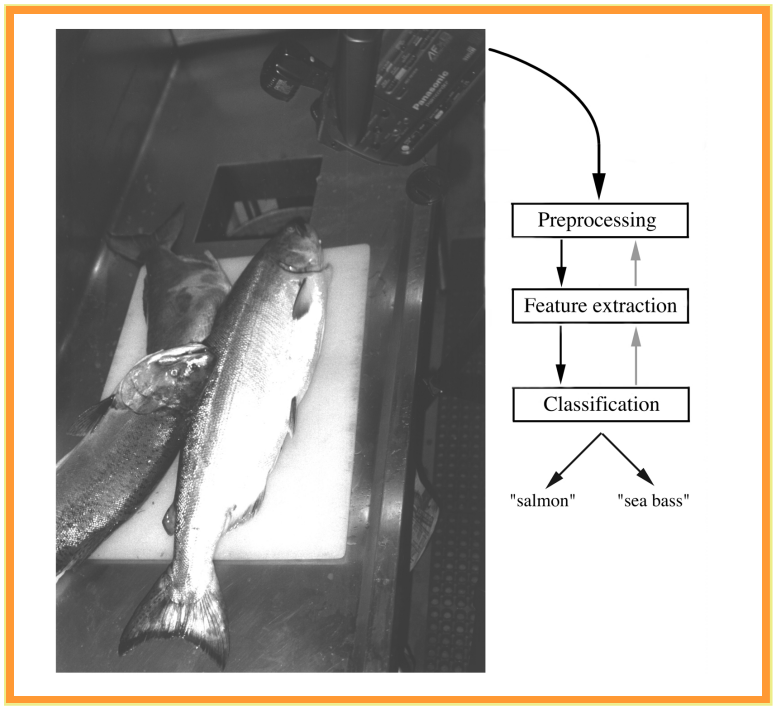
Cybernetics and Artificial Intelligence (2017), lecture 12

Classification – Perceptron, k-nn and relationship to Bayesian classifier

Dept. of Cybernetics
Czech Technical University in Prague
Matěj Hoffmann, Zdeněk Straka
Thanks to: Daniel Novák, Filip Železný

Motivation example – fish classification [Duda, Hart, Stork: Pattern Classification]



- Factory for fish processing
- 2 classes:
  - salmon
  - sea bass
- Features: length, width, lightness etc. from a camera

Last lecture – optimal fish classification using Bayes classifier

- Notation for classification problem
  - Classes $s_i \in S$ (e.g., salmon, sea bass)
  - Features $x_i \in X$ or feature vectors ($\vec{x_i}$) (also called attributes)
- Optimal classification of $\vec{x}$:

$$\delta^*(\vec{x}) = \arg\max_j p(s_j|\vec{x})$$

- Choosing the most probable class for a given feature vector.
- Both likelihood and prior are taken into account – recall Bayes rule:

$$p(s_j|x) = \frac{p(x|s_j)p(s_j)}{p(x)}$$

- E.g., what if 95% of fish are salmon?
  - Prior may become more relevant than features

Bayes classification in practice

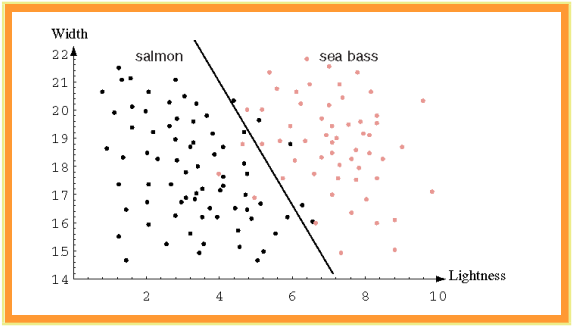
- Usually we **are not given** $P(s|\vec{x})$
- It has to be estimated from already classified examples – training data
- For discrete $\vec{x}$, **training examples** $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \ldots (\vec{x}_l, s_l)$
  - so-called i.i.d (independent, identically distributed) multiset
  - every $(\vec{x}_i, s)$ is drawn independently from $P(\vec{x}, s)$
- Without knowing anything about the distribution, a non-parametric estimate:

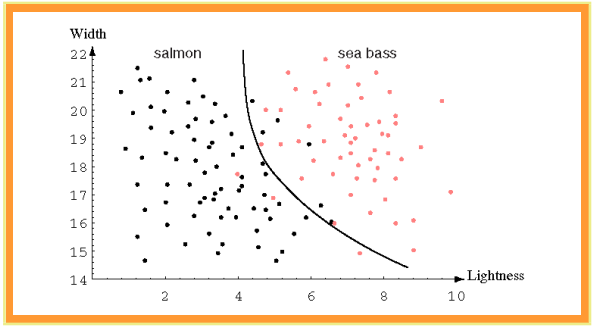
$$P(s|\vec{x}) \approx \frac{\#\text{ examples where } \vec{x}_i = \vec{x} \text{ and } s_i = s}{\#\text{ examples where } \vec{x}_i = \vec{x}}$$

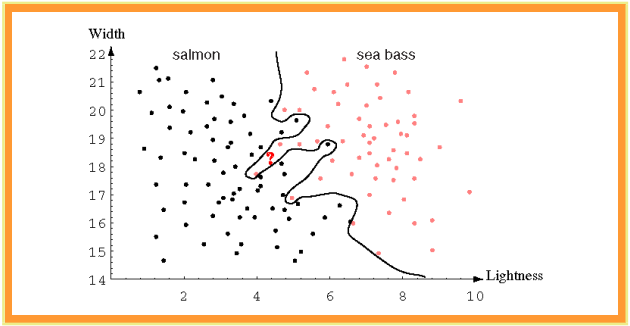
- This is hard in practice:
  - To reliably estimate $P(s|\vec{x})$, the number of examples grows exponentially with the number of elements of $\vec{x}$.
    - e.g. with the number of pixels in images
    - curse of dimensionality
    - denominator often 0
  - The computational curse would not manifest itself if components of $\vec{x}$ were statistically independent, but that is rarely the case.
  - Bayes classification provides a lower bound on classification error, but that is usually not achievable because $P(s|\vec{x})$ is not known.

Alternatives: classification without (probability) density estimation

- In other words, seeking to separate classes on training set in feature space
- Examples
  - **Linear classifier**
    - ∗ **Perceptron algorithm**
  - Quadratic classifier
  - **k-nn - k nearest neighbor**
  - SVM – Support Vector Machines
  - Decision trees







Linear Classifier: Direct Learning

- Assume a binary classification problem, i.e. $S = \{s_1, s_2\}$.
- One discriminant function $g(\vec{x})$ enough: classify $y = \begin{cases} s_1, & \text{if } g(\vec{x}) > 0; \\ s_2, & \text{otherwise.} \end{cases}$
- 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

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

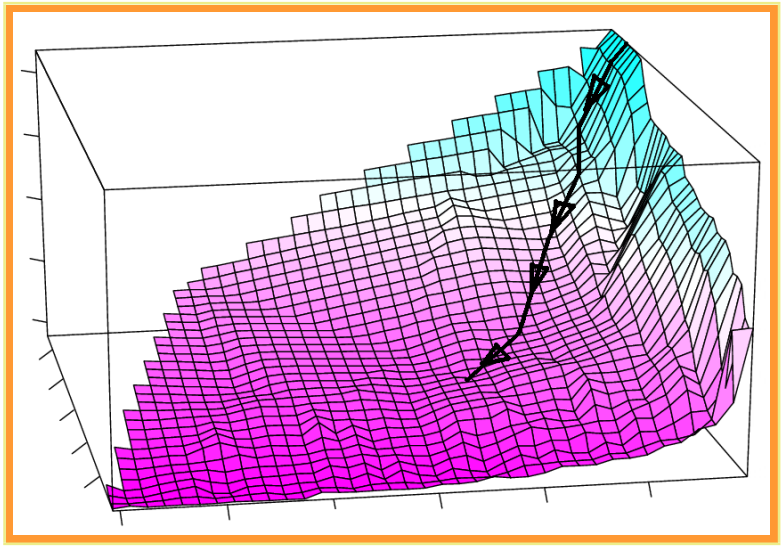
where M is the set of $\vec{z}_i$ that are misclassified.

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 descent algorithm** can be used to search for 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 $|\eta(k) \sum_{z_i \in M_k} \vec{z}| > \theta$ go to 2
  5. return $\vec{w}$
- η - the **learning rate**, θ - an error threshold.

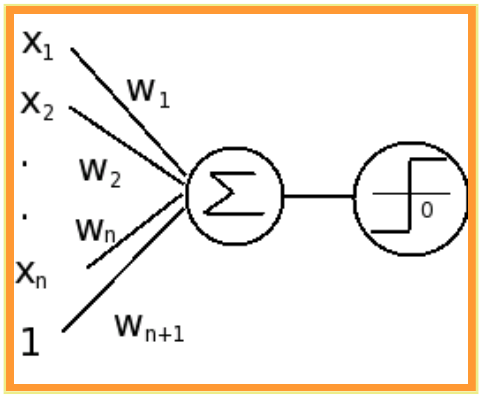


Perceptron: Linear separation

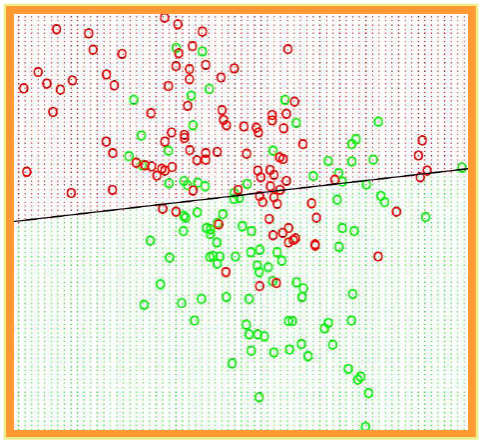
- If the two classes are linearly separable, the perceptron algorithm will terminate in a finite number of steps with zero training error.
- A problem that is linearly non-separable in $\Re^n$ may be separable after being *transformed* to $\Re^{n'}$ $n' > n$. For example, new coordinates may contain 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

Neighbor-based classification

- Assumption: similar objects fall in the same class.
- *Similarity* - small *distance* in X.
- A fuction, called a **metric**: $\rho : X \times X \to \Re$ such that $\forall x, y, z$
  - $\rho(x, y) \geq 0$
  - $\rho(x, x) = 0$
  - $\rho(x, y) = \rho(y, x)$
  - $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$
- **Examples:**
  - **Euclidean metric** for $X = \Re^n$:

$$\rho_E(\vec{x}_1, \vec{x}_2) = \sqrt{\sum_i (x_1(i) - x_2(i))^2}$$

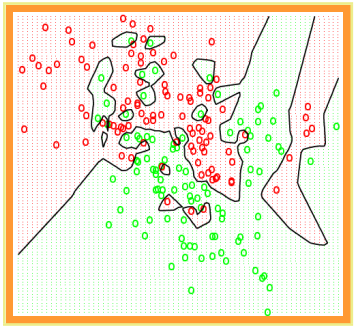
  - For $X = \{0, 1\}^n$, ρ_E^2 is equal to the **Hamming metric**, giving the number of non-equal corresponding components.

k-NN

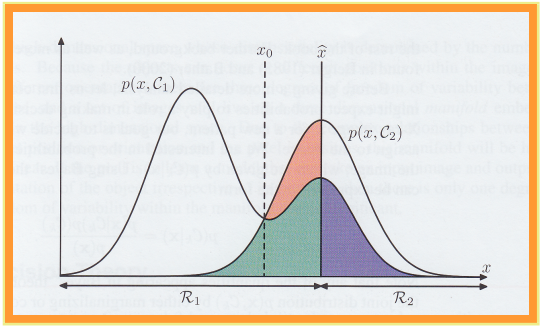
- **k-nearest neighbor classification, k-NN**.
- Given:
  - $k \in N$
  - Training examples: $(\vec{x}_1, s_1), (\vec{x}_2, s_2), \ldots (\vec{x}_l, s_l)$
  - Metric $\rho : X \times X \to \Re$
- Goal: classify $\vec{x}_{l+1}$
- Approach: choose k nearest (to $\vec{x}$ by ρ) examples. Let the majority class therein be the class for $\vec{x}_{l+1}$.

Classification flexibility

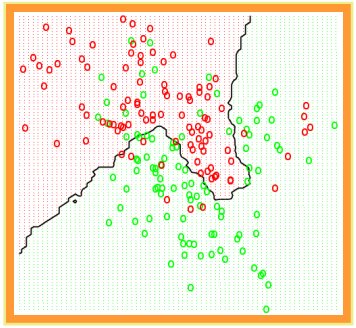
- How to choose k?
- **A general trend**: Consider a two-class problem (red/green) with noisy training examples (some s_i misclassified).



$k = 1$: Good fit of training data, small tolerance to noise.



Bayes classifier: less flexible than 1-nn, more flexible than 15-nn.



$k = 15$: Poor fit to training data. Small sensitivity to noise.

- Note: the shown Bayes classifier was constructed from **known** $P(s|\vec{x})$.
- Observation: with flexibility too large (small k) or too small (large k), one gets classifiers very different from the optimal B/C.
- Optimal k somewhere in the middle. Still pending: how to determine the best value?

Validation

- Mean risk $r(\delta)$ of classifier δ corresponds to the relative frequency of its misclassifications (convergence in the limit...), or 'error rate'.
- Define **training error** $TE(\delta)$ as the error rate on **v training data**.
- Is $TE(\delta)$ a good estimate of $r(\delta)$?
- Earlier: 1-nn is not a good classifier, despite having training error 0.
- $TE(\delta)$ is (usually) not a good estimate of $r(\delta)$ because it is biased. To estimate $r(\delta)$ in an unbiased way:
  - split available data into a **training set** $(\vec{x}_1, s_1), \ldots (\vec{x}_l, s_l)$ and an independent **testing set** $(\vec{x}_{l+1}, s_{l+1}), \ldots (\vec{x}_{l+m}, s_{l+m})$
  - (e.g. by a 75% - 25% split).
  - Construct (train) classifier on the training set.
- Error rate on the testing set is an **unbiased** estimate of $r(\delta)$.
- Unbiased does not mean accurate.

Specific probability distributions

- Recap - optimal classification possible when
  - Complete underlying (joint or conditional) probability distribution relating classes and features is known
  - using Bayes classifier
- However, this is difficult in practice.
- Remedy: assuming a specific probability distribution (with nice properties)

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

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

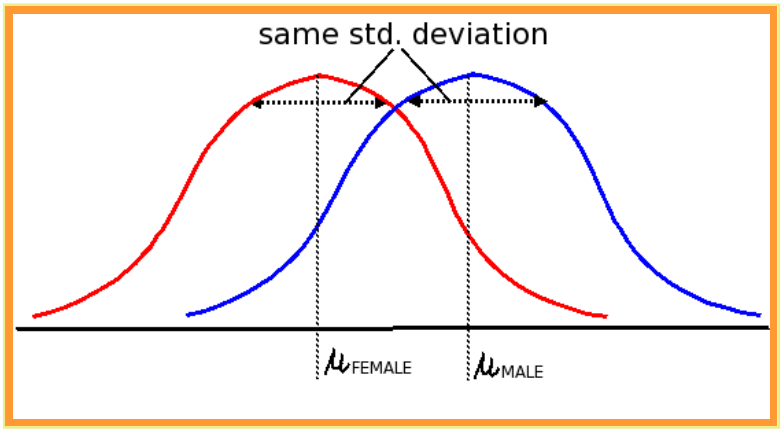
- A quadratic **discriminant function** thus defined **for each** $s \in S$,

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

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

Normal distribution, same std. deviation σ for each class

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



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

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

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, $\vec{x} \in \Re^n$)

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

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

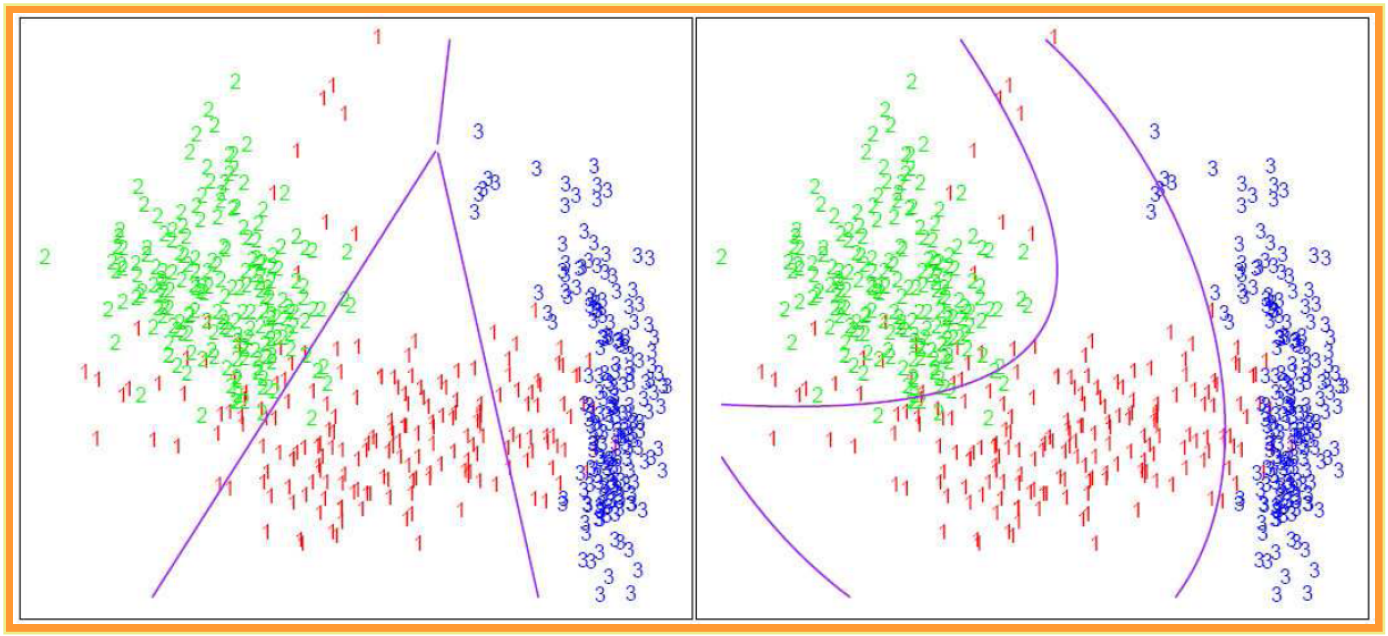
- Normal distribution assumption: $f(x|s, \vec{\mu}, \mathbf{\Sigma}) = N(x, \vec{\mu}_s, \mathbf{\Sigma}_s)$ for each class s.
- **Quadratic** discriminant function $\boxed{g_s(x) = \vec{x}^t \boldsymbol{A}_s \vec{x} + \vec{b}_s^t x + c_s}$ where

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

- 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.

Parameter estimation

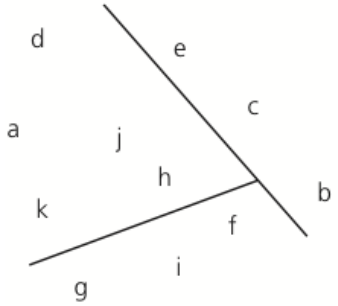
- Assuming $f(\vec{x}|s)$ **normal**: how does it help learning? Instead of estimating the unknown density function f, we only estimate parameters of the normal distribution $f(\vec{x}|s, \vec{\mu}, \mathbf{\Sigma})$
- That is, estimate $\vec{\mu}_s$ and $\mathbf{\Sigma}_s$ for each class s.
- Several options (next lecture)
  - Maximum Likelihood
  - Maximum Aposteriori
  - Bayesian inference

Unsupervised learning

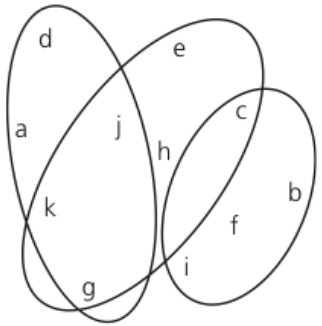
- Until now:
  - labeled samples $(\vec{x}, s)$ – features and category membership
  - **supervised learning**
- Unlabeled samples $\rightarrow$ **unsupervised learning**
- Why? [Duda, Hart, Stork: Pattern Classification, Ch. 10]
  1. labeled data sets are costly
  2. useful features can be extracted without supervision
  3. intrinsic structure in the data – e.g. natural clusters

Clustering

- (a) k-means, (b) fuzzy clustering, (c) probability using probability mixture, (d) hierarchical clustering (dendrogram)



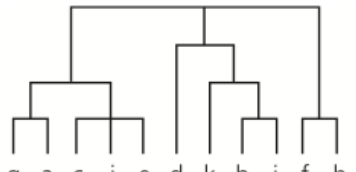
(a)



(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)

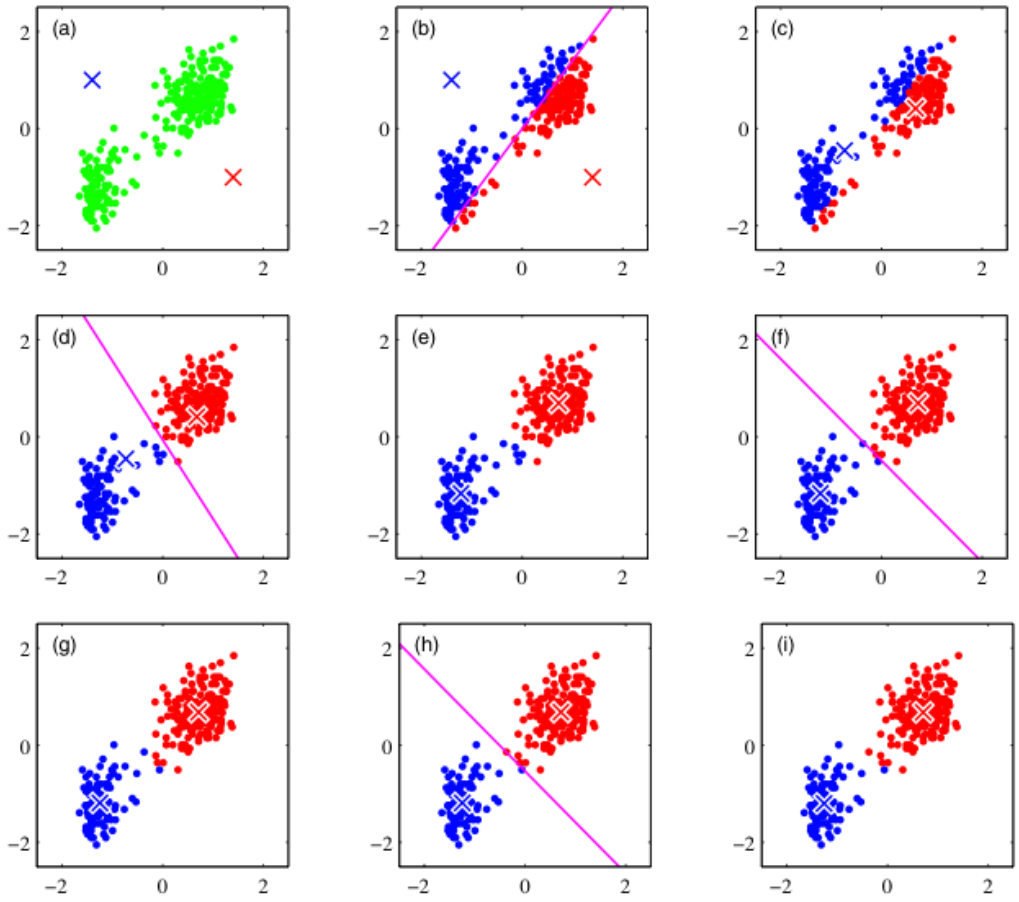


(d)

K-means

- with n input patterns
- searching for centers (means μ) of k clusters

1. begin Initialize $n, k, \mu_1, \mu_2, \ldots, \mu_k$
2. do classify n samples 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