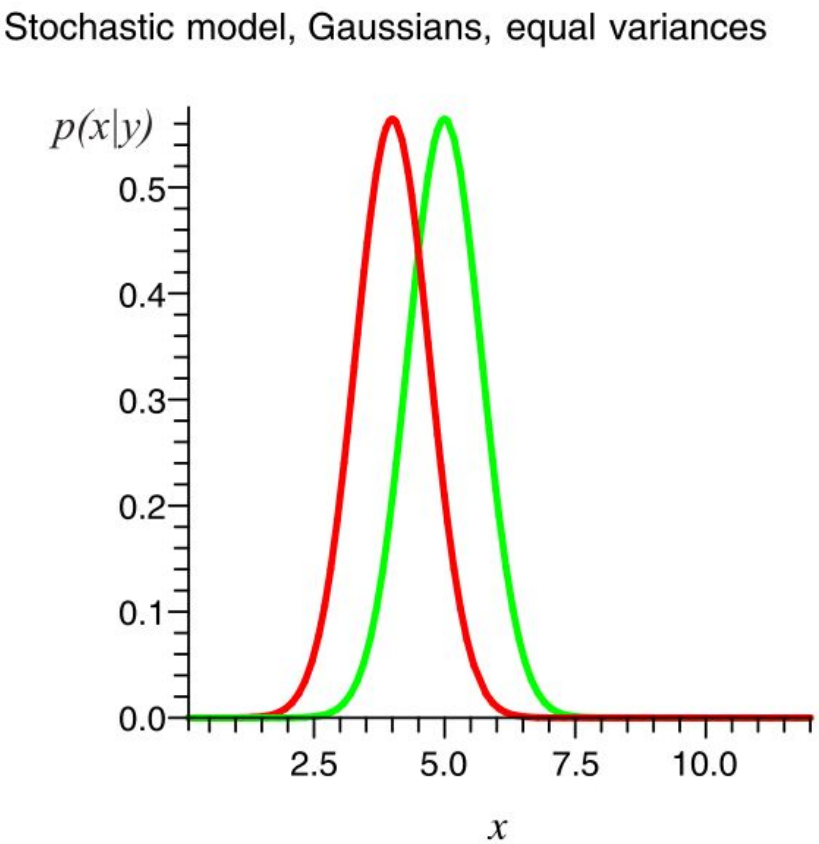
Classifier Performance

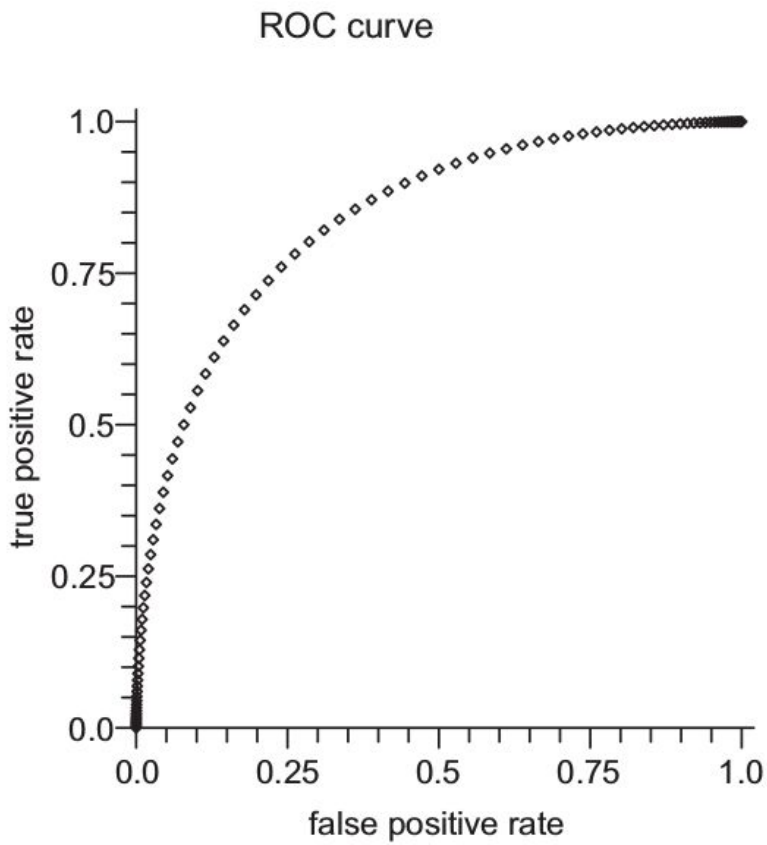




Two Gaussians, $\mu_1 = 4.0,\ \mu_2 = 6.0,\ \sigma_1 = \sigma_2 = 1.0$

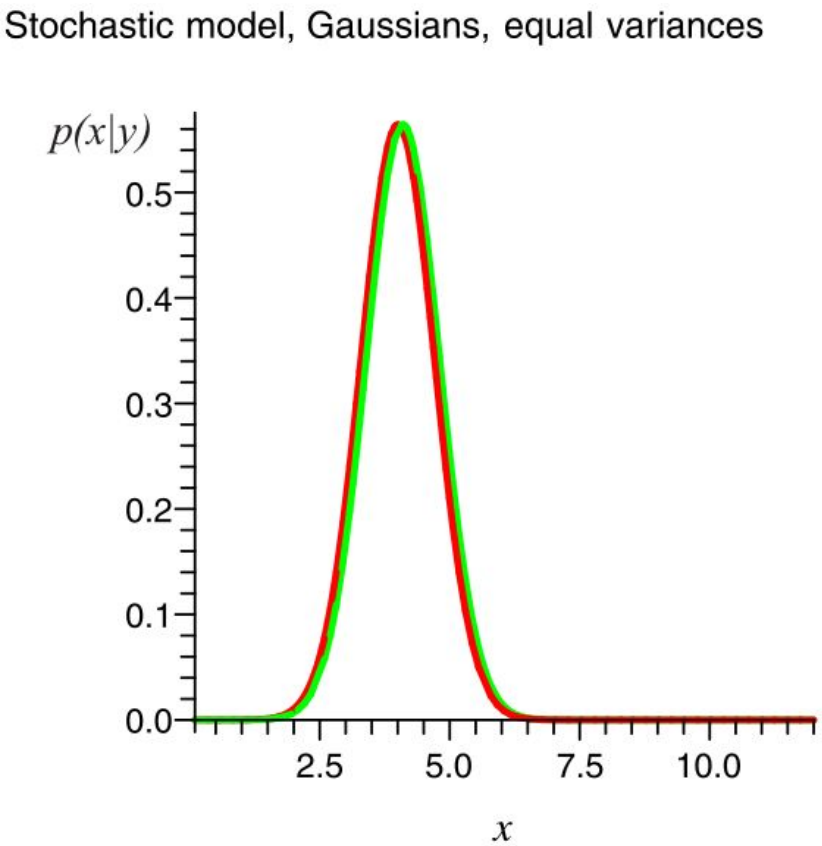
Classifier Performance

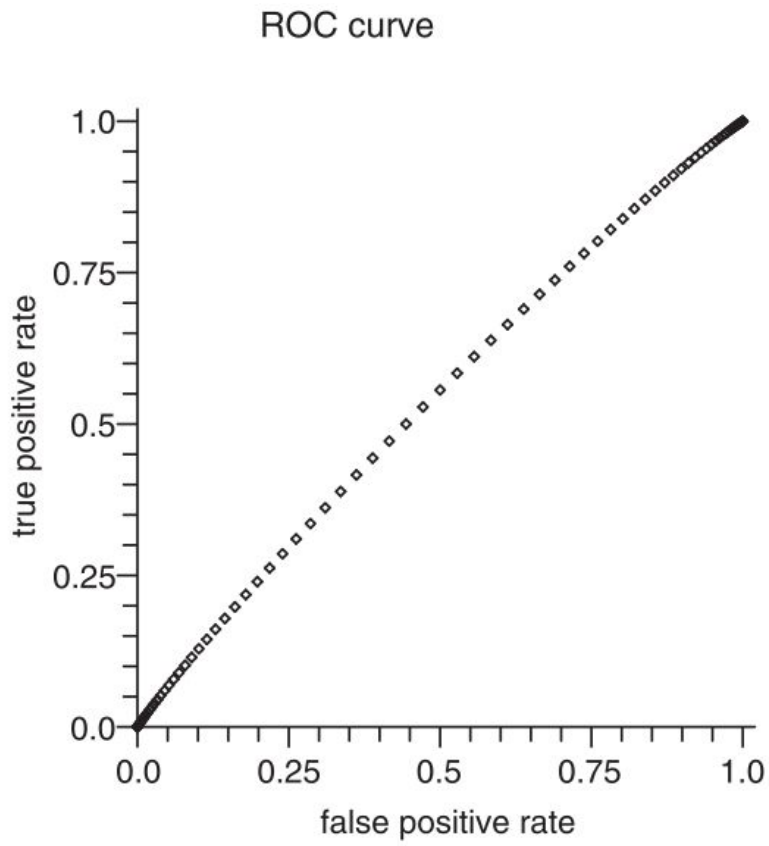




Two Gaussians, $\mu_1 = 4.0,\ \mu_2 = 5.0,\ \sigma_1 = \sigma_2 = 1.0$

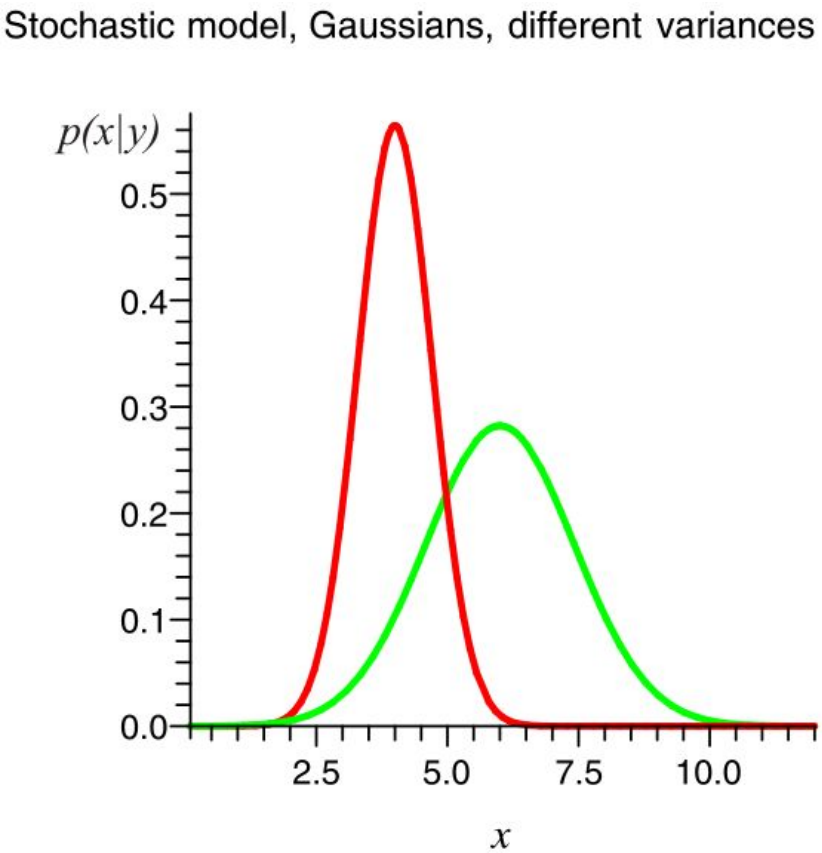
Classifier Performance

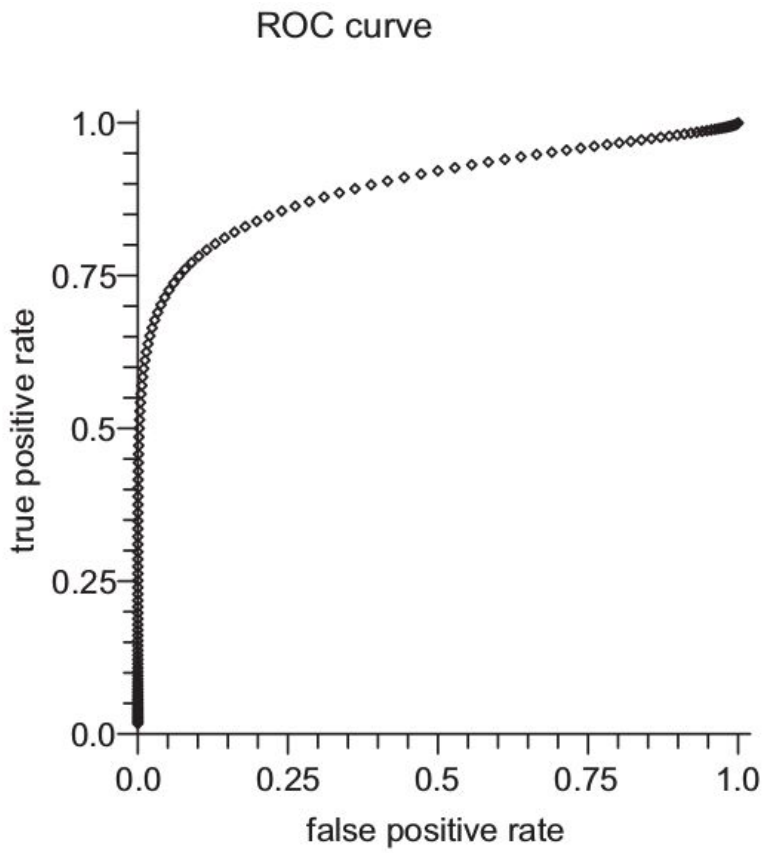




Two Gaussians, $\mu_1 = 4.0,\ \mu_2 = 4.1,\ \sigma_1 = \sigma_2 = 1.0$

Classifier Performance

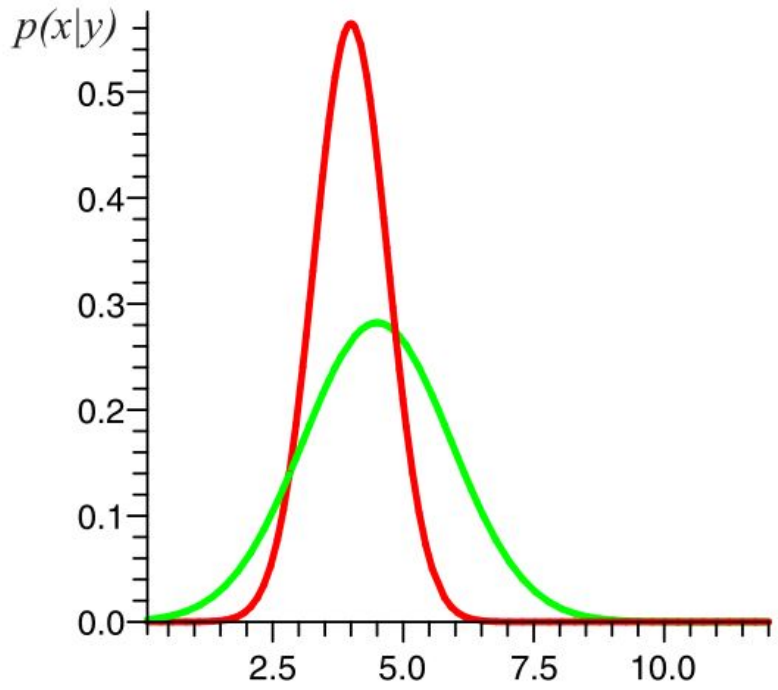




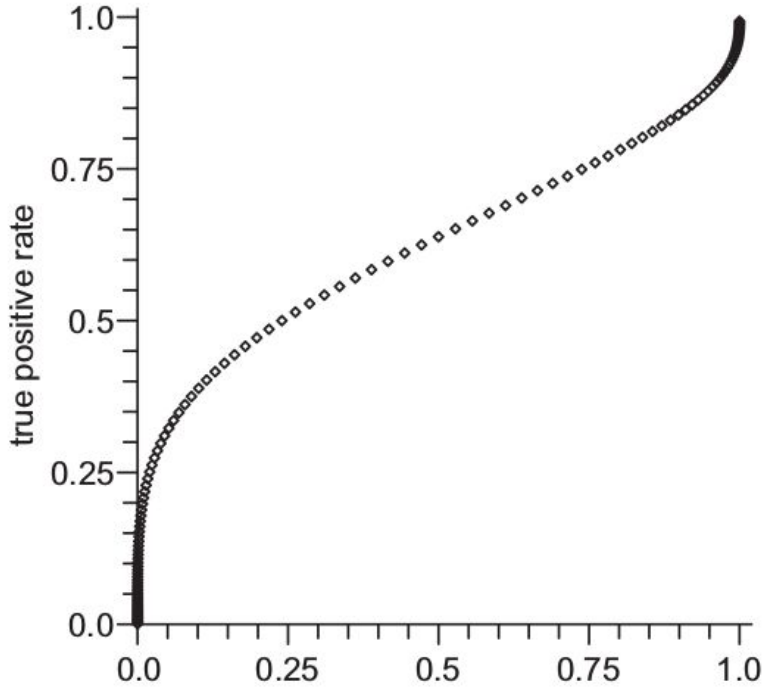
Two Gaussians, $\mu_1 = 4.0,\ \mu_2 = 6.0,\ \sigma_1 = 1.0,\ \sigma_2 = 2.0$

Classifier Performance

Stochastic model, Gaussians, different variances

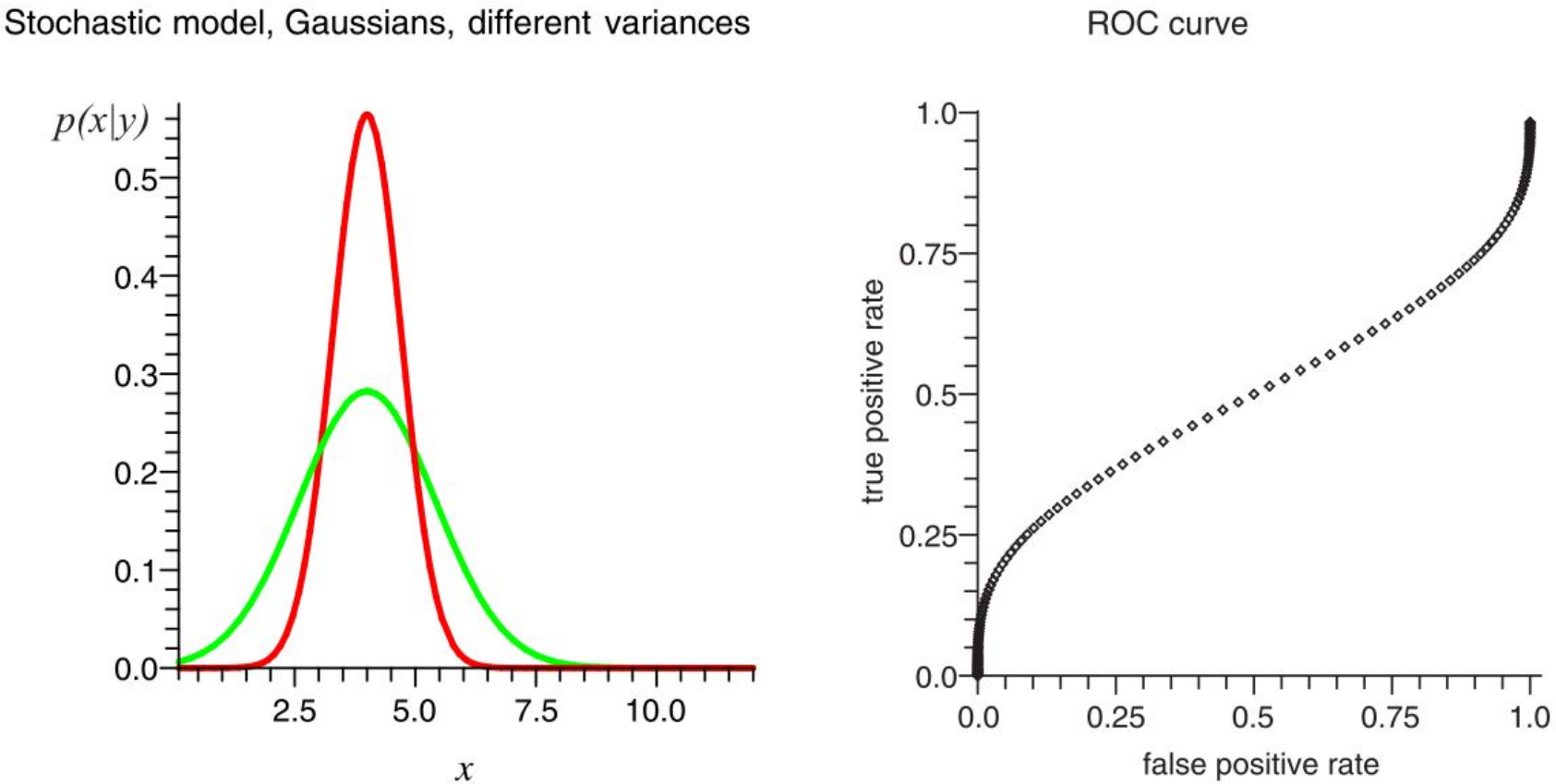


ROC curve



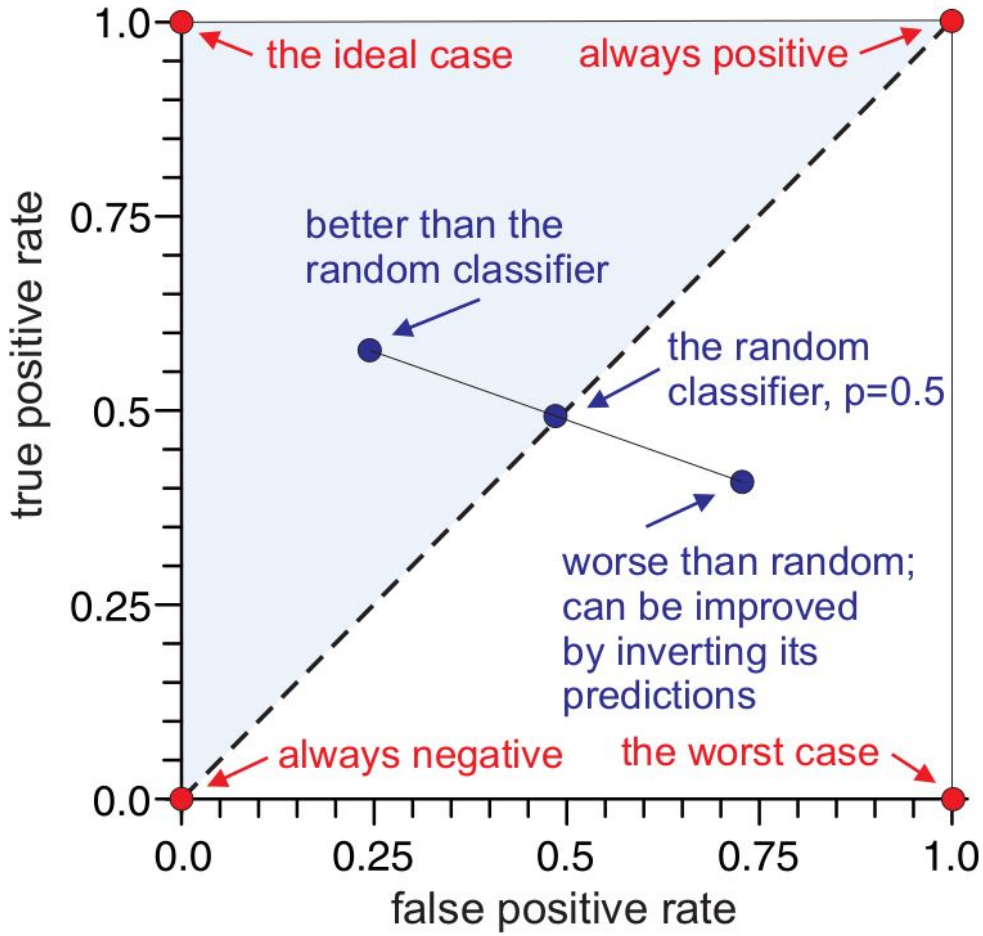
Two Gaussians, $\mu_1 = 4.0,\ \mu_2 = 4.5,\ \sigma_1 = 1.0,\ \sigma_2 = 2.0$

Classifier Performance



Two Gaussians, $\mu_1 = 4.0,\ \mu_2 = 4.0,\ \sigma_1 = 1.0,\ \sigma_2 = 2.0$

Classifier Performance



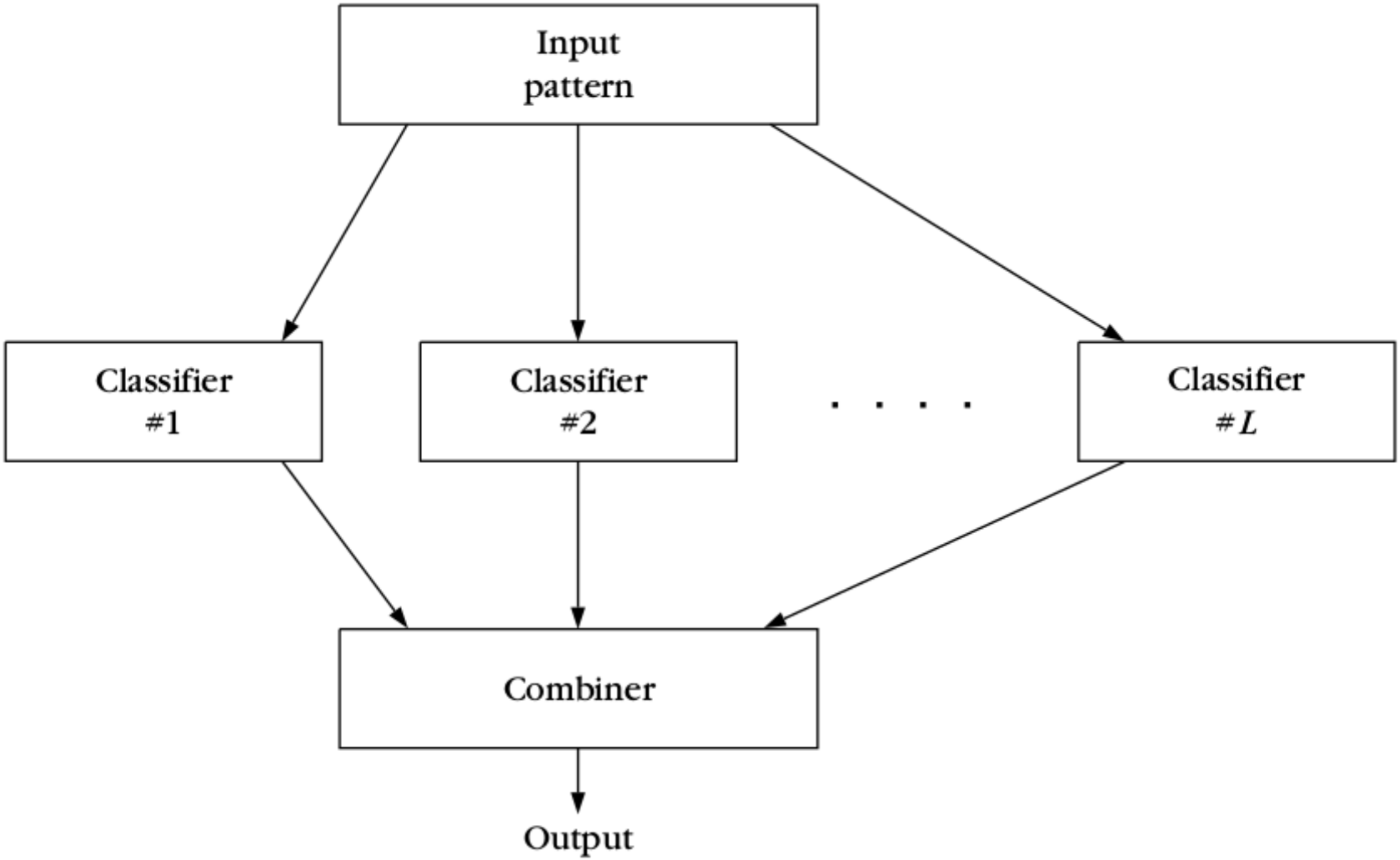
Properties of the ROC space

Outline

Combining Classifiers

- Another trend that offers more possibilities to the designer is to *combine different classifiers*
- Thus, one can exploit their individual advantages in order to reach an overall better performance than could be achieved by using each of them separately
- An important observation that justifies such an approach is the following:
  - From the different (candidate) classifiers we design in order to choose the one that fits our needs, one results in the best performance; that is, minimum classification error rate
  - However, different classifiers may fail on different patterns
  - That is, even the 'best' classifier can fail on patterns that other classifiers succeed on

Classifier Performance



Combining Classifiers

- Combining classifiers aims at exploiting this complementary information that seems to reside in the various classifiers
- Many interesting design issues have now come onto the scene
  - What is the strategy that one has to adopt for combining the individual outputs in order to reach the final conclusion?
  - Should all classifiers be fed with the same feature vectors, or must different feature vectors be selected for the different classifiers?
- In general, we assume that we are given a set of L classifiers, which have already been trained (in one way or another) to provide as outputs the class *a posteriori* probabilities

Combining Classifiers

- For a classification task of M classes, given an unknown feature vector $\mathbf{x}$ each classifier produces estimates of the a posteriori class probabilities

$$P_j(\omega_i|\mathbf{x}), \quad i = 1, 2, \ldots, M, \quad j = 1, 2, \ldots, L$$

- The goal is to devise a way to come up with an improved estimate of a 'final' *a posteriori* probability $P(\omega_i|\mathbf{x})$ based on all the resulting estimates from the individual classifiers, $P_j(\omega_i|\mathbf{x})$, $j = 1, 2, \ldots, L$
- An elegant way is to resort to information theoretic criteria by exploiting the Kullback–Leibler (KL) probability distance measure

Combining Individual Outputs

Geometric Average Rule

- According to this rule, one chooses $P(\omega_i|\mathbf{x})$ in order to minimize the *average* KL distance between probabilities
- That is

$$D_{av} = \frac{1}{L}\sum_{j=1}^{L} D_j$$

where

$$D_j = \sum_{i=1}^{M} P(\omega_i|\mathbf{x}) \ln \frac{P(\omega_i|\mathbf{x})}{P_j(\omega_i|\mathbf{x})}$$

Combining Individual Outputs

Geometric Average Rule

- Taking into account that

$$\sum_{i=1}^{M} P_j(\omega_i|\mathbf{x}) = 1$$

and employing Lagrange multipliers, the optimization problem can be solved

- The classification rule becomes equivalent to assigning the unknown pattern to the class maximizing the product
- That is

$$\max_{\omega_i} \prod_{j=1}^{L} P_j(\omega_i|\mathbf{x})$$

Combining Individual Outputs

Arithmetic Average Rule

- The KL probability dissimilarity cost is not a true distance measure, in the sense that it is not symmetric
- A different combination rule results if we choose to measure the probability distance via the alternative KL distance

$$D_j = \sum_{i=1}^{M} P_j(\omega_i|\mathbf{x}) \ln \frac{P_j(\omega_i|\mathbf{x})}{P(\omega_i|\mathbf{x})}$$

- This optimization problem leads to

$$\max_{\omega_i} \sum_{j=1}^{L} P_j(\omega_i|\mathbf{x})$$

Combining Individual Outputs

Majority Voting Rule

- The product and the summation schemes of combining classifiers belong to the so-called soft type rules
- Hard type combination rules are also very popular, owing to their simplicity and their robust performance
- According to the majority vote scheme, one decides in favor of the class for which there is a consensus, or when at least l_c of the classifiers agree on the class label of the unknown pattern, where

$$l_c = \begin{cases} \frac{L}{2} + 1, L \text{ even} \\ \frac{L+1}{2}, L \text{ odd} \end{cases}$$

Combining Individual Outputs

Majority Voting Rule

- Otherwise, the decision is rejection (*i.e.*, no decision is taken)
- Assume now that we are given L independent, individually trained classifiers, where each classifier has the same probability p of correct classification
  - If $p > 0.5$, $P_c(L)$ is monotonically increasing in L and $P_c(L) \to 1$ as $L \to \infty$
  - If $p < 0.5$, $P_c(L)$ is monotonically decreasing in L and $P_c(L) \to 0$ as $L \to \infty$
  - If $p = 0.5$, $P_c(L) = 0.5$ for all L
- In other words, increasing the number of classifiers increases the probability of a correct decision

Combining Individual Outputs

Majority Voting Rule

- This result provides a theoretical framework that justifies the general trend observed from experimental studies
- In practice, a number of scenarios have been proposed aiming to make the decisions of the individual classifiers more independent
  - One approach is to train individual classifiers using different data points that reside in the same feature space
  - An alternative route that takes us closer to independence is to let each classifier operate in a different feature subspace
- In practice, one tries to combine classifiers that are as 'diverse' as possible, expecting to improve performance by exploiting the complementary information residing in the outputs of the individual classifiers

Combining Individual Outputs

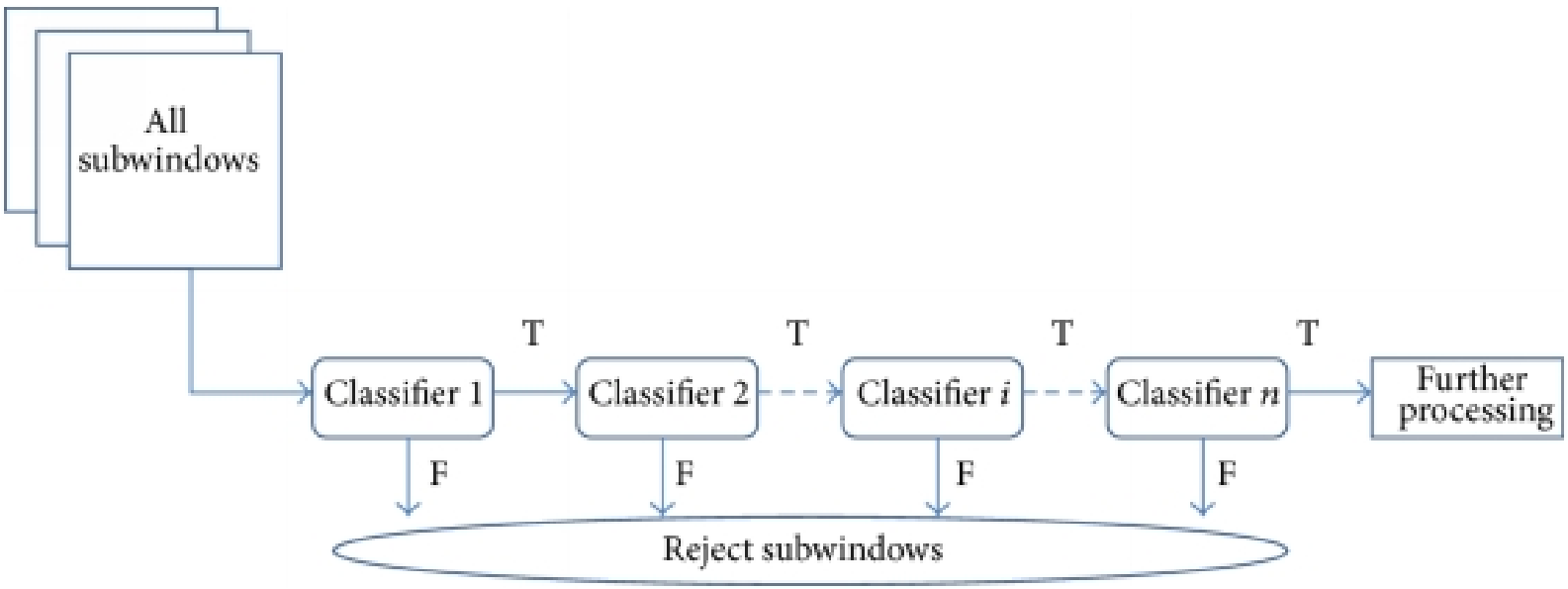
- There is not a single type of combination (*e.g.*, product rule, majority voting) that scores best for all cases
- It seems that the sum average and the majority vote rules are more popular and used the most frequently
- In other words, combining classifiers has to be considered carefully
- The common characteristic of all combination techniques presented so far is that the individual classifiers are separately trained and the combiner relies on a simple rule
- Boosting is a general approach to improve the performance of a given classifier and is one of the most powerful techniques

The Boosting Approach

- Although boosting can be considered an approach to combine classifiers, it is conceptually different from the techniques presented previously
- At the heart of a boosting method lies the so-called base classifier, which is a weak classifier
- A series of classifiers is then designed iteratively, employing each time the base classifier but using a different subset of the training set, according to an iteratively computed distribution, or a different weighting over the samples of the training set
- At each iteration, the computed weighting distribution gives emphasis to the 'hardest' (incorrectly classified) samples
- The final classifier is obtained as a weighted average of the previously *hierarchically* designed classifiers

The Boosting Approach

- It turns out that given a sufficient number of iterations the classification error of the final combination measured on the training set can become arbitrarily low
- One such algorithm is the so-called AdaBoost



Outline

Feature Selection

- As we pointed out, a major problem associated with pattern recognition is the so-called curse of dimensionality
  - The number of features at the disposal of the designer of a classification system is usually very large
- There is more than one reason to reduce the number of features to a sufficient minimum
  - Computational complexity is the obvious one
  - A related reason is that, although two features may carry good classification information when treated separately, there is little gain if they are combined into a feature vector because of a high mutual correlation
  - Another major reason is that imposed by the required generalization properties of the classifier

Feature Selection

- Given a number of features, how can one select the most important of them so as to reduce their number and at the same time retain as much as possible of their class discriminatory information?
- The procedure is known as *feature selection* or *reduction*
  - This step is very crucial
- In a more quantitative description, we should aim to select features leading to *large between-class distance* and *small within-class variance* in the feature vector space
- Sometimes the application of a linear or nonlinear transformation to a feature vector may lead to a new one with better discriminatory properties

Preprocessing

Outlier Removal

- An *outlier* is defined as a point that lies very far from the mean of the corresponding random variable
- This distance is measured with respect to a given threshold, usually a number of times the standard deviation
- Points with values very different from the mean value produce large errors during training and may have disastrous effects
- If the number of outliers is very small, they are usually discarded
- If this is not the case and they are the result of a distribution with long tails, then the designer may have to adopt cost functions that are not very sensitive in the presence of outliers

Preprocessing

Data Normalization

- In many practical situations a designer is confronted with features whose values lie within different dynamic ranges
- Thus, features with large values may have a larger influence in the cost function than features with small values
- The problem is overcome by normalizing the features so that their values lie within similar ranges

$$\hat{x}_{ik} = \frac{x_{ik} - \bar{x}_k}{\sigma_k}$$

- Nonlinear methods can also be employed in cases in which the data are not evenly distributed around the mean

$$y = \frac{x_{ik} - \bar{x}_k}{r\sigma_k}, \quad \hat{x}_{ik} = \frac{1}{1 + e^{-y}}$$

Preprocessing

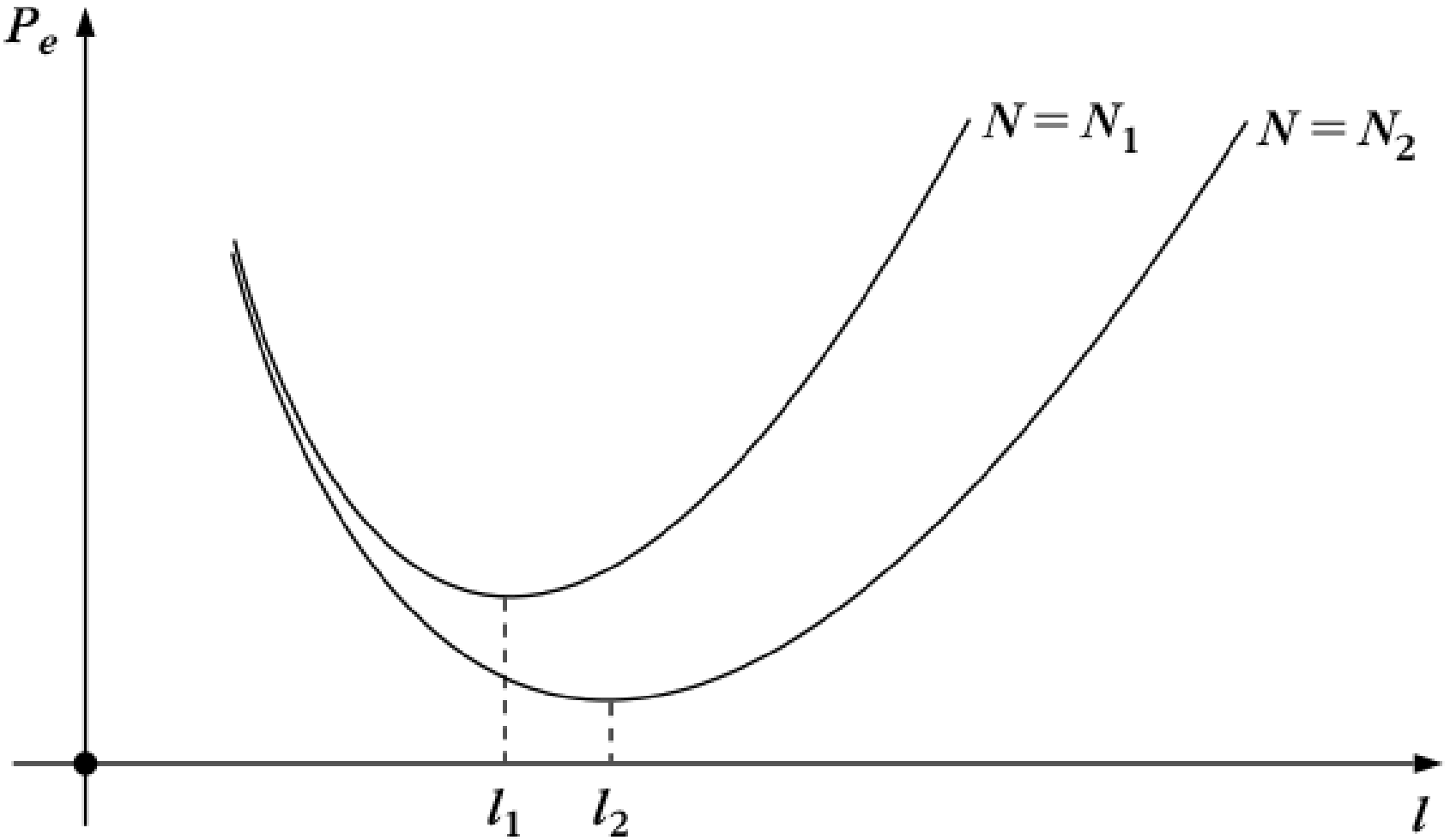
Missing Data

- In practice, certain features may be missing from some feature vectors
- The most traditional techniques in dealing with missing data include schemes that 'complete' the missing values by *(a)* zeros or *(b)* the unconditional mean, computed from the available values of the respective feature or (c) the conditional mean, if one has an estimate of the pdf of the missing values given the observed data
- Completing the missing values in a set of data is also known as *imputation*
- Another approach is to discard feature vectors with missing values

The Peaking Phenomenon

- In order to design a classifier with good generalization performance, the number of training points, N, must be large enough with respect to the number of features, l, that is, the dimensionality of the feature space
- In practice, for a finite N, by increasing the number of features one obtains an initial improvement in performance, but after a critical value further increase of the number of features results in an increase of the probability of error
- This phenomenon is also known as the *peaking phenomenon*
- Next figure illustrates the general trend that one expects to experience in practice by playing with the number of features, l, and the size of the training data set, N ($N_2 >> N_1$)

The Peaking Phenomenon



The Peaking Phenomenon

- The minimum in the curves occurs at some number $l = N/\alpha$, where α, usually, takes values in the range of 2 to 10
- Consequently, in practice, for a small number of training data, a small number of features must be used
- If a large number of training data is available, a larger number of features can be selected that yield better performance
- Although the above scenario covers a large set of 'traditional' classifiers, it is not always valid
  - We have already seen that adopting an appropriate kernel function to design a nonlinear SVM classifier implies a mapping to a high-dimensional space

Feature Selection Based on Hypothesis Testing

- A first step in feature selection is to look at each of the generated features *independently* and test their discriminatory capability for the problem at hand
- Although looking at the features independently is far from optimal, this procedure helps us to discard easily recognizable 'bad' choices and keeps the more elaborate techniques from unnecessary computational burden
- Let x be the random variable representing a specific feature
- We will try to investigate whether the values it takes for the different classes, say ω_1, ω_2, differ significantly
- To give an answer to this question, we will formulate the problem in the context of *statistical hypothesis testing*

Feature Selection Based on Hypothesis Testing

- A first step in feature selection is to look at each of the generated features *independently* and test their discriminatory capability for the problem at hand
- Although looking at the features independently is far from optimal, this procedure helps us to discard easily recognizable 'bad' choices and keeps the more elaborate techniques from unnecessary computational burden
- Let x be the random variable representing a specific feature
- We will try to investigate whether the values it takes for the different classes, say ω_1, ω_2, differ significantly
- To give an answer to this question, we will formulate the problem in the context of *statistical hypothesis testing*

Feature Selection Based on Hypothesis Testing

- We will try to answer which of the following hypotheses is correct:

  H_0: The values of the feature do not differ significantly

  H_1: The values of the feature differ significantly

- The decision is reached on the basis of experimental evidence supporting the rejection or not of H_0
- Let $x_i,\ i = 1, 2, \ldots, N$, be the sample values of the feature in class ω_1 with mean μ_1
- Correspondingly, for the other class ω_2 we have $y_i,\ i = 1, 2, \ldots, N$, with mean μ_2
- Let us now assume that the variance of the feature values is the same in both classes, $\sigma_1^2 = \sigma_2^2 = \sigma^2$

Feature Selection Based on Hypothesis Testing

- To decide about the closeness of the two mean values, we will test for the hypothesis

$$H_0 : \Delta\mu = \mu_1 - \mu_2 = 0$$
$$H_1 : \Delta\mu = \mu_1 - \mu_2 \neq 0$$

- To this end, let $z = x - y$ where x, y denote the random variables corresponding to the values of the feature in the two classes ω_1, ω_2, respectively, for which *statistical independence* has been assumed
- Obviously, $E[z] = \mu_1 - \mu_2$, and due to the independence assumption $\sigma_z^2 = 2\sigma^2$
- Following arguments similar to those used before $\bar{z} = \bar{x} - \bar{y}$

Feature Selection Based on Hypothesis Testing

- For the *known variance case*, $\bar{z}$ follows the normal $N(\mu_1 - \mu_2, \frac{2\sigma^2}{N})$ distribution for large N
- Then, we can define the test statistic

$$q = \frac{\bar{x} - \hat{u}}{\sigma/\sqrt{N}}$$

- Hence, the probability density function of q under H_0 is approximately $N(0, 1)$
- For a significance level ρ the acceptance interval $D = [-x_\rho, x_\rho]$, is chosen as the interval in which the random variable q lies with probability $1 - \rho$ (ρ the probability of being in $\bar{D}$)

Feature Selection Based on Hypothesis Testing

- In other words, if $q \in D$ decide H_0, if not decide H_1
- The acceptance intervals $[-x_\rho, x_\rho]$ corresponding to various probabilities for an $N(0,1)$ normal distribution are

$1-\rho$	0.8	0.85	0.9	0.95	0.98	0.99	0.998	0.999
x_ρ	1.282	1.44	1.645	1.967	2.326	2.576	3.09	3.291

- Example: Let us consider an experiment with a random variable x, and let us assume $N = 16$, $\mu_1 = 1.35$, $\mu_2 = 1.4$, $\sigma = 0.23$, and $\rho = 0.05$. Is x a good characteristic?

$$P(-1.97 < q < 1.97) = 0.95 \Rightarrow H_0 \text{ is accepted} \Rightarrow \text{No}$$

Feature Selection Based on Hypothesis Testing

- If the *variance is unknown*, then we choose the test statistic

$$q = \frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{s_z \sqrt{2/N}}$$

where

$$s_z^2 = \frac{1}{2N-2} \left(\sum_{i=1}^{N} (x_i - \bar{x})^2 + \sum_{i=1}^{N} (y_i - \bar{y})^2 \right)$$

- It can be shown that $s_z^2(2n-2)/\sigma^2$ follows a chi-square distribution with $2N-2$ degrees of freedom
- If x, y are normally distributed variables of the same variance σ^2, then the random variable q turns out to follow the t-distribution with $2N-2$ degrees of freedom

Feature Selection Based on Hypothesis Testing

Degrees of freedom	$1-\rho$				
	0.9	0.95	0.975	0.99	0.995
10	1.81	2.23	2.63	3.17	3.58
11	1.79	2.20	2.59	3.10	3.50
12	1.78	2.18	2.56	3.05	3.43
13	1.77	2.16	2.53	3.01	3.37
14	1.76	2.15	2.51	2.98	3.33
15	1.75	2.13	2.49	2.95	3.29
16	1.75	2.12	2.47	2.92	3.25
17	1.74	2.11	2.46	2.90	3.22
18	1.73	2.10	2.44	2.88	3.20
19	1.73	2.09	2.43	2.86	3.17
20	1.72	2.09	2.42	2.84	3.15

Interval values at various significance levels and degrees of freedom for a t-distribution

Feature Selection Based on Hypothesis Testing

- When the available number of samples is not the same in all classes, a slight modification is required

$$q = \frac{(\bar{x} - \bar{y}) - (\mu_1 - \mu_2)}{s_z \sqrt{1/N_1 + 1/N_2}}$$

where

$$s_z^2 = \frac{1}{N_1 + N_2 - 2} \left(\sum_{i=1}^{N} (x_i - \bar{x})^2 + \sum_{i=1}^{N} (y_i - \bar{y})^2 \right)$$

- Furthermore, in practice the variances may not be the same in the two classes
  - Sometimes this becomes the object of another hypothesis test to check whether the ratio $F = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2}$ is close to unity

Feature Selection Based on Hypothesis Testing

- Example: The sample measurements of a feature in two classes are

ω_1:	3.5	3.7	3.9	4.1	3.4	3.5	4.1	3.8	3.6	3.7
ω_2:	3.2	3.6	3.1	3.4	3.0	3.4	2.8	3.1	3.3	3.6

The question is to check whether this feature is informative enough with a significance level of $\rho = 0.05$

$$\omega_1 : \bar{x} = 3.73 \quad \hat{\sigma}_1^2 = 0.0601$$
$$\omega_2 : \bar{y} = 3.25 \quad \hat{\sigma}_2^2 = 0.0672$$

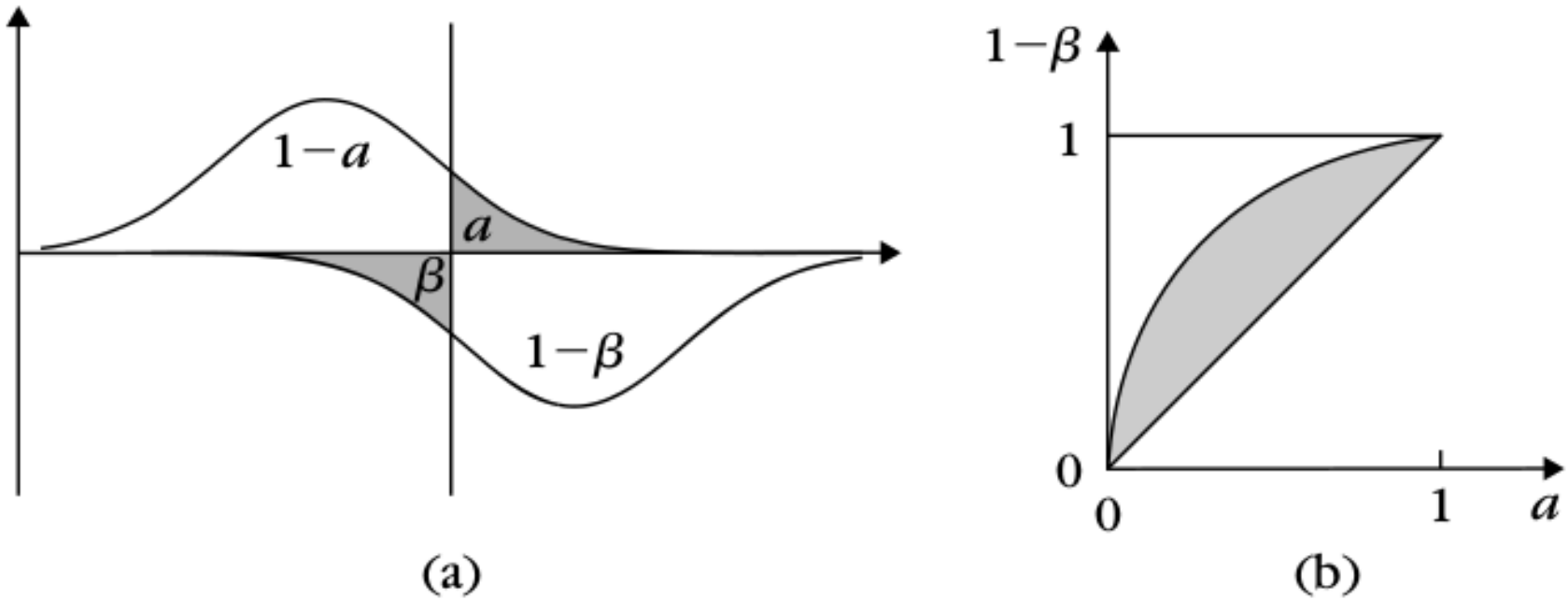
for $N = 10$ we have

$$s_z = \frac{1}{2}(\hat{\sigma}_1^2 + \hat{\sigma}_2^2) \Rightarrow q = \frac{\bar{x} - \bar{y} - 0}{s_z\sqrt{2/N}} = 4.25 \text{ and } D = [-2.1, 2.1]$$

The ROC Curve

- The hypothesis tests we have presented offer statistical evidence about the difference of the mean values of a single feature in the various classes
- However, although this is useful information for discarding features, if the corresponding mean values are closely located, this information may not be sufficient to guarantee good discrimination properties of a feature passing the test
- The ROC (Receiver Operating Characteristics) curve can easily be constructed by sweeping the threshold and computing percentages of wrong and correct classifications over the available training feature vectors
- More recently, the area under the ROC curve (AUC) has been used as an effective criterion to design classifiers

The ROC Curve



Two overlapping probability density functions describing the distribution of a feature in two classes, together with a threshold

Outline

Data Transformation and Dimensionality Reduction

- Feature generation is of paramount importance in any pattern recognition task
- Given a set of measurements, the goal is to discover compact and informative representations of the obtained data
- A similar process is also taking place in the human perception apparatus—our mental representation of the world is based on a relatively small number of perceptually relevant features
- These are generated after processing a large amount of sensory data, such as the intensity and the color of the pixels of the images sensed by our eyes, and the power spectra of the sound signals sensed by our ears

Data Transformation and Dimensionality Reduction

- The basic approach followed in this first section is to transform a given set of measurements to a new set of features
- If the transform is suitably chosen, transform domain features can exhibit high *information packing* properties compared with the original input samples
- This means that most of the classification-related information is 'squeezed' in a relatively small number of features, leading to a reduction of the necessary feature space dimension
- Sometimes we refer to such processing tasks as *dimensionality reduction* techniques

Data Transformation and Dimensionality Reduction

- The basic reasoning behind transform-based features is that an appropriately chosen transform can exploit and remove information redundancies, which usually exist in the set of samples obtained by the measuring devices
- Let us take for example an image resulting from a measuring device (*e.g.*, a camera)
  - The pixels at the various positions in the image have a large degree of correlation
  - If one uses the pixels as features, there will be a large degree of redundant information
  - Alternatively, if one obtains the Fourier transform of a typical real-world image, it turns out that most of the energy lies in the low-frequency components

The Karhunen–Loève Transform

- The Karhunen–Loève transform or principal component analysis (PCA), as it is also known, is one of the most popular methods for feature generation and dimensionality reduction in pattern recognition
- Though an old technique, it is still in use, and it forms the basis for a number of more advanced approaches
- Let $\mathbf{x}$ be the vector of input samples, and we will assume that the data samples have zero mean
- A desirable property of the generated features is to be mutually uncorrelated in an effort to avoid information redundancies

The Karhunen–Loève Transform

- The transformation can be established as

$$\mathbf{y} = A^T \mathbf{x}$$

where $E[y(i)y(j)] = 0,\ i \neq j$

- The correlation matrix matrix becomes

$$R_y \equiv E[\mathbf{y}\mathbf{y}^T] = E[A^T \mathbf{x}\mathbf{x}^T A] = A^T R_x A = \Lambda$$

where Λ is the diagonal matrix having as elements on its diagonal the respective eigenvalues $\lambda_i,\ i = 0, \ldots, N-1$ of R_x

- In practice, R_x is estimated as an average over the given set of training vectors

$$R_x \approx \frac{1}{n} \sum_{k=1}^{n} \mathbf{x}_k \mathbf{x}_k^T$$

The Karhunen–Loève Transform

- The relation between $\mathbf{x}$ and $\mathbf{y}$ can be rewritten as

$$\mathbf{x} = \sum i = 0^{N-1} y(i)\mathbf{a}_i \quad \text{and} \quad y(i) = \mathbf{a}_i^T \mathbf{x}$$

- Let us now define a new vector in the m-dimensional subspace

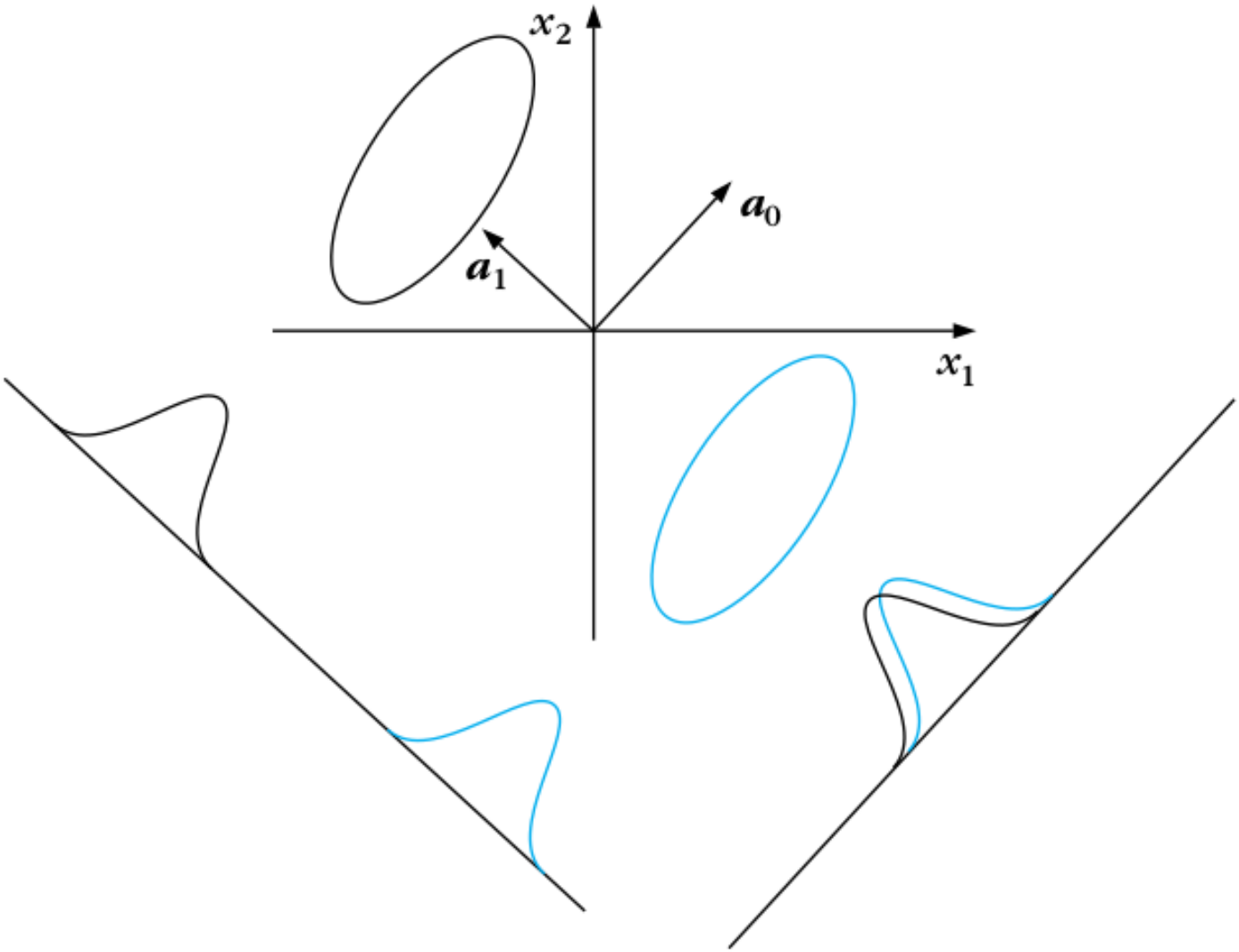
$$\hat{\mathbf{x}} = \sum_{i=0}^{m-1} y(i)\mathbf{a}_i$$

  where only m of the basis vectors are involved

- Obviously, this is nothing but the projection of $\mathbf{x}$ onto the subspace spanned by the m (orthonormal) eigenvectors involved in the summation

- If we choose the eigenvectors corresponding to the m largest eigenvalues of the correlation matrix, then the error is minimized

The Karhunen–Loève Transform



The Singular Value Decomposition

- The singular value decomposition of a matrix is one of the most elegant and powerful algorithms in linear algebra, and it has been extensively used for rank and dimension reduction in pattern recognition and information retrieval applications
- Given a $l \times n$ matrix X of rank r, we will show that there exist unitary matrices U and V of dimensions $l \times l$ and $n \times n$, respectively, so that

$$X = U \begin{bmatrix} \Lambda^{1/2} & O \\ O & 0 \end{bmatrix} V^H \quad \text{or} \quad Y \equiv \begin{bmatrix} \Lambda^{1/2} & O \\ O & 0 \end{bmatrix} = U^H X V$$

where $\Lambda^{1/2}$ is the $r \times r$ diagonal matrix with elements $\sqrt{\lambda_i}$, and λ_i are the r nonzero eigenvalues of the associated matrix $X^H X$

The Singular Value Decomposition

- In other words, there exist unitary matrices U and V that transform X into the special diagonal structure of Y
- If $\mathbf{u}_i$, $\mathbf{v}_i$ denote the column vectors of matrices U and V, respectively, then

$$X = \sum_{i=0}^{r-1} \sqrt{\lambda_i} \mathbf{u}_i \mathbf{v}_i^H$$

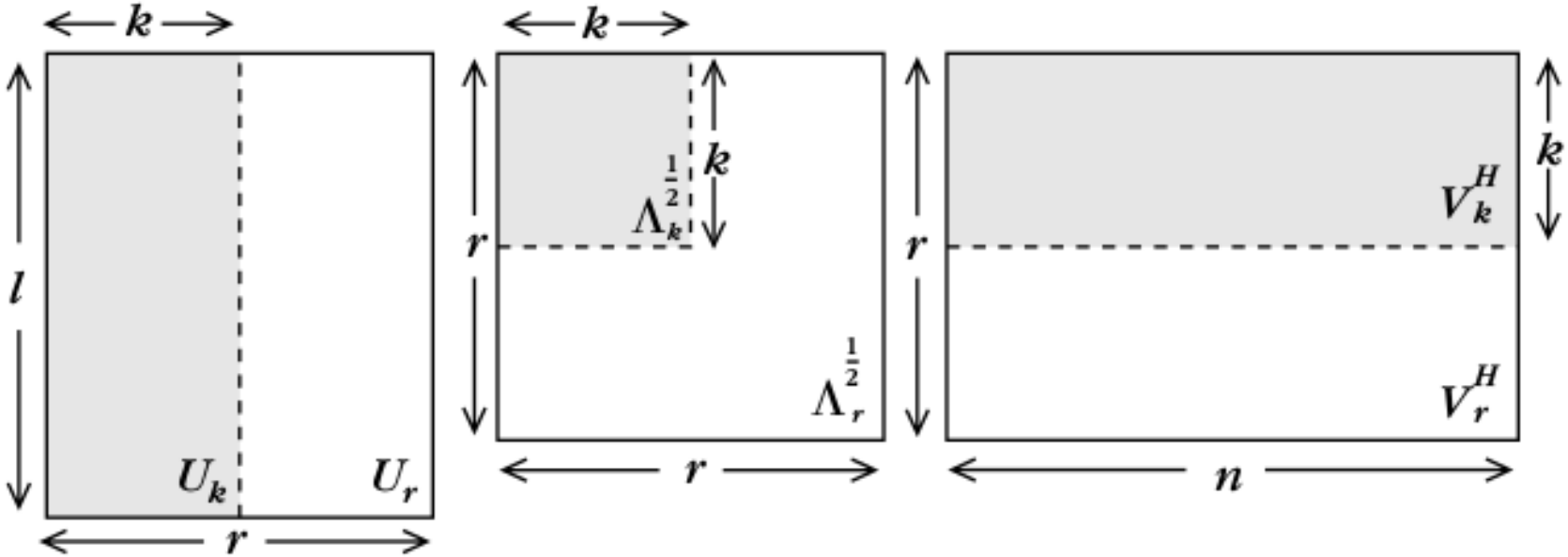
- A very interesting implication occurs if one uses less than r (the rank of X) terms in the summation

$$X \simeq \hat{X} = \sum_{i=0}^{k-1} \sqrt{\lambda_i} \mathbf{u}_i \mathbf{v}_i^H, \quad k \leq r$$

- Thus, $\hat{X}$ is the best rank-k approximation of X in the Frobenius norm sense

The Singular Value Decomposition

- SVD has been used extensively for dimension reduction in pattern recognition and information retrieval, and it forms the basis of what is known as *latent semantics indexing*



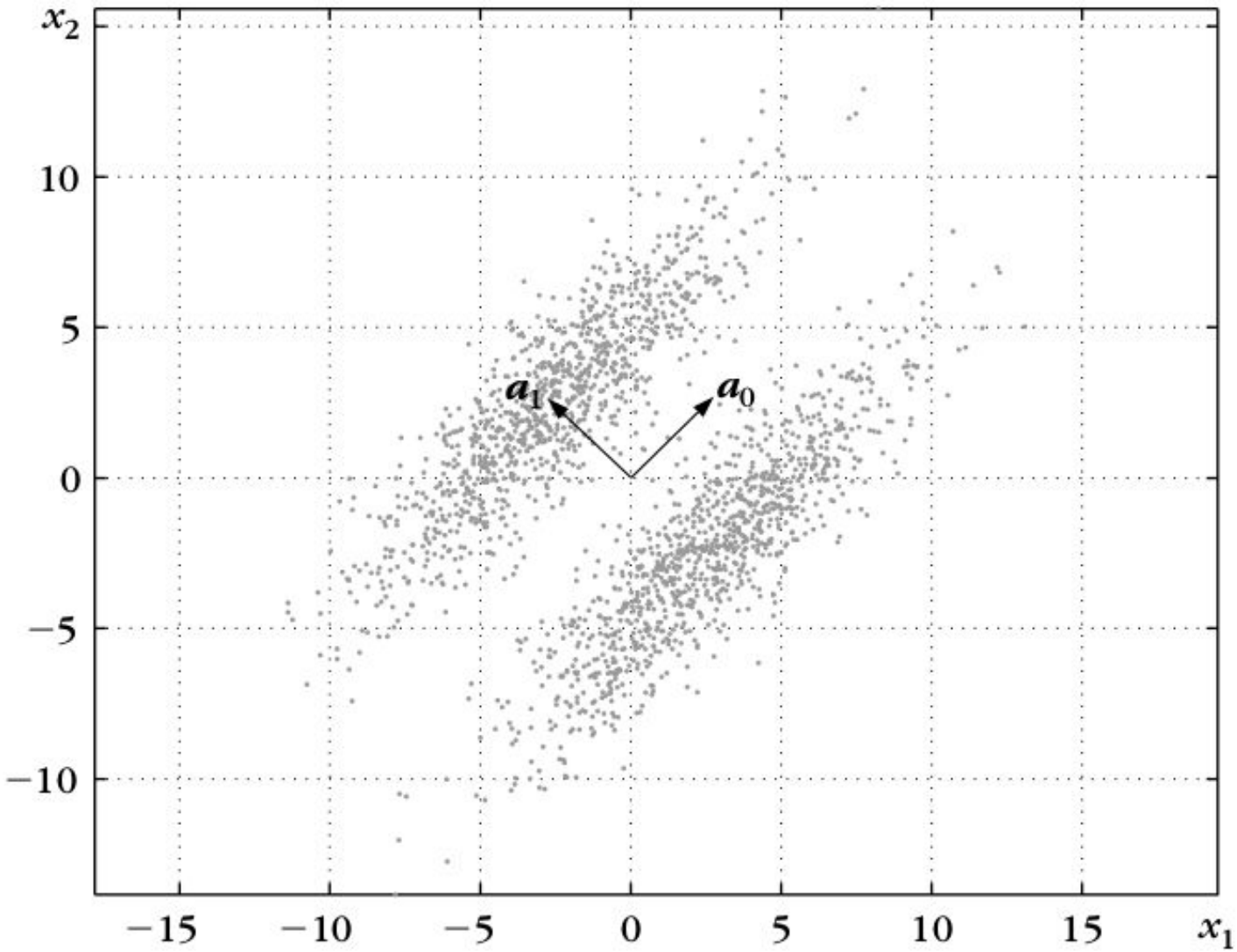
The Singular Value Decomposition

- SVD builds upon global information spread over all the data vectors in X
- Due to its optimal approximation properties, the SVD transform also has excellent 'information packing' properties, and an image array can be represented efficiently by a few of its singular values
- Thus, SVD is a natural candidate as a tool for feature generation/selection in classification
- Performing SVD of large matrices is a computationally expensive task

Independent Component Analysis

- The more recently developed ICA theory tries to achieve much more than simple decorrelation of the data
- Given the set of input samples $\mathbf{x}$, determine an $N \times N$ invertible matrix W such that the entries $y(i),\ i = 0, 1, \ldots, N-1$, of the transformed vector $\mathbf{y} = W\mathbf{x}$ are mutually independent
- The goal of statistical independence is a stronger condition than the uncorrelatedness required by the PCA
- The two conditions are equivalent only for Gaussian random variables

Independent Component Analysis



Other Discrete Transforms

- One-dimensional DFT

$$\mathbf{y} = W^H \mathbf{x}, \quad \mathbf{x} = W\mathbf{y}$$

where $W(n,k)^H = e^{-j2\pi nk/N}/\sqrt{N}$ and $W^{-1} = W^H = W^*$

- Two-dimensional DFT

$$Y = W^H X W^H, \quad X = WYW$$

- One-dimensional DCT

$$\mathbf{y} = C^T \mathbf{x}$$

where $C(n,k) = \sqrt{2/N}\cos[\pi k(2n+1)/(2N)]$ for $k \neq 0$, $C(n,k) = \sqrt{1/N}$ for $k = 0$, and $C^{-1} = C^T$

Other Discrete Transforms

- Two-dimensional DCT

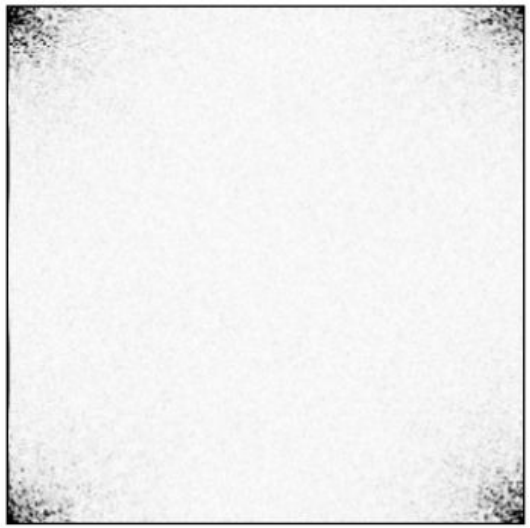
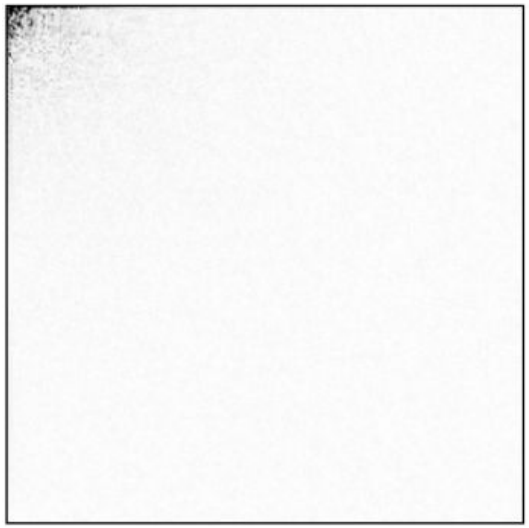
$$Y = C^T X C, \quad X = C Y C^T$$

- One-dimensional DST

$$S(k,n) = \sqrt{\frac{2}{N+1}} \sin\left(\frac{\pi(k+1)(n+1)}{N+1}\right)$$

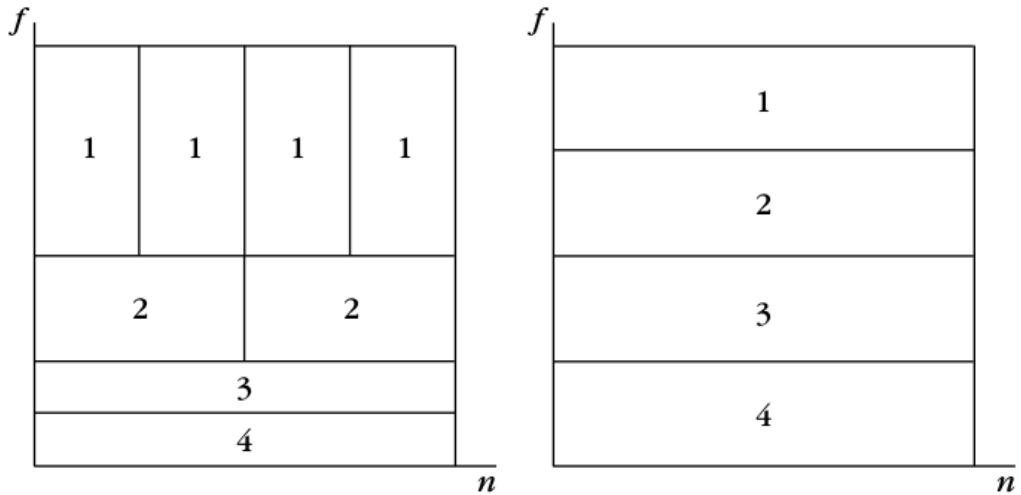
- The DFT, DCT and DST belong to the family of transforms that can be computed via a fast method in $O(N \log_2 N)$ operations
- Next figure shows the example of an image and its magnitude DFT, DCT, and DST transforms

Other Discrete Transforms


Other Discrete Transforms

- Nowadays, wavelet transformation is one of the most popular of the time-frequency transformations
- The fundamental idea of wavelet transforms is that the transformation should allow only changes in time extension, but not shape



Outline

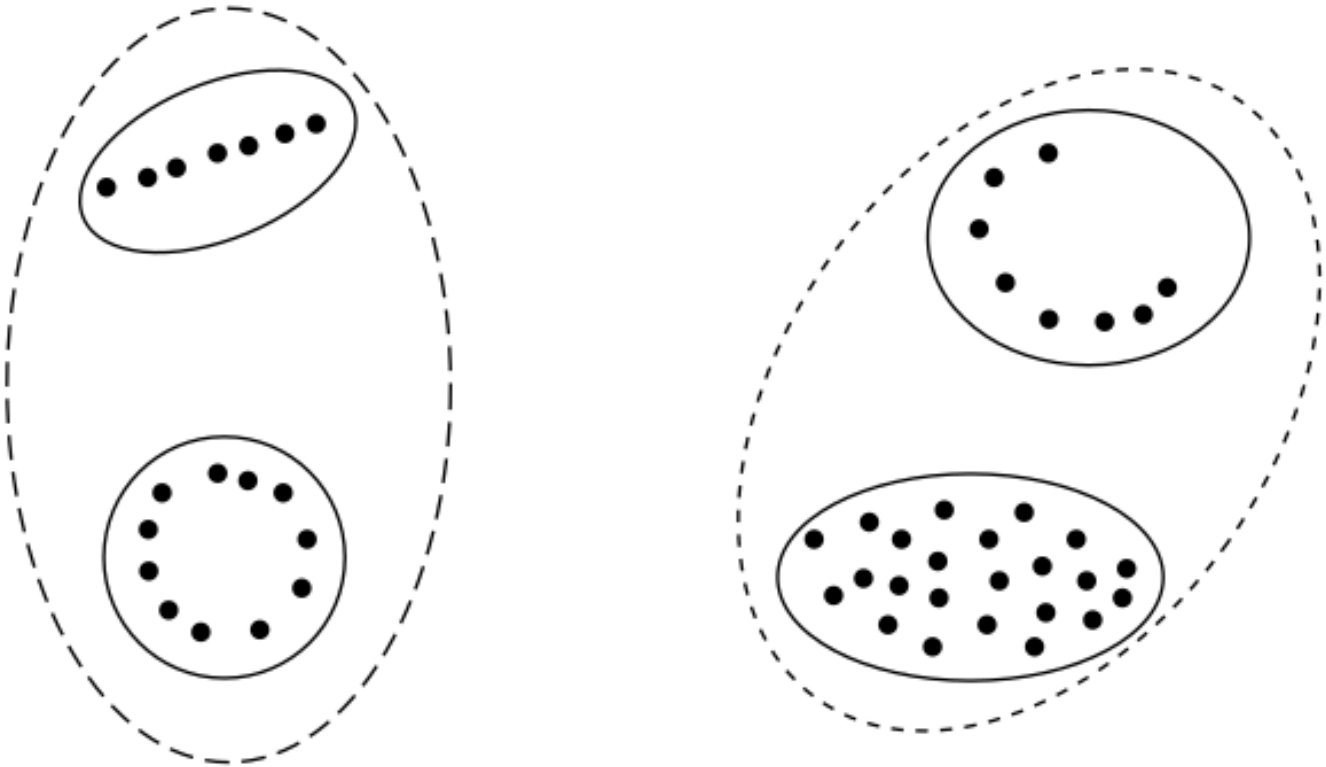
Introduction

- Now we turn to the unsupervised case, where class labeling of the training patterns is not available
- Thus, our major concern now is to 'reveal' the organization of patterns into 'sensible' clusters (groups), which will allow us to discover similarities and differences among patterns and to derive useful conclusions about them
- Clustering is one of the most primitive mental activities of humans, used to handle the huge amount of information they receive every day
- As was the case with supervised learning, we will assume that all patterns are represented in terms of features, which form l-dimensional feature vectors

Introduction

- The basic steps that an expert must follow in order to develop a clustering task are the following
  - Feature selection
  - Proximity measure
  - Clustering criterion
  - Clustering algorithms
  - Validation of the results
  - Interpretation of the results
- As one may have already suspected, different choices of features, proximity measures, clustering criteria, and clustering algorithms may lead to totally different clustering results
- Subjectivity is a reality we have to live with from now on

Introduction



A coarse clustering of the data results in two clusters, whereas a finer one results in four clusters

Introduction

- Clustering is a major tool used in a number of applications
- We summarize four basic directions in which clustering is of use:
  - Data reduction
  - Hypothesis generation
  - Hypothesis testing
  - Prediction based on groups
- A feature may take values from a continuous range (subset of $\mathbb{R}$) or from a finite discrete set
- A different categorization of the features is based on the relative significance of the values they take: nominal, ordinal, binary, interval-scaled, and ratio-scaled

Distance

- Minkowski distance:

$$d(i,j) = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \ldots + |x_{in} - x_{jn}|^p)^{1/p}$$

- The Manhattan (or city block) distance is when $p = 1$
- The Euclidean distance is when $p = 2$
- Weighting can also be applied to the Minkowski distance
- Note that *standardization* (normalization) may be useful in some applications
- For binary variables, a contingency table must be defined

Distance

		object j		
		1	0	sum
	1	q	r	$q+r$
object i	0	s	t	$s+t$
	sum	$q+s$	$r+t$	p

- *Symmetric binary dissimilarity* is defined as

$$d(i,j) = \frac{r+s}{q+r+s+t}$$

- *Asymmetric binary dissimilarity* is defined as

$$d(i,j) = \frac{r+s}{q+r+s}$$

Distance

- The dissimilarity between two nominal objects i and j can be computed based on the ratio of mismatches

$$d(i,j) = \frac{n-m}{n}$$

where m is the number of matches

- The dissimilarity $d(i,j)$ between mixed-type objects i and j is defined as

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d(i,j)^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}$$

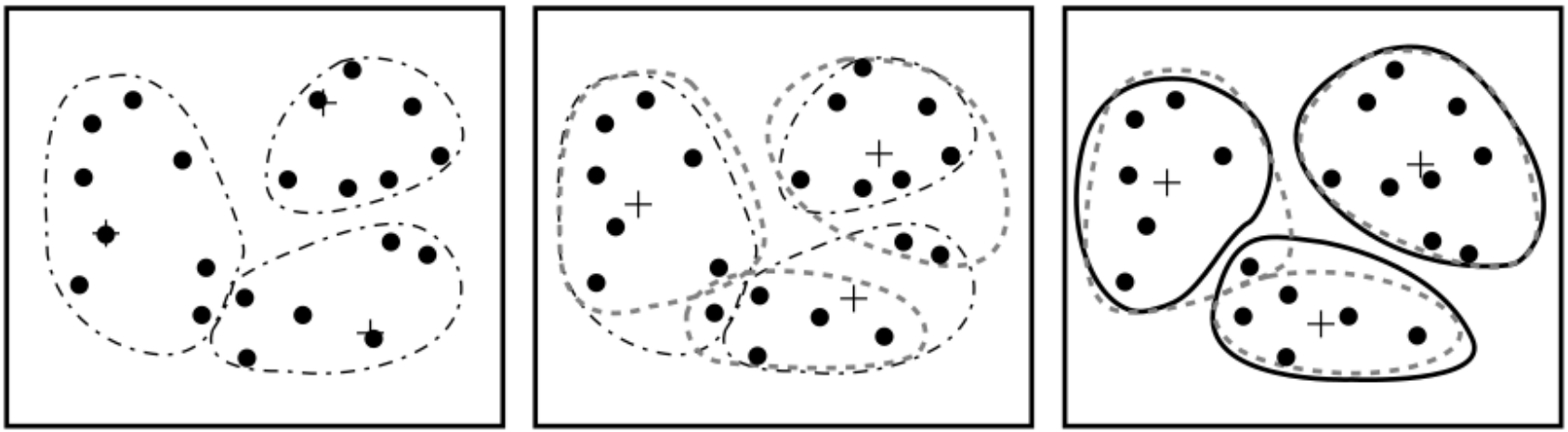
- Vector operations as cosine measure can be used for distance measurement

Major Clustering Methods

- Many clustering algorithms exist in the literature
- The major clustering methods can be classified into the following categories
  - Partitioning methods
  - Hierarchical methods
  - Density-based methods
  - Grid-based methods
  - Model-based methods
  - Clustering high-dimensional data
  - Constraint-based clustering

Major Clustering Methods

- The most well-known and commonly used partitioning methods are k-means, k-medoids, and their variations
- The k-means algorithm takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intracluster similarity is high but the intercluster similarity is low



Major Clustering Methods

- The k-means algorithm
  - Arbitrarily choose k objects from D as the initial cluster centers
  - Repeat until no change:

    (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster

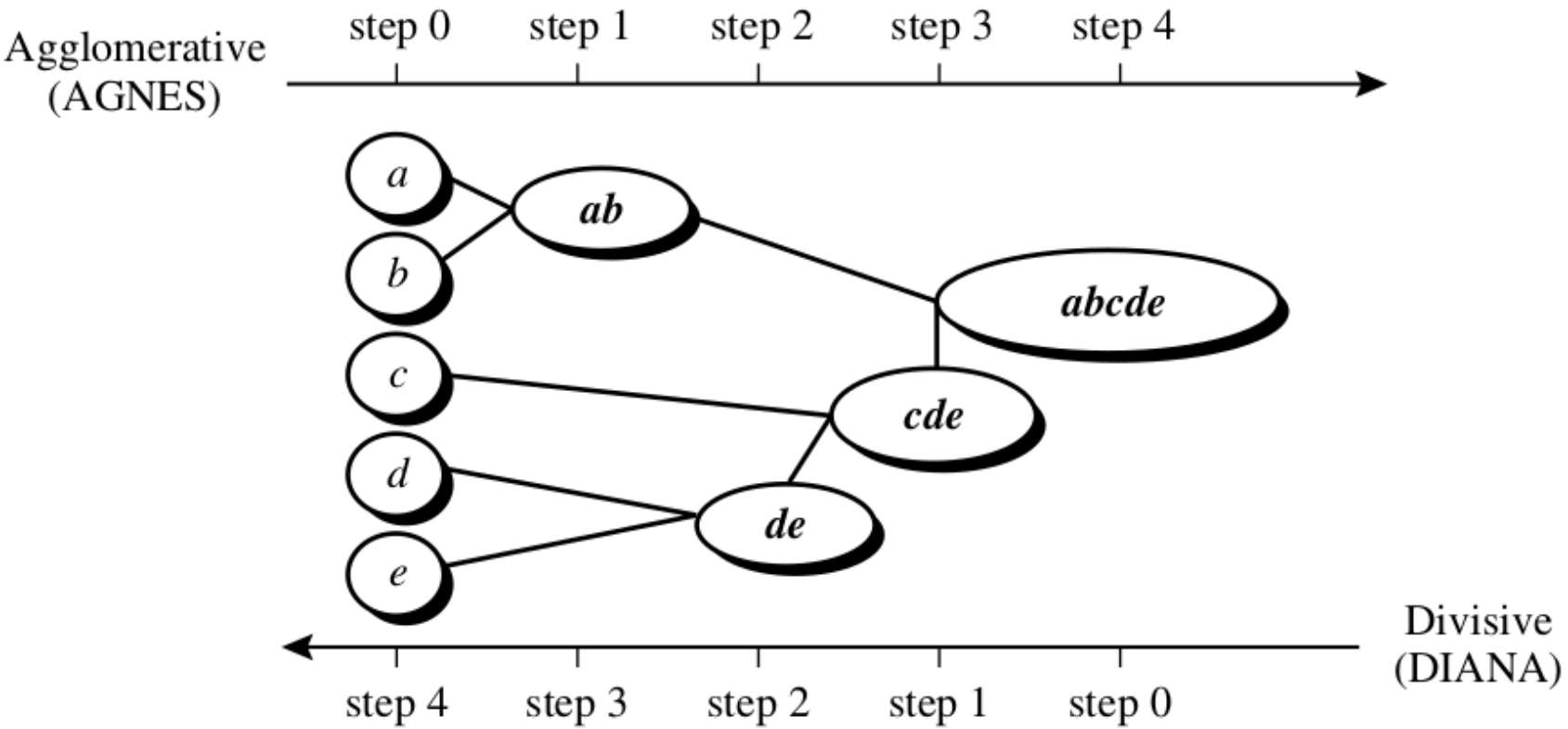
    update the cluster means, *i.e.*, calculate the mean value of the objects for each cluster
- The method often terminates at a local optimum
- The k-means method is not suitable for discovering clusters with nonconvex shapes or clusters of very different size

Major Clustering Methods

- The k-means algorithm is sensitive to outliers because an object with an extremely large value may substantially distort the distribution of data
- The basis of the k-medoids method for grouping n objects into k clusters: each representative object is actually the *medoid*, or most centrally located object, of its cluster
- PAM (Partitioning Around Medoids) was one of the first k-medoids algorithms introduced
- Variations of k-medois are CLARA (Clustering LARge Applications) and CLARANS (Clustering Large Applications based upon RANdomized Search)

Major Clustering Methods

- Agglomerative and divisive hierarchical clustering



Major Clustering Methods

- Four widely used measures for distance between clusters are:

$$d_{min}(C_i, C_j) = \min_{p \in C_i, p' \in C_j} |p - p'|$$

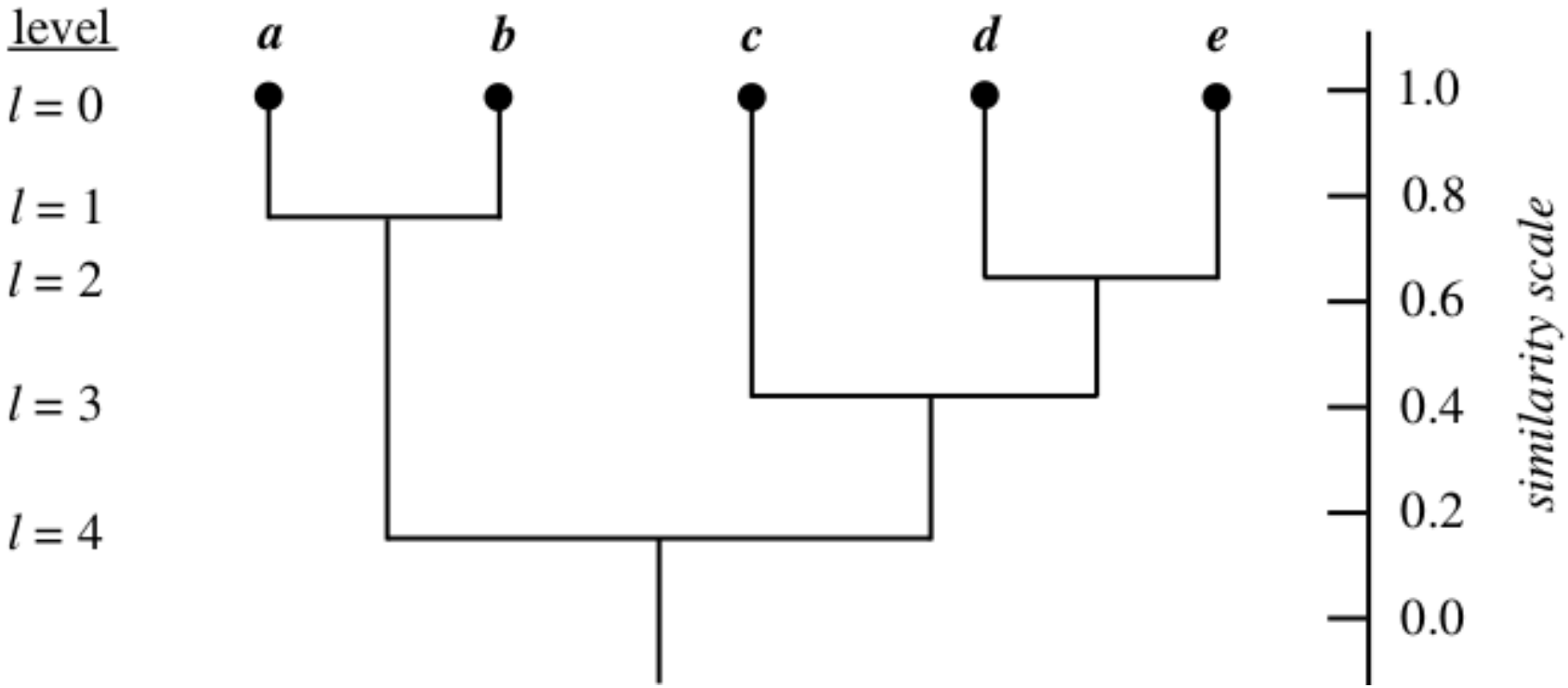
$$d_{max}(C_i, C_j) = \max_{p \in C_i, p' \in C_j} |p - p'|$$

$$d_{mean}(C_i, C_j) = |m_i - m_j|$$

$$d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i} \sum_{p' \in C_j} |p - p'|$$

- A tree structure called a *dendrogram* is commonly used to represent the process of hierarchical clustering

Major Clustering Methods



- BIRCH (Balanced Iterative Reducing and Clustering), ROCK (RObust Clustering using linKs), and Chameleon are examples of hierarchical clustering

Major Clustering Methods

- Density reachability and density connectivity in density-based clustering