

Machine Learning

A Bayesian and Optimization Perspective

Academic Press, 2015

Sergios Theodoridis¹

¹Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece.

Winter 2015, Version II

Chapter 7

Classification: A Tour Of The Classics

- In Chapter 3, the classification task was introduced and a linear classifier was designed via the sum of squared errors loss criterion. However, the LS estimator is an efficient one, **only** if the conditional distribution of the output variable, y , given the feature values, $\mathbf{x} = \mathbf{x}$, follows a **Gaussian distribution of a special type**. However, in classification, the dependent variable is **discrete**, hence it is **not** Gaussian; thus, the use of the LS criterion **cannot be justified**, in general.
- Here, the classification task will be treated via **Bayesian decision theory** arguments. Bayesian classification is a typical **generative method**, and its very essence comprises the estimation of the **joint probability** $p(y, \mathbf{x})$, $y \in D$, $\mathbf{x} \in \mathbb{R}^l$, where D is a discrete set of class labels.
- **Bayesian Classification Rule**: Given a set of M classes, ω_i , $i = 1, 2, \dots, M$, as well as the respective **posterior probabilities**, $P(\omega_i|\mathbf{x})$, classify an unknown feature vector, \mathbf{x} , according to the rule,

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} P(\omega_j|\mathbf{x}), \quad j = 1, 2, \dots, M.$$

In words, the unknown pattern, represented by \mathbf{x} , is **assigned to the class for which the posterior probability becomes maximum**.

- In Chapter 3, the classification task was introduced and a linear classifier was designed via the sum of squared errors loss criterion. However, the LS estimator is an efficient one, **only** if the conditional distribution of the output variable, y , given the feature values, $\mathbf{x} = \mathbf{x}$, follows a **Gaussian distribution of a special type**. However, in classification, the dependent variable is **discrete**, hence it is **not** Gaussian; thus, the use of the LS criterion **cannot be justified**, in general.
- Here, the classification task will be treated via **Bayesian decision theory** arguments. Bayesian classification is a typical **generative method**, and its very essence comprises the estimation of the **joint probability** $p(y, \mathbf{x})$, $y \in D$, $\mathbf{x} \in \mathbb{R}^l$, where D is a discrete set of class labels.
- **Bayesian Classification Rule:** Given a set of M classes, ω_i , $i = 1, 2, \dots, M$, as well as the respective **posterior probabilities**, $P(\omega_i|\mathbf{x})$, classify an unknown feature vector, \mathbf{x} , according to the rule,

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} P(\omega_j|\mathbf{x}), \quad j = 1, 2, \dots, M.$$

In words, the unknown pattern, represented by \mathbf{x} , is **assigned to the class for which the posterior probability becomes maximum**.

- In Chapter 3, the classification task was introduced and a linear classifier was designed via the sum of squared errors loss criterion. However, the LS estimator is an efficient one, **only** if the conditional distribution of the output variable, y , given the feature values, $\mathbf{x} = \mathbf{x}$, follows a **Gaussian distribution of a special type**. However, in classification, the dependent variable is **discrete**, hence it is **not** Gaussian; thus, the use of the LS criterion **cannot be justified**, in general.
- Here, the classification task will be treated via **Bayesian decision theory** arguments. Bayesian classification is a typical **generative method**, and its very essence comprises the estimation of the **joint probability** $p(y, \mathbf{x})$, $y \in D$, $\mathbf{x} \in \mathbb{R}^l$, where D is a discrete set of class labels.
- **Bayesian Classification Rule**: Given a set of M classes, ω_i , $i = 1, 2, \dots, M$, as well as the respective **posterior probabilities**, $P(\omega_i|\mathbf{x})$, classify an unknown feature vector, \mathbf{x} , according to the rule,

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} P(\omega_j|\mathbf{x}), \quad j = 1, 2, \dots, M.$$

In words, the unknown pattern, represented by \mathbf{x} , is **assigned to the class for which the posterior probability becomes maximum**.

- Note that prior to receiving any observation, our **uncertainty** concerning the classes is expressed via the **prior probabilities**, denoted by $P(\omega_i)$, $i = 1, 2, \dots, M$. Once the observation \mathbf{x} has been obtained, this extra information **removes part of our original uncertainty**, and the related statistical information is now provided by the **posterior probabilities**, which are then used for the classification.
- Employing Bayes' theorem,

$$P(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)P(\omega_j)}{p(\mathbf{x})}, \quad j = 1, 2, \dots, M,$$

where $p(\mathbf{x}|\omega_j)$ are the respective **conditional** probability distribution densities (pdf), the Bayesian classification rule becomes,

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} p(\mathbf{x}|\omega_j)P(\omega_j), \quad j = 1, 2, \dots, M.$$

In other words, the classifier depends on the a-priori class probabilities and the respective conditional pdfs.

- Also, note that

$$p(\mathbf{x}|\omega_j)P(\omega_j) = p(\omega_j, \mathbf{x}) := p(y, \mathbf{x}).$$

That is, the Bayesian classifier is a generative modeling technique.

- Note that prior to receiving any observation, our **uncertainty** concerning the classes is expressed via the **prior probabilities**, denoted by $P(\omega_i)$, $i = 1, 2, \dots, M$. Once the observation \mathbf{x} has been obtained, this extra information **removes part of our original uncertainty**, and the related statistical information is now provided by the **posterior probabilities**, which are then used for the classification.
- Employing Bayes' theorem,

$$P(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)P(\omega_j)}{p(\mathbf{x})}, \quad j = 1, 2, \dots, M,$$

where $p(\mathbf{x}|\omega_j)$ are the respective **conditional** probability distribution densities (pdf), the Bayesian classification rule becomes,

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} p(\mathbf{x}|\omega_j)P(\omega_j), \quad j = 1, 2, \dots, M.$$

In other words, the classifier depends on the a-priori class probabilities and the respective conditional pdfs.

- Also, note that

$$p(\mathbf{x}|\omega_j)P(\omega_j) = p(\omega_j, \mathbf{x}) := p(y, \mathbf{x}).$$

That is, the Bayesian classifier is a generative modeling technique.

- Note that prior to receiving any observation, our **uncertainty** concerning the classes is expressed via the **prior probabilities**, denoted by $P(\omega_i)$, $i = 1, 2, \dots, M$. Once the observation \mathbf{x} has been obtained, this extra information **removes part of our original uncertainty**, and the related statistical information is now provided by the **posterior probabilities**, which are then used for the classification.
- Employing Bayes' theorem,

$$P(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)P(\omega_j)}{p(\mathbf{x})}, \quad j = 1, 2, \dots, M,$$

where $p(\mathbf{x}|\omega_j)$ are the respective **conditional** probability distribution densities (pdf), the Bayesian classification rule becomes,

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} p(\mathbf{x}|\omega_j)P(\omega_j), \quad j = 1, 2, \dots, M.$$

In other words, the classifier depends on the a-priori class probabilities and the respective conditional pdfs.

- Also, note that

$$p(\mathbf{x}|\omega_j)P(\omega_j) = p(\omega_j, \mathbf{x}) := p(y, \mathbf{x}).$$

That is, the Bayesian classifier is a generative modeling technique.

- **Training Bayesian Classifiers:** Let us assume that we are given a set of training points, $(y_n, \mathbf{x}_n) \in D \times \mathbb{R}^l$, $n = 1, 2, \dots, N$, and consider the general task comprising M classes. Assume that each class, ω_i , $i = 1, 2, \dots, M$, is represented by N_i points in the training set, with $\sum_{i=1}^M N_i = N$. Then, the a-priori probabilities can be approximated by,

$$P(\omega_i) \approx \frac{N_i}{N}, \quad i = 1, 2, \dots, M.$$

- For the conditional pdfs, $p(\mathbf{x}|\omega_i)$, $i = 1, 2, \dots, M$, any method for estimating pdfs can be mobilized. For example, one can assume a known parametric form for each one of the conditionals and adopt the **Maximum Likelihood method (ML)**, or the **Maximum a-posteriori (MAP) estimator**, in order to obtain estimates of the parameters, using the training data from each one of the classes. Another alternative is to resort to **nonparametric histogram-like techniques**. Other methods for pdf estimation can also be employed, such as mixture modeling.

- **Training Bayesian Classifiers:** Let us assume that we are given a set of training points, $(y_n, \mathbf{x}_n) \in D \times \mathbb{R}^l$, $n = 1, 2, \dots, N$, and consider the general task comprising M classes. Assume that each class, ω_i , $i = 1, 2, \dots, M$, is represented by N_i points in the training set, with $\sum_{i=1}^M N_i = N$. Then, the a-priori probabilities can be approximated by,

$$P(\omega_i) \approx \frac{N_i}{N}, \quad i = 1, 2, \dots, M.$$

- For the conditional pdfs, $p(\mathbf{x}|\omega_i)$, $i = 1, 2, \dots, M$, any method for estimating pdfs can be mobilized. For example, one can assume a known parametric form for each one of the conditionals and adopt the **Maximum Likelihood method (ML)**, or the **Maximum a-posteriori (MAP) estimator**, in order to obtain estimates of the parameters, using the training data from each one of the classes. Another alternative is to resort to **nonparametric histogram-like techniques**. Other methods for pdf estimation can also be employed, such as mixture modeling.

The Bayesian Classifier Minimizes The Misclassification Error

- Recall that the goal of any classifier is to **partition the space**, in which the feature vectors lie, into regions and **associate each one of the regions to one and only one class**. For a two-class task, let $\mathcal{R}_1, \mathcal{R}_2$ be the two regions in, say, \mathbb{R}^l , where we decide in favor of class ω_1 and ω_2 , respectively. The probability of classification error is given by

$$P_e = P(\mathbf{x} \in \mathcal{R}_1, \mathbf{x} \in \omega_2) + P(\mathbf{x} \in \mathcal{R}_2, \mathbf{x} \in \omega_1).$$

That is, it is equal to the probability of the feature vector to belong to class ω_1 (ω_2) **and at the same time** to lie in the “wrong” region \mathcal{R}_2 (\mathcal{R}_1) in the feature space.

- The previous Equation can be written as,

$$P_e = P(\omega_2) \int_{\mathcal{R}_1} p(\mathbf{x}|\omega_2)d\mathbf{x} + P(\omega_1) \int_{\mathcal{R}_2} p(\mathbf{x}|\omega_1)d\mathbf{x}.$$

- It turns out that the Bayesian classifier **minimizes** P_e with respect to \mathcal{R}_1 and \mathcal{R}_2 . This is also true for the general case of M classes.

The Bayesian Classifier Minimizes The Misclassification Error

- Recall that the goal of any classifier is to **partition the space**, in which the feature vectors lie, into regions and **associate each one of the regions to one and only one class**. For a two-class task, let $\mathcal{R}_1, \mathcal{R}_2$ be the two regions in, say, \mathbb{R}^l , where we decide in favor of class ω_1 and ω_2 , respectively. The probability of classification error is given by

$$P_e = P(\mathbf{x} \in \mathcal{R}_1, \mathbf{x} \in \omega_2) + P(\mathbf{x} \in \mathcal{R}_2, \mathbf{x} \in \omega_1).$$

That is, it is equal to the probability of the feature vector to belong to class ω_1 (ω_2) **and at the same time** to lie in the “wrong” region \mathcal{R}_2 (\mathcal{R}_1) in the feature space.

- The previous Equation can be written as,

$$P_e = P(\omega_2) \int_{\mathcal{R}_1} p(\mathbf{x}|\omega_2)d\mathbf{x} + P(\omega_1) \int_{\mathcal{R}_2} p(\mathbf{x}|\omega_1)d\mathbf{x}.$$

- It turns out that the Bayesian classifier **minimizes** P_e with respect to \mathcal{R}_1 and \mathcal{R}_2 . This is also true for the general case of M classes.

The Bayesian Classifier Minimizes The Misclassification Error

- Recall that the goal of any classifier is to **partition the space**, in which the feature vectors lie, into regions and **associate each one of the regions to one and only one class**. For a two-class task, let $\mathcal{R}_1, \mathcal{R}_2$ be the two regions in, say, \mathbb{R}^l , where we decide in favor of class ω_1 and ω_2 , respectively. The probability of classification error is given by

$$P_e = P(\mathbf{x} \in \mathcal{R}_1, \mathbf{x} \in \omega_2) + P(\mathbf{x} \in \mathcal{R}_2, \mathbf{x} \in \omega_1).$$

That is, it is equal to the probability of the feature vector to belong to class ω_1 (ω_2) **and at the same time** to lie in the “wrong” region \mathcal{R}_2 (\mathcal{R}_1) in the feature space.

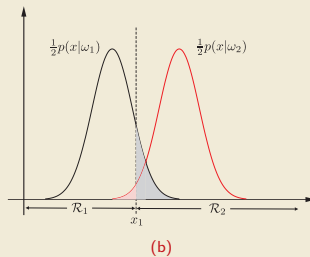
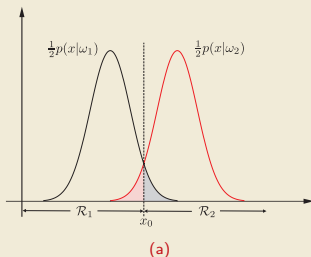
- The previous Equation can be written as,

$$P_e = P(\omega_2) \int_{\mathcal{R}_1} p(\mathbf{x}|\omega_2)d\mathbf{x} + P(\omega_1) \int_{\mathcal{R}_2} p(\mathbf{x}|\omega_1)d\mathbf{x}.$$

- It turns out that the Bayesian classifier **minimizes** P_e with respect to \mathcal{R}_1 and \mathcal{R}_2 . This is also true for the general case of M classes.

The Bayesian Classifier Minimizes The Misclassification Error

- The previous probability of error minimization property of the Bayesian classifier is illustrated in the following figures:



a) The classification error probability for partitioning the feature space, according to the Bayesian optimal classifier, is equal to the area of the shaded region. b) Moving the threshold value away from the value corresponding to the optimal Bayes rule increases the probability of error, as it is indicated by the increase of the area of the corresponding shaded region.

The Bayesian Classifier Minimizes The Misclassification Error

- Since in classification the dependent variable (label), y , is of a discrete nature, the classification error probability may sound to be the most natural cost function to be optimized. However, this is not always true. For example, in a medical diagnosis system, committing an error by predicting the class of a finding in an x-ray image as being “malignant”, while its true class is “normal”, is less significant than an error in the other way round.
- For such cases, **relative weights on the errors** according to their **importance** have to be used. The resulting function is known as the **average risk**.
- For the M -class problem, the **risk** or **loss** associated with class ω_k is defined as,

$$r_k = \sum_{i=1}^M \lambda_{ki} \int_{\mathcal{R}_i} p(\mathbf{x}|\omega_k) d\mathbf{x},$$

where, $\lambda_{kk} = 0$ and λ_{ki} is the weight that controls the significance of committing an error by assigning a pattern from class ω_k to class ω_i .

- The **average risk** is given by,

$$r = \sum_{k=1}^M P(\omega_k) r_k = \sum_{i=1}^M \int_{\mathcal{R}_i} \left(\sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) \right) d\mathbf{x}.$$

The Bayesian Classifier Minimizes The Misclassification Error

- Since in classification the dependent variable (label), y , is of a discrete nature, the classification error probability may sound to be the most natural cost function to be optimized. However, this is not always true. For example, in a medical diagnosis system, committing an error by predicting the class of a finding in an x-ray image as being “malignant”, while its true class is “normal”, is less significant than an error in the other way round.
- For such cases, **relative weights on the errors** according to their **importance** have to be used. The resulting function is known as the **average risk**.
- For the M -class problem, the **risk** or **loss** associated with class ω_k is defined as,

$$r_k = \sum_{i=1}^M \lambda_{ki} \int_{\mathcal{R}_i} p(\mathbf{x}|\omega_k) d\mathbf{x},$$

where, $\lambda_{kk} = 0$ and λ_{ki} is the weight that controls the significance of committing an error by assigning a pattern from class ω_k to class ω_i .

- The **average risk** is given by,

$$r = \sum_{k=1}^M P(\omega_k) r_k = \sum_{i=1}^M \int_{\mathcal{R}_i} \left(\sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) \right) d\mathbf{x}.$$

The Bayesian Classifier Minimizes The Misclassification Error

- Since in classification the dependent variable (label), y , is of a discrete nature, the classification error probability may sound to be the most natural cost function to be optimized. However, this is not always true. For example, in a medical diagnosis system, committing an error by predicting the class of a finding in an x-ray image as being “malignant”, while its true class is “normal”, is less significant than an error in the other way round.
- For such cases, **relative weights on the errors** according to their **importance** have to be used. The resulting function is known as the **average risk**.
- For the M -class problem, the **risk** or **loss** associated with class ω_k is defined as,

$$r_k = \sum_{i=1}^M \lambda_{ki} \int_{\mathcal{R}_i} p(\mathbf{x}|\omega_k) d\mathbf{x},$$

where, $\lambda_{kk} = 0$ and λ_{ki} is the weight that controls the significance of committing an error by assigning a pattern from class ω_k to class ω_i .

- The **average risk** is given by,

$$r = \sum_{k=1}^M P(\omega_k) r_k = \sum_{i=1}^M \int_{\mathcal{R}_i} \left(\sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) \right) d\mathbf{x}.$$

The Bayesian Classifier Minimizes The Misclassification Error

- Since in classification the dependent variable (label), y , is of a discrete nature, the classification error probability may sound to be the most natural cost function to be optimized. However, this is not always true. For example, in a medical diagnosis system, committing an error by predicting the class of a finding in an x-ray image as being “malignant”, while its true class is “normal”, is less significant than an error in the other way round.
- For such cases, **relative weights on the errors** according to their **importance** have to be used. The resulting function is known as the **average risk**.
- For the M -class problem, the **risk** or **loss** associated with class ω_k is defined as,

$$r_k = \sum_{i=1}^M \lambda_{ki} \int_{\mathcal{R}_i} p(\mathbf{x}|\omega_k) d\mathbf{x},$$

where, $\lambda_{kk} = 0$ and λ_{ki} is the weight that controls the significance of committing an error by assigning a pattern from class ω_k to class ω_i .

- The **average risk** is given by,

$$r = \sum_{k=1}^M P(\omega_k) r_k = \sum_{i=1}^M \int_{\mathcal{R}_i} \left(\sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) \right) d\mathbf{x}.$$

- The average risk is minimized if we partition the input space by selecting each \mathcal{R}_i (where we decide in favor of class ω_i) so that each one of the M integrals in the last summation becomes minimum; this is achieved if we adopt the rule:

$$\text{Assign } \mathbf{x} \text{ to } \omega_i : \sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) < \sum_{k=1}^M \lambda_{kj} P(\omega_k) p(\mathbf{x}|\omega_k), \quad \forall j \neq i,$$

- This is equivalent to

$$\text{Assign } \mathbf{x} \text{ to } \omega_i : \sum_{k=1}^M \lambda_{ki} P(\omega_k|\mathbf{x}) < \sum_{k=1}^M \lambda_{kj} P(\omega_k|\mathbf{x}), \quad \forall j \neq i.$$

- For the two-class case, it is readily seen that the rule becomes

$$\text{Assign } \mathbf{x} \text{ to } \omega_1 \text{ (}\omega_2\text{) if : } \lambda_{12} P(\omega_1|\mathbf{x}) > (<) \lambda_{21} P(\omega_2|\mathbf{x}),$$

or equivalently

$$\text{Assign } \mathbf{x} \text{ to } \omega_1 \text{ (}\omega_2\text{) if : } \underbrace{\lambda_{12} P(\omega_1)}_{P'(\omega_1)} p(\mathbf{x}|\omega_1) > (<) \underbrace{\lambda_{21} P(\omega_2)}_{P'(\omega_2)} p(\mathbf{x}|\omega_2).$$

Example Of Bayesian Classification

- In a two-class one-dimensional classification task, the data in the two classes are distributed according to the following two Gaussians,

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right),$$

and

$$p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-1)^2}{2}\right).$$

The problem is more sensitive with respect to errors committed on patterns from class ω_1 , which is expressed via the following **loss matrix**, which is the matrix comprising the respective weights,

$$L := \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0.5 & 0 \end{bmatrix}.$$

In other words, $\lambda_{12} = 1$ and $\lambda_{21} = 0.5$. The two classes are considered equiprobable.

- The goal is to derive the threshold values, x_r , and x_B , for the average risk and the Bayesian classification respectively.

Example Of Bayesian Classification

- In a two-class one-dimensional classification task, the data in the two classes are distributed according to the following two Gaussians,

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right),$$

and

$$p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-1)^2}{2}\right).$$

The problem is more sensitive with respect to errors committed on patterns from class ω_1 , which is expressed via the following **loss matrix**, which is the matrix comprising the respective weights,

$$L := \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0.5 & 0 \end{bmatrix}.$$

In other words, $\lambda_{12} = 1$ and $\lambda_{21} = 0.5$. The two classes are considered equiprobable.

- The goal is to derive the threshold values, x_r , and x_B , for the average risk and the Bayesian classification respectively.

Example Of Bayesian Classification

- Solution: According to the average risk rule, the region for which we decide in favor of class ω_1 is given by:

$$\mathcal{R}_1 : \lambda_{12} \frac{1}{2} p(x|\omega_1) > \lambda_{21} \frac{1}{2} p(x|\omega_2),$$

and the respective threshold value, x_r , is computed by the equation,

$$\exp\left(-\frac{x_r^2}{2}\right) = 0.5 \exp\left(-\frac{(x_r - 1)^2}{2}\right),$$

which after taking the logarithm and solving the respective equation, trivially results in

$$x_r = \frac{1}{2}(1 - 2 \ln 0.5).$$

- The threshold for the Bayesian classifier results if we set $\lambda_{21} = 1$, which gives

$$x_B = \frac{1}{2}.$$

Example Of Bayesian Classification

- Solution: According to the average risk rule, the region for which we decide in favor of class ω_1 is given by:

$$\mathcal{R}_1 : \lambda_{12} \frac{1}{2} p(x|\omega_1) > \lambda_{21} \frac{1}{2} p(x|\omega_2),$$

and the respective threshold value, x_r , is computed by the equation,

$$\exp\left(-\frac{x_r^2}{2}\right) = 0.5 \exp\left(-\frac{(x_r - 1)^2}{2}\right),$$

which after taking the logarithm and solving the respective equation, trivially results in

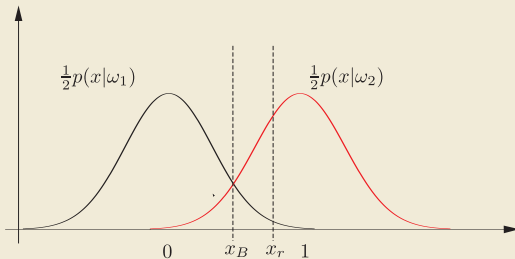
$$x_r = \frac{1}{2}(1 - 2 \ln 0.5).$$

- The threshold for the Bayesian classifier results if we set $\lambda_{21} = 1$, which gives

$$x_B = \frac{1}{2}.$$

Example Of Bayesian Classification

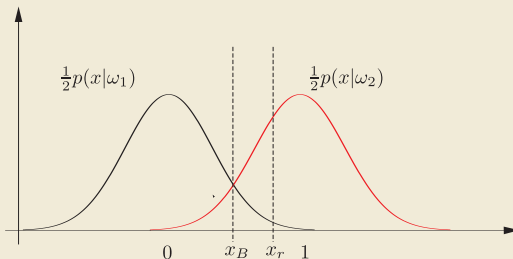
- The previous findings are illustrated in the figure below:



- Note that minimizing the average risk enlarges the region in which we decide in favor of the most sensitive class, ω_1 .

Example Of Bayesian Classification

- The previous findings are illustrated in the figure below:



- Note that minimizing the average risk **enlarges** the region in which we decide in favor of the **most sensitive class**, ω_1 .

- The goal of any classifier is to **partition the feature space into regions**. The partition is achieved via points in \mathbb{R} , curves in \mathbb{R}^2 , surfaces in \mathbb{R}^3 and hypersurfaces in \mathbb{R}^l . Any hypersurface, S , is expressed in terms of a function,

$$g : \mathbb{R}^l \mapsto \mathbb{R},$$

and it comprises all the points such that

$$S = \{\mathbf{x} \in \mathbb{R}^l : g(\mathbf{x}) = 0\}.$$

- Recall that all points lying on one side of this hypersurface score $g(\mathbf{x}) > 0$ and all the points on the other side score $g(\mathbf{x}) < 0$. The resulting (hyper)surfaces are known as **decision (hyper)surfaces**.
- For example, the respective decision hypersurface, which is formed by the Bayesian classifier for a two class classification task, is given by

$$g(\mathbf{x}) := P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0.$$

- Indeed, we decide in favor of class ω_1 if \mathbf{x} falls on the **positive side of the previous hypersurface**, and in favor of ω_2 for the points falling on the **negative side** (regions \mathcal{R}_1 and \mathcal{R}_2 in the next figure).

- The goal of any classifier is to **partition the feature space into regions**. The partition is achieved via points in \mathbb{R} , curves in \mathbb{R}^2 , surfaces in \mathbb{R}^3 and hypersurfaces in \mathbb{R}^l . Any hypersurface, S , is expressed in terms of a function,

$$g : \mathbb{R}^l \mapsto \mathbb{R},$$

and it comprises all the points such that

$$S = \{\mathbf{x} \in \mathbb{R}^l : g(\mathbf{x}) = 0\}.$$

- Recall that all points lying on one side of this hypersurface score $g(\mathbf{x}) > 0$ and all the points on the other side score $g(\mathbf{x}) < 0$. The resulting (hyper)surfaces are known as **decision (hyper)surfaces**.
- For example, the respective decision hypersurface, which is formed by the Bayesian classifier for a two class classification task, is given by

$$g(\mathbf{x}) := P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0.$$

- Indeed, we decide in favor of class ω_1 if \mathbf{x} falls on the **positive side of the previous hypersurface**, and in favor of ω_2 for the points falling on the **negative side** (regions \mathcal{R}_1 and \mathcal{R}_2 in the next figure).

- The goal of any classifier is to **partition the feature space into regions**. The partition is achieved via points in \mathbb{R} , curves in \mathbb{R}^2 , surfaces in \mathbb{R}^3 and hypersurfaces in \mathbb{R}^l . Any hypersurface, S , is expressed in terms of a function,

$$g : \mathbb{R}^l \mapsto \mathbb{R},$$

and it comprises all the points such that

$$S = \{\mathbf{x} \in \mathbb{R}^l : g(\mathbf{x}) = 0\}.$$

- Recall that all points lying on one side of this hypersurface score $g(\mathbf{x}) > 0$ and all the points on the other side score $g(\mathbf{x}) < 0$. The resulting (hyper)surfaces are known as **decision (hyper)surfaces**.
- For example, the respective decision hypersurface, which is formed by the Bayesian classifier for a two class classification task, is given by

$$g(\mathbf{x}) := P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0.$$

- Indeed, we decide in favor of class ω_1 if \mathbf{x} falls on the **positive side of the previous hypersurface**, and in favor of ω_2 for the points falling on the **negative side** (regions \mathcal{R}_1 and \mathcal{R}_2 in the next figure).

- The goal of any classifier is to **partition the feature space into regions**. The partition is achieved via points in \mathbb{R} , curves in \mathbb{R}^2 , surfaces in \mathbb{R}^3 and hypersurfaces in \mathbb{R}^l . Any hypersurface, S , is expressed in terms of a function,

$$g : \mathbb{R}^l \mapsto \mathbb{R},$$

and it comprises all the points such that

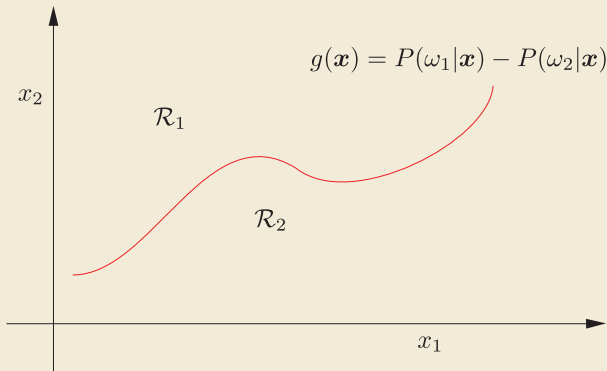
$$S = \{\mathbf{x} \in \mathbb{R}^l : g(\mathbf{x}) = 0\}.$$

- Recall that all points lying on one side of this hypersurface score $g(\mathbf{x}) > 0$ and all the points on the other side score $g(\mathbf{x}) < 0$. The resulting (hyper)surfaces are known as **decision (hyper)surfaces**.
- For example, the respective decision hypersurface, which is formed by the Bayesian classifier for a two class classification task, is given by

$$g(\mathbf{x}) := P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0.$$

- Indeed, we decide in favor of class ω_1 if \mathbf{x} falls on the **positive side of the previous hypersurface**, and in favor of ω_2 for the points falling on the **negative side** (regions \mathcal{R}_1 and \mathcal{R}_2 in the next figure).

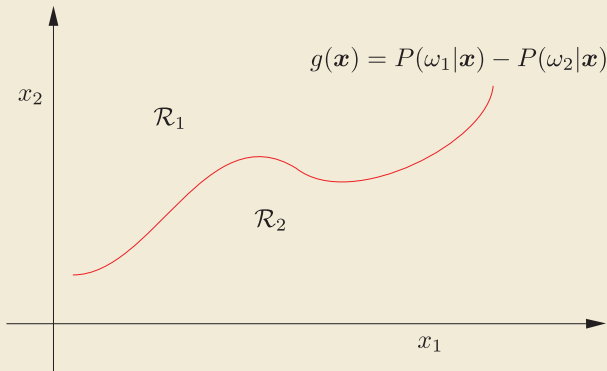
Decision Hypersurfaces



The Bayesian classifier implicitly forms hypersurfaces defined by $g(\mathbf{x}) = P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0$.

- Once we move away from the Bayesian concept of designing classifiers, different families of functions for selecting $g(\mathbf{x})$ can be adopted and the specific form will be obtained via **different optimization criteria**, which are not necessarily related to the probability of error/average risk.

Decision Hypersurfaces



The Bayesian classifier implicitly forms hypersurfaces defined by $g(\mathbf{x}) = P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0$.

- Once we move away from the Bayesian concept of designing classifiers, different families of functions for selecting $g(\mathbf{x})$ can be adopted and the specific form will be obtained via **different optimization criteria**, which are not necessarily related to the probability of error/average risk.

The Gaussian Distribution Case

- Plugging in the specific forms of the Gaussian conditionals, using logarithms and after a bit of trivial algebra, the decision hypersurface becomes,

$$\begin{aligned} g(\mathbf{x}) = & \underbrace{\frac{1}{2} (\mathbf{x}^T \Sigma_2^{-1} \mathbf{x} - \mathbf{x}^T \Sigma_1^{-1} \mathbf{x})}_{\text{quadratic terms}} \\ & + \underbrace{\mu_1^T \Sigma_1^{-1} \mathbf{x} - \mu_2^T \Sigma_2^{-1} \mathbf{x}}_{\text{linear terms}} \\ & - \underbrace{\frac{1}{2} \mu_1^T \Sigma_1^{-1} \mu_1 + \frac{1}{2} \mu_2^T \Sigma_2^{-1} \mu_2 + \ln \frac{P(\omega_1)}{P(\omega_2)} + \frac{1}{2} \ln \frac{|\Sigma_2|}{|\Sigma_1|}}_{\text{constant terms}} = 0. \end{aligned}$$

This is of a **quadratic nature**, hence the corresponding (hyper)surfaces are **(hyper)quadratics**, e.g., (hyper)ellipsoids, (hyper)parabolas, hyperbolas.

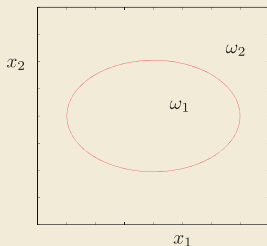
The Gaussian Distribution Case

- The following figure shows two examples, in the two-dimensional space, corresponding to $P(\omega_1) = P(\omega_2)$, and

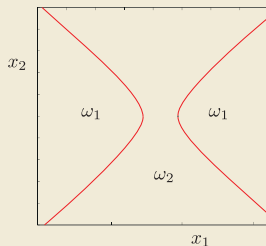
$$\boldsymbol{\mu}_1 = [0, 0]^T, \boldsymbol{\mu}_2 = [4, 0]^T, \boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.3 & 0.0 \\ 0.0 & 0.35 \end{bmatrix}, \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1.2 & 0.0 \\ 0.0 & 1.85 \end{bmatrix},$$

and

$$\boldsymbol{\mu}_1 = [0, 0]^T, \boldsymbol{\mu}_2 = [3.2, 0]^T, \boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.75 \end{bmatrix}, \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.75 & 0.0 \\ 0.0 & 0.1 \end{bmatrix},$$



(a)



(b)

When Covariance Matrices Are The Same In All Classes The Bayesian Classifier Becomes Linear

- Looking carefully at the decision hypersurface derived for the Bayesian classifier given before, it is readily noticed that once the **covariance matrices for the two classes become equal**, then the **quadratic terms cancel out** and the discriminant function becomes **linear**; thus, the corresponding hypersurface is a **hyperplane**. After some straightforward algebraic manipulations the decision hyperplane becomes:

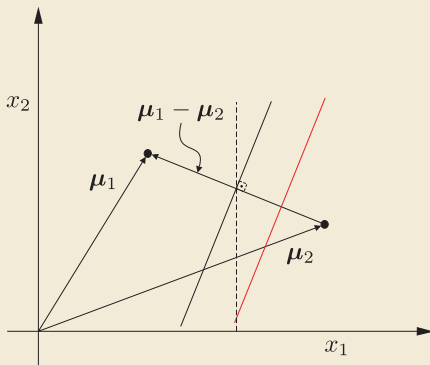
$$\begin{aligned}g(\mathbf{x}) &= \boldsymbol{\theta}^T(\mathbf{x} - \mathbf{x}_0) = 0, \text{ where} \\ \boldsymbol{\theta} &:= \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \\ \mathbf{x}_0 &:= \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) - \ln \frac{P(\omega_1)}{P(\omega_2)} \frac{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2}{\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_{\Sigma^{-1}}},\end{aligned}$$

where Σ is the common to the two classes covariance matrix and

$$\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_{\Sigma^{-1}} := \sqrt{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)},$$

is the so-called **Σ^{-1} -norm** of the vector $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$.

When Covariance Matrices Are The Same In All Classes The Bayesian Classifier Becomes Linear



The full gray line corresponds to the Bayesian classifier for two equiprobable Gaussian classes, which share a common covariance matrix of the specific form, $\Sigma = \sigma^2 I$; the line bisects the segment joining the two mean values (minimum Euclidean distance classifier). The red one is for the same case but for $P(\omega_1) > P(\omega_2)$. The dotted line is the optimal classifier for equiprobable classes, and a common covariance of a more general form, different than $\sigma^2 I$ (minimum Mahalanobis distance classifier).

- **Minimum Euclidean Distance Classifier:** Under the assumptions of a) Gaussian distributed data in each one of the classes, b) equiprobable classes, c) common covariance matrix in all classes of the special form $\Sigma = \sigma^2 I$ (individual features are independent and sharing a common variance), the Bayesian classification rule is equivalent with

Assign \mathbf{x} to class ω_i : $i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T (\mathbf{x} - \boldsymbol{\mu}_j), j = 1, 2, \dots M.$

In other words, the Euclidean distance of \mathbf{x} is computed from the mean values of all classes and it is assigned to the class for which this distance becomes smaller (full black line in the previous figure).

- **Minimum Mahalanobis Distance Classifier:** Under the previously adopted assumptions, but with the covariance matrix being of the more general form, $\Sigma \neq \sigma^2 I$, the rule becomes,

Assign \mathbf{x} to class ω_i : $i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_j), j = 1, 2, \dots M.$

Thus, instead of looking for the minimum Euclidean distance, one searches for the minimum Mahalanobis distance. For the two-class case, this rule corresponds to the dotted line of the previous figure.

- **Minimum Euclidean Distance Classifier:** Under the assumptions of a) Gaussian distributed data in each one of the classes, b) equiprobable classes, c) common covariance matrix in all classes of the special form $\Sigma = \sigma^2 I$ (individual features are independent and sharing a common variance), the Bayesian classification rule is equivalent with

Assign \mathbf{x} to class ω_i : $i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T (\mathbf{x} - \boldsymbol{\mu}_j), j = 1, 2, \dots M.$

In other words, the Euclidean distance of \mathbf{x} is computed from the mean values of all classes and it is assigned to the class for which this distance becomes smaller (full black line in the previous figure).

- **Minimum Mahalanobis Distance Classifier:** Under the previously adopted assumptions, but with the covariance matrix being of the more general form, $\Sigma \neq \sigma^2 I$, the rule becomes,

Assign \mathbf{x} to class ω_i : $i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_j), j = 1, 2, \dots M.$

Thus, instead of looking for the minimum Euclidean distance, one searches for the minimum Mahalanobis distance. For the two-class case, this rule corresponds to the dotted line of the previous figure.

- **Minimum Euclidean Distance Classifier:** Under the assumptions of a) Gaussian distributed data in each one of the classes, b) equiprobable classes, c) common covariance matrix in all classes of the special form $\Sigma = \sigma^2 I$ (individual features are independent and sharing a common variance), the Bayesian classification rule is equivalent with

Assign \mathbf{x} to class ω_i : $i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T (\mathbf{x} - \boldsymbol{\mu}_j), j = 1, 2, \dots M.$

In other words, the Euclidean distance of \mathbf{x} is computed from the mean values of all classes and it is assigned to the class for which this distance becomes smaller (full black line in the previous figure).

- **Minimum Mahalanobis Distance Classifier:** Under the previously adopted assumptions, but with the covariance matrix being of the more general form, $\Sigma \neq \sigma^2 I$, the rule becomes,

Assign \mathbf{x} to class ω_i : $i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_j), j = 1, 2, \dots M.$

Thus, instead of looking for the minimum Euclidean distance, one searches for the minimum **Mahalanobis distance**. For the two-class case, this rule corresponds to the dotted line of the previous figure.

Minimum Distance Classifier Example

- In a two-class classification task, the data in each one of the classes are distributed according to the Gaussian distribution, with mean values, $\mu_1 = [0, 0]^T$ and $\mu_2 = [3, 3]^T$, respectively, sharing a common covariance matrix,

$$\Sigma = \begin{bmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{bmatrix}.$$

Use the Bayesian classifier to classify the point $x = [1.0, 2.2]^T$ into one of the two classes.

- The classes are distributed according to the Gaussian distribution and share the same covariance matrix. Thus, the Bayesian classifier is equivalent with the **minimum Mahalanobis distance classifier**. The (square) Mahalanobis distance of the point x from the mean value of class ω_1 is,

$$d_1^2 = [1.0, 2.2] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.2 \end{bmatrix} = 2.95,$$

where the matrix in the middle of the left hand side is the inverse of the covariance matrix.

Minimum Distance Classifier Example

- In a two-class classification task, the data in each one of the classes are distributed according to the Gaussian distribution, with mean values, $\mu_1 = [0, 0]^T$ and $\mu_2 = [3, 3]^T$, respectively, sharing a common covariance matrix,

$$\Sigma = \begin{bmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{bmatrix}.$$

Use the Bayesian classifier to classify the point $x = [1.0, 2.2]^T$ into one of the two classes.

- The classes are distributed according to the Gaussian distribution and share the same covariance matrix. Thus, the Bayesian classifier is equivalent with the **minimum Mahalanobis distance classifier**. The (square) Mahalanobis distance of the point x from the mean value of class ω_1 is,

$$d_1^2 = [1.0, 2.2] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.2 \end{bmatrix} = 2.95,$$

where the matrix in the middle of the left hand side is the inverse of the covariance matrix.

Minimum Distance Classifier Example

- Similarly for class ω_2 , we obtain that

$$d_2^2 = [-2.0, -0.8] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} -2.0 \\ -0.8 \end{bmatrix} = 3.67.$$

- Hence, the pattern is assigned to class ω_1 , since its distance from μ_1 is smaller compared to that from μ_2 .
- Verify that if the Euclidean distance were used instead, the pattern would be assigned to class ω_1 .

- Assuming Gaussian classes, in order to estimate the required covariance matrix, the number of unknown parameters is of the order of $\mathcal{O}(l^2/2)$. Indeed, a possible estimate for the covariance matrix is the ML one,

$$\hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T,$$

which can be shown to correspond to a **biased** estimator. An estimate associated to an **unbiased** estimator is given by

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T.$$

where $\hat{\boldsymbol{\mu}}_{ML}$ is the sample mean.

- For high dimensional spaces, besides that this estimation task is a formidable one, it also requires a **large number of data points**, in order to obtain **statistically good estimates and avoid overfitting**, as it has already been discussed in Chapter 3. In such cases, one has to be content with **suboptimal solutions**. Indeed, adopting an optimal method, while using bad estimates of the involved parameters, it can necessarily lead to a bad overall performance.

- Assuming Gaussian classes, in order to estimate the required covariance matrix, the number of unknown parameters is of the order of $\mathcal{O}(l^2/2)$. Indeed, a possible estimate for the covariance matrix is the ML one,

$$\hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T,$$

which can be shown to correspond to a **biased** estimator. An estimate associated to an **unbiased** estimator is given by

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T.$$

where $\hat{\boldsymbol{\mu}}_{ML}$ is the sample mean.

- For high dimensional spaces, besides that this estimation task is a formidable one, it also requires a **large number of data points**, in order to obtain **statistically good estimates and avoid overfitting**, as it has already been discussed in Chapter 3. In such cases, one has to be content with **suboptimal solutions**. Indeed, adopting an optimal method, while using bad estimates of the involved parameters, it can necessarily lead to a bad overall performance.

The Naive Bayes Classifier

- The **Naive Bayes Classifier** is a typical and popular example of a **suboptimal classifier**. The basic assumption is that the components (features) in the feature vector are **statistically independent**; hence, the joint pdf can be written as a product of l marginals, i.e.,

$$p(\mathbf{x}|\omega_i) = \prod_{k=1}^l p(x_k|\omega_i), \quad i = 1, 2, \dots, M.$$

- Having adopted the Gaussian assumption, each one of the marginals is described by **two parameters, i.e., the mean and the variance**; this leads to a total of $2l$, per class, unknown parameters to be estimated. This is a substantial saving compared to the $O(l^2/2)$ number of parameters.
- We already discussed the curse of dimensionality issue and it was stressed out that high dimensional spaces are **sparsely populated**. Roughly speaking, if, say, N data points are needed in order get a good enough estimate of a pdf in the real axis, N^l data points would be needed for a similar accuracy in an l -dimensional space. Thus, by assuming the features to be mutually independent, one will end up in estimating l one-dimensional pdfs, hence substantially reducing the need for data.

- The **Naive Bayes Classifier** is a typical and popular example of a **suboptimal classifier**. The basic assumption is that the components (features) in the feature vector are **statistically independent**; hence, the joint pdf can be written as a product of l marginals, i.e.,

$$p(\mathbf{x}|\omega_i) = \prod_{k=1}^l p(x_k|\omega_i), \quad i = 1, 2, \dots, M.$$

- Having adopted the Gaussian assumption, each one of the marginals is described by **two parameters, i.e., the mean and the variance**; this leads to a total of $2l$, per class, unknown parameters to be estimated. This is a substantial saving compared to the $O(l^2/2)$ number of parameters.
- We already discussed the curse of dimensionality issue and it was stressed out that high dimensional spaces are **sparsely populated**. Roughly speaking, if, say, N data points are needed in order get a good enough estimate of a pdf in the real axis, N^l data points would be needed for a similar accuracy in an l -dimensional space. Thus, by assuming the features to be mutually independent, one will end up in estimating l one-dimensional pdfs, hence substantially reducing the need for data.

- The **Naive Bayes Classifier** is a typical and popular example of a **suboptimal classifier**. The basic assumption is that the components (features) in the feature vector are **statistically independent**; hence, the joint pdf can be written as a product of l marginals, i.e.,

$$p(\mathbf{x}|\omega_i) = \prod_{k=1}^l p(x_k|\omega_i), \quad i = 1, 2, \dots, M.$$

- Having adopted the Gaussian assumption, each one of the marginals is described by **two parameters, i.e., the mean and the variance**; this leads to a total of $2l$, per class, unknown parameters to be estimated. This is a substantial saving compared to the $O(l^2/2)$ number of parameters.
- We already discussed the curse of dimensionality issue and it was stressed out that high dimensional spaces are **sparsely populated**. Roughly speaking, if, say, N data points are needed in order get a good enough estimate of a pdf in the real axis, N^l data points would be needed for a similar accuracy in an l -dimensional space. Thus, by assuming the features to be mutually independent, one will end up in estimating l one-dimensional pdfs, hence substantially reducing the need for data.

The k -Nearest Neighbor Rule

- Although the Bayesian rule provides the optimal solution, with respect to the classification error probability, its application requires the estimation of the respective conditional pdfs; this is not an easy task, once the dimensionality of the feature space assumes relatively large values. This paves the way for considering alternative classification rules, which becomes our focus from now on.
- The k -nearest neighbor (k -NN) rule is a typical non-parametric classifier and it is one among the most popular and well known classifiers. In spite of its simplicity, it is still in use and stands next to more elaborate schemes.
- Consider N training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, for an M -class classification task. At the heart of the method lies a parameter k , which is a user-defined parameter. Once k is selected, then given a pattern, \mathbf{x} , assign it to the class in which the majority of its k nearest (according to a metric, e.g., Euclidean or Mahalanobis distance) neighbors, among the training points, belong. The parameter k should not be a multiple of M , in order to avoid ties. The simplest form of this rule is to assign the pattern to the class in which its nearest neighbor belongs, i.e., $k = 1$.

The k -Nearest Neighbor Rule

- Although the Bayesian rule provides the optimal solution, with respect to the classification error probability, its application requires the estimation of the respective conditional pdfs; this is not an easy task, once the dimensionality of the feature space assumes relatively large values. This paves the way for considering alternative classification rules, which becomes our focus from now on.
- The k -nearest neighbor (k -NN) rule is a typical non-parametric classifier and it is one among the most popular and well known classifiers. In spite of its simplicity, it is still in use and stands next to more elaborate schemes.
- Consider N training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, for an M -class classification task. At the heart of the method lies a parameter k , which is a user-defined parameter. Once k is selected, then given a pattern, \mathbf{x} , assign it to the class in which the majority of its k nearest (according to a metric, e.g., Euclidean or Mahalanobis distance) neighbors, among the training points, belong. The parameter k should not be a multiple of M , in order to avoid ties. The simplest form of this rule is to assign the pattern to the class in which its nearest neighbor belongs, i.e., $k = 1$.

The k -Nearest Neighbor Rule

- Although the Bayesian rule provides the optimal solution, with respect to the classification error probability, its application requires the estimation of the respective conditional pdfs; this is not an easy task, once the dimensionality of the feature space assumes relatively large values. This paves the way for considering alternative classification rules, which becomes our focus from now on.
- The k -nearest neighbor (k -NN) rule is a typical non-parametric classifier and it is one among the most popular and well known classifiers. In spite of its simplicity, it is still in use and stands next to more elaborate schemes.
- Consider N training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, for an M -class classification task. At the heart of the method lies a parameter k , which is a user-defined parameter. Once k is selected, then given a pattern, \mathbf{x} , assign it to the class in which the majority of its k nearest (according to a metric, e.g., Euclidean or Mahalanobis distance) neighbors, among the training points, belong. The parameter k should not be a multiple of M , in order to avoid ties. The simplest form of this rule is to assign the pattern to the class in which its nearest neighbor belongs, i.e., $k = 1$.

The k -Nearest Neighbor Rule

- It turns out that this conceptually simple rule, tends to the Bayesian classifier, if a) $N \rightarrow \infty$, b) $k \rightarrow \infty$ and c) $k/N \rightarrow 0$. More specifically, it can be shown that the classification errors, P_{NN} and P_{kNN} satisfy, **asymptotically**, the following bounds,

$$P_B \leq P_{NN} \leq 2P_B, \text{ for } k = 1,$$

$$P_B \leq P_{kNN} \leq P_B + \sqrt{\frac{2P_{NN}}{k}}, \text{ for } k \neq 1.$$

P_B is the error corresponding to the optimal Bayesian classifier.

- For $k = 1$, the bound says that the simple NN rule will never give an error larger than **twice the optimal one**. If, for example, $P_B = 0.01$, then $P_{NN} \leq 0.02$. That is, if one has an easy task (as this is indicated by the very low value of P_B) the NN rule can also do a good job. This, of course, is not the case for harder tasks.
- The bound for $k \neq 1$ says that for **large values of k** (provided, of course, N is large enough) the performance of the k -NN **tends to that of the optimal classifier**. In practice, one has to make sure that k does not get values close to N , but it remains a fraction of it.

The k -Nearest Neighbor Rule

- It turns out that this conceptually simple rule, tends to the Bayesian classifier, if a) $N \rightarrow \infty$, b) $k \rightarrow \infty$ and c) $k/N \rightarrow 0$. More specifically, it can be shown that the classification errors, P_{NN} and P_{kNN} satisfy, **asymptotically**, the following bounds,

$$P_B \leq P_{NN} \leq 2P_B, \text{ for } k = 1,$$

$$P_B \leq P_{kNN} \leq P_B + \sqrt{\frac{2P_{NN}}{k}}, \text{ for } k \neq 1.$$

P_B is the error corresponding to the optimal Bayesian classifier.

- For $k = 1$, the bound says that the simple NN rule will never give an error larger than **twice the optimal one**. If, for example, $P_B = 0.01$, then $P_{NN} \leq 0.02$. That is, if one has an easy task (as this is indicated by the very low value of P_B) the NN rule can also do a good job. This, of course, is not the case for harder tasks.
- The bound for $k \neq 1$ says that for **large values of k** (provided, of course, N is large enough) the performance of the k -NN **tends to that of the optimal classifier**. In practice, one has to make sure that k does not get values close to N , but it remains a fraction of it.

The k -Nearest Neighbor Rule

- It turns out that this conceptually simple rule, tends to the Bayesian classifier, if a) $N \rightarrow \infty$, b) $k \rightarrow \infty$ and c) $k/N \rightarrow 0$. More specifically, it can be shown that the classification errors, P_{NN} and P_{kNN} satisfy, **asymptotically**, the following bounds,

$$P_B \leq P_{NN} \leq 2P_B, \text{ for } k = 1,$$

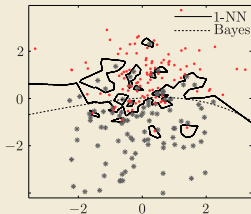
$$P_B \leq P_{kNN} \leq P_B + \sqrt{\frac{2P_{NN}}{k}}, \text{ for } k \neq 1.$$

P_B is the error corresponding to the optimal Bayesian classifier.

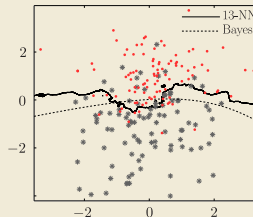
- For $k = 1$, the bound says that the simple NN rule will never give an error larger than **twice the optimal one**. If, for example, $P_B = 0.01$, then $P_{NN} \leq 0.02$. That is, if one has an easy task (as this is indicated by the very low value of P_B) the NN rule can also do a good job. This, of course, is not the case for harder tasks.
- The bound for $k \neq 1$ says that for **large values of k** (provided, of course, N is large enough) the performance of the k -NN **tends to that of the optimal classifier**. In practice, one has to make sure that k does not get values close to N , but it remains a fraction of it.

Nearest Neighbor Classifier Example

- The following figures illustrate the decision curves for a two-class classification task in the two-dimensional space, obtained by the Bayesian, the 1-NN and the 13-NN classifier. A number of $N = 100$ data are generated, for each class, by Gaussian distributions. The decision curve of the Bayes classifier has the form of a parabola, while the 1-NN classifier exhibits a highly nonlinear nature. The 13-NN rule forms a decision line close to the Bayesian one.



The 1-NN Rule with the Bayesian classifier



The 13-NN Rule with the Bayesian classifier

Logistic Regression

- In Bayesian classification, the posteriors are estimated via the respective conditional pdfs, which is not, in general, an easy task. The goal now becomes to model the posterior probabilities **directly**, via the so-called **logistic regression** method. This name has been established in the statistics community, although the model refers to classification and not to regression. This is a typical example of the **discriminative modeling** technique, where the distribution of data is of no interest.
- We will focus on the two-class case. The starting point is to model the ratio of the posteriors as:

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = \theta^T \mathbf{x},$$

where the constant term, θ_0 , has been absorbed in θ .

- Taking into account that $P(\omega_1|\mathbf{x}) + P(\omega_2|\mathbf{x}) = 1$ and defining

$$t := \theta^T \mathbf{x},$$

it is readily seen that the previous model is equivalent to

$$P(\omega_1|\mathbf{x}) = \sigma(t) := \frac{1}{1 + \exp(-t)}, \quad P(\omega_2|\mathbf{x}) = 1 - \sigma(t) = \frac{\exp(-t)}{1 + \exp(-t)}.$$

Logistic Regression

- In Bayesian classification, the posteriors are estimated via the respective conditional pdfs, which is not, in general, an easy task. The goal now becomes to model the posterior probabilities **directly**, via the so-called **logistic regression** method. This name has been established in the statistics community, although the model refers to classification and not to regression. This is a typical example of the **discriminative modeling** technique, where the distribution of data is of no interest.
- We will focus on the two-class case. The starting point is to model the ratio of the posteriors as:

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = \boldsymbol{\theta}^T \mathbf{x},$$

where the constant term, θ_0 , has been absorbed in $\boldsymbol{\theta}$.

- Taking into account that $P(\omega_1|\mathbf{x}) + P(\omega_2|\mathbf{x}) = 1$ and defining

$$t := \boldsymbol{\theta}^T \mathbf{x},$$

it is readily seen that the previous model is equivalent to

$$P(\omega_1|\mathbf{x}) = \sigma(t) := \frac{1}{1 + \exp(-t)}, \quad P(\omega_2|\mathbf{x}) = 1 - \sigma(t) = \frac{\exp(-t)}{1 + \exp(-t)}.$$

- In Bayesian classification, the posteriors are estimated via the respective conditional pdfs, which is not, in general, an easy task. The goal now becomes to model the posterior probabilities **directly**, via the so-called **logistic regression** method. This name has been established in the statistics community, although the model refers to classification and not to regression. This is a typical example of the **discriminative modeling** technique, where the distribution of data is of no interest.
- We will focus on the two-class case. The starting point is to model the ratio of the posteriors as:

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = \boldsymbol{\theta}^T \mathbf{x},$$

where the constant term, θ_0 , has been absorbed in $\boldsymbol{\theta}$.

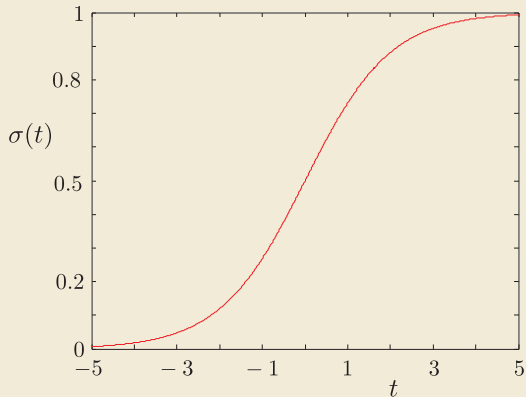
- Taking into account that $P(\omega_1|\mathbf{x}) + P(\omega_2|\mathbf{x}) = 1$ and defining

$$t := \boldsymbol{\theta}^T \mathbf{x},$$

it is readily seen that the previous model is equivalent to

$$P(\omega_1|\mathbf{x}) = \sigma(t) := \frac{1}{1 + \exp(-t)}, \quad P(\omega_2|\mathbf{x}) = 1 - \sigma(t) = \frac{\exp(-t)}{1 + \exp(-t)}.$$

- The function $\sigma(t)$ is known as the **logistic sigmoid** or **sigmoid link** function and it is shown in the figure below:



Training The Logistic Regression Model

- The parameter vector, θ , is estimated via the **Maximum Likelihood method** applied on a set of training samples, (y_n, \mathbf{x}_n) , $n = 1, \dots, N$, $y_n \in \{0, 1\}$. The likelihood function can be written as,

$$P(y_1, \dots, y_N; \theta) = \prod_{n=1}^N \left(\sigma(\theta^T \mathbf{x}_n) \right)^{y_n} \left(1 - \sigma(\theta^T \mathbf{x}_n) \right)^{1-y_n}.$$

- Usually, we consider the **negative** log-likelihood given by,

$$L(\theta) = - \sum_{n=1}^N \left(y_n \ln s_n + (1 - y_n) \ln(1 - s_n) \right), \quad s_n := \sigma(\theta^T \mathbf{x}_n).$$

The log-likelihood cost function is also known as the **cross-entropy** error.

Training The Logistic Regression Model

- The parameter vector, θ , is estimated via the **Maximum Likelihood method** applied on a set of training samples, (y_n, \mathbf{x}_n) , $n = 1, \dots, N$, $y_n \in \{0, 1\}$. The likelihood function can be written as,

$$P(y_1, \dots, y_N; \theta) = \prod_{n=1}^N \left(\sigma(\theta^T \mathbf{x}_n) \right)^{y_n} \left(1 - \sigma(\theta^T \mathbf{x}_n) \right)^{1-y_n}.$$

- Usually, we consider the **negative** log-likelihood given by,

$$L(\theta) = - \sum_{n=1}^N \left(y_n \ln s_n + (1 - y_n) \ln(1 - s_n) \right), \quad s_n := \sigma(\theta^T \mathbf{x}_n).$$

The log-likelihood cost function is also known as the **cross-entropy** error.

- **Minimizing the negative log-likelihood:** Minimization of $L(\theta)$ with respect to θ is carried out iteratively by any iterative minimization scheme, such as the gradient descent. $L(\theta)$ is convex which guarantees a **unique minimum**. The scheme needs the computation of the respective gradient, which in turn is based on the derivative of the sigmoid link function, which can be shown to be equal to

$$\frac{d\sigma(t)}{dt} = \sigma(t) (1 - \sigma(t)).$$

- The gradient is shown to be equal to

$$\nabla L(\theta) = \sum_{n=1}^N (s_n - y_n) \mathbf{x}_n = X^T (\mathbf{s} - \mathbf{y}),$$

where

$$X^T = [\mathbf{x}_1, \dots, \mathbf{x}_N], \quad \mathbf{s} := [s_1, \dots, s_N]^T, \quad \mathbf{y} = [y_1, \dots, y_N]^T.$$

- Hence, the **gradient descent** scheme for minimization becomes

$$\theta^{(i)} = \theta^{(i-1)} - \mu_i X^T (\mathbf{s}^{(i-1)} - \mathbf{y}).$$

- **Minimizing the negative log-likelihood:** Minimization of $L(\theta)$ with respect to θ is carried out iteratively by any iterative minimization scheme, such as the gradient descent. $L(\theta)$ is convex which guarantees a **unique minimum**. The scheme needs the computation of the respective gradient, which in turn is based on the derivative of the sigmoid link function, which can be shown to be equal to

$$\frac{d\sigma(t)}{dt} = \sigma(t) (1 - \sigma(t)).$$

- The gradient is shown to be equal to

$$\nabla L(\theta) = \sum_{n=1}^N (s_n - y_n) \mathbf{x}_n = X^T (\mathbf{s} - \mathbf{y}),$$

where

$$X^T = [\mathbf{x}_1, \dots, \mathbf{x}_N], \quad \mathbf{s} := [s_1, \dots, s_N]^T, \quad \mathbf{y} = [y_1, \dots, y_N]^T.$$

- Hence, the **gradient descent** scheme for minimization becomes

$$\theta^{(i)} = \theta^{(i-1)} - \mu_i X^T (\mathbf{s}^{(i-1)} - \mathbf{y}).$$

- **Minimizing the negative log-likelihood:** Minimization of $L(\theta)$ with respect to θ is carried out iteratively by any iterative minimization scheme, such as the gradient descent. $L(\theta)$ is convex which guarantees a **unique minimum**. The scheme needs the computation of the respective gradient, which in turn is based on the derivative of the sigmoid link function, which can be shown to be equal to

$$\frac{d\sigma(t)}{dt} = \sigma(t) (1 - \sigma(t)).$$

- The gradient is shown to be equal to

$$\nabla L(\theta) = \sum_{n=1}^N (s_n - y_n) \mathbf{x}_n = X^T (\mathbf{s} - \mathbf{y}),$$

where

$$X^T = [\mathbf{x}_1, \dots, \mathbf{x}_N], \quad \mathbf{s} := [s_1, \dots, s_N]^T, \quad \mathbf{y} = [y_1, \dots, y_N]^T.$$

- Hence, the **gradient descent** scheme for minimization becomes

$$\theta^{(i)} = \theta^{(i-1)} - \mu_i X^T (\mathbf{s}^{(i-1)} - \mathbf{y}).$$

Feature Generation and Scatter Matrices

- Two of the major phases in designing a pattern recognition system are the so-called **feature generation** and **feature selection** phases. Selecting **information-rich features** is of paramount importance. If “bad” features are selected, whatever smart classifier one adopts, the performance is bound to be poor. The main goal in selecting features, i.e., in selecting the **feature space** in which one is going to work, can be summarized as:
- Select the features to create a feature space in which the points, which represent the training patterns, are distributed so that to have:

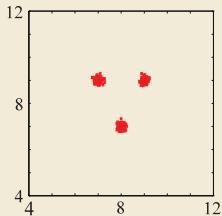
Large Between-Classes Distance
and
Small Within-Class Variance

Feature Generation and Scatter Matrices

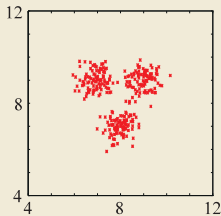
- Two of the major phases in designing a pattern recognition system are the so-called **feature generation** and **feature selection** phases. Selecting **information-rich features** is of paramount importance. If “bad” features are selected, whatever smart classifier one adopts, the performance is bound to be poor. The main goal in selecting features, i.e., in selecting the **feature space** in which one is going to work, can be summarized as:
- Select the features to create a feature space in which the points, which represent the training patterns, are distributed so that to have:

Large Between-Classes Distance
and
Small Within-Class Variance

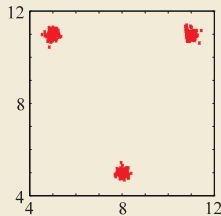
Feature Generation and Scatter Matrices



(a)



(b)



(c)

Three different choices of two-dimensional feature spaces: a) Small within-class variance and small between-classes distance. b) Large within-class variance and small between-classes distance. c) Small within-class variance and large between-classes distance. The last one is the best choice out of the three.

- A possible way to quantify the previously stated rule is via the concept of **Scatter Matrices**. There are various ways to define scatter matrices and also different ways to use them.

- Within-class scatter matrix

$$\Sigma_w = \sum_{m=1}^M P(\omega_m) \Sigma_m,$$

where, Σ_m is the covariance matrix of the points in the m th among M classes. In words, Σ_w is the **average covariance matrix** of the data in the **specific l -dimensional feature space**.

- Between-classes scatter matrix

$$\Sigma_b = \sum_{m=1}^M P(\omega_m) (\mu_m - \mu_0)(\mu_m - \mu_0)^T,$$

where μ_0 is the overall mean vector, i.e., $\mu_0 = \sum_{m=1}^M P(\omega_m) \mu_m$.

- Mixture scatter matrix

$$\Sigma_m = \Sigma_w + \Sigma_b.$$

- Three possible criteria that measure “goodness” of the selected feature space are ($|\cdot|$ denotes the determinant of a matrix):

$$J_1 := \frac{\text{trace}\{\Sigma_m\}}{\text{trace}\{\Sigma_w\}}, \quad J_2 = \frac{|\Sigma_m|}{|\Sigma_w|}, \quad J_3 = \text{trace}\{\Sigma_w^{-1} \Sigma_b\}.$$

- Within-class scatter matrix

$$\Sigma_w = \sum_{m=1}^M P(\omega_m) \Sigma_m,$$

where, Σ_m is the covariance matrix of the points in the m th among M classes. In words, Σ_w is the **average covariance matrix** of the data in the **specific l -dimensional feature space**.

- Between-classes scatter matrix

$$\Sigma_b = \sum_{m=1}^M P(\omega_m) (\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)^T,$$

where $\boldsymbol{\mu}_0$ is the overall mean vector, i.e., $\boldsymbol{\mu}_0 = \sum_{m=1}^M P(\omega_m) \boldsymbol{\mu}_m$.

- Mixture scatter matrix

$$\Sigma_m = \Sigma_w + \Sigma_b.$$

- Three possible criteria that measure “goodness” of the selected feature space are ($|\cdot|$ denotes the determinant of a matrix):

$$J_1 := \frac{\text{trace}\{\Sigma_m\}}{\text{trace}\{\Sigma_w\}}, \quad J_2 = \frac{|\Sigma_m|}{|\Sigma_w|}, \quad J_3 = \text{trace}\{\Sigma_w^{-1} \Sigma_b\}.$$

- Within-class scatter matrix

$$\Sigma_w = \sum_{m=1}^M P(\omega_m) \Sigma_m,$$

where, Σ_m is the covariance matrix of the points in the m th among M classes. In words, Σ_w is the **average covariance matrix** of the data in the **specific** l -dimensional feature space.

- Between-classes scatter matrix

$$\Sigma_b = \sum_{m=1}^M P(\omega_m) (\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)^T,$$

where $\boldsymbol{\mu}_0$ is the overall mean vector, i.e., $\boldsymbol{\mu}_0 = \sum_{m=1}^M P(\omega_m) \boldsymbol{\mu}_m$.

- Mixture scatter matrix

$$\Sigma_m = \Sigma_w + \Sigma_b.$$

- Three possible criteria that measure “goodness” of the selected feature space are ($|\cdot|$ denotes the determinant of a matrix):

$$J_1 := \frac{\text{trace}\{\Sigma_m\}}{\text{trace}\{\Sigma_w\}}, \quad J_2 = \frac{|\Sigma_m|}{|\Sigma_w|}, \quad J_3 = \text{trace}\{\Sigma_w^{-1} \Sigma_b\}.$$

- Within-class scatter matrix

$$\Sigma_w = \sum_{m=1}^M P(\omega_m) \Sigma_m,$$

where, Σ_m is the covariance matrix of the points in the m th among M classes. In words, Σ_w is the **average covariance matrix** of the data in the **specific l -dimensional feature space**.

- Between-classes scatter matrix

$$\Sigma_b = \sum_{m=1}^M P(\omega_m) (\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_m - \boldsymbol{\mu}_0)^T,$$

where $\boldsymbol{\mu}_0$ is the overall mean vector, i.e., $\boldsymbol{\mu}_0 = \sum_{m=1}^M P(\omega_m) \boldsymbol{\mu}_m$.

- Mixture scatter matrix

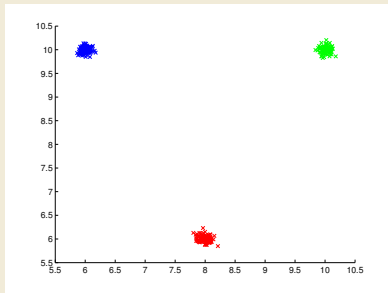
$$\Sigma_m = \Sigma_w + \Sigma_b.$$

- Three possible criteria that measure “**goodness**” of the selected feature space are ($|\cdot|$ denotes the determinant of a matrix):

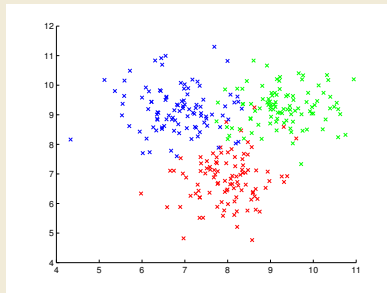
$$J_1 := \frac{\text{trace}\{\Sigma_m\}}{\text{trace}\{\Sigma_w\}}, \quad J_2 = \frac{|\Sigma_m|}{|\Sigma_w|}, \quad J_3 = \text{trace}\{\Sigma_w^{-1} \Sigma_b\}.$$

Example: Scatter Matrices

- Figure (a) corresponds to a very good feature selection case with large between-class distance and small within-class variance. The opposite is the case in Figure (b). The values of the resulting J_3 index are 1350 and 4.5, respectively. Similarly, the differences in the resulting values of the other two indices, J_1 and J_2 , are very large.



(a)



(b)

- Our interest now turns on designing **linear classifiers**. We focus on the two-class classification task. In other words, irrespective of the data distribution in each class, we decide to **partition the space in terms of hyperplanes**, i.e.,

$$g(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 = 0.$$

- In Fisher's linear discriminant analysis, the emphasis in estimating is only on $\boldsymbol{\theta}$; the bias term, θ_0 , is left out of the discussion. The inner product $\boldsymbol{\theta}^T \mathbf{x}$ can be viewed as the **projection of \mathbf{x} along the vector $\boldsymbol{\theta}$** . From geometry, we know that the respective projection is also a vector, \mathbf{y} , given by

$$\mathbf{y} = \frac{\boldsymbol{\theta}^T \mathbf{x}}{\|\boldsymbol{\theta}\|} \frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}.$$

- From now on, we will focus on the **scalar value of the projection**, $y := \boldsymbol{\theta}^T \mathbf{x}$, and ignore the scaling factor in the denominator, since scaling all features by the same value has no effect on our discussion. The goal, now, is to select **that direction, $\boldsymbol{\theta}$** , so that after projecting along this direction, a) the data in the two classes to be **as far away as possible from each other** and b) the respective **variances** of the points around their means, in each one of the classes, to be **as small as possible**.

- Our interest now turns on designing **linear classifiers**. We focus on the two-class classification task. In other words, irrespective of the data distribution in each class, we decide to **partition the space in terms of hyperplanes**, i.e.,

$$g(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 = 0.$$

- In Fisher's linear discriminant analysis, the emphasis in estimating is only on $\boldsymbol{\theta}$; the bias term, θ_0 , is left out of the discussion. The inner product $\boldsymbol{\theta}^T \mathbf{x}$ can be viewed as the **projection of \mathbf{x} along the vector $\boldsymbol{\theta}$** . From geometry, we know that the respective projection is also a vector, \mathbf{y} , given by

$$\mathbf{y} = \frac{\boldsymbol{\theta}^T \mathbf{x}}{\|\boldsymbol{\theta}\|} \frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}.$$

- From now on, we will focus on the **scalar value of the projection**, $y := \boldsymbol{\theta}^T \mathbf{x}$, and ignore the scaling factor in the denominator, since scaling all features by the same value has no effect on our discussion. The goal, now, is to select **that direction, $\boldsymbol{\theta}$** , so that after projecting along this direction, a) the data in the two classes to be **as far away as possible from each other** and b) the respective **variances** of the points around their means, in each one of the classes, to be **as small as possible**.

- Our interest now turns on designing **linear classifiers**. We focus on the two-class classification task. In other words, irrespective of the data distribution in each class, we decide to **partition the space in terms of hyperplanes**, i.e.,

$$g(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 = 0.$$

- In Fisher's linear discriminant analysis, the emphasis in estimating is only on $\boldsymbol{\theta}$; the bias term, θ_0 , is left out of the discussion. The inner product $\boldsymbol{\theta}^T \mathbf{x}$ can be viewed as the **projection of \mathbf{x} along the vector $\boldsymbol{\theta}$** . From geometry, we know that the respective projection is also a vector, \mathbf{y} , given by

$$\mathbf{y} = \frac{\boldsymbol{\theta}^T \mathbf{x}}{\|\boldsymbol{\theta}\|} \frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}.$$

- From now on, we will focus on the **scalar value of the projection**, $y := \boldsymbol{\theta}^T \mathbf{x}$, and ignore the scaling factor in the denominator, since scaling all features by the same value has no effect on our discussion. The goal, now, is to select **that direction, $\boldsymbol{\theta}$** , so that after projecting along this direction, a) the data in the two classes to be **as far away as possible from each other** and b) the respective **variances** of the points around their means, in each one of the classes, to be **as small as possible**.

- A criterion, that quantifies the aforementioned goal, is the so-called **Fisher's discriminant ratio** (FDR), defined as:

$$\text{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}, \quad (1)$$

where, μ_1 and μ_2 are the (scalar) mean values of the two classes, after the projection along θ , i.e.,

$$\mu_i = \theta^T \mu_i, \quad i = 1, 2.$$

- It can be shown that

$$\text{FDR} = \frac{\theta^T \Sigma_b \theta}{\theta^T \Sigma_w \theta}.$$

- **Proof:** We have that,

$$\begin{aligned} (\mu_1 - \mu_2)^2 &= \theta^T (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T \theta = \theta^T S_b \theta, \\ S_b &:= (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T. \end{aligned}$$

Note that if the classes are equiprobable, S_b is a **scaled version of the between-classes scatter matrix** Σ_b (under this assumption, $\mu_0 = 1/2(\mu_1 + \mu_2)$).

- A criterion, that quantifies the aforementioned goal, is the so-called **Fisher's discriminant ratio** (FDR), defined as:

$$\text{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}, \quad (1)$$

where, μ_1 and μ_2 are the (scalar) mean values of the two classes, after the projection along θ , i.e.,

$$\mu_i = \theta^T \mu_i, \quad i = 1, 2.$$

- It can be shown that

$$\text{FDR} = \frac{\theta^T \Sigma_b \theta}{\theta^T \Sigma_w \theta}.$$

- **Proof:** We have that,

$$\begin{aligned} (\mu_1 - \mu_2)^2 &= \theta^T (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T \theta = \theta^T S_b \theta, \\ S_b &:= (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T. \end{aligned}$$

Note that if the classes are equiprobable, S_b is a **scaled version of the between-classes scatter matrix** Σ_b (under this assumption, $\mu_0 = 1/2(\mu_1 + \mu_2)$).

- A criterion, that quantifies the aforementioned goal, is the so-called **Fisher's discriminant ratio** (FDR), defined as:

$$\text{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}, \quad (1)$$

where, μ_1 and μ_2 are the (scalar) mean values of the two classes, after the projection along θ , i.e.,

$$\mu_i = \theta^T \mu_i, \quad i = 1, 2.$$

- It can be shown that

$$\text{FDR} = \frac{\theta^T \Sigma_b \theta}{\theta^T \Sigma_w \theta}.$$

- **Proof:** We have that,

$$\begin{aligned} (\mu_1 - \mu_2)^2 &= \theta^T (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T \theta = \theta^T S_b \theta, \\ S_b &:= (\mu_1 - \mu_2) (\mu_1 - \mu_2)^T. \end{aligned}$$

Note that if the classes are equiprobable, S_b is a **scaled version** of the **between-classes scatter matrix** Σ_b (under this assumption, $\mu_0 = 1/2(\mu_1 + \mu_2)$).

- Thus we have,

$$(\mu_1 - \mu_2)^2 \propto \boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}. \quad (2)$$

Moreover,

$$\sigma_i^2 = \mathbb{E} [(y - \mu_i)^2] = \mathbb{E} [\boldsymbol{\theta}^T (x - \mu_i)(x - \mu_i)^T \boldsymbol{\theta}] = \boldsymbol{\theta}^T \Sigma_i \boldsymbol{\theta}, \quad i = 1, 2,$$

which leads to,

$$\sigma_1^2 + \sigma_2^2 = \boldsymbol{\theta}^T S_w \boldsymbol{\theta},$$

where $S_w = \Sigma_1 + \Sigma_2$.

- Note that if the classes are equiprobable, S_w becomes a **scaled version of the within-class scatter matrix**, and we have that

$$\sigma_1^2 + \sigma_2^2 \propto \boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}, \quad (3)$$

Combining (1), (2) and (3) and neglecting the proportionality constants, the claim has been proved. That is,

$$\text{FDR} = \frac{\boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}}{\boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}}.$$

- The ratio in this form is also known as the **generalized Rayleigh quotient**.

- Thus we have,

$$(\mu_1 - \mu_2)^2 \propto \boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}. \quad (2)$$

Moreover,

$$\sigma_i^2 = \mathbb{E} [(y - \mu_i)^2] = \mathbb{E} [\boldsymbol{\theta}^T (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\theta}] = \boldsymbol{\theta}^T \Sigma_i \boldsymbol{\theta}, \quad i = 1, 2,$$

which leads to,

$$\sigma_1^2 + \sigma_2^2 = \boldsymbol{\theta}^T S_w \boldsymbol{\theta},$$

where $S_w = \Sigma_1 + \Sigma_2$.

- Note that if the classes are equiprobable, S_w becomes a scaled version of the within-class scatter matrix, and we have that

$$\sigma_1^2 + \sigma_2^2 \propto \boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}, \quad (3)$$

Combining (1), (2) and (3) and neglecting the proportionality constants, the claim has been proved. That is,

$$\text{FDR} = \frac{\boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}}{\boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}}.$$

- The ratio in this form is also known as the generalized Rayleigh quotient.

- Thus we have,

$$(\mu_1 - \mu_2)^2 \propto \boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}. \quad (2)$$

Moreover,

$$\sigma_i^2 = \mathbb{E} [(y - \mu_i)^2] = \mathbb{E} [\boldsymbol{\theta}^T (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\theta}] = \boldsymbol{\theta}^T \Sigma_i \boldsymbol{\theta}, \quad i = 1, 2,$$

which leads to,

$$\sigma_1^2 + \sigma_2^2 = \boldsymbol{\theta}^T S_w \boldsymbol{\theta},$$

where $S_w = \Sigma_1 + \Sigma_2$.

- Note that if the classes are equiprobable, S_w becomes a **scaled version of the within-class scatter matrix**, and we have that

$$\sigma_1^2 + \sigma_2^2 \propto \boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}, \quad (3)$$

Combining (1), (2) and (3) and neglecting the proportionality constants, the claim has been proved. That is,

$$\text{FDR} = \frac{\boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}}{\boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}}.$$

- The ratio in this form is also known as the **generalized Rayleigh quotient**.

- Thus we have,

$$(\mu_1 - \mu_2)^2 \propto \boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}. \quad (2)$$

Moreover,

$$\sigma_i^2 = \mathbb{E} [(y - \mu_i)^2] = \mathbb{E} [\boldsymbol{\theta}^T (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\theta}] = \boldsymbol{\theta}^T \Sigma_i \boldsymbol{\theta}, \quad i = 1, 2,$$

which leads to,

$$\sigma_1^2 + \sigma_2^2 = \boldsymbol{\theta}^T S_w \boldsymbol{\theta},$$

where $S_w = \Sigma_1 + \Sigma_2$.

- Note that if the classes are equiprobable, S_w becomes a **scaled version of the within-class scatter matrix**, and we have that

$$\sigma_1^2 + \sigma_2^2 \propto \boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}, \quad (3)$$

Combining (1), (2) and (3) and neglecting the proportionality constants, the claim has been proved. That is,

$$\text{FDR} = \frac{\boldsymbol{\theta}^T \Sigma_b \boldsymbol{\theta}}{\boldsymbol{\theta}^T \Sigma_w \boldsymbol{\theta}}.$$

- The ratio in this form is also known as the **generalized Rayleigh quotient**.

Maximizing the Generalized Rayleigh Quotient

- **Maximizing FDR:** Our goal now becomes that of maximizing the FDR with respect to θ . This is a case of the **generalized Rayleigh ratio**, which is known that it is maximized if θ satisfies,

$$\Sigma_b \theta = \lambda \Sigma_w \theta,$$

where λ is the **maximum eigenvalue** of the matrix $\Sigma_w^{-1} \Sigma_b$.

- However, for our specific case, where Σ_b is a **rank-one matrix** and there is **only one nonzero eigenvalue**, we can bypass the need for solving an eigenvalue-eigenvector problem. Observe that,

$$\lambda \Sigma_w \theta \propto (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \theta \propto (\mu_1 - \mu_2).$$

- In other words, $\Sigma_w \theta$ lies in the direction of $(\mu_1 - \mu_2)$, and since we are only interested in the direction, we can finally write that,

$$\theta = \Sigma_w^{-1}(\mu_1 - \mu_2),$$

assuming, of course, that Σ_w is invertible. In practice, Σ_w is obtained as the respective sample mean using the available observations.

Maximizing the Generalized Rayleigh Quotient

- **Maximizing FDR:** Our goal now becomes that of maximizing the FDR with respect to θ . This is a case of the **generalized Rayleigh ratio**, which is known that it is maximized if θ satisfies,

$$\Sigma_b \theta = \lambda \Sigma_w \theta,$$

where λ is the **maximum eigenvalue** of the matrix $\Sigma_w^{-1} \Sigma_b$.

- However, for our specific case, where Σ_b is a **rank-one matrix** and there is **only one nonzero eigenvalue**, we can bypass the need for solving an eigenvalue-eigenvector problem. Observe that,

$$\lambda \Sigma_w \theta \propto (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \theta \propto (\mu_1 - \mu_2).$$

- In other words, $\Sigma_w \theta$ lies in the direction of $(\mu_1 - \mu_2)$, and since we are only interested in the direction, we can finally write that,

$$\theta = \Sigma_w^{-1}(\mu_1 - \mu_2),$$

assuming, of course, that Σ_w is invertible. In practice, Σ_w is obtained as the respective sample mean using the available observations.

Maximizing the Generalized Rayleigh Quotient

- **Maximizing FDR:** Our goal now becomes that of maximizing the FDR with respect to θ . This is a case of the **generalized Rayleigh ratio**, which is known that it is maximized if θ satisfies,

$$\Sigma_b \theta = \lambda \Sigma_w \theta,$$

where λ is the **maximum eigenvalue** of the matrix $\Sigma_w^{-1} \Sigma_b$.

- However, for our specific case, where Σ_b is a **rank-one matrix** and there is **only one nonzero eigenvalue**, we can bypass the need for solving an eigenvalue-eigenvector problem. Observe that,

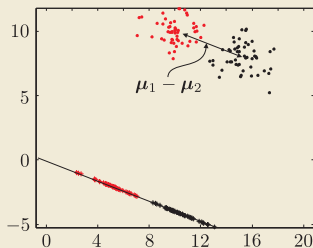
$$\lambda \Sigma_w \theta \propto (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \theta \propto (\mu_1 - \mu_2).$$

- In other words, $\Sigma_w \theta$ lies in the direction of $(\mu_1 - \mu_2)$, and since we are only interested in the direction, we can finally write that,

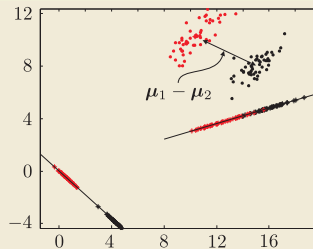
$$\theta = \Sigma_w^{-1}(\mu_1 - \mu_2),$$

assuming, of course, that Σ_w is invertible. In practice, Σ_w is obtained as the respective sample mean using the available observations.

Fisher's Linear Discriminant



(a)



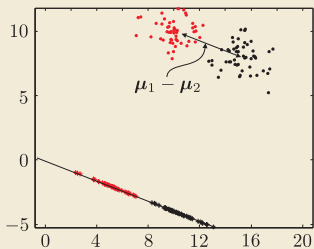
(b)

a) The optimal direction resulting from Fisher's discriminant, for two spherically distributed classes. The direction on which projection takes place is parallel to the segment joining the mean values of the data in the two classes. b) The line on the bottom left of the figure corresponds to the direction that results from Fisher's discriminant; observe that it is no more parallel to $\mu_1 - \mu_2$, because the classes are not spherically distributed. Note that, projecting on the line on the right, results in class overlap.

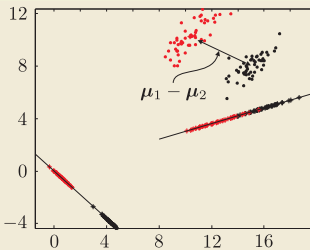
- In order the Fisher's discriminant method to be used as a classifier, a threshold θ_0 must be adopted, and the decision rule becomes:

$$y = (\mu_1 - \mu_2)^T \Sigma_w^{-1} x + \theta_0 \begin{cases} > 0, \text{ class } \omega_1, \\ < 0, \text{ class } \omega_2. \end{cases}$$

Various rules for θ_0 can be used. Note that the previous rule is the same with the optimal (linear) Bayesian one; however, now, no assumption concerning Gaussianity was adopted.



(a)



(b)

a) The optimal direction resulting from Fisher's discriminant, for two spherically distributed classes. The direction on which projection takes place is parallel to the segment joining the mean values of the data in the two classes. b) The line on the bottom left of the figure corresponds to the direction that results from Fisher's discriminant; observe that it is no more parallel to $\mu_1 - \mu_2$, because the classes are not spherically distributed. Note that, projecting on the line on the right, results in class overlap.

- In order the Fisher's discriminant method to be used as a classifier, a threshold θ_0 must be adopted, and the decision rule becomes:

$$y = (\mu_1 - \mu_2)^T \Sigma_w^{-1} \mathbf{x} + \theta_0 \begin{cases} > 0, \text{ class } \omega_1, \\ < 0, \text{ class } \omega_2. \end{cases}$$

Various rules for θ_0 can be used. Note that the previous rule is the same with the optimal (linear) Bayesian one; however, now, no assumption concerning Gaussianity was adopted.

Classification Trees

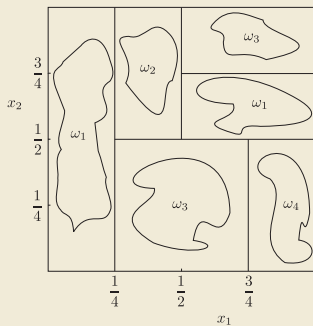
- **Classification trees** are **multistage** systems and classification of a pattern into a class is achieved **sequentially**. Via a series of tests, classes are **rejected** in a sequential fashion, till a decision is finally reached in favor of one, remaining, class. Each one of the tests, whose outcome decides which classes are rejected, is of a **binary “Yes” or “No”** type and it is applied to a **single feature**.
- Our goal is to present the main philosophy around a special type of trees, known as **ordinary binary classification trees** (OBCT). They belong to a more general class of methods which construct trees, both for classification as well as regression, known as **classification and regression trees** (CART)
- The basic idea around OBCTs is to partition the feature space into **(hyper)rectangles**; that is, the space is **partitioned via hyperplanes, which are parallel to the axes**. The division of the space in **(hyper)rectangles** is performed via a series of “questions”, of the form: is the value of the feature $x_i < a$? This is also known as the **splitting criterion**. The sequence of questions can nicely be realized via the use of a tree.

Classification Trees

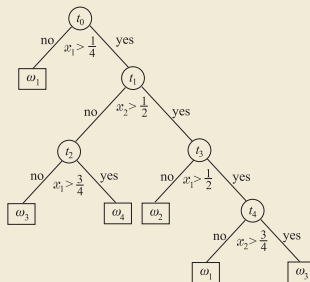
- **Classification trees** are **multistage** systems and classification of a pattern into a class is achieved **sequentially**. Via a series of tests, classes are **rejected** in a sequential fashion, till a decision is finally reached in favor of one, remaining, class. Each one of the tests, whose outcome decides which classes are rejected, is of a **binary “Yes” or “No”** type and it is applied to a **single feature**.
- Our goal is to present the main philosophy around a special type of trees, known as **ordinary binary classification trees** (OBCT). They belong to a more general class of methods which construct trees, both for classification as well as regression, known as **classification and regression trees** (CART)
- The basic idea around OBCTs is to partition the feature space into (hyper)rectangles; that is, the space is **partitioned via hyperplanes, which are parallel to the axes**. The division of the space in (hyper)rectangles is performed via a series of “questions”, of the form: is the value of the feature $x_i < a$? This is also known as the **splitting criterion**. The sequence of questions can nicely be realized via the use of a tree.

- **Classification trees** are **multistage** systems and classification of a pattern into a class is achieved **sequentially**. Via a series of tests, classes are **rejected** in a sequential fashion, till a decision is finally reached in favor of one, remaining, class. Each one of the tests, whose outcome decides which classes are rejected, is of a **binary “Yes” or “No”** type and it is applied to a **single feature**.
- Our goal is to present the main philosophy around a special type of trees, known as **ordinary binary classification trees** (OBCT). They belong to a more general class of methods which construct trees, both for classification as well as regression, known as **classification and regression trees** (CART)
- The basic idea around OBCTs is to partition the feature space into **(hyper)rectangles**; that is, the space is **partitioned via hyperplanes, which are parallel to the axes**. The division of the space in (hyper)rectangles is performed via a series of “questions”, of the form: is the value of the feature $x_i < a$? This is also known as the **splitting criterion**. The sequence of questions can nicely be realized via the use of a tree.

- The previous descriptions are illustrated in the following figures:



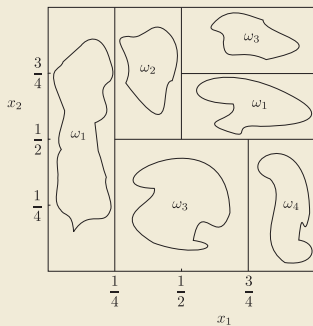
Partition of the features space, via an OBCT tree.



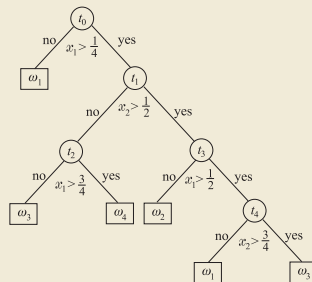
The corresponding classification tree.

- Starting from the **root node**, a path of successive decisions is realized, till a **leaf node** is reached. Each leaf node is associated with a **single class**. The assignment of a point to a class is done according to the label of the respective leaf node. This type of classification is conceptually simple and **easily interpretable**.

- The previous descriptions are illustrated in the following figures:



Partition of the features space, via an OBCT tree.



The corresponding classification tree.

- Starting from the **root node**, a path of successive decisions is realized, till a **leaf node** is reached. Each leaf node is associated with a **single** class. The assignment of a point to a class is done according to the label of the respective leaf node. This type of classification is conceptually simple and **easily interpretable**.

- Once a tree has been developed, classification is straightforward. The major challenge lies in **constructing the tree, by exploiting the information which resides in the training data set**. The main questions one is confronted with, while designing a tree, are:

- Which splitting criterion to be adopted?
- When to stop growing a tree and declare a node as final?
- How a leaf node is associated with a specific class?

- **Splitting Criterion:** The questions asked at each node are of the type,

$$\text{is } x_i \leq a?$$

- The goal is to **select an appropriate value for the threshold value a** . Assume that starting from the root node, the tree has grown up to the current node, say t . Each node, t , is associated with a subset $X_t \subseteq X$ of the training data set, X . This is the set of the training points that have survived to this node after the tests, which have taken place at the previous nodes in the tree. The goal is to **split X_t** into two **disjoint** subsets, namely X_{tY} , and X_{tN} , depending on the answer in the specific question at node t .

- Once a tree has been developed, classification is straightforward. The major challenge lies in **constructing the tree, by exploiting the information which resides in the training data set**. The main questions one is confronted with, while designing a tree, are:
 - Which splitting criterion to be adopted?
 - When to stop growing a tree and declare a node as final?
 - How a leaf node is associated with a specific class?
- **Splitting Criterion:** The questions asked at each node are of the type,

$$\text{is } x_i \leq a?$$

- The goal is to **select an appropriate value for the threshold value a** . Assume that starting from the root node, the tree has grown up to the current node, say t . Each node, t , is associated with a subset $X_t \subseteq X$ of the training data set, X . This is the set of the training points that have survived to this node after the tests, which have taken place at the previous nodes in the tree. The goal is to **split X_t** into two **disjoint** subsets, namely X_{tY} , and X_{tN} , depending on the answer in the specific question at node t .

- Once a tree has been developed, classification is straightforward. The major challenge lies in **constructing the tree, by exploiting the information which resides in the training data set**. The main questions one is confronted with, while designing a tree, are:

- Which splitting criterion to be adopted?
- When to stop growing a tree and declare a node as final?
- How a leaf node is associated with a specific class?

- **Splitting Criterion**: The questions asked at each node are of the type,

$$\text{is } x_i \leq a?$$

- The goal is to **select an appropriate value for the threshold value a** . Assume that starting from the root node, the tree has grown up to the current node, say t . Each node, t , is associated with a subset $X_t \subseteq X$ of the training data set, X . This is the set of the training points that have survived to this node after the tests, which have taken place at the previous nodes in the tree. The goal is to **split X_t** into two **disjoint** subsets, namely X_{tY} , and X_{tN} , depending on the answer in the specific question at node t .

- For every split, the following is true,

$$\begin{aligned}X_{tY} \cap X_{tN} &= \emptyset, \\ X_{tY} \cup X_{tN} &= X_t.\end{aligned}$$

- The goal in each node is to select **which feature is to be tested** and also what is the **best value of the corresponding threshold value a** . The adopted philosophy is to make the choice so that every split to generate sets, X_{tY} , X_{tN} , which are more **class-homogeneous** compared to X_t . In other words, the data in each one of the two descendant sets must show a **higher preference** to specific classes, compared to the ancestor set.
- For example, assume that the data in X_t consist of points which belong to four classes, e.g., ω_1 , ω_2 , ω_3 , ω_4 . The idea is to perform the splitting so that most of the data in X_{tY} to belong to, say, ω_1 , ω_2 and most of the data in X_{tN} to ω_3 , ω_4 .
- In the adopted terminology, the sets X_{tY} and X_{tN} should be **purier** compared to X_t . Thus, we must first select a criterion, which measures node **impurity** and then compute a) the threshold value and b) choose the specific feature (to be tested), so that to **maximize the decrease in node impurity**.

- For every split, the following is true,

$$\begin{aligned}X_{tY} \cap X_{tN} &= \emptyset, \\X_{tY} \cup X_{tN} &= X_t.\end{aligned}$$

- The goal in each node is to select **which feature is to be tested** and also what is the **best value of the corresponding threshold value a** . The adopted philosophy is to make the choice so that every split to generate sets, X_{tY} , X_{tN} , which are more **class-homogeneous** compared to X_t . In other words, the data in each one of the two descendant sets must show a **higher preference** to specific classes, compared to the ancestor set.
- For example, assume that the data in X_t consist of points which belong to four classes, e.g., ω_1 , ω_2 , ω_3 , ω_4 . The idea is to perform the splitting so that most of the data in X_{tY} to belong to, say, ω_1 , ω_2 and most of the data in X_{tN} to ω_3 , ω_4 .
- In the adopted terminology, the sets X_{tY} and X_{tN} should be **purier** compared to X_t . Thus, we must first select a criterion, which measures node **impurity** and then compute a) the threshold value and b) choose the specific feature (to be tested), so that to **maximize the decrease in node impurity**.

- For every split, the following is true,

$$\begin{aligned}X_{tY} \cap X_{tN} &= \emptyset, \\X_{tY} \cup X_{tN} &= X_t.\end{aligned}$$

- The goal in each node is to select **which feature is to be tested** and also what is the **best value of the corresponding threshold value a** . The adopted philosophy is to make the choice so that every split to generate sets, X_{tY} , X_{tN} , which are more **class-homogeneous** compared to X_t . In other words, the data in each one of the two descendant sets must show a **higher preference** to specific classes, compared to the ancestor set.
- For example, assume that the data in X_t consist of points which belong to four classes, e.g., ω_1 , ω_2 , ω_3 , ω_4 . The idea is to perform the splitting so that most of the data in X_{tY} to belong to, say, ω_1 , ω_2 and most of the data in X_{tN} to ω_3 , ω_4 .
- In the adopted terminology, the sets X_{tY} and X_{tN} should be **purier** compared to X_t . Thus, we must first select a criterion, which measures node **impurity** and then compute a) the threshold value and b) choose the specific feature (to be tested), so that to **maximize the decrease in node impurity**.

- For every split, the following is true,

$$\begin{aligned}X_{tY} \cap X_{tN} &= \emptyset, \\ X_{tY} \cup X_{tN} &= X_t.\end{aligned}$$

- The goal in each node is to select **which feature is to be tested** and also what is the **best value of the corresponding threshold value a** . The adopted philosophy is to make the choice so that every split to generate sets, X_{tY} , X_{tN} , which are more **class-homogeneous** compared to X_t . In other words, the data in each one of the two descendant sets must show a **higher preference** to specific classes, compared to the ancestor set.
- For example, assume that the data in X_t consist of points which belong to four classes, e.g., ω_1 , ω_2 , ω_3 , ω_4 . The idea is to perform the splitting so that most of the data in X_{tY} to belong to, say, ω_1 , ω_2 and most of the data in X_{tN} to ω_3 , ω_4 .
- In the adopted terminology, the sets X_{tY} and X_{tN} should be **purier** compared to X_t . Thus, we must first select a criterion, which measures node **impurity** and then compute a) the threshold value and b) choose the specific feature (to be tested), so that to **maximize the decrease in node impurity**.

- A common measure to quantify node impurity is the associated with a node, t , **entropy**, defined as

$$I(t) = - \sum_{m=1}^M P(\omega_m|t) \log_2 P(\omega_m|t),$$

where $\log_2(\cdot)$ is the base-two logarithm. The **maximum value of $I(t)$** occurs if all **probabilities are equal** (maximum impurity) and the **smallest value**, which is equal to zero, when **only one of the probability values is one** and the rest equal to zero. Probabilities are approximated as,

$$P(\omega_m|t) = \frac{N_t^m}{N_t}, \quad m = 1, 2, \dots, M,$$

where, N_t^m is the number of the points from class m in X_t , and N_t the total number of points in X_t . The decrease in node impurity, after splitting the data into two sets, is defined as:

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

where $I(t_Y)$ and $I(t_N)$ are the impurities associated with the two new sets, respectively.

- Another popular impurity measuring index, which results in slightly sharper maximum compared to the entropy one, is the so-called **Gini index**, defined as,

$$I(t) = \sum_{m=1}^M P(\omega_m|t) (1 - P(\omega_m|t)) .$$

This index is also zero if one of the probability values is equal to 1 and the rest are zero, and it takes its maximum value when all classes are equiprobable.

- Besides the splitting criterion, the following three steps are needed for completing the construction of a tree:

Stop splitting rule: The obvious question that is posed, while constructing a tree, is when to stop growing it. One possible way is to adopt a threshold value, T , and stop splitting a node once the maximum value $\Delta I(t)$, for all possible splits, is smaller than T . Another possibility is to stop when the cardinality of X_t becomes smaller than a certain number or if the node is pure, in the sense that all points in the node belong to a single class.

- Another popular impurity measuring index, which results in slightly sharper maximum compared to the entropy one, is the so-called **Gini index**, defined as,

$$I(t) = \sum_{m=1}^M P(\omega_m|t) (1 - P(\omega_m|t)) .$$

This index is also zero if one of the probability values is equal to 1 and the rest are zero, and it takes its maximum value when all classes are equiprobable.

- Besides the splitting criterion, the following three steps are needed for completing the construction of a tree:

Stop splitting rule: The obvious question that is posed, while constructing a tree, is when to stop growing it. One possible way is to adopt a threshold value, T , and stop splitting a node once the maximum value $\Delta I(t)$, for all possible splits, is smaller than T . Another possibility is to stop when the cardinality of X_t becomes smaller than a certain number or if the node is pure, in the sense that all points in the node belong to a single class.

- Another popular impurity measuring index, which results in slightly sharper maximum compared to the entropy one, is the so-called **Gini index**, defined as,

$$I(t) = \sum_{m=1}^M P(\omega_m|t) (1 - P(\omega_m|t)) .$$

This index is also zero if one of the probability values is equal to 1 and the rest are zero, and it takes its maximum value when all classes are equiprobable.

- Besides the splitting criterion, the following three steps are needed for completing the construction of a tree:

Stop splitting rule: The obvious question that is posed, while constructing a tree, is when to stop growing it. One possible way is to adopt a threshold value, T , and stop splitting a node once the maximum value $\Delta I(t)$, for all possible splits, is smaller than T . Another possibility is to stop when the cardinality of X_t becomes smaller than a certain number or if the node is pure, in the sense that all points in the node belong to a single class.

Constructing Classification Trees

- **Class Assignment Rule:** Once a node, t , is declared to be a leaf node, it is assigned a class label; usually this is done on a majority voting rationale. That is, it is assigned the label of the class where most of the data points in X_t belong.
- **Pruning a Tree:** Experience has shown that growing a tree and using a stopping rule does not always work well in practice; growing may either stop early or may result in trees of very large size. A common recipe in practice is to first grow a tree up to a large size and then adopt a **pruning technique to eliminate nodes**. Different pruning criteria can be used; a popular one is to combine an estimate of the **error probability** with a **complexity** measuring index.
- **Advantages of Classification Trees:** a) They can naturally treat mixtures of numeric and categorical variables, b) they scale well with large data sets, c) they can treat in an effective way missing variables and c) they are easily **interpretable**; in other words, it is possible for a human to understand the reason of the output of the learning algorithm. In some applications, such as in financial decisions, this is a legal requirement.

Constructing Classification Trees

- **Class Assignment Rule:** Once a node, t , is declared to be a leaf node, it is assigned a class label; usually this is done on a majority voting rationale. That is, it is assigned the label of the class where most of the data points in X_t belong.
- **Pruning a Tree:** Experience has shown that growing a tree and using a stopping rule does not always work well in practice; growing may either stop early or may result in trees of very large size. A common recipe in practice is to first grow a tree up to a large size and then adopt a **pruning technique to eliminate nodes**. Different pruning criteria can be used; a popular one is to combine an estimate of the **error probability** with a **complexity** measuring index.
- **Advantages of Classification Trees:** a) They can naturally treat mixtures of numeric and categorical variables, b) they scale well with large data sets, c) they can treat in an effective way missing variables and c) they are easily **interpretable**; in other words, it is possible for a human to understand the reason of the output of the learning algorithm. In some applications, such as in financial decisions, this is a legal requirement.

Constructing Classification Trees

- **Class Assignment Rule:** Once a node, t , is declared to be a leaf node, it is assigned a class label; usually this is done on a majority voting rationale. That is, it is assigned the label of the class where most of the data points in X_t belong.
- **Pruning a Tree:** Experience has shown that growing a tree and using a stopping rule does not always work well in practice; growing may either stop early or may result in trees of very large size. A common recipe in practice is to first grow a tree up to a large size and then adopt a **pruning technique to eliminate nodes**. Different pruning criteria can be used; a popular one is to combine an estimate of the **error probability** with a **complexity** measuring index.
- **Advantages of Classification Trees:** a) They can naturally treat mixtures of numeric and categorical variables, b) they scale well with large data sets, c) they can treat in an effective way missing variables and c) they are easily **interpretable**; in other words, it is possible for a human to understand the reason of the output of the learning algorithm. In some applications, such as in financial decisions, this is a legal requirement.

Constructing Classification Trees

- **Drawbacks of Classification Trees:**
 - The prediction performance of the tree classifiers is not, in general, as good as other methods, e.g., support vector machines and neural networks.
 - Another major drawback is that they are **unstable** classifiers. That is, a **small change** in the training data set can result in a **very different tree**. The reason for this lies in the hierarchical nature of the tree classifiers. An error that occurs in a node at a high level of the tree propagates all the way down to the leaves below it.
- **Bagging** (Bootstrap Aggregating) is a technique that can reduce the variance and improve the generalization error performance. The basic idea is to create a number of, say, B variants, X_1, X_2, \dots, X_B , of the training set, X , using **bootstrap** techniques, by uniformly sampling from X with replacement. For each of the training set variants, X_i , a tree, T_i , is constructed. The final decision for the classification of a given point is in favor of the class predicted by the majority of the subclassifiers, T_i , $i = 1, 2, \dots, B$.

- **Drawbacks of Classification Trees:**
 - The prediction performance of the tree classifiers is not, in general, as good as other methods, e.g., support vector machines and neural networks.
 - Another major drawback is that they are **unstable** classifiers. That is, a **small change** in the training data set can result in a **very different tree**. The reason for this lies in the hierarchical nature of the tree classifiers. An error that occurs in a node at a high level of the tree propagates all the way down to the leaves below it.
- **Bagging** (Bootstrap Aggregating) is a technique that can reduce the variance and improve the generalization error performance. The basic idea is to create a number of, say, B variants, X_1, X_2, \dots, X_B , of the training set, X , using **bootstrap** techniques, by uniformly sampling from X with replacement. For each of the training set variants, X_i , a tree, T_i , is constructed. The final decision for the classification of a given point is in favor of the class predicted by the majority of the subclassifiers, T_i , $i = 1, 2, \dots, B$.

Which Classifier Then? The No-Free Lunch Theorem

- We have discussed a number of classifiers and more methods are discussed in the book, such as concerning support vector machines, Bayesian methods and neural/deep networks. The obvious question that may be naturally raised is: which method then? Unfortunately, there is no definite answer to it.
- **No Free Lunch Theorem:** The goal of the design of any classifier, and in general of any learning scheme, is to provide a **good generalization performance**. However, there is **no context-independent or usage-independent reasons to support one learning technique from another**. Each learning task, represented by the available data set, will show a preference to a **specific learning scheme, which fits the specificities of the particular problem at hand**. An algorithm, which scores top in one problem, can score low for another.

Which Classifier Then? The No-Free Lunch Theorem

- We have discussed a number of classifiers and more methods are discussed in the book, such as concerning support vector machines, Bayesian methods and neural/deep networks. The obvious question that may be naturally raised is: which method then? Unfortunately, there is no definite answer to it.
- **No Free Lunch Theorem:** The goal of the design of any classifier, and in general of any learning scheme, is to provide a **good generalization performance**. However, there is **no context-independent or usage-independent reasons to support one learning technique from another**. Each learning task, represented by the available data set, will show a preference to a **specific learning scheme, which fits the specificities of the particular problem at hand**. An algorithm, which scores top in one problem, can score low for another.

Combining Classifiers

- A trend in order to improve performance is to **combine different classifiers together** and exploit their individual advantages. An observation that justifies such an approach is that during testing, there are patterns on which even the best classifier, for a particular task, fails to predict their true class. In contrast, the same patterns can be classified correctly by other classifiers, with an inferior overall performance. This points out that there may be some **complimentarily among different classifiers** and combination can lead to a boosted performance, compared to that obtained from the best (single) classifier. Recall that bagging, previously mentioned for trees, is a type of classifier combination. There are a number of different combination schemes. For example, having trained a number of different classifiers, on the same training set, then use:
 - The arithmetic averaging rule.
 - The geometric averaging rule.
 - The Majority voting rule.

The Boosting Approach

- This approach is built around the **boosting** rationale. That is, a **weak learning algorithm**, i.e., one which does slightly better than a random guessing, can be **boosted** into a strong one, with a good performance index. At the heart of such techniques lies the so-called **base learner**, which is a **weak** one. Boosting consists of an iterative scheme, where at each step the base learner is optimally computed, using a **different training set**; the set at the current iteration is generated via a weighting of the training samples, each time using a **different set of weights**. The latter are computed so that to take into account the achieved performance up to the current iteration step. The final learner is obtained via a **weighted average** of all the **hierarchically** designed base learners. Thus, boosting can also be considered as a scheme for combining different learners.
- It turns out that, one can significantly improve the (poor) performance of the weak learner. This is very interesting indeed. Training a weak classifier, by **appropriate manipulation of the training data** (as a matter of fact, the weighting mechanism identifies hard samples, i.e., the ones which keep failing, and places more emphasis on them) one can obtain a strong classifier.

The Boosting Approach

- This approach is built around the **boosting** rationale. That is, a **weak learning algorithm**, i.e., one which does slightly better than a random guessing, can be **boosted** into a strong one, with a good performance index. At the heart of such techniques lies the so-called **base learner**, which is a **weak** one. Boosting consists of an iterative scheme, where at each step the base learner is optimally computed, using a **different training set**; the set at the current iteration is generated via a weighting of the training samples, each time using a **different set of weights**. The latter are computed so that to take into account the achieved performance up to the current iteration step. The final learner is obtained via a **weighted average** of all the **hierarchically** designed base learners. Thus, boosting can also be considered as a scheme for combining different learners.
- It turns out that, one can significantly improve the (poor) performance of the weak learner. This is very interesting indeed. Training a weak classifier, by **appropriate manipulation of the training data** (as a matter of fact, the weighting mechanism identifies hard samples, i.e., the ones which keep failing, and places more emphasis on them) one can obtain a strong classifier.