Cybernetics and Artificial Intelligence

2. Machine Learning

Gerstner laboratory
Dept. of Cybernetics
Czech Technical University in Prague

Last lecture's wrap-up

- Bayes classification: classify into $\arg\max_s P(s|\vec{x})$.
- The Bayes classifier has the smallest risk (classification error) among all classifiers.
- Bayes classification rests upon knowing the true distribution $P(s|\vec{x})$.
- Usually we are not given $P(s|\vec{x})$ or $P(\vec{x}, s)$, only a i.i.d random sample therefrom. Without any prior knowledge on $P(\vec{x}, s)$, it gets very hard to estimate it from the sample as the number of components in $\vec{x}$ grows.
- The computational curse would not manifest itself if components of $\vec{x}$ were statistically independent, but that is rarely the case.
- A more realistic assumption, also avoiding the computational curse, is that the form of distribution $P(\vec{x}|s)$ is known and only its parameters should be estimated from the training sample.

Distributional Assumption

- The normal density

$$N(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2}$$

- Notable properties:
  - Central limit theorem: The effect of a sum of a large number of small independent random disturbances (however distributed) leads to the normal distribution.
  - Of all densities $f(x)$ of a random variable X with given mean and variance, the normal density has the **greatest entropy** $H(X) = \int_{-\infty}^{\infty} f(x) \log_2 f(x) dx$.
- Given a single **real scalar** attribute, the *normal distribution* assumption proposes that for each class s, the conditional density of x is:

$$f(x|s) = N(x, \mu_s, \sigma_s)$$

- Often, distributional parameters are explicitly shown in the conditional part:

$$f(x|s, \mu_s, \sigma_s) = N(x, \mu_s, \sigma_s)$$

Classifying under normal attribute distribution

- Under the normal distribution assumption, for Bayes optimal classification we proceed as follows

$$\arg\max_s f(s|x,\mu_s,\sigma_s) = \arg\max_s \frac{f(x|s,\mu_s,\sigma_s)P(s)}{f(x)} = \arg\max_s f(x|s,\vec{\phi})P(s)$$

$$= \arg\max_s \frac{1}{\sigma_s\sqrt{2\pi}} \exp\frac{-(x-\mu_s)^2}{2\sigma_s^2} \cdot P(s) = \arg\max_s \ln\left(\frac{1}{\sigma_s\sqrt{2\pi}} \exp\frac{-(x-\mu_s)^2}{2\sigma_s^2} \cdot P(s)\right)$$

$$= \arg\max_s \left(-\frac{1}{2}\ln\sigma_s^2 \underbrace{-\frac{1}{2}\ln 2\pi}_{\text{can drop}} + \frac{-(x-\mu_s)^2}{2\sigma_s^2} + \ln P(s)\right)$$

$$= \arg\max_s \left(-\frac{1}{2}\ln\sigma_s^2 - \frac{1}{2\sigma_s^2}\left(x^2 - 2x\mu_s + \mu_s^2\right) + \ln P(s)\right) = \arg\max_s a_s x^2 + b_s x + c_s$$

where

$$a_s = -\tfrac{1}{2}\ln\sigma_s^2 \qquad b_s = \tfrac{\mu_s}{\sigma_s^2} \qquad c_s = -\tfrac{1}{2}\ln\sigma_s^2 - \tfrac{\mu_s^2}{2\sigma_s^2} + \ln P(s)$$

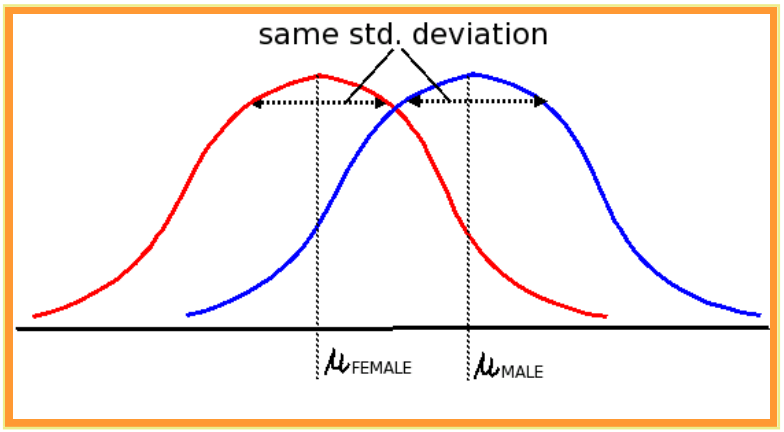
- A quadratic **discriminant function** thus defined **for each** $s \in S$,

$$\boxed{g_s(x) = a_s x^2 + b_s x + c_s}$$

Using discriminant functions: for a given x, **classify into** $\max_s g_s(x)$.

Normal distribution, same std. deviation σ (same variance σ^2)

- **Simple case:** same std. deviations. Example: $s = \{male, female\}$, $x =$ height.



- Since $\forall s\ \sigma_s = \sigma$, further simplification is possible

$$\max_s P(s|x, \mu_s, \sigma) = \max_s \left(\underbrace{-\frac{x^2}{2\sigma^2}}_{\text{can drop}} + \frac{1}{2\sigma^2}\left(2x\mu_s - \mu_s^2\right) + \ln P(s) \right) = \max_s \left(b_s \cdot x + c_s\right)$$

  where $b_s = \frac{\mu_s}{\sigma^2}$ and $c_s = -\frac{\mu_s^2}{2\sigma^2} + \ln P(s)$.

- Here, the discriminant function is **linear**:

$$g_s(x) = b_s x + c_s$$

The multivariate case

- The multivariate case ($\vec{x}$ now a n-component real vector, $\vec{x} \in \Re^n$)

$$N(x, \vec{\mu}, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n \det(\mathbf{\Sigma})}} \exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu})^t |\mathbf{\Sigma}| (\vec{x} - \vec{\mu})\right]$$

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_{1,1} & \sigma_{2,1} & \dots & \sigma_{n,1} \\ \sigma_{1,2} & \sigma_{2,2} & \dots & \sigma_{n,2} \\ \vdots & \vdots & & \vdots \\ \sigma_{1,n} & \sigma_{2,n} & \dots & \sigma_{n,n} \end{bmatrix} \dots \text{ the } \textbf{covariance} \text{ matrix:} \quad \begin{aligned} \sigma_{i,j} &= \overline{(x_i - \mu_i)(x_j - \mu_j)} \\ \sigma_{i,i} &= \sigma_i^2 \end{aligned}$$

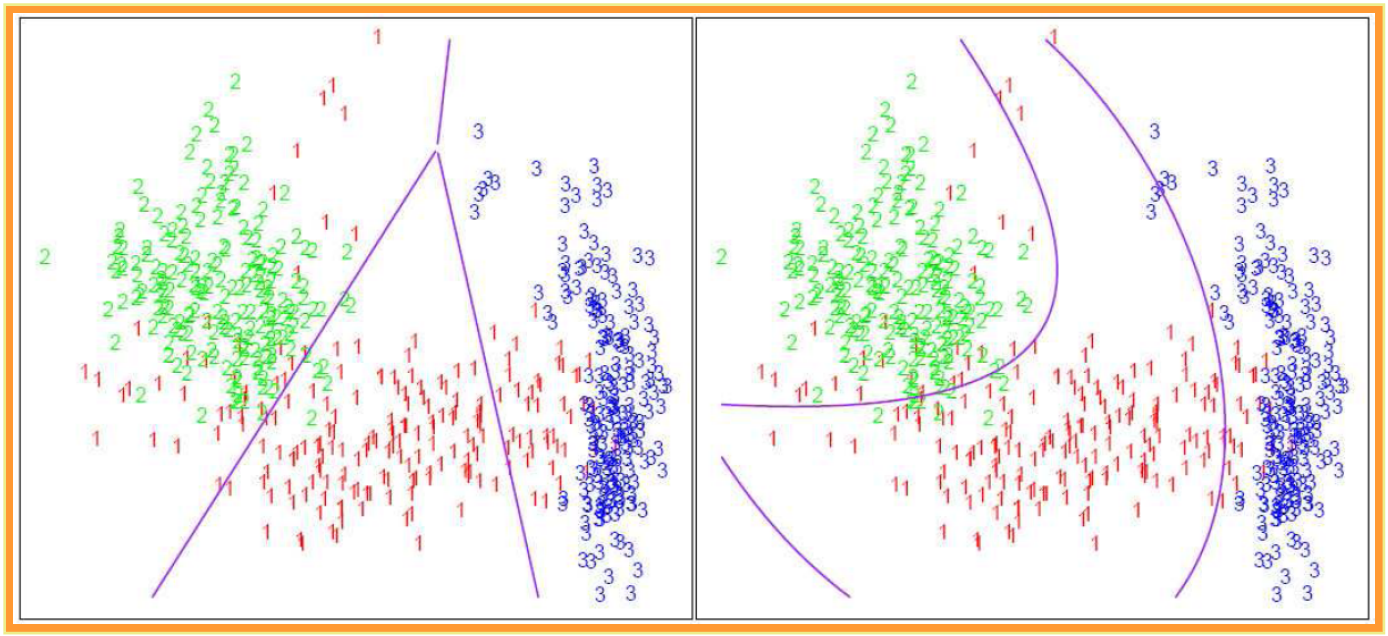
- Normal distribution assumption: $f(x|s, \vec{\mu}, \mathbf{\Sigma}) = N(x, \vec{\mu}_s, \mathbf{\Sigma}_s)$ for each class s.
- **Quadratic** discriminant function $\boxed{g_s(x) = \vec{x}^t \boldsymbol{A}_s \vec{x} + \vec{b}_s^t x + c_s}$ where

$$\boldsymbol{A}_s = -\tfrac{1}{2}\mathbf{\Sigma}_s^{-1} \qquad \vec{b}_s = \mathbf{\Sigma}_s^{-1}\mu_s \qquad c_s = -\tfrac{1}{2}\mu_s^t \mathbf{\Sigma}_s^{-1}\mu_s - \tfrac{1}{2}\ln\det(\mathbf{\Sigma}_s) + \ln P(s)$$

- Special Case:$\forall s\ \Sigma_s = \Sigma$: **Linear** discriminant function $\boxed{g_s(x) = \vec{b}_s^t x + c_s}$ where

$$\vec{b}_s = \mathbf{\Sigma}_s^{-1}\mu_s \qquad c_s = -\tfrac{1}{2}\mu_s^t \mathbf{\Sigma}_s^{-1}\mu_s + \ln P(s)$$

Linear vs. Quadratic Discrimination



- Left: linear discrimination in $\Re^2$. Points where $g_s(\vec{x})$ is maximal for a given s form convex regions with piece-wise linear boundaries.
- Right: quadratic discrimination in $\Re^2$. Points where $g_s(\vec{x})$ is maximal for a given s form regions with piece-wise quadratic boundaries.

Learning: Maximum Likelihood Approach

- Assuming $f(\vec{x}|s)$ **normal**: how does it help learning? Instead of estimating the unknown density function f, we only estimate parameters of the normal distribution $f(\vec{x}|s, \vec{\mu}, \mathbf{\Sigma})$
- That is, estimate $\vec{\mu}_s$ and $\mathbf{\Sigma}_s$ for each class s.
- **Maximum likelihood**: given a sample $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_m$ of class s, find

$$(\hat{\vec{\mu}}_s, \hat{\mathbf{\Sigma}}_s) = \arg\max_{\vec{\mu}, \mathbf{\Sigma}} f(\vec{x}_1, \vec{x}_2, \ldots | s, \vec{\mu}, \mathbf{\Sigma})$$

$$= \arg\max_{\vec{\mu}, \mathbf{\Sigma}} \prod_{i=1}^{m} f(\vec{x}_i|s, \vec{\mu}, \mathbf{\Sigma}) = \arg\max_{\vec{\mu}, \mathbf{\Sigma}} \sum_{i=1}^{m} \ln f(\vec{x}_i|s, \vec{\mu}, \mathbf{\Sigma})$$

  i.e. maximize the likelihood of generating this sample from class s under parameters $\vec{\mu}, \mathbf{\Sigma}$.
- Homework: verify that the solution is, as one would expect:

$$\hat{\vec{\mu}}_s = \frac{1}{m} \sum_{i=1}^{m} \vec{x}_i \qquad\qquad \hat{\mathbf{\Sigma}}_s = \frac{1}{m} \sum_{i=1}^{m} (\vec{x}_i - \hat{\vec{\mu}}_s)(\vec{x}_i - \hat{\vec{\mu}}_s)^t$$

- That is: just calculate the sample mean and the average of m **matrices** $(\vec{x}_i - \hat{\vec{\mu}}_s)(\vec{x}_i - \hat{\vec{\mu}}_s)^t$.
- Do this for all classes s.

Linear Classifier: Direct Learning

- Assume a binary classification problem, i.e. $S = \{s_1, s_2\}$.
- One discriminant function $g(\vec{x})$ enough: classify $y = \begin{cases} s_1, & \text{if } g(\vec{x}) > 0; \\ s_2, & \text{otherwise.} \end{cases}$
- Under the normal distribution assumption, if $\Sigma_{s_1} = \Sigma_{s_2}$, $g(\vec{x})$ is linear, i.e. $g(\vec{x}) = \vec{b}^t\vec{x} + c$.
- Instead of estimating $\vec{\mu}, \mathbf{\Sigma}_s$ and subsequent calculation of $\vec{b}$ and c, we may estimate $\vec{b}, c$ directly from the given sample $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2) \dots (\vec{x}_m, y_m)\}$.
- We want $\left(\vec{b}^t\vec{x}_i + c\right) > 0$ if $y_i = s_1$ and $\left(\vec{b}^t\vec{x}_i + c\right) < 0$ otherwise.
- Same as requesting $\left(\vec{b}^t\vec{z}_i + c\right) > 0$ for all z_i, where $z_i = x_i$ if $y_i = s_1$ and $z_i = -x_i$ otherwise.
- Let formally $z_i^{n+1} = 1 \ \forall i$ and $\vec{w} = [\vec{b}, c]$ (add c as the last component of $\vec{w}$).
- Thus we can write simply $g(\vec{z}) = \vec{w}^t\vec{z}$ and request $\vec{w}^t\vec{z}_i > 0$ for all z_i .
- Let

$$E(\vec{w}) = \sum_{\vec{z}_i \in M} -\vec{w}^t\vec{z}_i$$

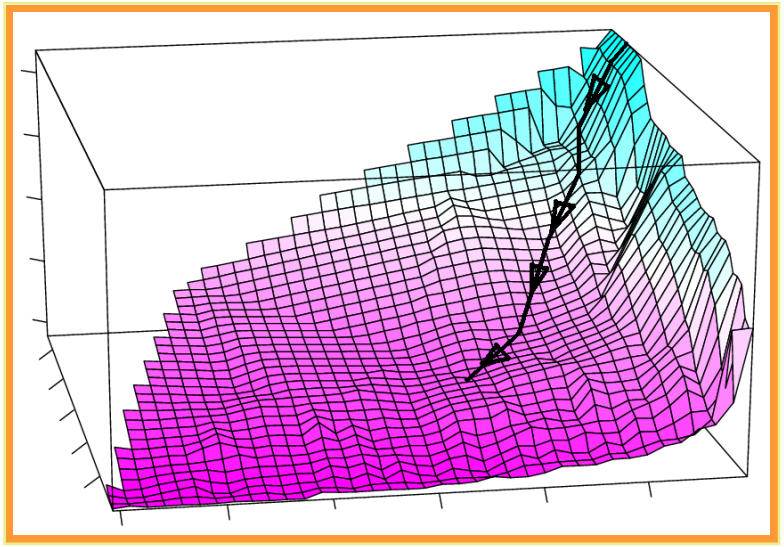
  where M is the set $\vec{z}_i$ that are misclassified.

Perceptron

- $E(\vec{b}, c)$ is always non-negative.
- If $E(\vec{w}) = 0$ then all examples in D are correctly classified and D is **linearly separable**. We want to find the minimum of $E(\vec{w})$.
- $E(\vec{w})$ is piece-wise linear. A **gradient algorithm** can be used to search a minimum.
- Gradient algorithm: go towards a minimum by making discrete steps in $\Re^{n+1}$ in the direction opposite to the gradient of $E(\vec{w})$.

$$\nabla(E(\vec{w})) = \left(\frac{\partial E(\vec{w})}{\partial w_1}, \frac{\partial E(\vec{w})}{\partial w_2}, \cdots \frac{\partial E(\vec{w})}{\partial w_{n+1}}\right) = \sum_{z_i \in M} -\vec{z}$$

- The **perceptron gradient algorithm**:
  1. $k = 0$. Choose a random $\vec{w}$.
  2. $k \leftarrow k + 1$
  3. $\vec{w} \leftarrow \vec{w} + \eta(k) \sum_{z_i \in M_k} \vec{z}$
  4. if $|\nu(k) \sum_{z_i \in M_k} \vec{z}| > \theta$ go to 2
  5. return $\vec{w}$
- η - the **learning rate**, θ - an error threshold.

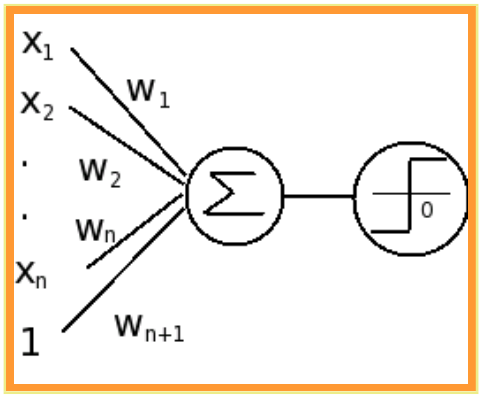


Perceptron: Linear separation

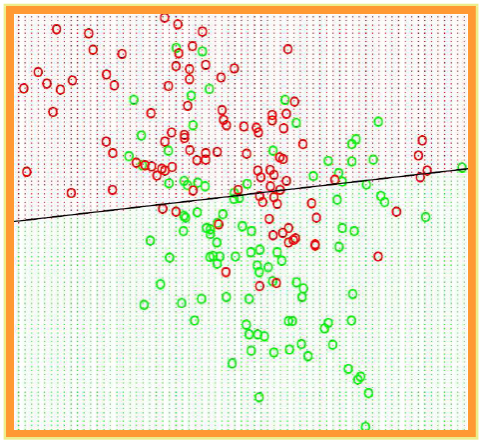
- Perceptrons used in the general tasks of linear discrimination, not constrained to the normal distribution assumption.
- If the two classes are linearly separable, the perceptron algorithm will terminate in a finite number of steps with zero training error.
- A problem that is linearly non-separable in $\Re^n$ may be separable after being *transformed* to $\Re^{n'}$ $n' > n$. For example, new coordinates may contain all quadratic terms:

$$[x(1), \ldots x(n), x^2(1), x(1)x(2), x(1)x(3), \ldots x^2(n)]$$

- This is called **basis expansion**. A linear separation in the expanded space corresponds to a non-linear (here quadratic) separation in the original space $\Re^n$.
- A linear separation method such as the perceptron may be applied in the extended space, generating nonlinear separation in the original space.

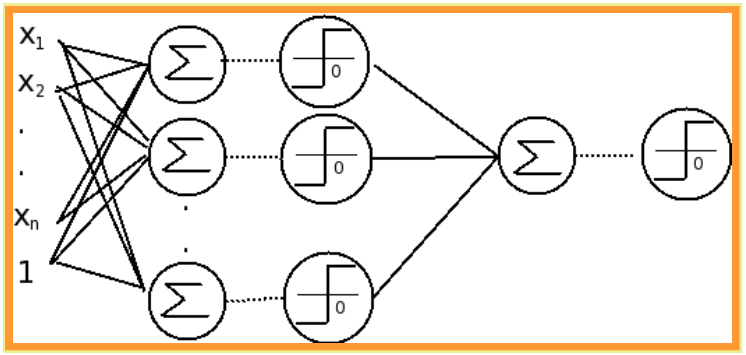


A perceptron scheme


A linearly non-separable problem

A Feedforward Network

- Besides basis expansion, nonlinear separation may also be achieved directly through a more complex, network architecture; p *hidden* units construct new 'features'.



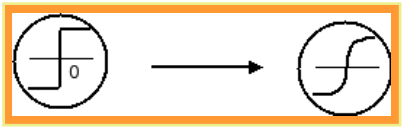
- Here, each full line corresponds to a multiplication coefficient. Denoting the threshold function Θ and assuming that $\vec{x}$ contains the constant 1 as the last component, this network implements a function of the form

$$t(\vec{x}) = \Theta(\vec{v}^t \cdot [\Theta(\vec{w}_1^t \cdot \vec{x}), \ldots \Theta(\vec{w}_p^t \cdot \vec{x})]$$

- We would like to minimize the error on training data D (where $y_i \in \{-1, 1\}$), e.g.

$$E(\vec{v}, \vec{w}_1, \ldots \vec{w}_p) = \sum_{(\vec{x}_i, y_i) \in D} (t(\vec{x}_i) - y_i)^2$$

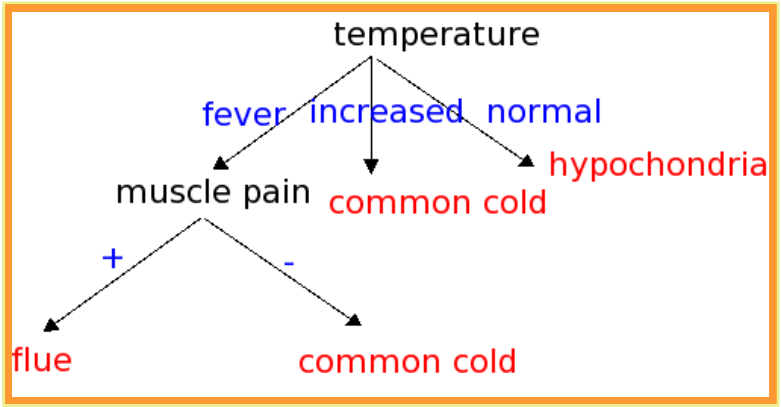
- Due to the thresholds Θ, t is non-differentiable, its gradient not defined and a gradient approach cannot be applied. This can be cured by replacing Θ with a similar, but differentiable function.



- The resulting network is also known as the multi-layer feedforward **artificial neural network**. A gradient algorithm, called the **backpropagation** algorithm is available for minimizing E.

Decision trees

- For many purposes, a classification model is required that a human can directly understand and interpret.
- **Decision trees** are examples of such interpretable models.



- Denote $x(i)$ the i-th component of the example's attribute tuple. for attributes with finite domain (typically nominal attributes), non-leaf vertices correspond to attribute tests in the form

$$x(i) = \textit{value}$$

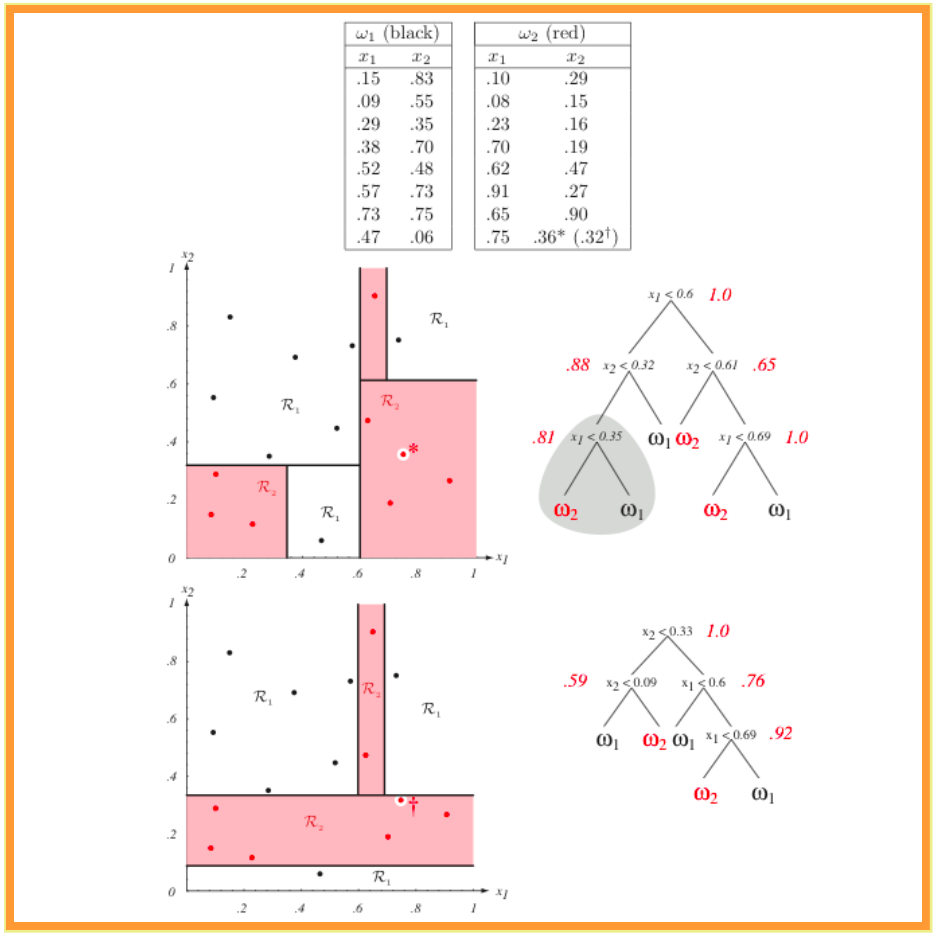
  For attributes with real domain, they may be in the form

$$x(i) \geq \textit{value}, \text{ or } x(i) \leq \textit{value}$$

- Leaves contain predicted classes.
- The predicted class is conditioned by the tests on the path from the root to the leaf.

Decision tree discrimination boundary

- Decision tree classification boundaries in $\Re^n$ are given by axis-parallel hyperplanes.
- Example in $\Re^2$ for binary classification:



Example-weather data

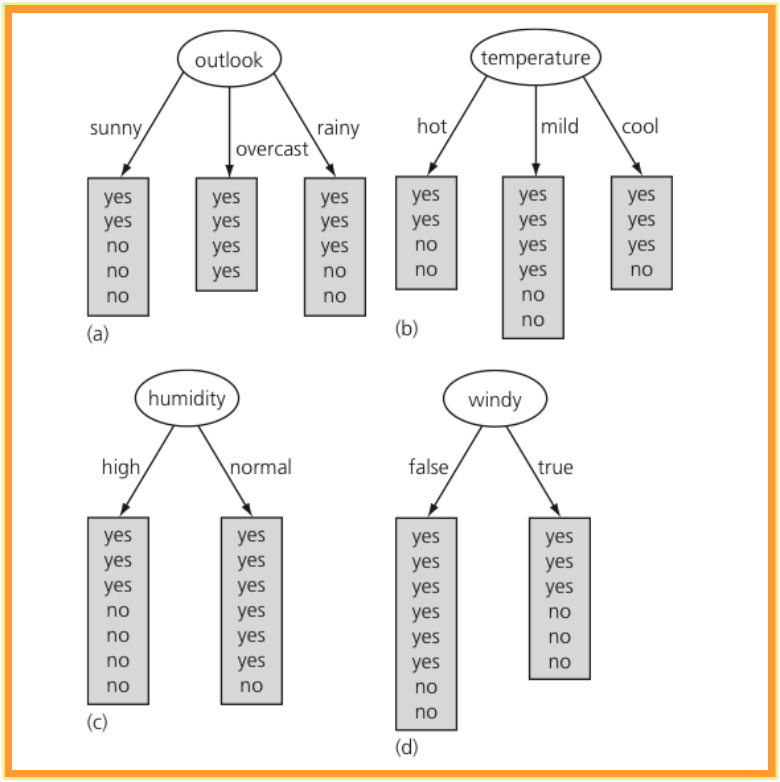
- 9 pozitive YES and 5 negative NO cases

Table 1.2 The weather data.

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

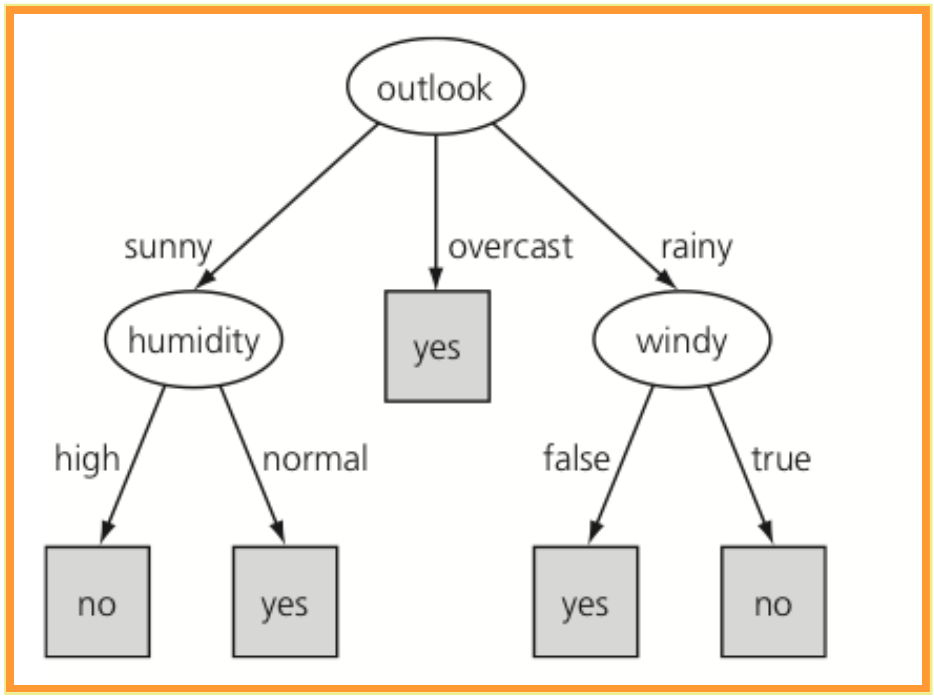
Decision tree construction

- A 'divide-and-conquer' strategy is used for decision tree building from examples.
- Let's define information meassure
- $info([2,3]) = 0.971$,$info([4,0]) = 0.0$, $info([3,2]) = 0.971$
- $info([2,3],[4,0],[3,2]) = (\frac{5}{14})0.971 + (\frac{4}{14})0 + (\frac{5}{14})0.971 = 0.693.$



Decision tree construction

- Let's define information gain
- at root: $info([9,5]) = 0.94$
- Information gain $gain(outlook) = info([9,5]) - info([2,3],[4,0],[3,2]) = 0.940 - 0.693 = 0.247$ bits,
- $gain(temperature) = 0.029$ bits,$gain(humidity) = 0.152$ bits,$gain(windy) = 0.048$ bits
- Select attribute outlook!



Choosing a split attribute

- Entropy of sample D with distribution $p_1, p_2, \ldots p_\gamma$ among γ classes:

$$H(D) = \sum_{i=1}^{\gamma} -p_i \log_2 p_i$$

  $p_i \ldots$ relative frequencies

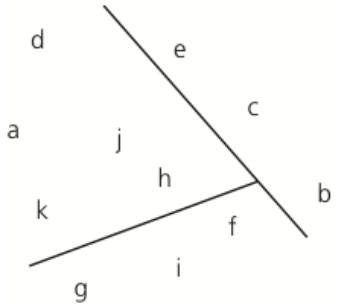
- Minimum $H(D) = 0$, if all examples in the same class.
- Maximum $H(D) = \log_2 \gamma$, if the distribution is uniform.
- Selection heuristic: reduction in entropy after adding a split on attribute x_i

$$G(D, i) = H(D) - \sum_{v_j \in \mathit{Domain}(x(i))} \frac{|E_j|}{|E|} H(E_j)$$

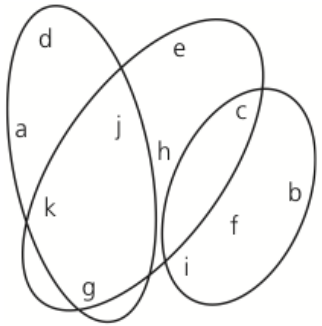
- Sum of entropies in the offsprings weighted by relative sizes of their examples subsets.

Clustering

- No training data
- Natural clusters
- (a) k-means, (b) fuzzy clustering (c) probability using probability mixture , (d) hierarchical clustering (dendogram)



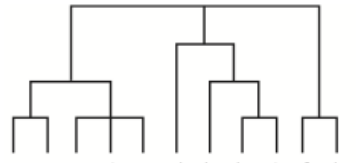
(a)



(b)

	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

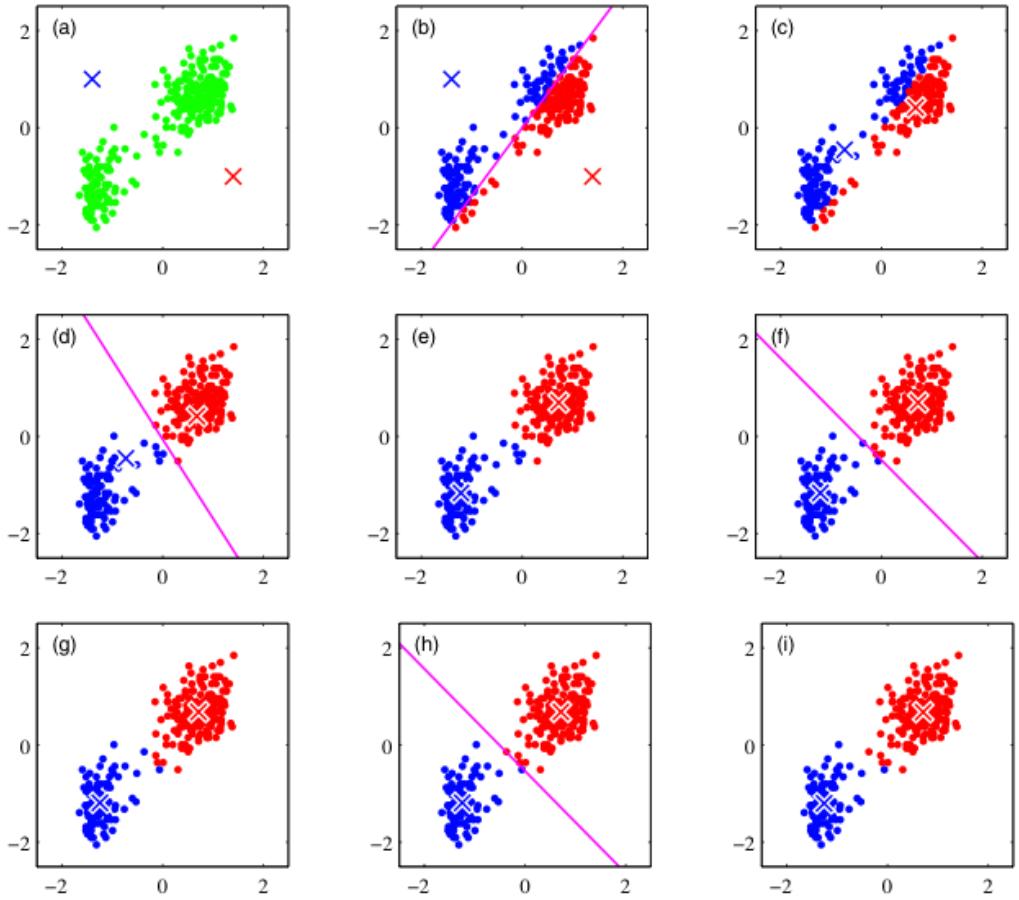
(c)



(d)

K-means

1. beginInicialize $k, \mu_1, \mu_2, \ldots, \mu_k$
2. do classify sample according to nearest μ_i
3. update μ_i
4. until no change μ_i
5. return $\mu_1, \mu_2, \ldots, \mu_k$
6. end



Hierarchical clustering

- agglomerative: bottom-up $\rightarrow$ merging
- divisive: top-down $\rightarrow$ splitting

1. begin Initialize$k, \hat{k} \leftarrow n, \mathcal{D}_i \leftarrow \{X_i\}, i = 1, \ldots, n$
2. do $\hat{k} = \hat{k} - 1$
3. find nearest clusters. $\mathcal{D}_i$ a $\mathcal{D}_j$
4. until $k = \hat{k}$
5. return k clusters
6. end

- $d_{min}(x, x^{'}) = \min \|x - x^{'}\|$, $x \in \mathcal{D}_i$, $x^{'} \in \mathcal{D}_i$

Hierarchical clustering - example