Outline

- Recognition Based on Decision-Theoretic Methods
  - Optimum Statistical Classifiers
  - Neural Networks

Optimum Statistical Classifiers

- Previously, we had considered very simple classifiers without using any high order statistics of the underlying patterns
- If we have knowledge about some sort of statistics of the patterns, we can use information to design a classified that would minimize the probability of incorrect classifications
- Let us define the probability that the pattern $\boldsymbol{x}$ belongs to ω_i as $p(\omega_i/\boldsymbol{x})$
- If our classifier incorrectly decides on ω_j, then we define this error as L_{ij}
- Averaging this error for all classes we obtain the average error as

$$r_j(\boldsymbol{x}) = \sum_{k=1}^{W} L_{kj} p(\omega_k/\boldsymbol{x})$$

Optimum Statistical Classifiers

- Using the Bayes' rule we obtain

$$r_j(\boldsymbol{x}) = \frac{1}{p(\boldsymbol{x})} \sum_{k=1}^{W} L_{kj} p(\boldsymbol{x}/\omega_k) P(\omega_k)$$

- We use this average error as our cost function, and choose the class that results in the minimum error
- All r_j's are calculated for $j = 1, 2, \ldots, W$ and the class with minimum error is selected
- Now let us select a value for L_{ij}, a reasonable choice is to make it zero when $i = j$ and one otherwise
- Then we have

$$d_j(\boldsymbol{x}) = r_j(\boldsymbol{x}) = p(\boldsymbol{x}/\omega_j) P(\omega_j)$$

Optimum Statistical Classifiers

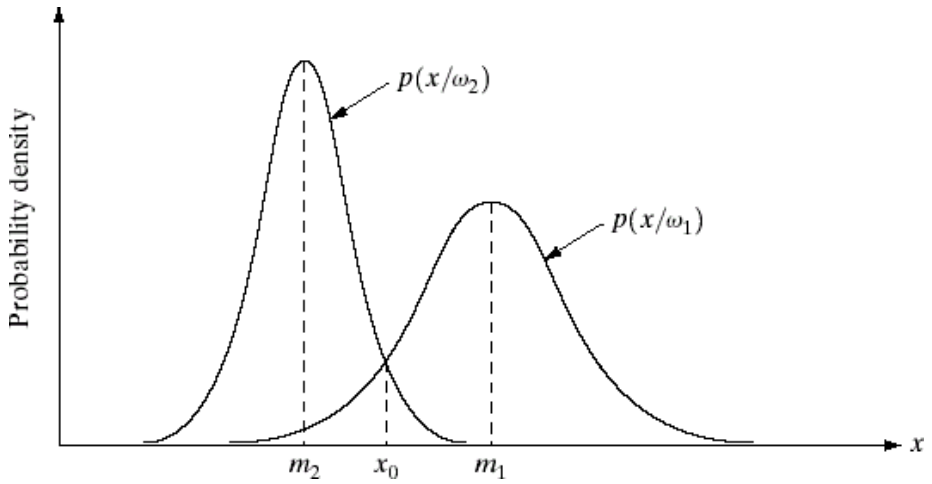
- To be able to calculate this optimum classifier we need to know the probability of each class ($P(\omega)$) and the pdf of the individual patters must be known
- It is a reasonable assumption that we can infer the knowledge of $P(\omega)$ based on the problem at hand
- However, it is not an easy task to estimate $p(\boldsymbol{x}/\omega_j)$
- Therefore, instead of estimating these directly, we use a parametric model and estimate a few parameters hence obtain an approximate of $p(\boldsymbol{x}/\omega_j)$'s

Optimum Statistical Classifiers

- Let us use a Gaussian model and assume a scalar x for simplicity. Then the decision function is

$$d_j(x) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x-m_j)^2}{2\sigma_j^2}} P(\omega_j)$$

- If we assume that we have only two classes the problem of classifying is summarized in the following figure



Optimum Statistical Classifiers

- The 1-D example illustrates the idea, but in reality we will almost always have multidimensional patterns
- Then the Gaussian will be multivariate

$$p(\boldsymbol{x}/\omega_j) = \frac{1}{(2\pi)^{\frac{n}{2}}|C_j|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_j)^{\mathrm{T}} C_j^{-1}(\boldsymbol{x}-\boldsymbol{m}_j)}$$

  where C_j is the covariance matrix and $\boldsymbol{m}_j$ is the mean vector

- These quantities can be estimated based on the samples

$$\hat{\boldsymbol{m}}_j = \frac{1}{N_j}\sum \boldsymbol{x}$$

$$\hat{C}_j = \frac{1}{N_j}\sum \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} - \boldsymbol{m}_j\boldsymbol{m}_j^{\mathrm{T}}$$

Optimum Statistical Classifiers

- Now let us minimize $d_j(\boldsymbol{x})$ by minimizing its log (which is a monoton function)

$$\log P(\omega_j) - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log|C_j| - \frac{1}{2}[(\boldsymbol{x}-\boldsymbol{m}_j)^{\mathrm{T}} C_j^{-1}(\boldsymbol{x}-\boldsymbol{m}_j)]$$

- This expression has a special form when all covariance matrices are identity and all classes are equally probable

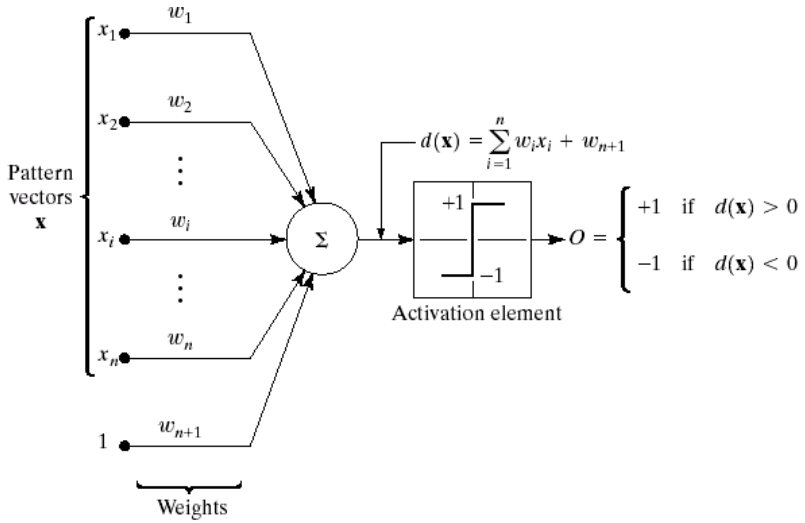
$$d_j(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{m}_j - \frac{1}{2}\boldsymbol{m}_j^{\mathrm{T}}\boldsymbol{m}_j$$

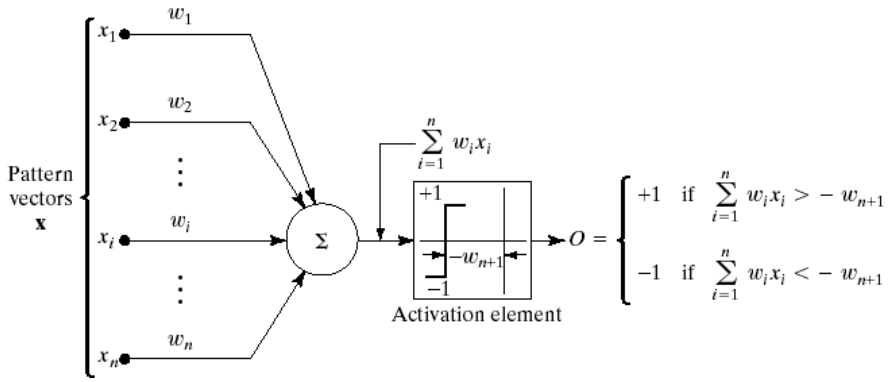
- Remember this! Then the minimum distance classifier is the optimum bayesian classifier when all pattern classes are Gaussian distributed with identity covariance matrix and all classes are equally probable

Neural Networks: Introduction

- Instead of assuming a parametric model for the pdf's of patterns, we can use an approach called "training"
- Known patterns are used first for "training" and learning. In this way, the decision functions are created based on these known patterns
- This approach brings us to neural networks

Neural Networks: A simple two pattern classes example





Neural Networks: A simple two pattern classes example

- The decision is based on $d(\boldsymbol{x})$, for positive $d(\boldsymbol{x})$ the particular class is selected, for negative $d(\boldsymbol{x})$ it is not selected
- When $d(\boldsymbol{x})$ is exactly zero, we can obtain the decision boundary

$$d(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_{n+1}$$

- To be able to reconstruct a decision function we need to calculate the weights with training algorithms

Neural Networks: Training Algorithms - Linearly Separable Classes

- We select an initial coefficient vector $\boldsymbol{w}(1)$ and then using the training data at each class, update the coefficient vector as follows

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + c\boldsymbol{y}(k) \quad \text{when} \quad \boldsymbol{w}^{\mathrm{T}}(k)\boldsymbol{y}(k) < 0, \boldsymbol{y}(k)\epsilon\omega_1$$
$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - c\boldsymbol{y}(k) \quad \text{when} \quad \boldsymbol{w}^{\mathrm{T}}(k)\boldsymbol{y}(k) > 0, \boldsymbol{y}(k)\epsilon\omega_2$$

  where c is a positive constant $\boldsymbol{y} = [x_1, x_2, \ldots, x_n, 1]$, and $\boldsymbol{w} = [w_1, w_2, \ldots, w_{n+1}]$

- This algorithm converges when the two sets are linearly separable

Neural Networks: Training Algorithms - Nonseparable classes, Widrow-Hoff Method

- This training method can be used with linearly nonseparable classes
- The algorithm minimizes the error between the desired and actual response

$$[r - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{y}]^2$$

where r is the desired response, 1 if the training pattern belongs to class 1, and -1 if it belongs to class 2

- This cost function can be minimized using gradiend based numerical optimization methods

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \alpha \left[\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}\right]_{\boldsymbol{w}=\boldsymbol{w}(k)}$$

Neural Networks: Training Algorithms - Nonseparable classes, Widrow-Hoff Method

- Substituting the derivative of the cost function we can obtain

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \alpha[r - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{y}]\boldsymbol{y}$$

- The only parameter to be selected is α, a reasonable choice is between 0.1 and 1