Data mining and Machine learning

Part 18. Probabilistic Classification

Bayes Classifier

Let the training dataset $\boldsymbol{D}$ consists of n points $\boldsymbol{x}_i$ in a d-dimensional space, and let y_i denote the class for each point, with $y_i \in \{c_1, c_2, \ldots, c_k\}$.

The Bayes classifier estimates the posterior probability $P(c_i|\boldsymbol{x})$ for each class c_i, and chooses the class that has the largest probability. The predicted class for $\boldsymbol{x}$ is given as

$$\hat{y} = \arg\max_{c_i}\{P(c_i|\boldsymbol{x})\}$$

According to the Bayes theorem, we have

$$P(c_i|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|c_i) \cdot P(c_i)}{P(\boldsymbol{x})}$$

Because $P(\boldsymbol{x})$ is fixed for a given point, Bayes rule can be rewritten as

$$\hat{y} = \arg\max_{c_i}\{P(c_i|\boldsymbol{x})\} = \arg\max_{c_i}\left\{\frac{P(\boldsymbol{x}|c_i)P(c_i)}{P(\boldsymbol{x})}\right\} = \arg\max_{c_i}\{P(\boldsymbol{x}|c_i)P(c_i)\}$$

Estimating the Prior Probability: $P(c_i)$

Let $\boldsymbol{D}_i$ denote the subset of points in $\boldsymbol{D}$ that are labeled with class c_i:

$$\boldsymbol{D}_i = \{\boldsymbol{x}_j \in \boldsymbol{D} \mid \boldsymbol{x}_j \text{ has class } y_j = c_i\}$$

Let the size of the dataset $\boldsymbol{D}$ be given as $|\boldsymbol{D}| = n$, and let the size of each class-specific subset $\boldsymbol{D}_i$ be given as $|\boldsymbol{D}_i| = n_i$.

The prior probability for class c_i can be estimated as follows:

$$\hat{P}(c_i) = \frac{n_i}{n}$$

Estimating the Likelihood

Numeric Attributes, Parametric Approach

To estimate the likelihood $P(\boldsymbol{x}|c_i)$, we have to estimate the joint probability of $\boldsymbol{x}$ across all the d dimensions, i.e., we have to estimate $P\big(\boldsymbol{x} = (x_1, x_2, \ldots, x_d)|c_i\big)$.

In the parametric approach we assume that each class c_i is normally distributed, and we use the estimated mean $\hat{\boldsymbol{\mu}}_i$ and covariance matrix $\widehat{\boldsymbol{\Sigma}}_i$ to compute the probability density at $\boldsymbol{x}$

$$\hat{f}_i(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\widehat{\boldsymbol{\Sigma}}_i|}} \exp\left\{-\frac{(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_i)^T \widehat{\boldsymbol{\Sigma}}_i^{-1} (\boldsymbol{x} - \hat{\boldsymbol{\mu}}_i)}{2}\right\}$$

The posterior probability is then given as

$$P(c_i|\boldsymbol{x}) = \frac{\hat{f}_i(\boldsymbol{x})P(c_i)}{\sum_{j=1}^{k} \hat{f}_j(\boldsymbol{x})P(c_j)}$$

The predicted class for $\boldsymbol{x}$ is:

$$\hat{y} = \arg\max_{c_i}\Big\{\hat{f}_i(\boldsymbol{x})P(c_i)\Big\}$$

Bayes Classifier Algorithm

BayesClassifier ($\boldsymbol{D} = \{(\boldsymbol{x}_j, y_j)\}_{j=1}^{n}$)**:**

for $i = 1, \ldots, k$ **do**
- $\boldsymbol{D}_i \leftarrow \{\boldsymbol{x}_j \mid y_j = c_i, j = 1, \ldots, n\}$ // class-specific subsets
- $n_i \leftarrow |\boldsymbol{D}_i|$ // cardinality
- $\hat{P}(c_i) \leftarrow n_i / n$ // prior probability
- $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in \boldsymbol{D}_i} \boldsymbol{x}_j$ // mean
- $\boldsymbol{Z}_i \leftarrow \boldsymbol{D}_i - \mathbf{1}_{n_i} \hat{\boldsymbol{\mu}}_i^T$ // centered data
- $\widehat{\boldsymbol{\Sigma}}_i \leftarrow \frac{1}{n_i} \boldsymbol{Z}_i^T \boldsymbol{Z}_i$ // covariance matrix

return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i$ *for all* $i = 1, \ldots, k$

Testing ($\boldsymbol{x}$ **and** $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\widehat{\boldsymbol{\Sigma}}_i$, **for all** $i \in [1, k]$)**:**

$\hat{y} \leftarrow \arg\max_{c_i} \left\{ f(\boldsymbol{x} | \hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i) \cdot P(c_i) \right\}$

return $\hat{y}$

Bayes Classifier: Iris Data

Consider the 2D Iris data (`sepal length` and `sepal width`). Class c_1, which corresponds to `iris-setosa` (circles), has $n_1 = 50$ points, whereas the other class c_2 (triangles) has $n_2 = 100$ points.

$$\hat{P}(c_1) = \frac{n_1}{n} = \frac{50}{150} = 0.33 \qquad \hat{P}(c_2) = \frac{n_2}{n} = \frac{100}{150} = 0.67$$

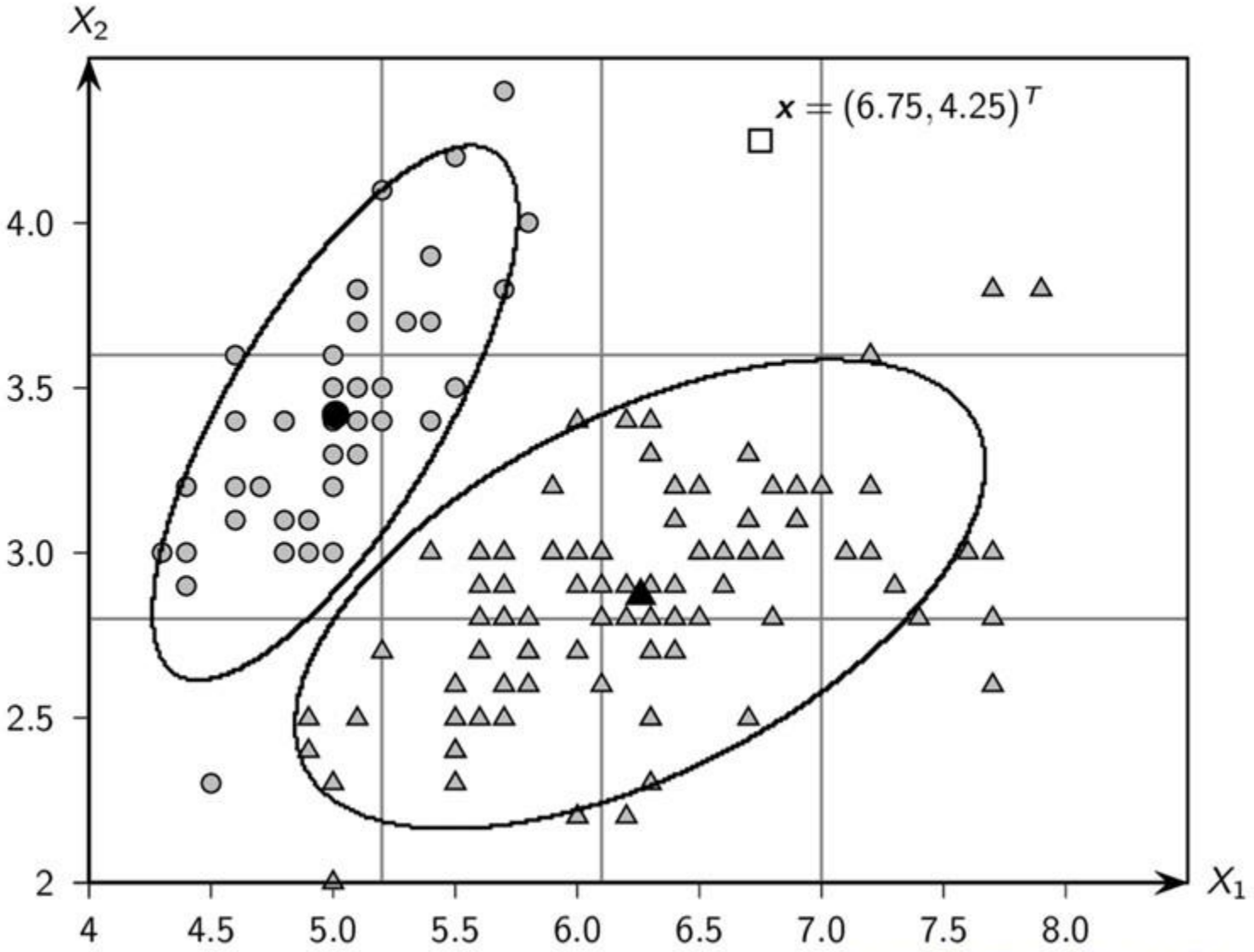
$$\hat{\boldsymbol{\mu}}_1 = \begin{pmatrix} 5.006 \\ 3.418 \end{pmatrix} \qquad \hat{\boldsymbol{\mu}}_2 = \begin{pmatrix} 6.262 \\ 2.872 \end{pmatrix}$$

$$\widehat{\boldsymbol{\Sigma}}_1 = \begin{pmatrix} 0.1218 & 0.0983 \\ 0.0983 & 0.1423 \end{pmatrix} \qquad \widehat{\boldsymbol{\Sigma}}_2 = \begin{pmatrix} 0.435 & 0.1209 \\ 0.1209 & 0.1096 \end{pmatrix}$$

Plot shows the contour or level curve (corresponding to 1% of the peak density) for the multivariate normal distribution modeling the probability density for both classes.

Bayes Classifier: Iris Data

X_1:sepal length versus X_2:sepal width



Bayes Classifier: Iris Data

Let $\boldsymbol{x} = (6.75, 4.25)^T$ be a test point (shown as white square). The posterior probabilities for c_1 and c_2 can be computed as:

$$\hat{P}(c_1|\boldsymbol{x}) \propto \hat{f}(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_1, \widehat{\boldsymbol{\Sigma}}_1)\hat{P}(c_1) = (4.951 \times 10^{-7}) \times 0.33 = 1.634 \times 10^{-7}$$

$$\hat{P}(c_2|\boldsymbol{x}) \propto \hat{f}(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_2, \widehat{\boldsymbol{\Sigma}}_2)\hat{P}(c_2) = (2.589 \times 10^{-5}) \times 0.67 = 1.735 \times 10^{-5}$$

Because $\hat{P}(c_2|\boldsymbol{x}) > \hat{P}(c_1|\boldsymbol{x})$ the class for $\boldsymbol{x}$ is predicted as $\hat{y} = c_2$.

Bayes Classifier

Categorical Attributes

Let X_j be a categorical attribute over the domain $dom(X_j) = \{a_{j1}, a_{j2}, \ldots, a_{jm_j}\}$. Each categorical attribute X_j is modeled as an m_j-dimensional multivariate Bernoulli random variable $\boldsymbol{X}_j$ that takes on m_j distinct vector values $\boldsymbol{e}_{j1}, \boldsymbol{e}_{j2}, \ldots, \boldsymbol{e}_{jm_j}$, where $\boldsymbol{e}_{jr}$ is the rth standard basis vector in $\mathbb{R}^{m_j}$ and corresponds to the rth value or symbol $a_{jr} \in dom(X_j)$.

The entire d-dimensional dataset is modeled as the vector random variable $\boldsymbol{X} = (\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_d)^T$. Let $d' = \sum_{j=1}^{d} m_j$; a categorical point $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T$ is therefore represented as the d'-dimensional binary vector

$$\boldsymbol{v} = \begin{pmatrix} \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{v}_d \end{pmatrix} = \begin{pmatrix} \boldsymbol{e}_{1r_1} \\ \vdots \\ \boldsymbol{e}_{dr_d} \end{pmatrix}$$

where $\boldsymbol{v}_j = \boldsymbol{e}_{jr_j}$ provided $x_j = a_{jr_j}$ is the r_jth value in the domain of X_j.

Bayes Classifier

Categorical Attributes

The probability of the categorical point $\boldsymbol{x}$ is obtained from the joint probability mass function (PMF) for the vector random variable $\boldsymbol{X}$:

$$P(\boldsymbol{x}|c_i) = f(\boldsymbol{v}|c_i) = f\left(\boldsymbol{X}_1 = \boldsymbol{e}_{1r_1}, \ldots, \boldsymbol{X}_d = \boldsymbol{e}_{dr_d} | \; c_i\right)$$

The joint PMF can be estimated directly from the data $\boldsymbol{D}_i$ for each class c_i as follows:

$$\hat{f}(\boldsymbol{v}|c_i) = \frac{n_i(\boldsymbol{v})}{n_i}$$

where $n_i(\boldsymbol{v})$ is the number of times the value $\boldsymbol{v}$ occurs in class c_i.
However, to avoid zero probabilities we add a *pseudo-count* of 1 for each value

$$\hat{f}(\boldsymbol{v}|c_i) = \frac{n_i(\boldsymbol{v}) + 1}{n_i + \prod_{j=1}^{d} m_j}$$

Discretized Iris Data

Assume that the `sepal length` and `sepal width` attributes in the Iris dataset have been discretized as shown below.

Bins	Domain
$[4.3, 5.2]$	Very Short (a_{11})
$(5.2, 6.1]$	Short (a_{12})
$(6.1, 7.0]$	Long (a_{13})
$(7.0, 7.9]$	Very Long (a_{14})

(a) Discretized `sepal length`

Bins	Domain
$[2.0, 2.8]$	Short (a_{21})
$(2.8, 3.6]$	Medium (a_{22})
$(3.6, 4.4]$	Long (a_{23})

(b) Discretized `sepal width`

We have $|dom(X_1)| = m_1 = 4$ and $|dom(X_2)| = m_2 = 3$.

Discretized Iris Data

Class-specific Empirical Joint Probability Mass Function

	Class: c_1	X_2			$\hat{f}_{X_1}$
		Short (e_{21})	Medium (e_{22})	Long (e_{23})	
X_1	Very Short (e_{11})	1/50	33/50	5/50	39/50
	Short (e_{12})	0	3/50	8/50	13/50
	Long (e_{13})	0	0	0	0
	Very Long (e_{14})	0	0	0	0
	$\hat{f}_{X_2}$	1/50	36/50	13/50	

	Class: c_2	X_2			$\hat{f}_{X_1}$
		Short (e_{21})	Medium (e_{22})	Long (e_{23})	
X_1	Very Short (e_{11})	6/100	0	0	6/100
	Short (e_{12})	24/100	15/100	0	39/100
	Long (e_{13})	13/100	30/100	0	43/100
	Very Long (e_{14})	3/100	7/100	2/100	12/100
	$\hat{f}_{X_2}$	46/100	52/100	2/100	

Discretized Iris Data: Test Case

Consider a test point $\boldsymbol{x} = (5.3, 3.0)^T$ corresponding to the categorical point (`Short, Medium`), which is represented as $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{e}_{12}^T & \boldsymbol{e}_{22}^T \end{pmatrix}^T$.

The prior probabilities of the classes are $\hat{P}(c_1) = 0.33$ and $\hat{P}(c_2) = 0.67$. The likelihood and posterior probability for each class is given as

$$\hat{P}(\boldsymbol{x}|c_1) = \hat{f}(\boldsymbol{v}|c_1) = 3/50 = 0.06$$
$$\hat{P}(\boldsymbol{x}|c_2) = \hat{f}(\boldsymbol{v}|c_2) = 15/100 = 0.15$$
$$\hat{P}(c_1|\boldsymbol{x}) \propto 0.06 \times 0.33 = 0.0198$$
$$\hat{P}(c_2|\boldsymbol{x}) \propto 0.15 \times 0.67 = 0.1005$$

In this case the predicted class is $\hat{y} = c_2$.

Discretized Iris Data: Test Case with Pseudo-counts

The test point $\boldsymbol{x} = (6.75, 4.25)^T$ corresponds to the categorical point (Long, Long), and it is represented as $\boldsymbol{v} = \begin{pmatrix} \boldsymbol{e}_{13}^T & \boldsymbol{e}_{23}^T \end{pmatrix}^T$.

Unfortunately the probability mass at $\boldsymbol{v}$ is zero for both classes. We adjust the PMF via pseudo-counts noting that the number of possible values are $m_1 \times m_2 = 4 \times 3 = 12$.

The likelihood and prior probability can then be computed as

$$\hat{P}(\boldsymbol{x}|c_1) = \hat{f}(\boldsymbol{v}|c_1) = \frac{0+1}{50+12} = 1.61 \times 10^{-2}$$

$$\hat{P}(\boldsymbol{x}|c_2) = \hat{f}(\boldsymbol{v}|c_2) = \frac{0+1}{100+12} = 8.93 \times 10^{-3}$$

$$\hat{P}(c_1|\boldsymbol{x}) \propto (1.61 \times 10^{-2}) \times 0.33 = 5.32 \times 10^{-3}$$

$$\hat{P}(c_2|\boldsymbol{x}) \propto (8.93 \times 10^{-3}) \times 0.67 = 5.98 \times 10^{-3}$$

Thus, the predicted class is $\hat{y} = c_2$.

Bayes Classifier: Challenges

The main problem with the Bayes classifier is the lack of enough data to reliably estimate the joint probability density or mass function, especially for high-dimensional data.

For numeric attributes we have to estimate $O(d^2)$ covariances, and as the dimensionality increases, this requires us to estimate too many parameters.

For categorical attributes we have to estimate the joint probability for all the possible values of $\boldsymbol{v}$, given as $\prod_j |\text{dom}(X_j)|$. Even if each categorical attribute has only two values, we would need to estimate the probability for 2^d values. However, because there can be at most n distinct values for $\boldsymbol{v}$, most of the counts will be zero.

Naive Bayes classifier addresses these concerns.

Naive Bayes Classifier

Numeric Attributes

The naive Bayes approach makes the simple assumption that all the attributes are independent, which implies that the likelihood can be decomposed into a product of dimension-wise probabilities:

$$P(\boldsymbol{x}|c_i) = P(x_1, x_2, \ldots, x_d|c_i) = \prod_{j=1}^{d} P(x_j|c_i)$$

The likelihood for class c_i, for dimension X_j, is given as

$$P(x_j|c_i) \propto f(x_j|\hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) = \frac{1}{\sqrt{2\pi}\hat{\sigma}_{ij}} \exp\left\{-\frac{(x_j - \hat{\mu}_{ij})^2}{2\hat{\sigma}_{ij}^2}\right\}$$

where $\hat{\mu}_{ij}$ and $\hat{\sigma}_{ij}^2$ denote the estimated mean and variance for attribute X_j, for class c_i.

Naive Bayes Classifier

Numeric Attributes

The naive assumption corresponds to setting all the covariances to zero in $\widehat{\Sigma}_i$, that is,

$$\Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{i2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & \sigma_{id}^2 \end{pmatrix}$$

The naive Bayes classifier thus uses the sample mean $\hat{\boldsymbol{\mu}}_i = (\hat{\mu}_{i1}, \dots, \hat{\mu}_{id})^T$ and a *diagonal* sample covariance matrix $\widehat{\Sigma}_i = diag(\sigma_{i1}^2, \dots, \sigma_{id}^2)$ for each class c_i. In total $2d$ parameters have to be estimated, corresponding to the sample mean and sample variance for each dimension X_j.

Naive Bayes Algorithm

NaiveBayes ($\boldsymbol{D} = \{(\boldsymbol{x}_j, y_j)\}_{j=1}^{n}$):

for $i = 1, \ldots, k$ **do**
- $\boldsymbol{D}_i \leftarrow \{\boldsymbol{x}_j^T \mid y_j = c_i, j = 1, \ldots, n\}$ // class-specific subsets
- $n_i \leftarrow |\boldsymbol{D}_i|$ // cardinality
- $\hat{P}(c_i) \leftarrow n_i / n$ // prior probability
- $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in \boldsymbol{D}_i} \boldsymbol{x}_j$ // mean
- $\overline{\boldsymbol{D}}_i = \boldsymbol{D}_i - 1 \cdot \hat{\boldsymbol{\mu}}_i^T$ // centered data for class c_i
- **for** $j = 1, .., d$ **do** // class-specific var for jth attribute
  - $\hat{\sigma}_{ij}^2 \leftarrow \frac{1}{n_i} (\overline{X}_j^i)^T (\overline{X}_j^i)$ // variance
- $\hat{\boldsymbol{\sigma}}_i \leftarrow \left(\hat{\sigma}_{i1}^2, \ldots, \hat{\sigma}_{id}^2\right)^T$ // class-specific attribute variances

return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\sigma}}_i$ for all $i = 1, \ldots, k$

Testing (x **and** $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\sigma}}_i$, **for all** $i \in [1, k]$):

$$\hat{y} \leftarrow \arg\max_{c_i} \left\{ \hat{P}(c_i) \prod_{j=1}^{d} f(x_j | \hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) \right\}$$

return $\hat{y}$

Iris Data: Naive Bayes

Consider 2D Iris Data consisting of `sepal length` and `sepal width`. $\hat{P}(c_i)$ and $\hat{\boldsymbol{\mu}}_i$ remain unchanged. $\widehat{\Sigma}$ are assumed to be diagonal:

$$\widehat{\Sigma}_1 = \begin{pmatrix} 0.1218 & 0 \\ 0 & 0.1423 \end{pmatrix} \qquad \widehat{\Sigma}_2 = \begin{pmatrix} 0.435 & 0 \\ 0 & 0.1096 \end{pmatrix}$$

Figure shows the contour or level curve (corresponding to 1% of the peak density) of the multivariate normal distribution for both classes. The diagonal assumption leads to contours that are axis-parallel ellipses.

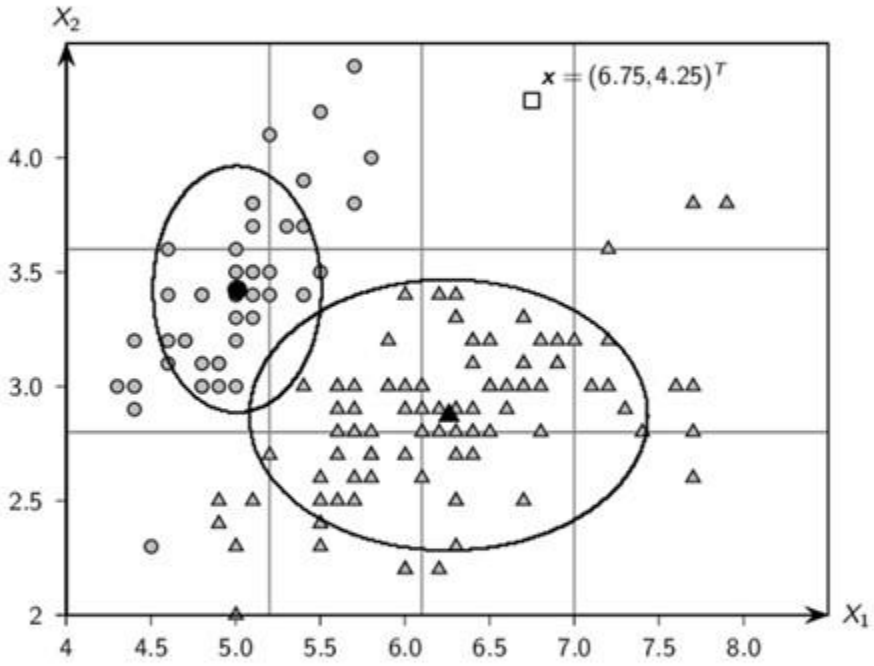
For the test point $\boldsymbol{x} = (6.75, 4.25)^T$, the posterior probabilities for c_1 and c_2 are as follows:

$$\hat{P}(c_1|\boldsymbol{x}) \propto \hat{f}(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_1, \widehat{\Sigma}_1)\hat{P}(c_1) = (4.014 \times 10^{-7}) \times 0.33 = 1.325 \times 10^{-7}$$
$$\hat{P}(c_2|\boldsymbol{x}) \propto \hat{f}(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_2, \widehat{\Sigma}_2)\hat{P}(c_2) = (9.585 \times 10^{-5}) \times 0.67 = 6.422 \times 10^{-5}$$

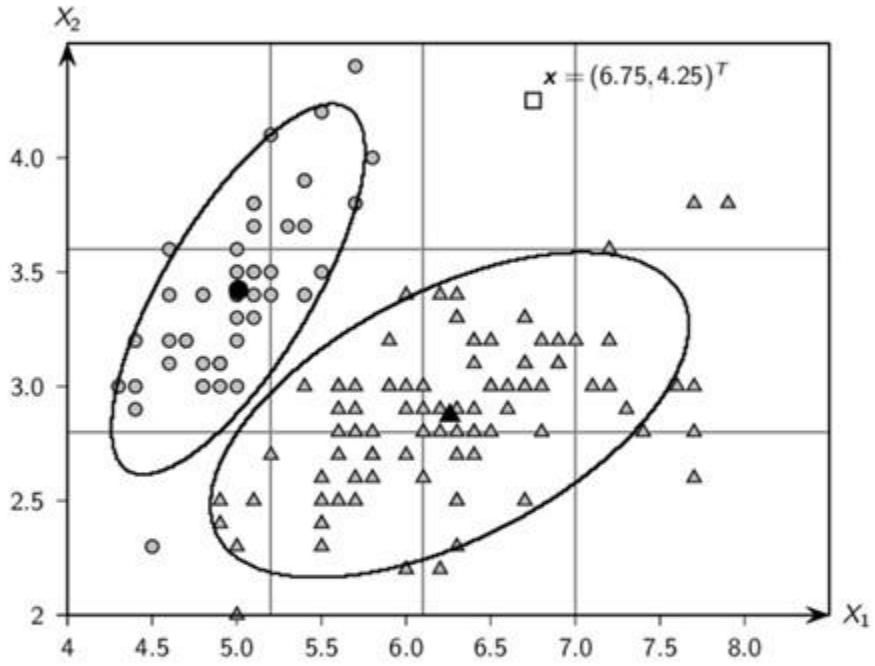
Because $\hat{P}(c_2|\boldsymbol{x}) > \hat{P}(c_1|\boldsymbol{x})$, it predicted $\hat{y} = c_2$ to $\boldsymbol{x}$.

Naive Bayes versus Full Bayes Classifier

X_1:sepal length versus X_2:sepal width



(a) Naive Bayes



(b) Full Bayes

Naive Bayes

Categorical Attributes

The independence assumption leads to a simplification:

$$P(\boldsymbol{x}|c_i) = \prod_{j=1}^{d} P(x_j|c_i) = \prod_{j=1}^{d} f\left(\boldsymbol{X}_j = \boldsymbol{e}_{jr_j} \mid c_i\right)$$

where $f(\boldsymbol{X}_j = \boldsymbol{e}_{jr_j}|c_i)$ is the probability mass function for $\boldsymbol{X}_j$, estimated as:

$$\hat{f}(\boldsymbol{v}_j|c_i) = \frac{n_i(\boldsymbol{v}_j)}{n_i}$$

where $n_i(\boldsymbol{v}_j)$ is the observed frequency of the value $\boldsymbol{v}_j = \boldsymbol{e}_j r_j$ corresponding to the r_jth categorical value a_{jr_j} for the attribute X_j for class c_i.
If the count is zero, we can use the pseudo-count method to obtain a prior probability. The adjusted estimates with pseudo-counts are given as

$$\hat{f}(\boldsymbol{v}_j|c_i) = \frac{n_i(\boldsymbol{v}_j) + 1}{n_i + m_j}$$

where $m_j = |dom(X_j)|$.

Discretized Iris Data: Naive Bayes

Class-specific Empirical Joint Probability Mass Function

	Class: c_1	X_2			$\hat{f}_{X_1}$
		Short (e_{21})	Medium (e_{22})	Long (e_{23})	
X_1	Very Short (e_{11})	1/50	33/50	5/50	39/50
	Short (e_{12})	0	3/50	8/50	13/50
	Long (e_{13})	0	0	0	0
	Very Long (e_{14})	0	0	0	0
	$\hat{f}_{X_2}$	1/50	36/50	13/50	

	Class: c_2	X_2			$\hat{f}_{X_1}$
		Short (e_{21})	Medium (e_{22})	Long (e_{23})	
X_1	Very Short (e_{11})	6/100	0	0	6/100
	Short (e_{12})	24/100	15/100	0	39/100
	Long (e_{13})	13/100	30/100	0	43/100
	Very Long (e_{14})	3/100	7/100	2/100	12/100
	$\hat{f}_{X_2}$	46/100	52/100	2/100	

Discretized Iris Data: Naive Bayes

The test point $\boldsymbol{x} = (6.75, 4.25)$, corresponding to (Long, Long) or $\boldsymbol{v} = (\boldsymbol{e}_{13}, \boldsymbol{e}_{23})$, is classified as follows:

$$\hat{P}(\boldsymbol{v}|c_1) = \hat{P}(\boldsymbol{e}_{13}|c_1) \cdot \hat{P}(\boldsymbol{e}_{23}|c_1) = \left(\frac{0+1}{50+4}\right) \cdot \left(\frac{13}{50}\right) = 4.81 \times 10^{-3}$$

$$\hat{P}(\boldsymbol{v}|c_2) = \hat{P}(\boldsymbol{e}_{13}|c_2) \cdot \hat{P}(\boldsymbol{e}_{23}|c_2) = \left(\frac{43}{100}\right) \cdot \left(\frac{2}{100}\right) = 8.60 \times 10^{-3}$$

$$\hat{P}(c_1|\boldsymbol{v}) \propto (4.81 \times 10^{-3}) \times 0.33 = 1.59 \times 10^{-3}$$

$$\hat{P}(c_2|\boldsymbol{v}) \propto (8.6 \times 10^{-3}) \times 0.67 = 5.76 \times 10^{-3}$$

Thus, the predicted class is $\hat{y} = c_2$.

Nonparametric Approach

K Nearest Neighbors Classifier

We consider a non-parametric approach for likelihood estimation using the nearest neighbors density estimation.

Let $\boldsymbol{D}$ be a training dataset comprising n points $\boldsymbol{x}_i \in \mathbb{R}^d$, and let $\boldsymbol{D}_i$ denote the subset of points in $\boldsymbol{D}$ that are labeled with class c_i, with $n_i = |\boldsymbol{D}_i|$.

Given a test point $\boldsymbol{x} \in \mathbb{R}^d$, and K, the number of neighbors to consider, let r denote the distance from $\boldsymbol{x}$ to its Kth nearest neighbor in $\boldsymbol{D}$.

Consider the d-dimensional hyperball of radius r around the test point $\boldsymbol{x}$, defined as

$$B_d(\boldsymbol{x}, r) = \{\boldsymbol{x}_i \in \boldsymbol{D} \mid \delta(\boldsymbol{x}, \boldsymbol{x}_i) \leq r\}$$

Here $\delta(\boldsymbol{x}, \boldsymbol{x}_i)$ is the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$, which is usually assumed to be the Euclidean distance, i.e., $\delta(\boldsymbol{x}, \boldsymbol{x}_i) = \|\boldsymbol{x} - \boldsymbol{x}_i\|_2$. We assume that $|B_d(\boldsymbol{x}, r)| = K$.

Nonparametric Approach

K Nearest Neighbors Classifier

Let K_i denote the number of points among the K nearest neighbors of $\boldsymbol{x}$ that are labeled with class c_i, that is

$$K_i = \{\boldsymbol{x}_j \in B_d(\boldsymbol{x}, r) \mid y_j = c_i\}$$

The class conditional probability density at $\boldsymbol{x}$ can be estimated as the fraction of points from class c_i that lie within the hyperball divided by its volume, that is

$$\hat{f}(\boldsymbol{x}|c_i) = \frac{K_i/n_i}{V} = \frac{K_i}{n_i V}$$

where $V = \text{vol}(B_d(\boldsymbol{x}, r))$ is the volume of the d-dimensional hyperball.
The posterior probability $P(c_i|\boldsymbol{x})$ can be estimated as

$$P(c_i|\boldsymbol{x}) = \frac{\hat{f}(\boldsymbol{x}|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{f}(\boldsymbol{x}|c_j)\hat{P}(c_j)}$$

However, because $\hat{P}(c_i) = \frac{n_i}{n}$, we have

$$\hat{f}(\boldsymbol{x}|c_i)\hat{P}(c_i) = \frac{K_i}{n_i V} \cdot \frac{n_i}{n} = \frac{K_i}{nV}$$

The posterior probability is given as

$$P(c_i|\boldsymbol{x}) = \frac{\frac{K_i}{nV}}{\sum_{j=1}^{k} \frac{K_j}{nV}} = \frac{K_i}{K}$$

Finally, the predicted class for $\boldsymbol{x}$ is

$$\hat{y} = \arg\max_{c_i} \{P(c_i|\boldsymbol{x})\} = \arg\max_{c_i} \left\{ \frac{K_i}{K} \right\} = \arg\max_{c_i} \{K_i\}$$

Because K is fixed, the KNN classifier predicts the class of $\boldsymbol{x}$ as the majority class among its K nearest neighbors.

Iris Data

K Nearest Neighbors Classifier

Consider the 2D Iris dataset. Class c_1 (circles) has $n_1 = 50$ points and c_2 (triangles) has $n_2 = 100$ points.

We use KNN ($K = 5$) to classify the test point $\boldsymbol{x} = (6.75, 4.25)^T$.

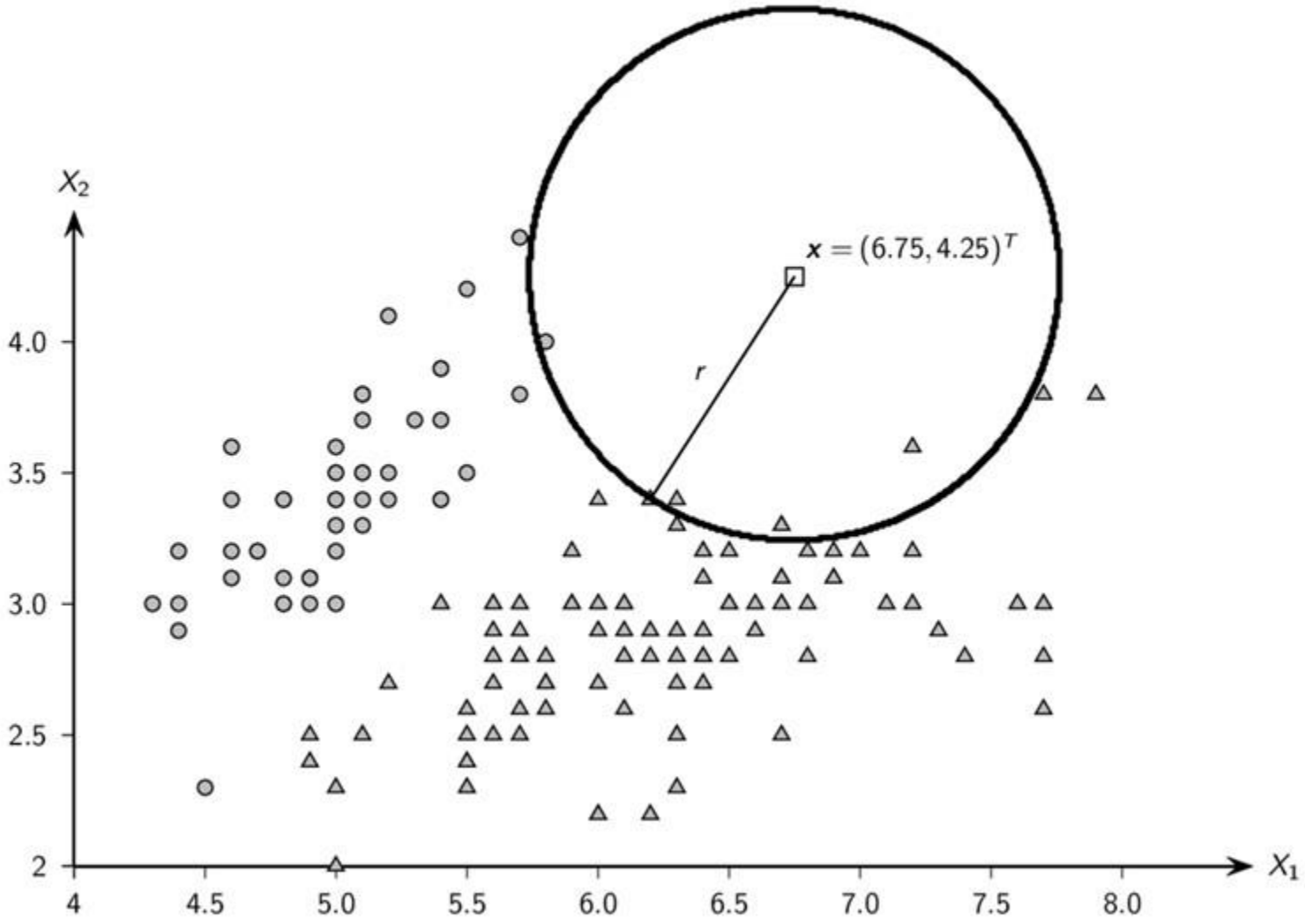
The distance from $\boldsymbol{x}$ to its 5th nearest neighbor, namely $(6.2, 3.4)^T$, is given as $r = \sqrt{1.025} = 1.012$.

The enclosing ball or circle of radius r encompasses $K_1 = 1$ point from c_1 and $K_2 = 4$ points from c_2.

Therefore, the predicted class for $\boldsymbol{x}$ is $\hat{y} = c_2$.

Iris Data

K Nearest Neighbors Classifier



Data mining and Machine learning

Part 19. Decision Tree Classifier

Decision Tree Classifier

Let the training dataset $\boldsymbol{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$ consist of n points in a d-dimensional space, with y_i being the class label for point $\boldsymbol{x}_i$.

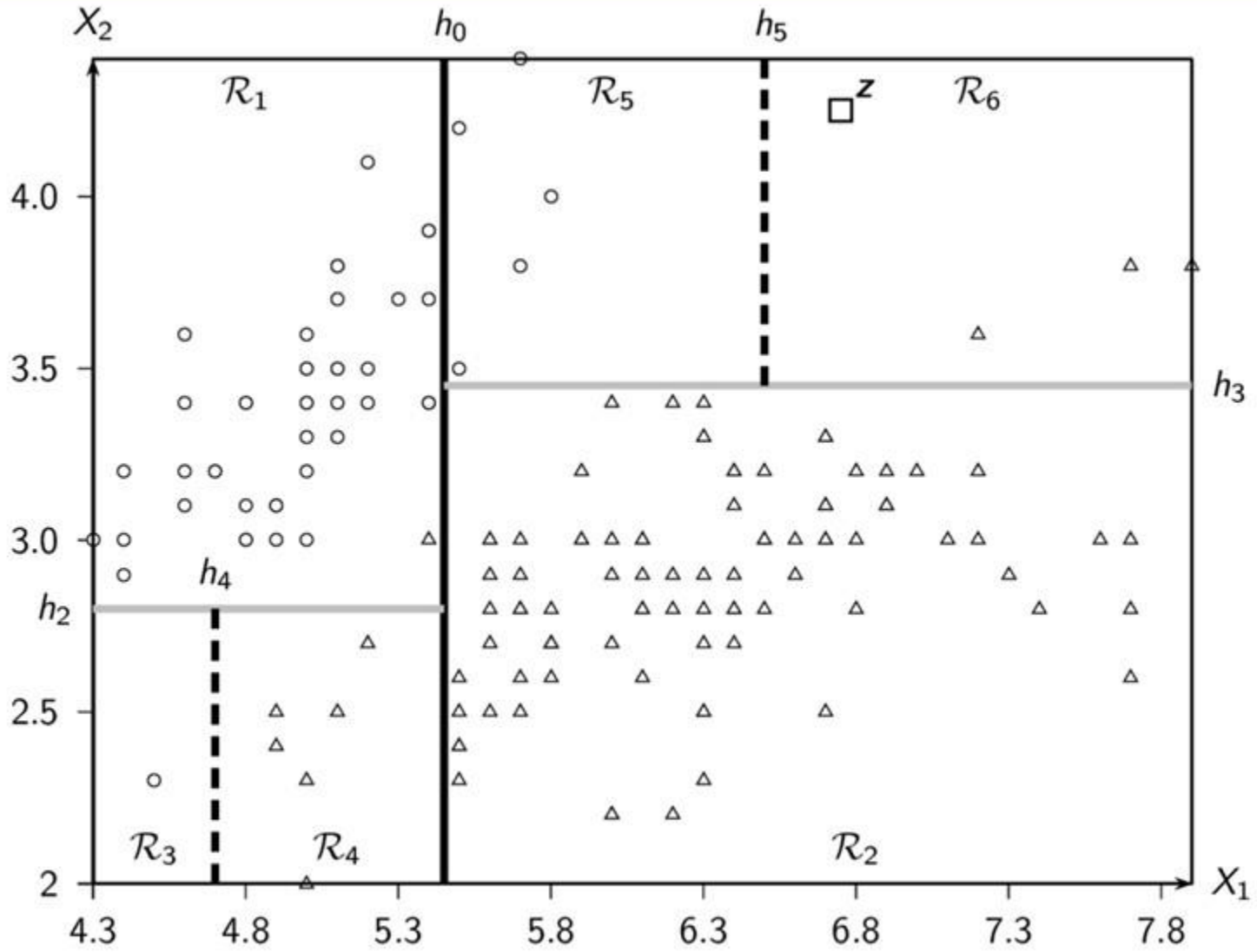
A decision tree classifier is a recursive, partition-based tree model that predicts the class $\hat{y}_i$ for each point $\boldsymbol{x}_i$.

Let $\mathcal{R}$ denote the data space that encompasses the set of input points $\boldsymbol{D}$. A decision tree uses an axis-parallel hyperplane to split the data space $\mathcal{R}$ into two resulting half-spaces or regions, say $\mathcal{R}_1$ and $\mathcal{R}_2$, which also induces a partition of the input points into $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$, respectively.

Each of these regions is recursively split via axis-parallel hyperplanes until most of the points belong to the same class.

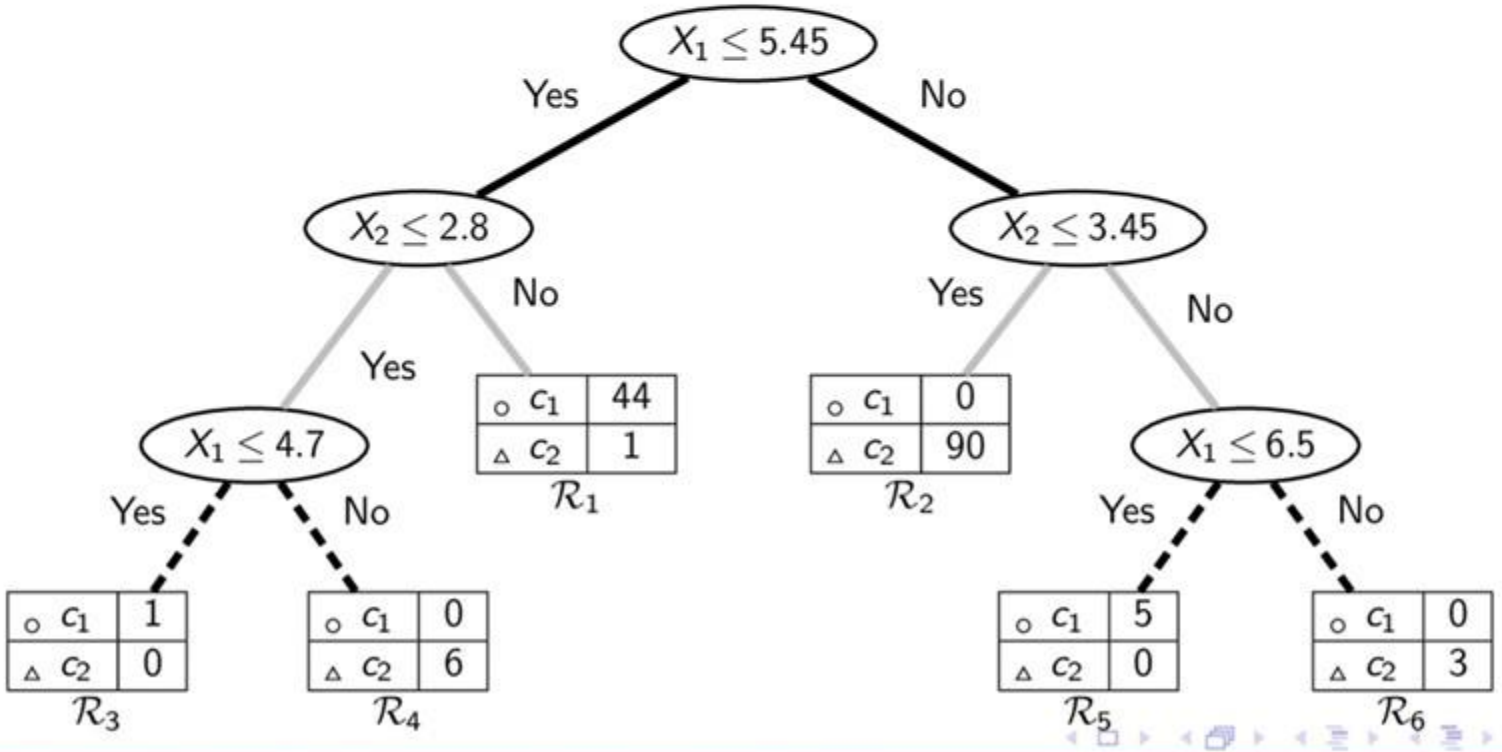
To classify a new *test* point we have to recursively evaluate which half-space it belongs to until we reach a leaf node in the decision tree, at which point we predict its class as the label of the leaf.

Decision Tree: Recursive Splits



Decision Tree

A decision tree consists of internal nodes that represent the decisions corresponding to the hyperplanes or split points (i.e., which half-space a given point lies in), and leaf nodes that represent regions or partitions of the data space, which are labeled with the majority class. A region is characterized by the subset of data points that lie in that region.



Decision Trees: Axis-Parallel Hyperplanes

A hyperplane $h(\boldsymbol{x})$ is defined as the set of all points $\boldsymbol{x}$ that satisfy the following equation

$$h(\boldsymbol{x}): \boldsymbol{w}^T \boldsymbol{x} + b = 0$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is a *weight vector* that is normal to the hyperplane, and b is the offset of the hyperplane from the origin.

A decision tree considers only *axis-parallel hyperplanes*, that is, the weight vector must be parallel to one of the original dimensions or axes X_j:

$$h(\boldsymbol{x}): x_j + b = 0$$

where the choice of the offset b yields different hyperplanes along dimension X_j.

Decision Trees: Split Points

A hyperplane specifies a decision or *split point* because it splits the data space $\mathcal{R}$ into two half-spaces. All points $\boldsymbol{x}$ such that $h(\boldsymbol{x}) \leq 0$ are on the hyperplane or to one side of the hyperplane, whereas all points such that $h(\boldsymbol{x}) > 0$ are on the other side.

The split point is written as $h(\boldsymbol{x}) \leq 0$, i.e.

$$X_j \leq v$$

where $v = -b$ is some value in the domain of attribute X_j.

The decision or split point $X_j \leq v$ thus splits the input data space $\mathcal{R}$ into two regions $\mathcal{R}_Y$ and $\mathcal{R}_N$, which denote the set of *all possible points* that satisfy the decision and those that do not.

Categorical Attributes: For a categorical attribute X_j, the split points or decisions are of the $X_j \in V$, where $V \subset dom(X_j)$, and $dom(X_j)$ denotes the domain for X_j.

Decision Trees: Data Partition and Purity

Each split of $\mathcal{R}$ into $\mathcal{R}_Y$ and $\mathcal{R}_N$ also induces a binary partition of the corresponding input data points $\boldsymbol{D}$. A split point of the form $X_j \leq v$ induces the data partition

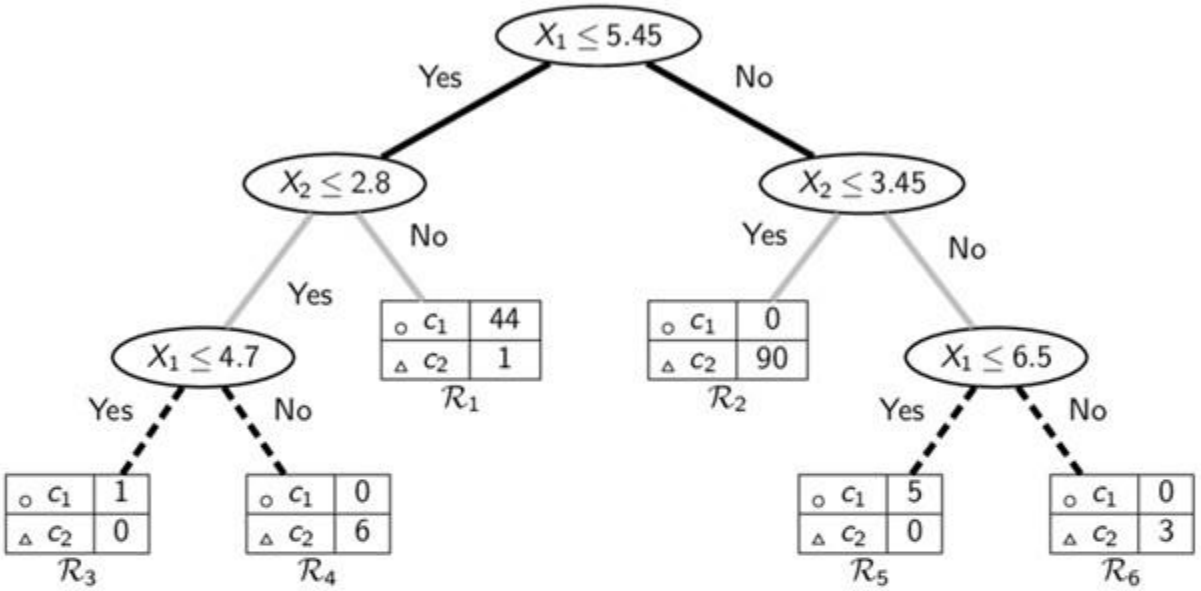
$$\boldsymbol{D}_Y = \{\boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{D}, x_j \leq v\}$$
$$\boldsymbol{D}_N = \{\boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{D}, x_j > v\}$$

The purity of a region $\mathcal{R}_j$ is the fraction of points with the majority label in $\boldsymbol{D}_j$, that is,

$$purity(\boldsymbol{D}_j) = \max_i \left\{ \frac{n_{ji}}{n_j} \right\}$$

where $n_j = |\boldsymbol{D}_j|$ is the total number of data points in the region $\mathcal{R}_j$, and n_{ji} is the number of points in $\boldsymbol{D}_j$ with class label c_i.

Decision Trees to Rules



A tree is a set of decision rules; each comprising the decisions on the path to a leaf:

$\mathcal{R}_3$: If $X_1 \leq 5.45$ and $X_2 \leq 2.8$ and $X_1 \leq 4.7$, then class is c_1, or

$\mathcal{R}_4$: If $X_1 \leq 5.45$ and $X_2 \leq 2.8$ and $X_1 > 4.7$, then class is c_2, or

$\mathcal{R}_1$: If $X_1 \leq 5.45$ and $X_2 > 2.8$, then class is c_1, or

$\mathcal{R}_2$: If $X_1 > 5.45$ and $X_2 \leq 3.45$, then class is c_2, or

$\mathcal{R}_5$: If $X_1 > 5.45$ and $X_2 > 3.45$ and $X_1 \leq 6.5$, then class is c_1, or

$\mathcal{R}_6$: If $X_1 > 5.45$ and $X_2 > 3.45$ and $X_1 > 6.5$, then class is c_2

Decision Tree Algorithm

The method takes as input a training dataset $\boldsymbol{D}$, and two parameters η and π, where η is the leaf size and π the leaf purity threshold.

Different split points are evaluated for each attribute in $\boldsymbol{D}$. Numeric decisions are of the form $X_j \leq v$ for some value v in the value range for attribute X_j, and categorical decisions are of the form $X_j \in V$ for some subset of values in the domain of X_j.

The best split point is chosen to partition the data into two subsets, $\boldsymbol{D}_Y$ and $\boldsymbol{D}_N$, where $\boldsymbol{D}_Y$ corresponds to all points $\boldsymbol{x} \in \boldsymbol{D}$ that satisfy the split decision, and $\boldsymbol{D}_N$ corresponds to all points that do not satisfy the split decision. The decision tree method is then called recursively on $\boldsymbol{D}_Y$ and $\boldsymbol{D}_N$.

We stop the process if the leaf size drops below η or if the purity is at least π.

Decision Tree Algorithm

DecisionTree $(\boldsymbol{D}, \eta, \pi)$**:**

$n \leftarrow |\boldsymbol{D}|$ // partition size
$n_i \leftarrow |\{\boldsymbol{x}_j | \boldsymbol{x}_j \in \boldsymbol{D}, y_j = c_i\}|$ // size of class c_i
$purity(\boldsymbol{D}) \leftarrow \max_i \left\{ \frac{n_i}{n} \right\}$
if $n \leq \eta$ *or* $purity(\boldsymbol{D}) \geq \pi$ **then** // stopping condition
    $c^* \leftarrow \arg\max_{c_i} \left\{ \frac{n_i}{n} \right\}$ // majority class
    create leaf node, and label it with class c^*
    return
$(split\ point^*, score^*) \leftarrow (\emptyset, 0)$ // initialize best split point
foreach *(attribute* X_j*)* **do**
    if *(*X_j *is numeric)* **then**
        $(v, score) \leftarrow$ Evaluate-Numeric-Attribute$(\boldsymbol{D}, X_j)$
        if $score > score^*$ **then** $(split\ point^*, score^*) \leftarrow (X_j \leq v, score)$
    else if *(*X_j *is categorical)* **then**
        $(V, score) \leftarrow$ Evaluate-Categorical-Attribute$(\boldsymbol{D}, X_j)$
        if $score > score^*$ **then** $(split\ point^*, score^*) \leftarrow (X_j \in V, score)$
$\boldsymbol{D}_Y \leftarrow \{\boldsymbol{x} \in \boldsymbol{D} \mid \boldsymbol{x}$ satisfies $split\ point^*\}$
$\boldsymbol{D}_N \leftarrow \{\boldsymbol{x} \in \boldsymbol{D} \mid \boldsymbol{x}$ does not satisfy $split\ point^*\}$
create internal node $split\ point^*$, with two child nodes, $\boldsymbol{D}_Y$ and $\boldsymbol{D}_N$
DecisionTree$(\boldsymbol{D}_Y, \eta, \pi)$; DecisionTree$(\boldsymbol{D}_N, \eta, \pi)$

Split Point Evaluation Measures: Entropy

Intuitively, we want to select a split point that gives the best separation or discrimination between the different class labels.

Entropy measures the amount of disorder or uncertainty in a system. A partition has lower entropy (or low disorder) if it is relatively pure, that is, if most of the points have the same label. On the other hand, a partition has higher entropy (or more disorder) if the class labels are mixed, and there is no majority class as such.

The entropy of a set of labeled points $\boldsymbol{D}$ is defined as follows:

$$H(\boldsymbol{D}) = -\sum_{i=1}^{k} P(c_i|\boldsymbol{D}) \log_2 P(c_i|\boldsymbol{D})$$

where $P(c_i|\boldsymbol{D})$ is the probability of class c_i in $\boldsymbol{D}$, and k is the number of classes.

If a region is pure, that is, has points from the same class, then the entropy is zero. On the other hand, if the classes are all mixed up, and each appears with equal probability $P(c_i|\boldsymbol{D}) = \frac{1}{k}$, then the entropy has the highest value, $H(\boldsymbol{D}) = \log_2 k$.

Split Point Evaluation Measures: Entropy

Define the *split entropy* as the weighted entropy of each of the resulting partitions

$$H(\boldsymbol{D}_Y, \boldsymbol{D}_N) = \frac{n_Y}{n} H(\boldsymbol{D}_Y) + \frac{n_N}{n} H(\boldsymbol{D}_N)$$

where $n = |\boldsymbol{D}|$ is the number of points in $\boldsymbol{D}$, and $n_Y = |\boldsymbol{D}_Y|$ and $n_N = |\boldsymbol{D}_N|$ are the number of points in $\boldsymbol{D}_Y$ and $\boldsymbol{D}_N$.

Define the *information gain* for a split point as

$$Gain(\boldsymbol{D}, \boldsymbol{D}_Y, \boldsymbol{D}_N) = H(\boldsymbol{D}) - H(\boldsymbol{D}_Y, \boldsymbol{D}_N)$$

The higher the information gain, the more the reduction in entropy, and the better the split point.

We score each split point and choose the one that gives the highest information gain.

Split Point Evaluation Measures

Gini Index and CART Measure

Gini Index: The Gini index is defined as follows:

$$G(\boldsymbol{D}) = 1 - \sum_{i=1}^{k} P(c_i|\boldsymbol{D})^2$$

If the partition is pure, then Gini index is 0. The weighted Gini index is:

$$G(\boldsymbol{D}_Y, \boldsymbol{D}_N) = \frac{n_Y}{n} G(\boldsymbol{D}_Y) + \frac{n_N}{n} G(\boldsymbol{D}_N)$$

The lower the Gini index value, the better the split point.

CART: The CART measure is

$$CART(\boldsymbol{D}_Y, \boldsymbol{D}_N) = 2\frac{n_Y}{n}\frac{n_N}{n} \sum_{i=1}^{k} \left| P(c_i|\boldsymbol{D}_Y) - P(c_i|\boldsymbol{D}_N) \right|$$

CART maximizes the difference between the class PMF for the two partitions; the higher the CART measure, the better the split point.

Evaluating Split Points: Numeric Attributes

All of the split point evaluation measures depend on the class probability mass function (PMF) for $\boldsymbol{D}$, namely, $P(c_i|\boldsymbol{D})$, and the class PMFs for the resulting partitions $\boldsymbol{D}_Y$ and $\boldsymbol{D}_N$, namely $P(c_i|\boldsymbol{D}_Y)$ and $P(c_i|\boldsymbol{D}_N)$.

We have to evaluate split points of the form $X \leq v$. We consider only the midpoints between two successive distinct values for X in the sample $\boldsymbol{D}$. Let $\{v_1, \ldots, v_m\}$ denote the set of all such midpoints, such that $v_1 < v_2 < \cdots < v_m$.

For each split point $X \leq v$, we have to estimate the class PMFs:

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \hat{P}(c_i|X \leq v)$$
$$\hat{P}(c_i|\boldsymbol{D}_N) = \hat{P}(c_i|X > v)$$

Using Bayes theorem, we have

$$\hat{P}(c_i|X \leq v) = \frac{\hat{P}(X \leq v|c_i)\hat{P}(c_i)}{\hat{P}(X \leq v)} = \frac{\hat{P}(X \leq v|c_i)\hat{P}(c_i)}{\sum_{j=1}^{k} \hat{P}(X \leq v|c_j)\hat{P}(c_j)}$$

Thus we have to estimate the prior probability and likelihood for each class in each partition.

Evaluating Split Points: Numeric Attributes

The prior probability for each class in $\boldsymbol{D}$ can be estimated as

$$\hat{P}(c_i) = \frac{1}{n} \sum_{j=1}^{n} I(y_j = c_i) = \frac{n_i}{n}$$

where y_j is the class for point $\boldsymbol{x}_j$, $n = |\boldsymbol{D}|$ is the total number of points, and n_i is the number of points in $\boldsymbol{D}$ with class c_i.

Define N_{vi} as the number of points $x_j \leq v$ with class c_i, where x_j is the value of data point $\boldsymbol{x}_j$ for the attribute X, given as

$$N_{vi} = \sum_{j=1}^{n} I(x_j \leq v \text{ and } y_j = c_i)$$

Evaluating Split Points: Numeric Attributes

We can estimate $\hat{P}(X \leq v|c_i)$ and $\hat{P}(X > v|c_i)$ as follows:

$$\hat{P}(X \leq v|c_i) = \frac{N_{vi}}{n_i}$$

$$\hat{P}(X > v|c_i) = 1 - \hat{P}(X \leq v|c_i) = \frac{n_i - N_{vi}}{n_i}$$

Finally, we have

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \hat{P}(c_i|X \leq v) = \frac{N_{vi}}{\sum_{j=1}^{k} N_{vj}}$$

$$\hat{P}(c_i|\boldsymbol{D}_N) = \hat{P}(c_i|X > v) = \frac{n_i - N_{vi}}{\sum_{j=1}^{k}(n_j - N_{vj})}$$

The total cost of evaluating a numeric attribute is $O(n \log n + nk)$, where k is the number of classes, and n is the number of points.

Algorithm Evaluate-Numeric-Attribute

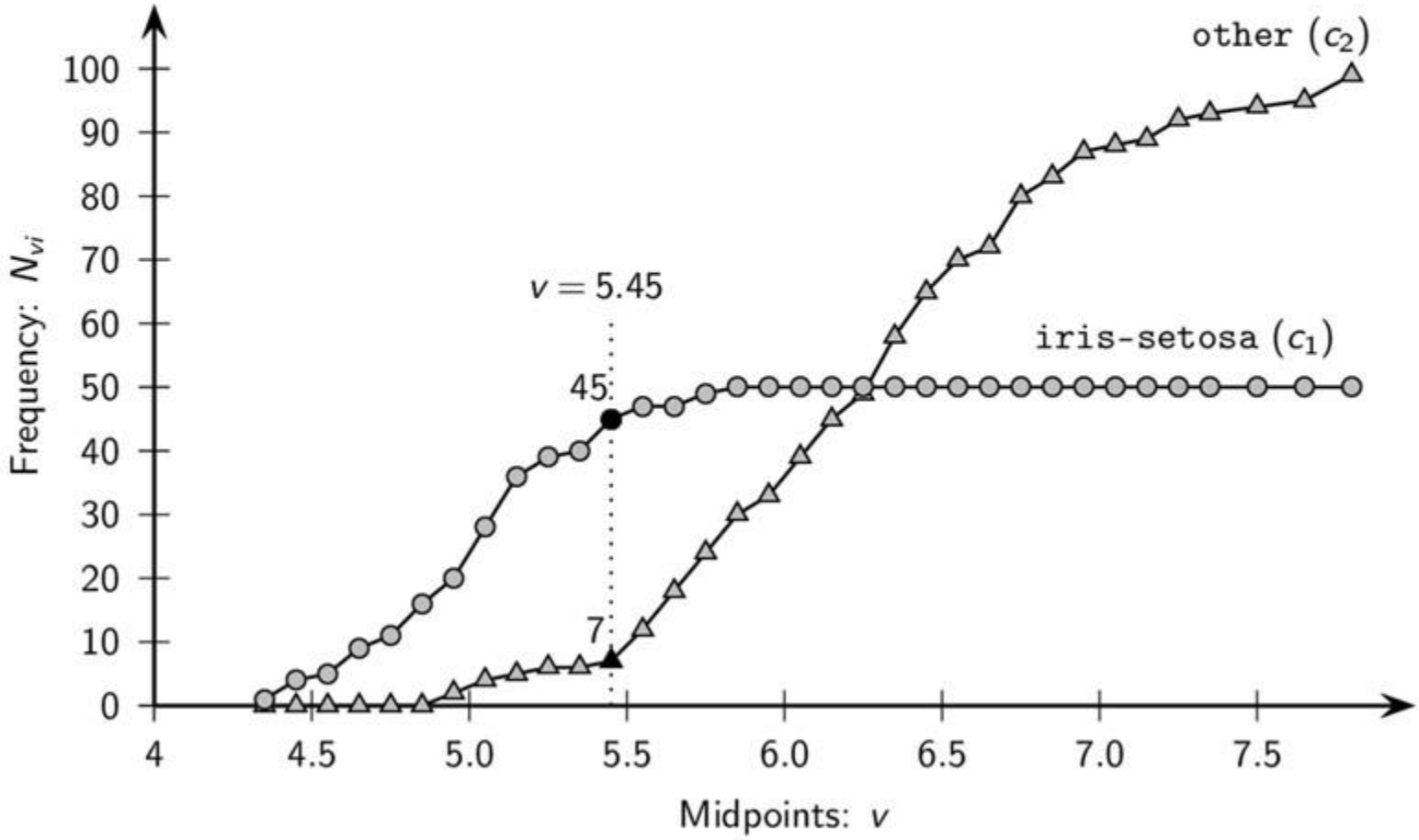
Evaluate-Numeric-Attribute $(\boldsymbol{D}, X)$:

1 sort $\boldsymbol{D}$ on attribute X, so that $x_j \leq x_{j+1}, \forall j = 1, \ldots, n-1$
2 $\mathcal{M} \leftarrow \emptyset$ // set of midpoints
3 **for** $i = 1, \ldots, k$ **do** $n_i \leftarrow 0$
4 **for** $j = 1, \ldots, n-1$ **do**
5 **if** $y_j = c_i$ **then** $n_i \leftarrow n_i + 1$
6 **if** $x_{j+1} \neq x_j$ **then**
7 $v \leftarrow (x_{j+1} + x_j)/2$; $\mathcal{M} \leftarrow \mathcal{M} \cup \{v\}$ // midpoints
8 **for** $i = 1, \ldots, k$ **do**
9 $N_{vi} \leftarrow n_i$ // Number of points such that $x_j \leq v$ and $y_j = c_i$
10 **if** $y_n = c_i$ **then** $n_i \leftarrow n_i + 1$
11 $v^* \leftarrow \emptyset$; $score^* \leftarrow 0$ // initialize best split point
12 **forall** $v \in \mathcal{M}$ **do**
13 **for** $i = 1, \ldots, k$ **do**
14 $\hat{P}(c_i | \boldsymbol{D}_Y) \leftarrow \frac{N_{vi}}{\sum_{j=1}^{k} N_{vj}}$
15 $\hat{P}(c_i | \boldsymbol{D}_N) \leftarrow \frac{n_i - N_{vi}}{\sum_{j=1}^{k} n_j - N_{vj}}$
16 $score(X \leq v) \leftarrow Gain(\boldsymbol{D}, \boldsymbol{D}_Y, \boldsymbol{D}_N)$
17 **if** $score(X \leq v) > score^*$ **then**
18 $v^* \leftarrow v$; $score^* \leftarrow score(X \leq v)$
19 **return** $(v^*, score^*)$

Iris Data: Class-specific Frequencies N_{vi}

Classes c_1 and c_2 for attribute X_1 - `sepal length`

We first compute the frequencies N_{vi}. Consider the split point $X_1 \leq 5.45$:



Iris Data: Finding the best split point

We do have two classes c_1 and c_2:

$$\hat{P}(c_1) = 50/150 = 1/3$$
$$\hat{P}(c_2) = 100/150 = 2/3$$

The entropy of the dataset $\boldsymbol{D}$ is therefore

$$H(\boldsymbol{D}) = -\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right) = 0.918$$

After computing the frequencies N_{vi}, we check each split point, e.g., $X_1 = 5.45$:

$$N_{v1} = 45 \qquad N_{v2} = 7$$

$$\hat{P}(c_1|\boldsymbol{D}_Y) = \frac{N_{v1}}{N_{v1}+N_{v2}} = \frac{45}{45+7} = 0.865$$
$$\hat{P}(c_2|\boldsymbol{D}_Y) = \frac{N_{v2}}{N_{v1}+N_{v2}} = \frac{7}{45+7} = 0.135$$

Iris Data: Finding the best split point

$$\hat{P}(c_1|\boldsymbol{D}_N) = \frac{n_1 - N_{v1}}{(n_1 - N_{v1}) + (n_2 - N_{v2})} = \frac{50 - 45}{(50 - 45) + (100 - 7)} = 0.051$$

$$\hat{P}(c_2|\boldsymbol{D}_N) = \frac{n_2 - N_{v2}}{(n_1 - N_{v1}) + (n_2 - N_{v2})} = \frac{(100 - 7)}{(50 - 45) + (100 - 7)} = 0.949$$

Computing the entropy of the partitions $\boldsymbol{D}_Y$ and $\boldsymbol{D}_N$:

$$H(\boldsymbol{D}_Y) = -(0.865 \log_2 0.865 + 0.135 \log_2 0.135) = 0.571$$

$$H(\boldsymbol{D}_N) = -(0.051 \log_2 0.051 + 0.949 \log_2 0.949) = 0.291$$

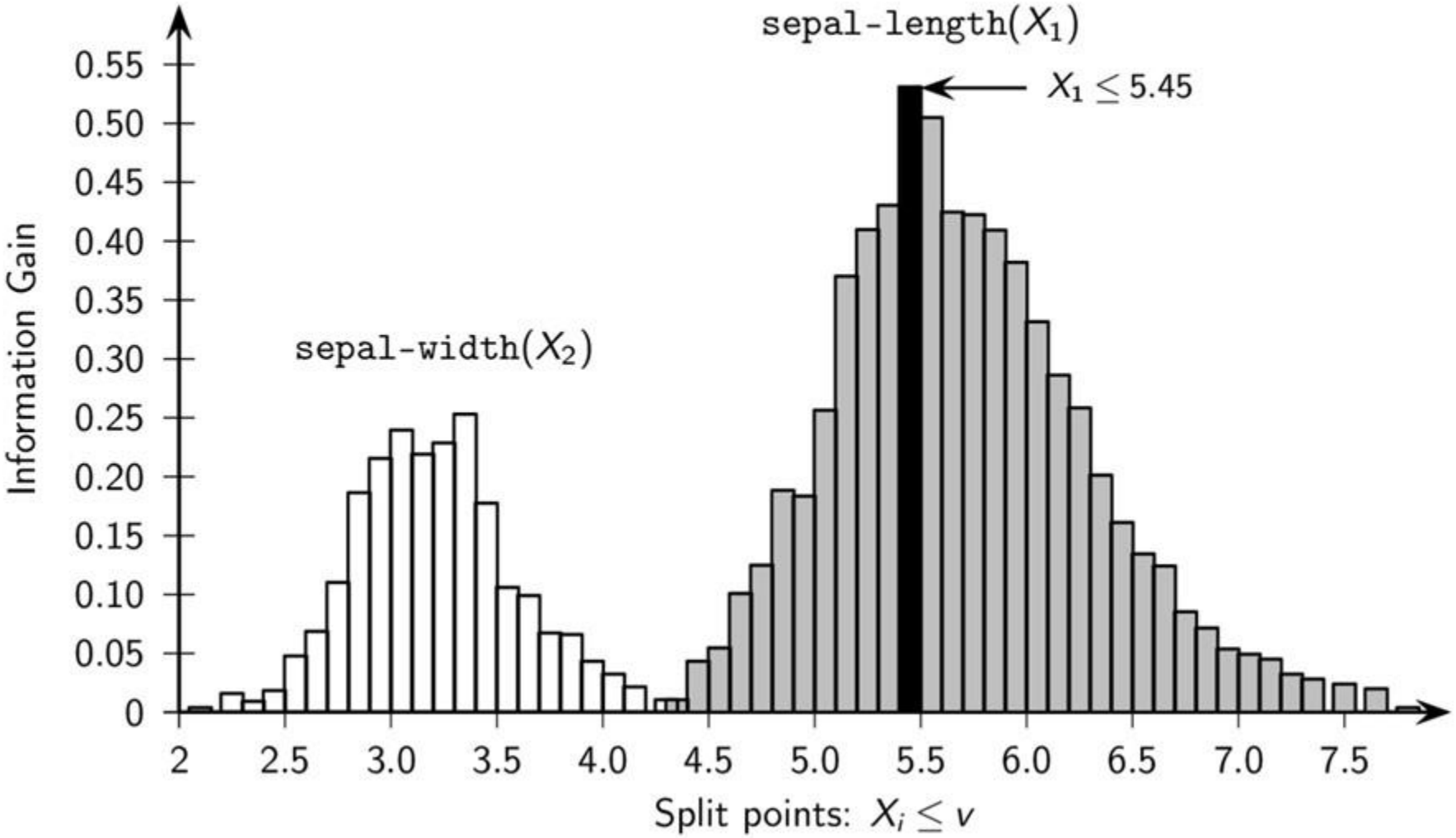
The entropy of $X \leq 5.45$ is:

$$H(\boldsymbol{D}_Y, \boldsymbol{D}_N) = \frac{52}{150} H(\boldsymbol{D}_Y) + \frac{98}{150} H(\boldsymbol{D}_N) = 0.388$$

The information gain for $X = 5.45$ is therefore

$$Gain = H(\boldsymbol{D}) - H(\boldsymbol{D}_Y, \boldsymbol{D}_N) = 0.918 - 0.388 = 0.53$$

Iris Data: Information Gain for Different Splits



Categorical Attributes

For categorical X the split points are of the form $X \in V$, where $V \subset dom(X)$ and $V \neq \emptyset$. All distinct partitions of the set of values of X are considered.

If $m = |dom(X)|$, then there are $O(2^{m-1})$ distinct partitions, which can be too many. One simplification is to restrict V to be of size one, so that there are only m split points of the form $X_j \in \{v\}$, where $v \in dom(X_j)$.

Define n_{vi} as the number of points $\boldsymbol{x}_j \in \boldsymbol{D}$, with value $x_j = v$ for attribute X and having class $y_j = c_i$:

$$n_{vi} = \sum_{j=1}^{n} I(x_j = v \text{ and } y_j = c_i)$$

The class conditional empirical PMF for X is then given as

$$\hat{P}(X = v|c_i) = \frac{\hat{P}(X = v \text{ and } c_i)}{\hat{P}(c_i)} = \frac{n_{vi}}{n_i}$$

We then have

$$\hat{P}(c_i|\boldsymbol{D}_Y) = \frac{\sum_{v \in V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \in V} n_{vj}} \qquad \hat{P}(c_i|\boldsymbol{D}_N) = \frac{\sum_{v \notin V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \notin V} n_{vj}}$$

Algorithm Evaluate-Categorical-Attribute

Evaluate-Categorical-Attribute (D, X, l):

1 **for** $i = 1, \ldots, k$ **do**
2 $\quad n_i \leftarrow 0$
3 $\quad$ **forall** $v \in dom(X)$ **do** $n_{vi} \leftarrow 0$
4 **for** $j = 1, \ldots, n$ **do**
5 $\quad$ **if** $x_j = v$ *and* $y_j = c_i$ **then** $n_{vi} \leftarrow n_{vi} + 1$ `// frequency statistics`
6

`// evaluate split points of the form` $X \in V$

7 $V^* \leftarrow \emptyset$; $score^* \leftarrow 0$ `// initialize best split point`
8 **forall** $V \subset dom(X)$, *such that* $1 \leq |V| \leq l$ **do**
9 $\quad$ **for** $i = 1, \ldots, k$ **do**
10 $\quad\quad \hat{P}(c_i|\boldsymbol{D}_Y) \leftarrow \frac{\sum_{v \in V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \in V} n_{vj}}$
11 $\quad\quad \hat{P}(c_i|\boldsymbol{D}_N) \leftarrow \frac{\sum_{v \notin V} n_{vi}}{\sum_{j=1}^{k} \sum_{v \notin V} n_{vj}}$
12 $\quad score(X \in V) \leftarrow Gain(\boldsymbol{D}, \boldsymbol{D}_Y, \boldsymbol{D}_N)$
13 $\quad$ **if** $score(X \in V) > score^*$ **then**
14 $\quad\quad V^* \leftarrow V; score^* \leftarrow score(X \in V)$
15 **return** $(V^*, score^*)$

Discretized sepal length: Class Frequencies

Consider the 2-dimensional Iris dataset comprising the sepal length and sepal width attributes.

sepal length has been discretized as shown in Table, where we can see the class frequencies n_{vi}:

Bins	v: values	Class frequencies (n_{vi})	
		c_1:iris-setosa	c_2:other
[4.3, 5.2]	Very Short (a_1)	39	6
(5.2, 6.1]	Short (a_2)	11	39
(6.1, 7.0]	Long (a_3)	0	43
(7.0, 7.9]	Very Long (a_4)	0	12

Discretized sepal length: Best Split Point

Consider the split point $X_1 \in \{a_1, a_3\}$:

$$\hat{P}(c_1|\boldsymbol{D}_Y) = \frac{n_{a_1 1} + n_{a_3 1}}{(n_{a_1 1} + n_{a_3 1}) + (n_{a_1 2} + n_{a_3 2})} = \frac{39 + 0}{(39 + 0) + (6 + 43)} = 0.443$$

$$\hat{P}(c_2|\boldsymbol{D}_Y) = 1 - \hat{P}(c_1|\boldsymbol{D}_Y) = 0.557$$

with the entropy given as

$$H(\boldsymbol{D}_Y) = -(0.443 \log_2 0.443 + 0.557 \log_2 0.557) = 0.991$$

Compute the class PMF for $\boldsymbol{D}_N$:

$$\hat{P}(c_1|\boldsymbol{D}_N) = \frac{n_{a_2 1} + n_{a_4 1}}{(n_{a_2 1} + n_{a_4 1}) + (n_{a_2 2} + n_{a_4 2})} = \frac{11 + 0}{(11 + 0) + (39 + 12)} = 0.177$$

$$\hat{P}(c_2|\boldsymbol{D}_N) = 1 - \hat{P}(c_1|\boldsymbol{D}_N) = 0.823$$

with the entropy given as

$$H(\boldsymbol{D}_N) = -(0.177 \log_2 0.177 + 0.823 \log_2 0.823) = 0.673$$

Discretized sepal length: Best Split Point

$V \in \{a_1, a_3\}$ splits the input data $\boldsymbol{D}$ into partitions of size

$$|\boldsymbol{D}_Y| = 39 + 6 + 43 = 88$$

$$|\boldsymbol{D}_N| = 150 - 88 = 62.$$

The entropy of the split is therefore given as

$$H(\boldsymbol{D}_Y, \boldsymbol{D}_N) = \frac{88}{150} H(\boldsymbol{D}_Y) + \frac{62}{150} H(\boldsymbol{D}_N) = 0.86$$

Since the entropy of $\boldsymbol{D}$ is $H(\boldsymbol{D}) = 0.918$. The gain is given as:

$$Gain = H(\boldsymbol{D}) - H(\boldsymbol{D}_Y, \boldsymbol{D}_N) = 0.918 - 0.86 = 0.058$$

Categorical Split Points for Discretized `sepal length`

V	Split entropy	Info. gain
$\{a_1\}$	**0.509**	**0.410**
$\{a_2\}$	0.897	0.217
$\{a_3\}$	0.711	0.207
$\{a_4\}$	0.869	0.049
$\{a_1, a_2\}$	0.632	0.286
$\{a_1, a_3\}$	0.860	0.058
$\{a_1, a_4\}$	0.667	0.251
$\{a_2, a_3\}$	0.667	0.251
$\{a_2, a_4\}$	0.860	0.058
$\{a_3, a_4\}$	0.632	0.286

Best split: $X \in \{a_1\}$.

Data mining and Machine learning

Part 20. Linear Discriminant Analysis

Linear Discriminant Analysis

Given labeled data consisting of d-dimensional points $\boldsymbol{x}_i$ along with their classes y_i, the goal of linear discriminant analysis (LDA) is to find a vector $\boldsymbol{w}$ that maximizes the separation between the classes after projection onto $\boldsymbol{w}$.

The key difference between principal component analysis and LDA is that the former deals with unlabeled data and tries to maximize variance, whereas the latter deals with labeled data and tries to maximize the discrimination between the classes.

Projection onto a Line

Let $\boldsymbol{D}_i$ denote the subset of points labeled with class c_i, i.e., $\boldsymbol{D}_i = \{\boldsymbol{x}_j | y_j = c_i\}$, and let $|\boldsymbol{D}_i| = n_i$ denote the number of points with class c_i. We assume that there are only $k = 2$ classes.

The projection of any d-dimensional point $\boldsymbol{x}_i$ onto a unit vector $\boldsymbol{w}$ is given as

$$\boldsymbol{x}_i' = \left(\frac{\boldsymbol{w}^T \boldsymbol{x}_i}{\boldsymbol{w}^T \boldsymbol{w}}\right) \boldsymbol{w} = \left(\boldsymbol{w}^T \boldsymbol{x}_i\right) \boldsymbol{w} = a_i \boldsymbol{w}$$

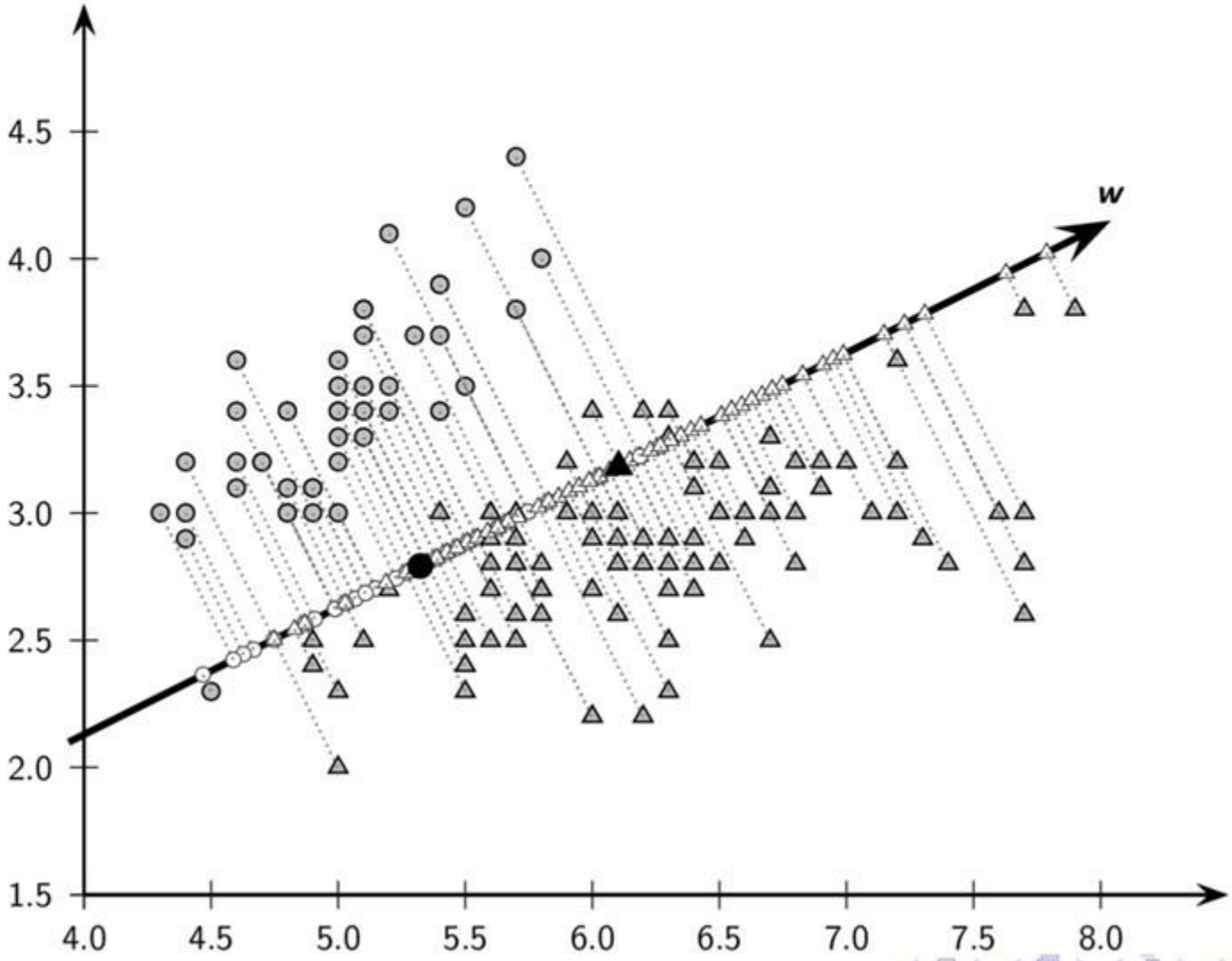
where a_i specifies the offset or coordinate of $\boldsymbol{x}_i'$ along the line $\boldsymbol{w}$:

$$a_i = \boldsymbol{w}^T \boldsymbol{x}_i$$

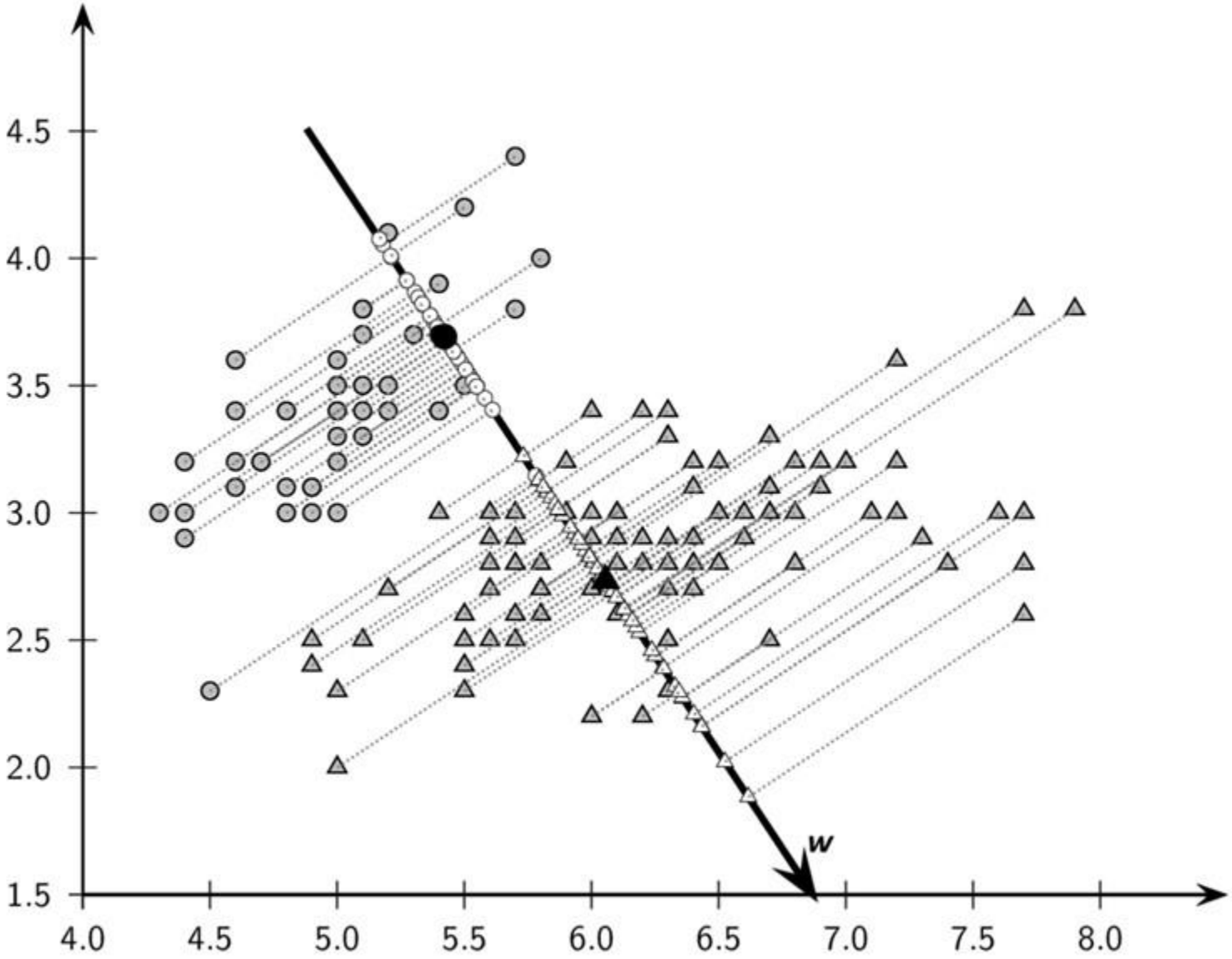
The set of n scalars $\{a_1, a_2, \ldots, a_n\}$ represents the mapping from $\mathbb{R}^d$ to $\mathbb{R}$, that is, from the original d-dimensional space to a 1-dimensional space (along $\boldsymbol{w}$).

Projection onto $\boldsymbol{w}$: Iris 2D Data

`iris-setosa` as class c_1 (circles), and the other two Iris types as class c_2 (triangles)



Iris 2D Data: Optimal Linear Discriminant Direction



Optimal Linear Discriminant

The mean of the projected points is given as:

$$m_1 = \boldsymbol{w}^T \boldsymbol{\mu}_1 \qquad\qquad m_2 = \boldsymbol{w}^T \boldsymbol{\mu}_2$$

To maximize the separation between the classes, we maximize the difference between the projected means, $|m_1 - m_2|$. However, for good separation, the variance of the projected points for each class should also not be too large. LDA maximizes the separation by ensuring that the *scatter* s_i^2 for the projected points within each class is small, where scatter is defined as

$$s_i^2 = \sum_{\boldsymbol{x}_j \in \boldsymbol{D}_i} (a_j - m_i)^2 = n_i \sigma_i^2$$

where σ_i^2 is the variance for class c_i.

Linear Discriminant Analysis: Fisher Objective

We incorporate the two LDA criteria, namely, maximizing the distance between projected means and minimizing the sum of projected scatter, into a single maximization criterion called the *Fisher LDA objective*:

$$\max_{\boldsymbol{w}} \; J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

In matrix terms, we can rewrite $(m_1 - m_2)^2$ as follows:

$$(m_1 - m_2)^2 = \left(\boldsymbol{w}^T(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\right)^2 = \boldsymbol{w}^T \boldsymbol{B} \boldsymbol{w}$$

where $\boldsymbol{B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ is a $d \times d$ rank-one matrix called the *between-class scatter matrix*.

The projected scatter for class c_i is given as

$$s_i^2 = \sum_{\boldsymbol{x}_j \in \boldsymbol{D_1}} (\boldsymbol{w}^T \boldsymbol{x}_j - \boldsymbol{w}^T \boldsymbol{\mu}_i)^2 = \boldsymbol{w}^T \left(\sum_{\boldsymbol{x}_j \in \boldsymbol{D}_i} (\boldsymbol{x}_j - \boldsymbol{\mu}_i)(\boldsymbol{x}_j - \boldsymbol{\mu}_i)^T \right) \boldsymbol{w} = \boldsymbol{w}^T \boldsymbol{S}_i \boldsymbol{w}$$

where $\boldsymbol{S}_i$ is the *scatter matrix* for $\boldsymbol{D}_i$.

Linear Discriminant Analysis: Fisher Objective

The combined scatter for both classes is given as

$$s_1^2 + s_2^2 = \boldsymbol{w}^T \boldsymbol{S}_1 \boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{S}_2 \boldsymbol{w} = \boldsymbol{w}^T (\boldsymbol{S}_1 + \boldsymbol{S}_2) \boldsymbol{w} = \boldsymbol{w}^T \boldsymbol{S} \boldsymbol{w}$$

where the symmetric positive semidefinite matrix $\boldsymbol{S} = \boldsymbol{S}_1 + \boldsymbol{S}_2$ denotes the *within-class scatter matrix* for the pooled data.
The LDA objective function in matrix form is

$$\max_{\boldsymbol{w}} \; J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{B} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S} \boldsymbol{w}}$$

To solve for the best direction $\boldsymbol{w}$, we differentiate the objective function with respect to $\boldsymbol{w}$; after simplification it yields the *generalized eigenvalue problem*

$$\boldsymbol{B} \boldsymbol{w} = \lambda \boldsymbol{S} \boldsymbol{w}$$

where $\lambda = J(\boldsymbol{w})$ is a generalized eigenvalue of $\boldsymbol{B}$ and $\boldsymbol{S}$. To maximize the objective λ should be chosen to be the largest generalized eigenvalue, and $\boldsymbol{w}$ to be the corresponding eigenvector.

Linear Discriminant Algorithm

LinearDiscriminant $(\boldsymbol{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n)$**:**

1 $\boldsymbol{D}_i \leftarrow \{\boldsymbol{x}_j \mid y_j = c_i, j = 1, \ldots, n\}, i = 1, 2$ // class-specific subsets

2 $\boldsymbol{\mu}_i \leftarrow \text{mean}(\boldsymbol{D}_i), i = 1, 2$ // class means

3 $\boldsymbol{B} \leftarrow (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ // between-class scatter matrix

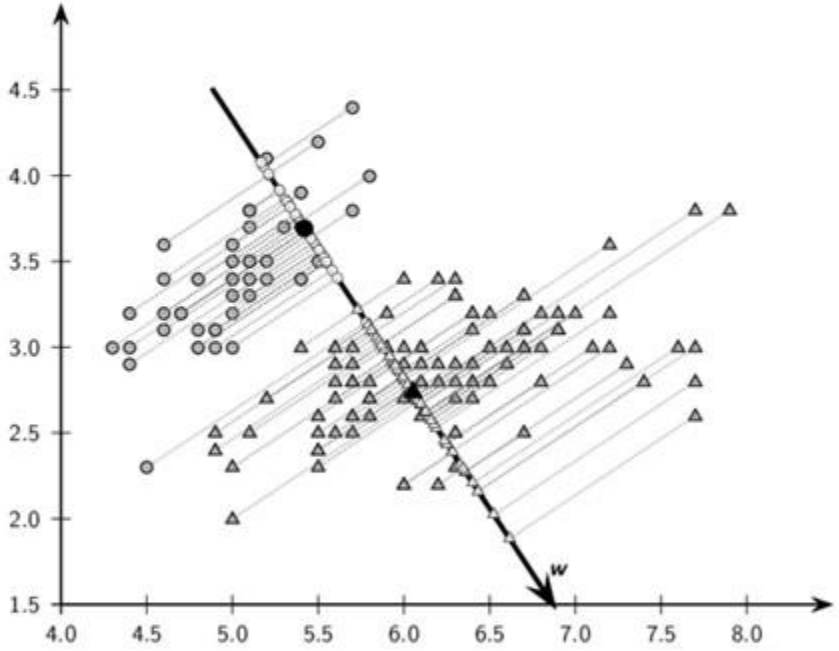
4 $\boldsymbol{Z}_i \leftarrow \boldsymbol{D}_i - 1_{n_i} \boldsymbol{\mu}_i^T, i = 1, 2$ // center class matrices

5 $\boldsymbol{S}_i \leftarrow \boldsymbol{Z}_i^T \boldsymbol{Z}_i, i = 1, 2$ // class scatter matrices

6 $\boldsymbol{S} \leftarrow \boldsymbol{S}_1 + \boldsymbol{S}_2$ // within-class scatter matrix

7 $\lambda_1, \boldsymbol{w} \leftarrow \text{eigen}(\boldsymbol{S}^{-1}\boldsymbol{B})$ // compute dominant eigenvector

Linear Discriminant Direction: Iris 2D Data



The between-class scatter matrix is

$$\boldsymbol{B} = \begin{pmatrix} 1.587 & -0.693 \\ -0.693 & 0.303 \end{pmatrix}$$

and the within-class scatter matrix is

$$\boldsymbol{S_1} = \begin{pmatrix} 6.09 & 4.91 \\ 4.91 & 7.11 \end{pmatrix}$$

$$\boldsymbol{S_2} = \begin{pmatrix} 43.5 & 12.09 \\ 12.09 & 10.96 \end{pmatrix}$$

$$\boldsymbol{S} = \begin{pmatrix} 49.58 & 17.01 \\ 17.01 & 18.08 \end{pmatrix}$$

The direction of most separation between c_1 and c_2 is the dominant eigenvector corresponding to the largest eigenvalue of the matrix $\boldsymbol{S^{-1}B}$. The solution is

$$J(\boldsymbol{w}) = \lambda_1 = 0.11$$

$$\boldsymbol{w} = \begin{pmatrix} 0.551 \\ -0.834 \end{pmatrix}$$

Linear Discriminant Analysis: Two Classes

For the two class scenario, if $\boldsymbol{S}$ is nonsingular, we can directly solve for $\boldsymbol{w}$ without computing the eigenvalues and eigenvectors.

The between-class scatter matrix $\boldsymbol{B}$ points in the same direction as $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ because

$$\begin{aligned}\boldsymbol{B}\boldsymbol{w} &= \Big((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\Big)\boldsymbol{w} \\ &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\Big((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\boldsymbol{w}\Big) \\ &= b(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\end{aligned}$$

The generalized eigenvectors equation can then be rewritten as

$$\boldsymbol{w} = \frac{b}{\lambda}\boldsymbol{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

Because $\frac{b}{\lambda}$ is just a scalar, we can solve for the best linear discriminant as

$$\boxed{\boldsymbol{w} = \boldsymbol{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)}$$

We can finally normalize $\boldsymbol{w}$ to be a unit vector.

Linear Discriminant Direction: Iris 2D Data

We can directly compute $\boldsymbol{w}$ as follows:

$$\begin{aligned}\boldsymbol{w} &= \boldsymbol{S}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ &= \begin{pmatrix} 0.066 & -0.029 \\ -0.100 & 0.044 \end{pmatrix} \begin{pmatrix} -1.246 \\ 0.546 \end{pmatrix} = \begin{pmatrix} -0.0527 \\ 0.0798 \end{pmatrix}\end{aligned}$$

After normalizing, we have

$$\boldsymbol{w} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \frac{1}{0.0956} \begin{pmatrix} -0.0527 \\ 0.0798 \end{pmatrix} = \begin{pmatrix} -0.551 \\ 0.834 \end{pmatrix}$$

Note that even though the sign is reversed for $\boldsymbol{w}$, they represent the same direction; only the scalar multiplier is different.

Kernel Discriminant Analysis

The goal of kernel LDA is to find the direction vector $\boldsymbol{w}$ in feature space that maximizes

$$\max_{\boldsymbol{w}} \; J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

It is well known that $\boldsymbol{w}$ can be expressed as a linear combination of the points in feature space

$$\boldsymbol{w} = \sum_{j=1}^{n} a_j \phi(\boldsymbol{x}_j)$$

The mean for class c_i in feature space is given as

$$\boldsymbol{\mu}_i^{\phi} = \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in \boldsymbol{D}_i} \phi(\boldsymbol{x}_j)$$

and the covariance matrix for class c_i in feature space is

$$\Sigma_i^{\phi} = \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in \boldsymbol{D}_i} \left(\phi(\boldsymbol{x}_j) - \boldsymbol{\mu}_i^{\phi}\right)\left(\phi(\boldsymbol{x}_j) - \boldsymbol{\mu}_i^{\phi}\right)^T$$

Kernel Discriminant Analysis

The between-class scatter matrix in feature space is

$$\boldsymbol{B}_\phi = \left(\boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi\right)\left(\boldsymbol{\mu}_1^\phi - \boldsymbol{\mu}_2^\phi\right)^T$$

and the within-class scatter matrix in feature space is

$$\boldsymbol{S}_\phi = n_1\Sigma_1^\phi + n_2\Sigma_2^\phi$$

$\boldsymbol{S}_\phi$ is a $d \times d$ symmetric, positive semidefinite matrix, where d is the dimensionality of the feature space.

The best linear discriminant vector $\boldsymbol{w}$ in feature space is the dominant eigenvector, which satisfies the expression

$$\left(\boldsymbol{S}_\phi^{-1}\boldsymbol{B}_\phi\right)\boldsymbol{w} = \lambda\boldsymbol{w}$$

where we assume that $\boldsymbol{S}_\phi$ is non-singular.

LDA Objective via Kernel Matrix: Between-class Scatter

The projected mean for class c_i is given as

$$m_i = \boldsymbol{w}^T \boldsymbol{\mu}_i^{\phi} = \frac{1}{n_i} \sum_{j=1}^{n} \sum_{\boldsymbol{x}_k \in \boldsymbol{D}_i} a_j K(\boldsymbol{x}_j, \boldsymbol{x}_k) = \boldsymbol{a}^T \boldsymbol{m}_i$$

where $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)^T$ is the weight vector, and

$$\boldsymbol{m}_i = \frac{1}{n_i} \begin{pmatrix} \sum_{\boldsymbol{x}_k \in \boldsymbol{D}_i} K(\boldsymbol{x}_1, \boldsymbol{x}_k) \\ \sum_{\boldsymbol{x}_k \in \boldsymbol{D}_i} K(\boldsymbol{x}_2, \boldsymbol{x}_k) \\ \vdots \\ \sum_{\boldsymbol{x}_k \in \boldsymbol{D}_i} K(\boldsymbol{x}_n, \boldsymbol{x}_k) \end{pmatrix} = \frac{1}{n_i} \boldsymbol{K}^{c_i} 1_{n_i}$$

where $\boldsymbol{K}^{c_i}$ is the $n \times n_i$ subset of the kernel matrix, restricted to columns belonging to points only in $\boldsymbol{D}_i$, and 1_{n_i} is the n_i-dimensional vector all of whose entries are one.

The separation between the projected means is thus

$$(m_1 - m_2)^2 = \left(\boldsymbol{a}^T \boldsymbol{m}_1 - \boldsymbol{a}^T \boldsymbol{m}_2\right)^2 = \boldsymbol{a}^T \boldsymbol{M} \boldsymbol{a}$$

where $\boldsymbol{M} = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T$ is the between-class scatter matrix.

LDA Objective via Kernel Matrix: Within-class Scatter

We can compute the projected scatter for each class, s_1^2 and s_2^2, purely in terms of the kernel function, as follows

$$s_1^2 = \sum_{x_i \in D_1} \left\| \boldsymbol{w}^T \phi(\boldsymbol{x}_i) - \boldsymbol{w}^T \boldsymbol{\mu}_1^{\phi} \right\|^2 = \boldsymbol{a}^T \left(\left(\sum_{x_i \in D_1} \boldsymbol{K}_i \boldsymbol{K}_i^T \right) - n_1 \boldsymbol{m}_1 \boldsymbol{m}_1^T \right) \boldsymbol{a} = \boldsymbol{a}^T \boldsymbol{N}_1 \boldsymbol{a}$$

where $\boldsymbol{K}_i$ is the ith column of the kernel matrix, and $\boldsymbol{N}_1$ is the class scatter matrix for c_1.

The sum of projected scatter values is then given as

$$s_1^2 + s_2^2 = \boldsymbol{a}^T (\boldsymbol{N}_1 + \boldsymbol{N}_2) \boldsymbol{a} = \boldsymbol{a}^T \boldsymbol{N} \boldsymbol{a}$$

where $\boldsymbol{N}$ is the $n \times n$ within-class scatter matrix.

Kernel LDA

The kernel LDA maximization condition is

$$\max_{\boldsymbol{w}} J(\boldsymbol{w}) = \max_{\boldsymbol{a}} J(\boldsymbol{a}) = \frac{\boldsymbol{a}^T \boldsymbol{M} \boldsymbol{a}}{\boldsymbol{a}^T \boldsymbol{N} \boldsymbol{a}}$$

The weight vector $\boldsymbol{a}$ is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem:

$$\boldsymbol{M}\boldsymbol{a} = \lambda_1 \boldsymbol{N}\boldsymbol{a}$$

When there are only two classes $\boldsymbol{a}$ can be obtained directly:

$$\boldsymbol{a} = \boldsymbol{N}^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2)$$

To normalize $\boldsymbol{w}$ to be a unit vector we scale $\boldsymbol{a}$ by $\frac{1}{\sqrt{\boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a}}}$.

We can project any point $\boldsymbol{x}$ onto the discriminant direction as follows:

$$\boldsymbol{w}^T \phi(\boldsymbol{x}) = \sum_{j=1}^{n} a_j \phi(\boldsymbol{x}_j)^T \phi(\boldsymbol{x}) = \sum_{j=1}^{n} a_j K(\boldsymbol{x}_j, \boldsymbol{x})$$

Kernel Discriminant Analysis Algorithm

KernelDiscriminant $(\boldsymbol{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}, K)$**:**

1 $\boldsymbol{K} \leftarrow \left\{K(\boldsymbol{x}_i, \boldsymbol{x}_j)\right\}_{i,j=1,\ldots,n}$ // compute $n \times n$ kernel matrix

2 $\boldsymbol{K}^{c_i} \leftarrow \left\{\boldsymbol{K}(j,k) \mid y_k = c_i, 1 \leq j,k \leq n\right\}, i = 1,2$ // class kernel matrix

3 $\boldsymbol{m}_i \leftarrow \frac{1}{n_i}\boldsymbol{K}^{c_i} 1_{n_i}, i = 1,2$ // class means

4 $\boldsymbol{M} \leftarrow (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T$ // between-class scatter matrix

5 $\boldsymbol{N}_i \leftarrow \boldsymbol{K}^{c_i}(\boldsymbol{I}_{n_i} - \frac{1}{n_i} 1_{n_i \times n_i})(\boldsymbol{K}^{c_i})^T,\ i = 1,2$ // class scatter matrices

6 $\boldsymbol{N} \leftarrow \boldsymbol{N}_1 + \boldsymbol{N}_2$ // within-class scatter matrix

7 $\lambda_1, \boldsymbol{a} \leftarrow \text{eigen}(\boldsymbol{N}^{-1}\boldsymbol{M})$ // compute weight vector

8 $\boldsymbol{a} \leftarrow \frac{\boldsymbol{a}}{\sqrt{\boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a}}}$ // normalize $\boldsymbol{w}$ to be unit vector

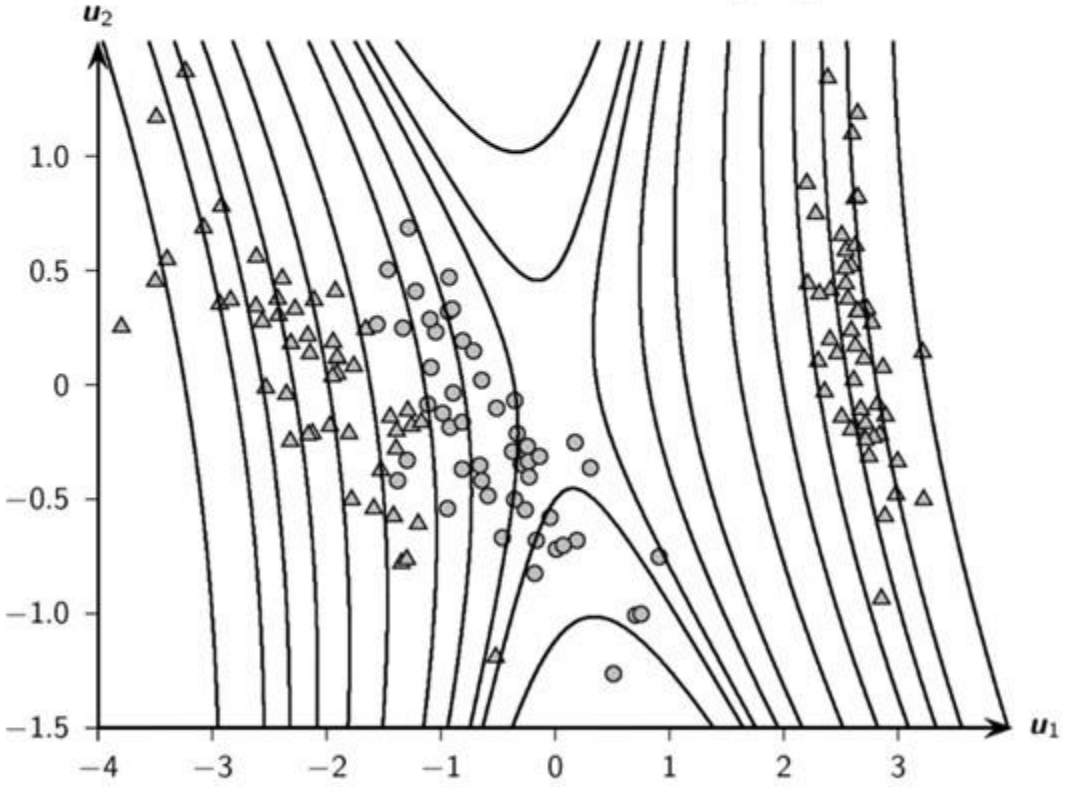
Kernel Discriminant Analysis

Quadratic Homogeneous Kernel

Iris 2D Data: c_1 (circles; `iris-virginica`) and c_2 (triangles; other two types).

Kernel Function: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \boldsymbol{x}_j)^2$.

Contours of constant projection onto optimal kernel discriminant, i.e., points along both the curves have the same value when projected onto $\boldsymbol{w}$.



Kernel Discriminant Analysis

Quadratic Homogeneous Kernel

Projecting $\boldsymbol{x}_i \in \boldsymbol{D}$ onto $\boldsymbol{w}$, which separates the two classes.

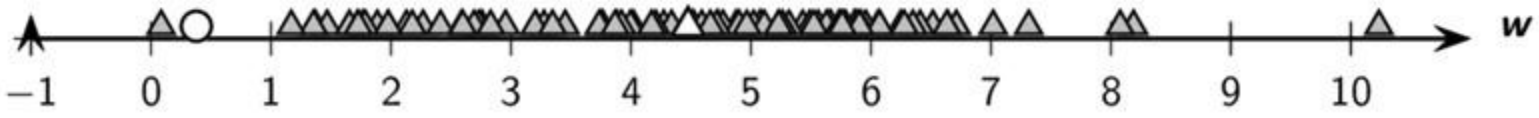
The projected scatters and means for both classes are as follows:

$$m_1 = 0.338 \qquad m_2 = 4.476$$
$$s_1^2 = 13.862 \qquad s_2^2 = 320.934$$

The value of $J(\boldsymbol{w})$ is given as

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{(0.338 - 4.476)^2}{13.862 + 320.934} = \frac{17.123}{334.796} = 0.0511$$

which, as expected, matches $\lambda_1 = 0.0511$ from above.



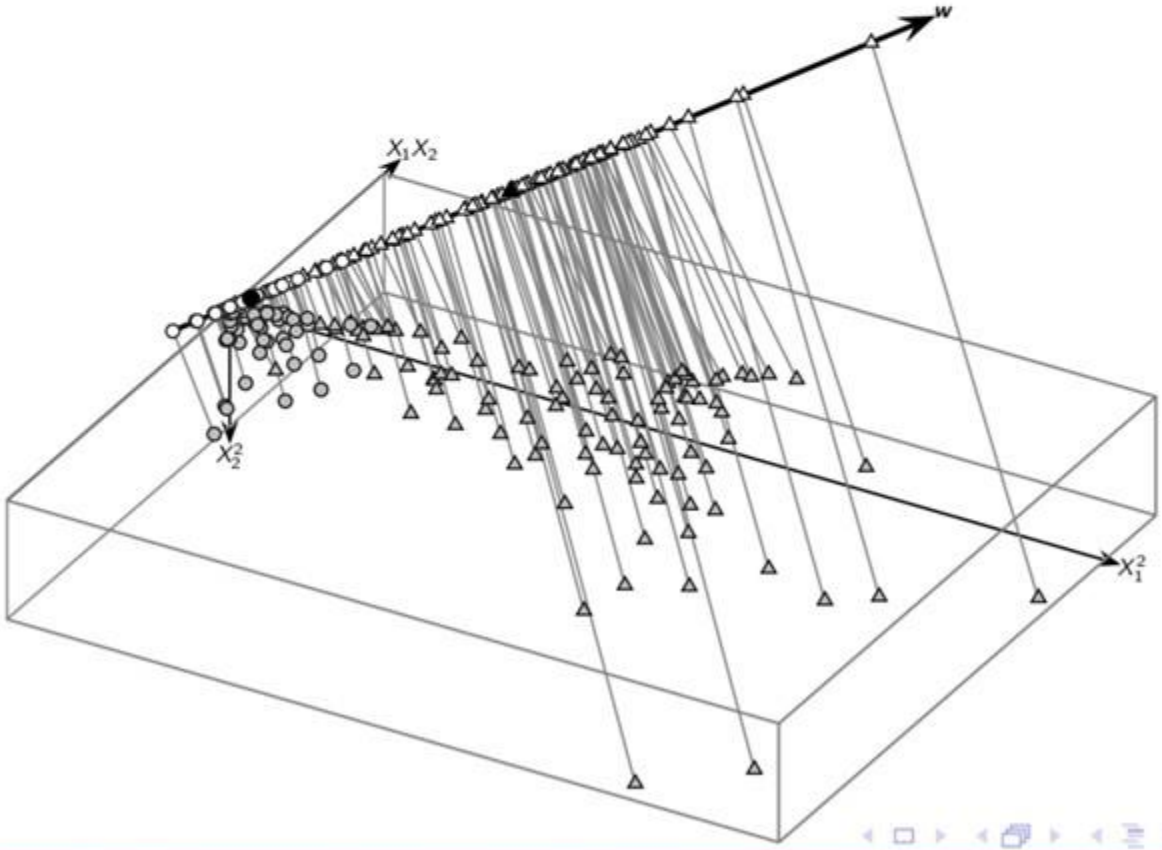
Kernel Feature Space and Optimal Discriminant

It is not desirable or possible to obtain an explicit discriminant vector $\boldsymbol{w}$.

$\boldsymbol{x} = (x_1, x_2)^T \in \mathbb{R}^2$ is mapped to $\phi(\boldsymbol{x}) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)^T \in \mathbb{R}^3$.

The projection of $\phi(\boldsymbol{x}_i)$ onto $\boldsymbol{w}$ is also shown, where

$$\boldsymbol{w} = 0.511x_1x_2 + 0.761x_1^2 - 0.4x_2^2$$



Data mining and Machine learning

Part 21. Support Vector Machine

Hyperplanes

Let $\boldsymbol{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ be a classification dataset, with n points in a d-dimensional space. We assume that there are only two class labels, that is, $y_i \in \{+1, -1\}$, denoting the positive and negative classes.

A hyperplane in d dimensions is given as the set of all points $\boldsymbol{x} \in \mathbb{R}^d$ that satisfy the equation $h(\boldsymbol{x}) = 0$, where $h(\boldsymbol{x})$ is the *hyperplane function*:

$$h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$$

Here, $\boldsymbol{w}$ is a d dimensional *weight vector* and b is a scalar, called the *bias*.

For points that lie on the hyperplane, we have

$$h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0$$

The weight vector $\boldsymbol{w}$ specifies the direction that is orthogonal or normal to the hyperplane, which fixes the orientation of the hyperplane, whereas the bias b fixes the offset of the hyperplane in the d-dimensional space, i.e., where the hyperplane intersects each of the axes:

$$w_ix_i = -b \quad \text{or} \quad x_i = \frac{-b}{w_i}$$

Separating Hyperplane

A hyperplane splits the d-dimensional data space into two *half-spaces*.

A dataset is said to be *linearly separable* if each half-space has points only from a single class.

If the input dataset is linearly separable, then we can find a *separating* hyperplane $h(\boldsymbol{x}) = 0$, such that for all points labeled $y_i = -1$, we have $h(\boldsymbol{x}_i) < 0$, and for all points labeled $y_i = +1$, we have $h(\boldsymbol{x}_i) > 0$.

The hyperplane function $h(\boldsymbol{x})$ thus serves as a linear classifier or a linear discriminant, which predicts the class y for any given point $\boldsymbol{x}$, according to the decision rule:

$$y = \begin{cases} +1 & \text{if } h(\boldsymbol{x}) > 0 \\ -1 & \text{if } h(\boldsymbol{x}) < 0 \end{cases}$$

Geometry of a Hyperplane: Distance

Consider a point $\boldsymbol{x} \in \mathbb{R}^d$ that does not lie on the hyperplane. Let $\boldsymbol{x}_p$ be the orthogonal projection of $\boldsymbol{x}$ on the hyperplane, and let $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{x}_p$. Then we can write $\boldsymbol{x}$ as

$$\boldsymbol{x} = \boldsymbol{x}_p + \boldsymbol{r} = \boldsymbol{x}_p + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

where r is the *directed distance* of the point $\boldsymbol{x}$ from $\boldsymbol{x}_p$.
To obtain an expression for r, consider the value $h(\boldsymbol{x})$, we have:

$$h(\boldsymbol{x}) = h\left(\boldsymbol{x}_p + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right) = \boldsymbol{w}^T\left(\boldsymbol{x}_p + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right) + b = r\|\boldsymbol{w}\|$$

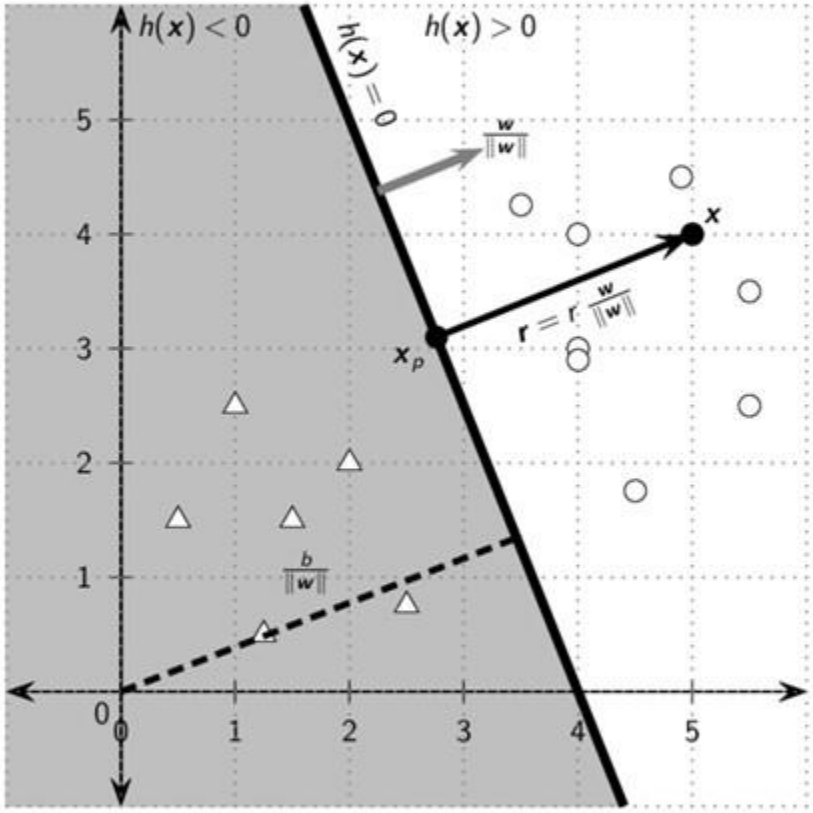
The directed distance r of point $\boldsymbol{x}$ to the hyperplane is thus:

$$r = \frac{h(\boldsymbol{x})}{\|\boldsymbol{w}\|}$$

To obtain distance, which must be non-negative, we multiply r by the class label y_i of the point $\boldsymbol{x}_i$ because when $h(\boldsymbol{x}_i) < 0$, the class is -1, and when $h(\boldsymbol{x}_i) > 0$ the class is $+1$:

$$\delta_i = \frac{y_i h(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

Geometry of a Hyperplane in 2D



$$\boldsymbol{p} = (p_1, p_2) = (4,0),\ \boldsymbol{q} = (q_1, q_2) = (2,5)$$

$$-\frac{w_1}{w_2} = \frac{q_2 - p_2}{q_1 - p_1} = \frac{5-0}{2-4} = -\frac{5}{2}$$

Given $(4,0)$, the offset b is:

$$b = -5x_1 - 2x_2 = -5 \cdot 4 - 2 \cdot 0 = -20$$

Given $\boldsymbol{w} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$ and $b = -20$:

$$h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b = \begin{pmatrix} 5 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 20 = 0$$

$$\delta = y\, r = -1\, r = \frac{-b}{\|\boldsymbol{w}\|} = \frac{-(-20)}{\sqrt{29}} = 3.71$$

Margin and Support Vectors

The distance of a point $\boldsymbol{x}$ from the hyperplane $h(\boldsymbol{x}) = 0$ is thus given as

$$\delta = y\, r = \frac{y\, h(\boldsymbol{x})}{\|\boldsymbol{w}\|}$$

The *margin* is the minimum distance of a point from the separating hyperplane:

$$\delta^* = \min_{\boldsymbol{x}_i} \left\{ \frac{y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|} \right\}$$

All the points (or vectors) that achieve the minimum distance are called *support vectors* for the hyperplane. They satisfy the condition:

$$\delta^* = \frac{y^*(\boldsymbol{w}^T \boldsymbol{x}^* + b)}{\|\boldsymbol{w}\|}$$

where y^* is the class label for $\boldsymbol{x}^*$.

Canonical Hyperplane

Multiplying the hyperplane equation on both sides by some scalar s yields an equivalent hyperplane:

$$s\, h(\boldsymbol{x}) = s\, \boldsymbol{w}^T \boldsymbol{x} + s\, b = (s\boldsymbol{w})^T \boldsymbol{x} + (sb) = 0$$

To obtain the unique or *canonical* hyperplane, we choose the scalar $s = \frac{1}{y^*(\boldsymbol{w}^T \boldsymbol{x}^* + b)}$ so that the absolute distance of a support vector from the hyperplane is 1, i.e., the margin is

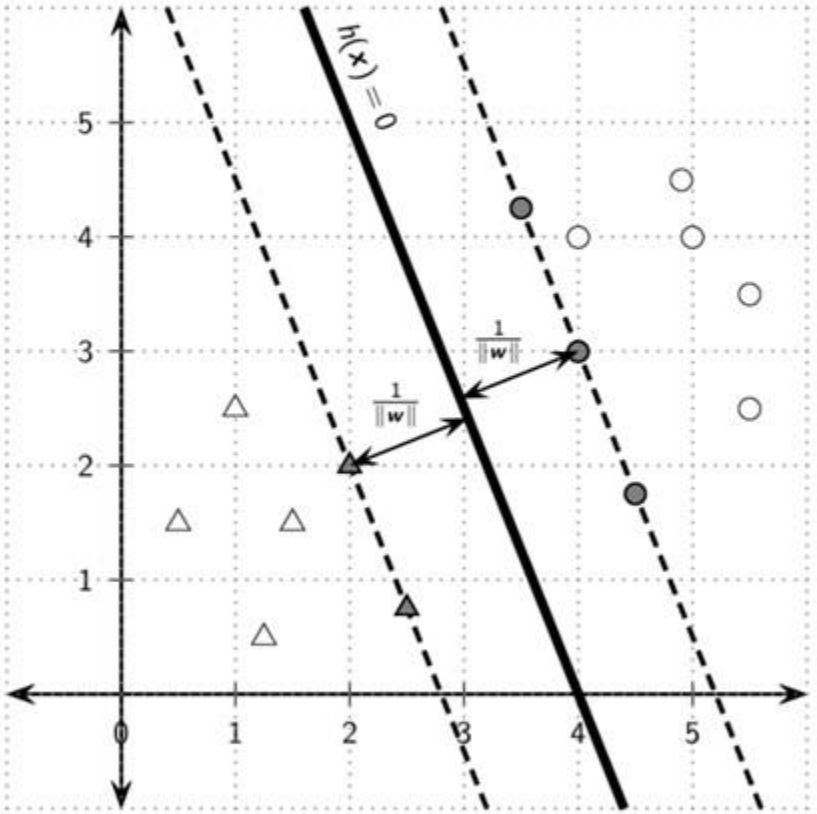
$$\delta^* = \frac{y^*(\boldsymbol{w}^T \boldsymbol{x}^* + b)}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}$$

For the canonical hyperplane, for each support vector $\boldsymbol{x}_i^*$ (with label y_i^*), we have $y_i^* h(\boldsymbol{x}_i^*) = 1$, and for any point that is not a support vector we have $y_i h(\boldsymbol{x}_i) > 1$. Over all points, we have

$$y_i\, (\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1, \text{ for all points } \boldsymbol{x}_i \in \boldsymbol{D}$$

Separating Hyperplane: Margin and Support Vectors

Shaded points are support vectors



$$h(x) = \begin{pmatrix} 5 \\ 2 \end{pmatrix}^T x - 20 = 0$$

Given $x^* = (2,2)^T$, $y^* = -1$.

$$s = \frac{1}{y^* h(x^*)} = \frac{1}{-1\left(\begin{pmatrix} 5 \\ 2 \end{pmatrix}^T \begin{pmatrix} 2 \\ 2 \end{pmatrix} - 20\right)} = \frac{1}{6}$$

$$w = \frac{1}{6}\begin{pmatrix} 5 \\ 2 \end{pmatrix} = \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix} \qquad b = \frac{-20}{6}$$

$$h(x) = \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix}^T x - 20/6 = \begin{pmatrix} 0.833 \\ 0.333 \end{pmatrix}^T x - 3.33$$

$$\delta^* = \frac{y^* h(x^*)}{\|w\|} = \frac{1}{\sqrt{\left(\frac{5}{6}\right)^2 + \left(\frac{2}{6}\right)^2}} = \frac{6}{\sqrt{29}} = 1.114$$

SVM: Linear and Separable Case

Assume that the points are linearly separable, that is, there exists a separating hyperplane that perfectly classifies each point.

The goal of SVMs is to choose the canonical hyperplane, h^*, that yields the maximum margin among all possible separating hyperplanes

$$h^* = \arg\max_{\boldsymbol{w},b} \left\{ \frac{1}{\|\boldsymbol{w}\|} \right\}$$

We can obtain an equivalent minimization formulation:

$$\textbf{Objective Function:} \quad \min_{\boldsymbol{w},b} \left\{ \frac{\|\boldsymbol{w}\|^2}{2} \right\}$$

$$\textbf{Linear Constraints:} \quad y_i \left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) \geq 1, \ \forall \boldsymbol{x}_i \in \boldsymbol{D}$$

SVM: Linear and Separable Case

We turn the constrained SVM optimization into an unconstrained one by introducing a Lagrange multiplier α_i for each constraint. The new objective function, called the *Lagrangian*, then becomes

$$\min L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \right)$$

L should be minimized w.r.t. $\boldsymbol{w}$ and b, and it should be maximized w.r.t. α_i.
Taking the derivative of L with respect to $\boldsymbol{w}$ and b, and setting those to zero, we obtain

$$\frac{\partial}{\partial \boldsymbol{w}} L = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0 \quad \text{or} \quad \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial}{\partial b} L = \sum_{i=1}^{n} \alpha_i y_i = 0$$

We can see that $\boldsymbol{w}$ can be expressed as a linear combination of the data points $\boldsymbol{x}_i$, with the signed Lagrange multipliers, $\alpha_i y_i$, serving as the coefficients.

Further, the sum of the signed Lagrange multipliers, $\alpha_i y_i$, must be zero.

SVM: Linear and Separable Case

Incorporating $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$ into the Lagrangian we obtain the new *dual Lagrangian* objective function, which is specified purely in terms of the Lagrange multipliers:

Objective Function: $\max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$

Linear Constraints: $\alpha_i \geq 0, \forall i \in \boldsymbol{D}$, and $\sum_{i=1}^{n} \alpha_i y_i = 0$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$ is the vector comprising the Lagrange multipliers.

L_{dual} is a convex quadratic programming problem (note the $\alpha_i \alpha_j$ terms), which admits a unique optimal solution.

SVM: Linear and Separable Case

Once we have obtained the α_i values for $i = 1, \ldots, n$, we can solve for the weight vector $\boldsymbol{w}$ and the bias b. Each of the Lagrange multipliers α_i satisfies the KKT conditions at the optimal solution:

$$\alpha_i \left(y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \right) = 0$$

which gives rise to two cases:

1. $\alpha_i = 0$, or
2. $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 = 0$, which implies $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1$

This is a very important result because if $\alpha_i > 0$, then $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1$, and thus the point $\boldsymbol{x}_i$ must be a support vector.

On the other hand, if $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) > 1$, then $\alpha_i = 0$, that is, if a point is not a support vector, then $\alpha_i = 0$.

Linear and Separable Case: Weight Vector and Bias

Once we know α_i for all points, we can compute the weight vector $\boldsymbol{w}$ by taking the summation only for the support vectors:

$$\boldsymbol{w} = \sum_{i,\alpha_i>0} \alpha_i y_i \boldsymbol{x}_i$$

Only the support vectors determine $\boldsymbol{w}$, since $\alpha_i = 0$ for other points.
To compute the bias b, we first compute one solution b_i, per support vector, as follows:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1, \text{ which implies } b_i = \frac{1}{y_i} - \boldsymbol{w}^T \boldsymbol{x}_i = y_i - \boldsymbol{w}^T \boldsymbol{x}_i$$

The bias b is taken as the average value:

$$b = \text{avg}_{\alpha_i>0}\{b_i\}$$

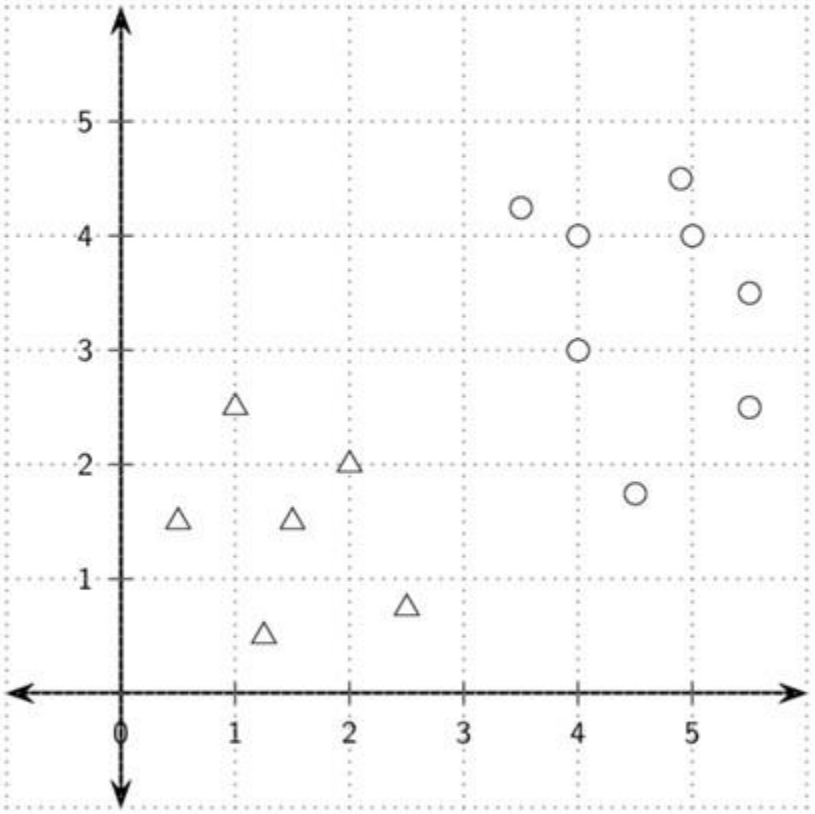
SVM Classifier

Given the optimal hyperplane function $h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$, for any new point $\boldsymbol{z}$, we predict its class as

$$\hat{y} = \text{sign}(h(\boldsymbol{z})) = \text{sign}(\boldsymbol{w}^T \boldsymbol{z} + b)$$

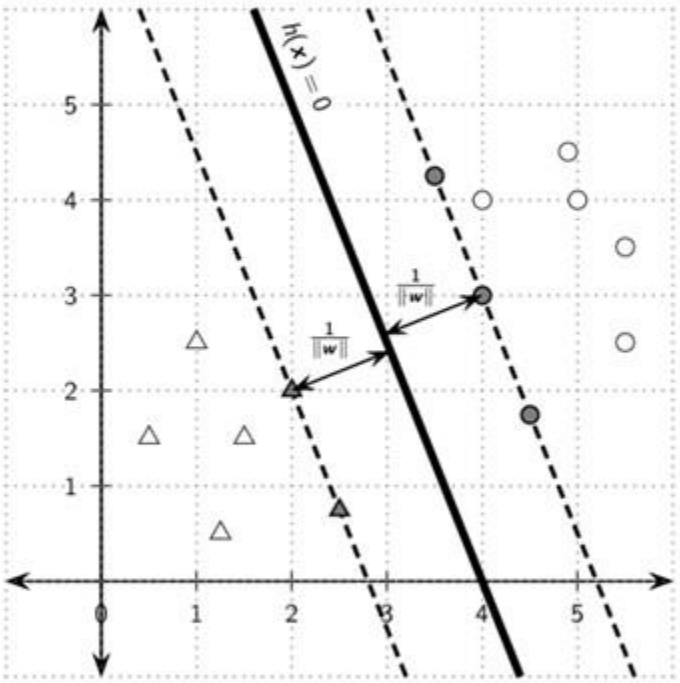
where the sign$(\cdot)$ function returns $+1$ if its argument is positive, and -1 if its argument is negative.

Example Dataset: Separable Case



$\boldsymbol{x}_i$	x_{i1}	x_{i2}	y_i
$\boldsymbol{x}_1$	3.5	4.25	$+1$
$\boldsymbol{x}_2$	4	3	$+1$
$\boldsymbol{x}_3$	4	4	$+1$
$\boldsymbol{x}_4$	4.5	1.75	$+1$
$\boldsymbol{x}_5$	4.9	4.5	$+1$
$\boldsymbol{x}_6$	5	4	$+1$
$\boldsymbol{x}_7$	5.5	2.5	$+1$
$\boldsymbol{x}_8$	5.5	3.5	$+1$
$\boldsymbol{x}_9$	0.5	1.5	-1
$\boldsymbol{x}_{10}$	1	2.5	-1
$\boldsymbol{x}_{11}$	1.25	0.5	-1
$\boldsymbol{x}_{12}$	1.5	1.5	-1
$\boldsymbol{x}_{13}$	2	2	-1
$\boldsymbol{x}_{14}$	2.5	0.75	-1

Optimal Separating Hyperplane



Solving the L_{dual} quadratic program yields

x_i	x_{i1}	x_{i2}	y_i	α_i
x_1	3.5	4.25	+1	0.0437
x_2	4	3	+1	0.2162
x_4	4.5	1.75	+1	0.1427
x_{13}	2	2	−1	0.3589
x_{14}	2.5	0.75	−1	0.0437

The weight vector and bias are:

$$\boldsymbol{w} = \sum_{i,\alpha_i>0} \alpha_i y_i \boldsymbol{x}_i = \begin{pmatrix} 0.833 \\ 0.334 \end{pmatrix}$$

$$b = \text{avg}\{b_i\} = -3.332$$

The optimal hyperplane is given as follows:

$$h(\boldsymbol{x}) = \begin{pmatrix} 0.833 \\ 0.334 \end{pmatrix}^T \boldsymbol{x} - 3.332 = 0$$

Soft Margin SVM: Linear and Nonseparable Case

The assumption that the dataset be perfectly linearly separable is unrealistic. SVMs can handle non-separable points by introducing *slack variables* ξ_i as follows:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i$$

where $\xi_i \geq 0$ is the slack variable for point $\boldsymbol{x}_i$, which indicates how much the point violates the separability condition, that is, the point may no longer be at least $1/\|\boldsymbol{w}\|$ away from the hyperplane.

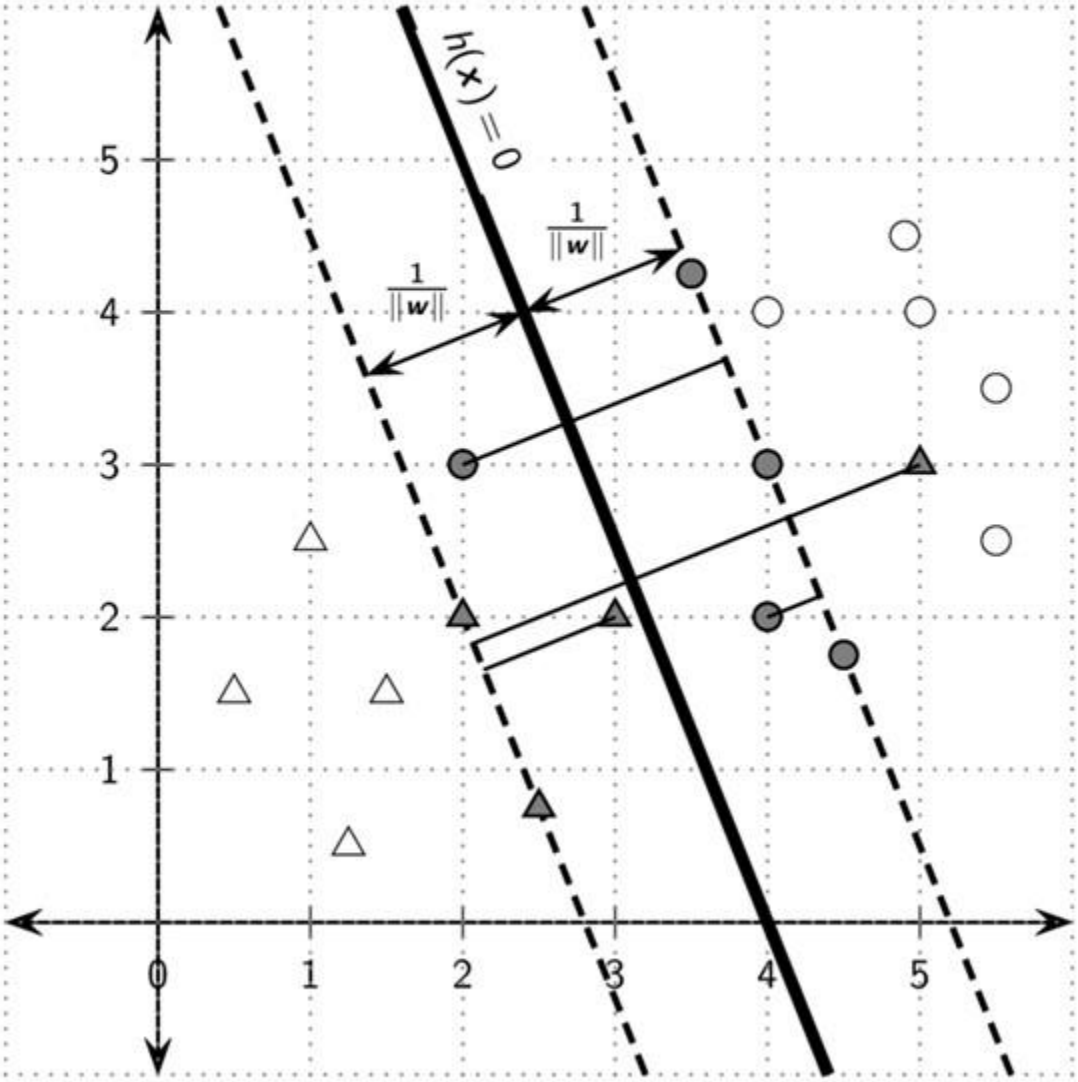
The slack values indicate three types of points. If $\xi_i = 0$, then the corresponding point $\boldsymbol{x}_i$ is at least $\frac{1}{\|\boldsymbol{w}\|}$ away from the hyperplane.

If $0 < \xi_i < 1$, then the point is within the margin and still correctly classified, that is, it is on the correct side of the hyperplane.

However, if $\xi_i \geq 1$ then the point is misclassified and appears on the wrong side of the hyperplane.

Soft Margin Hyperplane

Shaded points are the support vectors



SVM: Soft Margin or Linearly Non-separable Case

In the nonseparable case, also called the *soft margin* the SVM objective function is

$$\textbf{Objective Function:} \quad \min_{\boldsymbol{w},b,\xi_i} \left\{ \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{i=1}^{n}(\xi_i)^k \right\}$$

$$\textbf{Linear Constraints:} \quad y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geq 1 - \xi_i, \ \forall \boldsymbol{x}_i \in \boldsymbol{D}$$

$$\xi_i \geq 0 \ \forall \boldsymbol{x}_i \in \boldsymbol{D}$$

where C and k are constants that incorporate the cost of misclassification.

The term $\sum_{i=1}^{n}(\xi_i)^k$ gives the *loss*, that is, an estimate of the deviation from the separable case.

The scalar C is a *regularization constant* that controls the trade-off between maximizing the margin or minimizing the loss. For example, if $C \to 0$, then the loss component essentially disappears, and the objective defaults to maximizing the margin. On the other hand, if $C \to \infty$, then the margin ceases to have much effect, and the objective function tries to minimize the loss.

SVM: Soft Margin Loss Function

The constant k governs the form of the loss. When $k = 1$, called *hinge loss*, the goal is to minimize the sum of the slack variables, whereas when $k = 2$, called *quadratic loss*, the goal is to minimize the sum of the squared slack variables.

Hinge Loss: Assuming $k = 1$, the SVM dual Lagrangian is given as

$$\max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

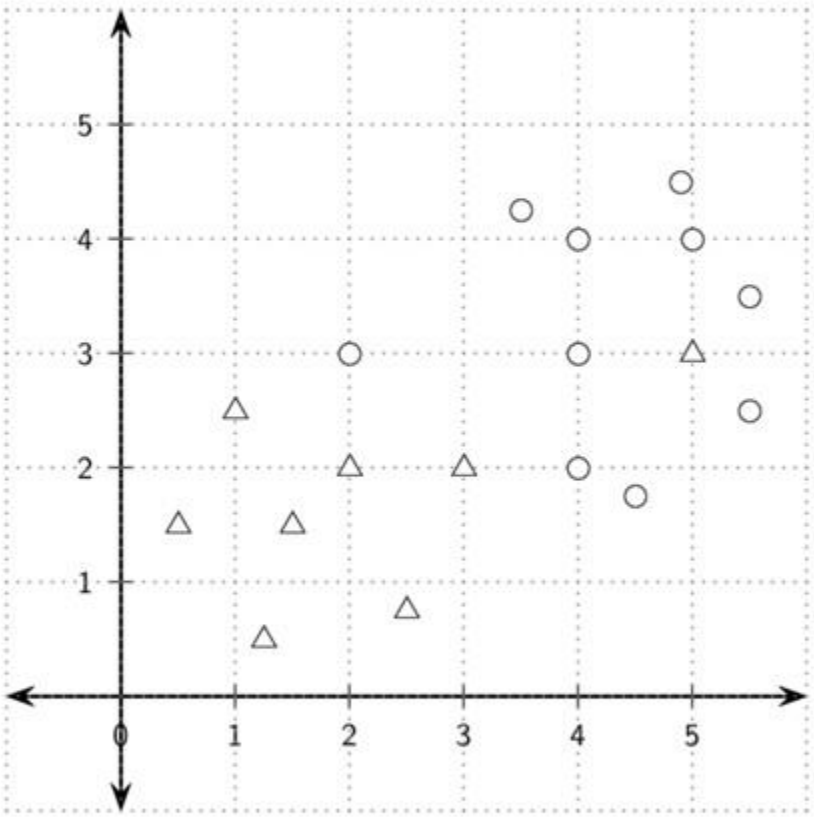
The only difference from the separable case is that $0 \leq \alpha_i \leq C$.

Quadratic Loss: Assuming $k = 2$, the dual objective is:

$$\max_{\boldsymbol{\alpha}} \; L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_i^T \boldsymbol{x}_j + \frac{1}{2C} \delta_{ij} \right)$$

where δ is the *Kronecker delta* function, defined as $\delta_{ij} = 1$ if and only if $i = j$.

Example Dataset: Linearly Non-separable Case



x_i	x_{i1}	x_{i2}	y_i
x_1	3.5	4.25	+1
x_2	4	3	+1
x_3	4	4	+1
x_4	4.5	1.75	+1
x_5	4.9	4.5	+1
x_6	5	4	+1
x_7	5.5	2.5	+1
x_8	5.5	3.5	+1
x_9	0.5	1.5	−1
x_{10}	1	2.5	−1
x_{11}	1.25	0.5	−1
x_{12}	1.5	1.5	−1
x_{13}	2	2	−1
x_{14}	2.5	0.75	−1
x_{15}	4	2	+1
x_{16}	2	3	+1
x_{17}	3	2	−1
x_{18}	5	3	−1

Example Dataset: Linearly Non-separable Case

Let $k = 1$ and $C = 1$, then solving the L_{dual} yields the following support vectors and Lagrangian values α_i:

$\boldsymbol{x}_i$	x_{i1}	x_{i2}	y_i	α_i
$\boldsymbol{x}_1$	3.5	4.25	$+1$	0.0271
$\boldsymbol{x}_2$	4	3	$+1$	0.2162
$\boldsymbol{x}_4$	4.5	1.75	$+1$	0.9928
$\boldsymbol{x}_{13}$	2	2	-1	0.9928
$\boldsymbol{x}_{14}$	2.5	0.75	-1	0.2434
$\boldsymbol{x}_{15}$	4	2	$+1$	1
$\boldsymbol{x}_{16}$	2	3	$+1$	1
$\boldsymbol{x}_{17}$	3	2	-1	1
$\boldsymbol{x}_{18}$	5	3	-1	1

The optimal hyperplane is given as follows:

$$h(\boldsymbol{x}) = \begin{pmatrix} 0.834 \\ 0.333 \end{pmatrix}^T \boldsymbol{x} - 3.334 = 0$$

Example Dataset: Linearly Non-separable Case

The slack $\xi_i = 0$ for all points that are not support vectors, and also for those support vectors that are on the margin. Slack is positive only for the remaining support vectors and it can be computed as: $\xi_i = 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)$.
Thus, for all support vectors not on the margin, we have

$\boldsymbol{x}_i$	$\boldsymbol{w}^T \boldsymbol{x}_i$	$\boldsymbol{w}^T \boldsymbol{x}_i + b$	$\xi_i = 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)$
$\boldsymbol{x}_{15}$	4.001	0.667	0.333
$\boldsymbol{x}_{16}$	2.667	−0.667	1.667
$\boldsymbol{x}_{17}$	3.167	−0.167	0.833
$\boldsymbol{x}_{18}$	5.168	1.834	2.834

The total slack is given as

$$\sum_i \xi_i = \xi_{15} + \xi_{16} + \xi_{17} + \xi_{18} = 0.333 + 1.667 + 0.833 + 2.834 = 5.667$$

The slack variable $\xi_i > 1$ for those points that are misclassified (i.e., are on the wrong side of the hyperplane), namely $\boldsymbol{x}_{16} = (3,3)^T$ and $\boldsymbol{x}_{18} = (5,3)^T$. The other two points are correctly classified, but lie within the margin, and thus satisfy $0 < \xi_i < 1$.

Kernel SVM: Nonlinear Case

The linear SVM approach can be used for datasets with a nonlinear decision boundary via the kernel trick.

Conceptually, the idea is to map the original d-dimensional points $\boldsymbol{x}_i$ in the input space to points $\phi(\boldsymbol{x}_i)$ in a high-dimensional feature space via some nonlinear transformation ϕ.

Given the extra flexibility, it is more likely that the points $\phi(\boldsymbol{x}_i)$ might be linearly separable in the feature space.

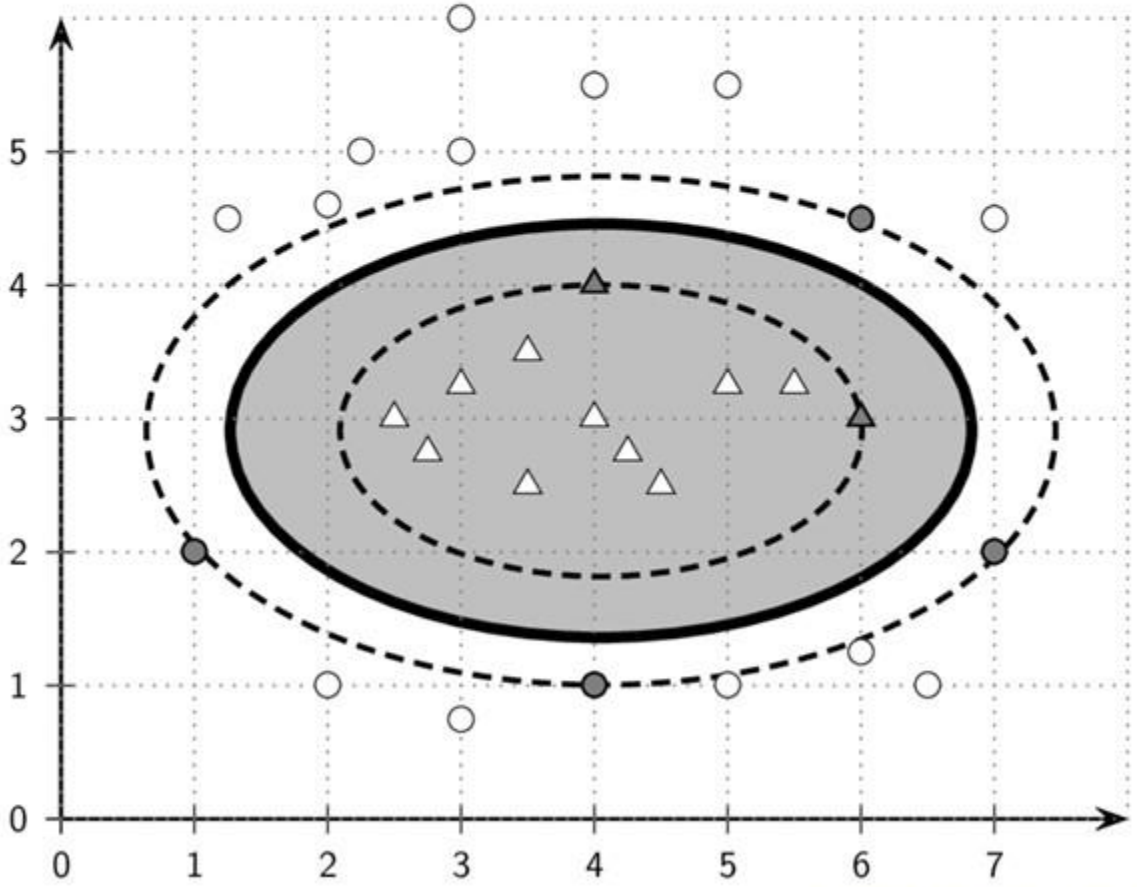
A linear decision surface in feature space actually corresponds to a nonlinear decision surface in the input space.

Further, the kernel trick allows us to carry out all operations via the kernel function computed in input space, rather than having to map the points into feature space.

Nonlinear SVM

There is no linear classifier that can discriminate between the points. However, there exists a perfect quadratic classifier that can separate the two classes.

$$\phi(\boldsymbol{x}) = (\sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)^T$$



Nonlinear SVMs: Kernel Trick

To apply the kernel trick for nonlinear SVM classification, we have to show that all operations require only the kernel function:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

Applying ϕ to each point, we can obtain the new dataset in the feature space $\boldsymbol{D}_\phi = \{\phi(\boldsymbol{x}_i), y_i\}_{i=1}^n$.

The SVM objective function in feature space is given as

Objective Function: $$\min_{\boldsymbol{w}, b, \xi_i} \left\{ \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{i=1}^{n} (\xi_i)^k \right\}$$

Linear Constraints: $$y_i \left(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b\right) \geq 1 - \xi_i, \text{and } \xi_i \geq 0, \ \forall \boldsymbol{x}_i \in \boldsymbol{D}$$

where $\boldsymbol{w}$ is the weight vector, b is the bias, and ξ_i are the slack variables, all in feature space.

Nonlinear SVMs: Kernel Trick

For hinge loss, the dual Lagrangian in feature space is given as

$$\max_{\boldsymbol{\alpha}} L_{dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$
$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Subject to the constraints that $0 \leq \alpha_i \leq C$, and $\sum_{i=1}^{n} \alpha_i y_i = 0$.

The dual Lagrangian depends only on the dot product between two vectors in feature space $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and thus we can solve the optimization problem using the kernel matrix $\boldsymbol{K} = \{K(\boldsymbol{x}_i, \boldsymbol{x}_j)\}_{i,j=1,\ldots,n}$.

For quadratic loss, the dual Lagrangian corresponds to the use of a new kernel

$$K_q(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j + \frac{1}{2C} \delta_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{2C} \delta_{ij}$$

Nonlinear SVMs: Weight Vector and Bias

We cannot directly obtain the weight vector without transforming the points, since

$$\boldsymbol{w} = \sum_{\alpha_i > 0} \alpha_i y_i \phi(\boldsymbol{x}_i)$$

However, we can compute the bias via kernel operations, since

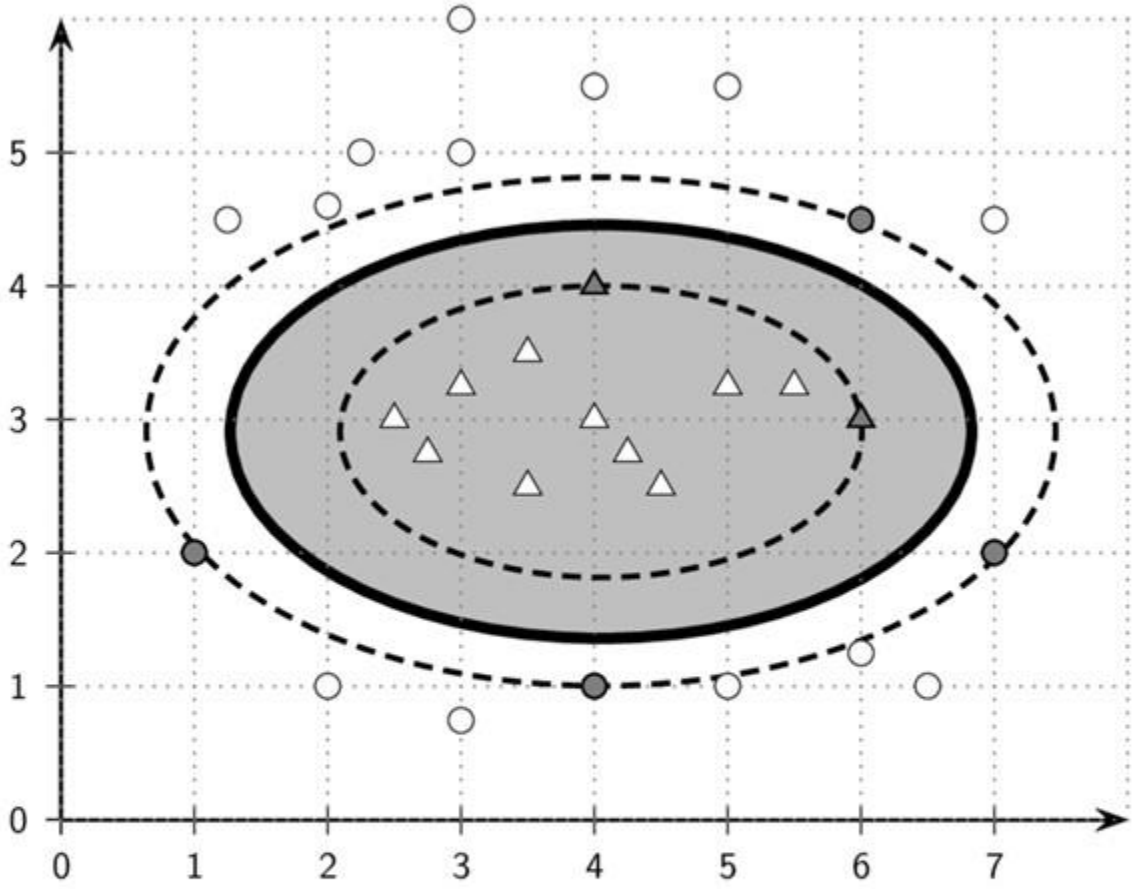
$$b_i = y_i - \boldsymbol{w}^T \phi(\boldsymbol{x}_i) = y_i - \sum_{\alpha_j > 0} \alpha_j y_j K(\boldsymbol{x}_j, \boldsymbol{x}_i)$$

Likewise, we can predict the class for a new point $\boldsymbol{z}$ as follows:

$$\hat{y} = \text{sign}(\boldsymbol{w}^T \phi(\boldsymbol{z}) + b) = \boxed{\text{sign}\left(\sum_{\alpha_i > 0} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{z}) + b\right)}$$

All SVM operations can be carried out in terms of the kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$. Thus, any nonlinear kernel function can be used to do nonlinear classification in the input space.

Nonlinear SVM: Inhomogeneous Quadratic Kernel



The optimal quadratic hyperplane is obtained by setting $C = 4$, and using an inhomogeneous polynomial kernel of degree $q = 2$:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T \boldsymbol{x}_j)^2$$

Nonlinear SVM: Inhomogeneous Quadratic Kernel

ϕ maps $\boldsymbol{x}_i$ into feature space as follows:

$$\phi\left(\boldsymbol{x} = (x_1, x_2)^T\right) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right)^T$$

$\boldsymbol{x}_1 = (1,2)^T$ is transformed into

$$\phi(\boldsymbol{x}_i) = \left(1, \sqrt{2}\cdot 1, \sqrt{2}\cdot 2, 1^2, 2^2, \sqrt{2}\cdot 1\cdot 2\right)^T = (1, 1.41, 2.83, 1, 2, 2.83)^T$$

Solving L_{dual}, we found the following six support vectors:

$\boldsymbol{x}_i$	$(x_{i1}, x_{i2})^T$	$\phi(\boldsymbol{x}_i)$	y_i	α_i
$\boldsymbol{x}_1$	$(1,2)^T$	$(1, 1.41, 2.83, 1, 4, 2.83)^T$	$+1$	0.6198
$\boldsymbol{x}_2$	$(4,1)^T$	$(1, 5.66, 1.41, 16, 1, 5.66)^T$	$+1$	2.069
$\boldsymbol{x}_3$	$(6,4.5)^T$	$(1, 8.49, 6.36, 36, 20.25, 38.18)^T$	$+1$	3.803
$\boldsymbol{x}_4$	$(7,2)^T$	$(1, 9.90, 2.83, 49, 4, 19.80)^T$	$+1$	0.3182
$\boldsymbol{x}_5$	$(4,4)^T$	$(1, 5.66, 5.66, 16, 16, 15.91)^T$	-1	2.9598
$\boldsymbol{x}_6$	$(6,3)^T$	$(1, 8.49, 4.24, 36, 9, 25.46)^T$	-1	3.8502

Nonlinear SVM: Inhomogeneous Quadratic Kernel

We compute the weight vector for the hyperplane:

$$\boldsymbol{w} = \sum_{\alpha_i > 0} \alpha_i y_i \phi(\boldsymbol{x}_i) = (0, -1.413, -3.298, 0.256, 0.82, -0.018)^T$$

as well as the bias:

$$b = -8.841$$

The decision boundary in input space corresponds to an ellipse, centered at $(4.046, 2.907)$, with axis lengths 2.78 and 1.55.

Notice that we explicitly transformed all the points into the feature space just for illustration purposes.

The kernel trick allows us to achieve the same goal using only the kernel function.

SVM Training Algorithms

Instead of dealing explicitly with the bias b, we map each point $\boldsymbol{x}_i \in \mathbb{R}^d$ to the point $\boldsymbol{x}'_i \in \mathbb{R}^{d+1}$ as follows:

$$\boldsymbol{x}'_i = (x_{i1}, \ldots, x_{id}, 1)^T$$

We also map the weight vector to $\mathbb{R}^{d+1}$, with $w_{d+1} = b$, so that

$$\boldsymbol{w} = (w_1, \ldots, w_d, b)^T$$

The equation of the hyperplane is then given as follows:

$$h(\boldsymbol{x}') : \boldsymbol{w}^T \boldsymbol{x}' = w_1 x_{i1} + \cdots + w_d x_{id} + b = 0$$

After the mapping, the constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ does not apply in the SVM dual formulations. The new set of constraints is given as

$$y_i \boldsymbol{w}^T \boldsymbol{x} \geq 1 - \xi_i$$

Dual Optimization: Gradient Ascent

The dual optimization objective for hinge loss is given as

$$\max_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

subject to the constraints $0 \leq \alpha_i \leq C$ for all $i = 1, \ldots, n$. Here $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)^T \in \mathbb{R}^n$.

The gradient or the rate of change in the objective function at $\boldsymbol{\alpha}$ is given as the partial derivative of $J(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$, that is, with respect to each α_k:

$$\nabla J(\boldsymbol{\alpha}) = \left(\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_1}, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_2}, \ldots, \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_n} \right)^T$$

where the kth component of the gradient is obtained by differentiating $J(\alpha_k)$ with respect to α_k:

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \frac{\partial J(\alpha_k)}{\partial \alpha_k} = 1 - y_k \left(\sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_k) \right)$$

Stochastic Gradient Ascent

Starting from an initial $\boldsymbol{\alpha}$, the gradient ascent approach successively updates by moving in the direction of the gradient $\nabla J(\boldsymbol{\alpha})$:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \eta_t \nabla J(\boldsymbol{\alpha}_t)$$

where $\boldsymbol{\alpha}_t$ is the estimate at the tth step, and η_t is the step size.

The optimal step size is:

$$\eta_k = \frac{1}{K(\boldsymbol{x}_k, \boldsymbol{x}_k)}$$

Instead of updating the entire $\boldsymbol{\alpha}$ vector in each step, in the stochastic gradient ascent approach, we update each component α_k independently and immediately use the new value to update other components. The update rule for the k-th component is given as

$$\alpha_k = \alpha_k + \eta_k \frac{\partial J(\boldsymbol{\alpha})}{\partial \alpha_k} = \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_k)\right)$$

Algorithm SVM-Dual

SVM-Dual (D, K, C, ϵ)**:**

1 **foreach** $x_i \in D$ **do** $x_i \leftarrow \begin{pmatrix} x_i \\ 1 \end{pmatrix}$

2 **if** *loss* = `hinge` **then**

3 $\quad K \leftarrow \{K(x_i, x_j)\}_{i,j=1,\ldots,n}$ `// kernel matrix, hinge loss`

4 **else if** *loss* = `quadratic` **then**

5 $\quad K \leftarrow \{K(x_i, x_j) + \frac{1}{2C}\delta_{ij}\}_{i,j=1,\ldots,n}$ `// kernel matrix, quadratic loss`

6 **for** $k = 1, \ldots, n$ **do** $\eta_k \leftarrow \frac{1}{K(x_k, x_k)}$

7 $t \leftarrow 0$

8 $\alpha_0 \leftarrow (0, \ldots, 0)^T$

9 **repeat**

10 $\quad \alpha \leftarrow \alpha_t$

11 $\quad$ **for** $k = 1$ *to* n **do**

$\quad\quad$ `// update kth component of` α

12 $\quad\quad \alpha_k \leftarrow \alpha_k + \eta_k \left(1 - y_k \sum_{i=1}^{n} \alpha_i y_i K(x_i, x_k)\right)$

13 $\quad\quad$ **if** $\alpha_k < 0$ **then** $\alpha_k \leftarrow 0$

14 $\quad\quad$ **if** $\alpha_k > C$ **then** $\alpha_k \leftarrow C$

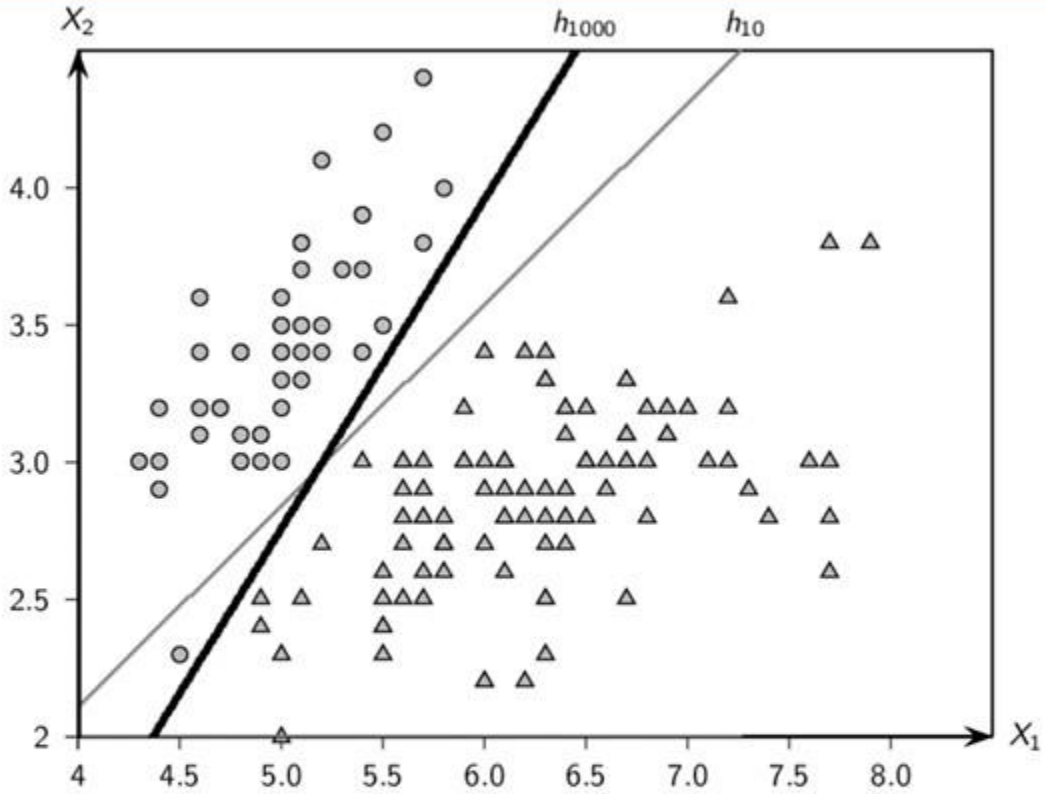
15 $\quad \alpha_{t+1} = \alpha$

16 $\quad t \leftarrow t + 1$

17 **until** $\|\alpha_t - \alpha_{t-1}\| \leq \epsilon$

SVM Dual Algorithm: Iris Data – Linear Kernel

c_1: `Iris-setosa` (circles) and c_2: other types of Iris flowers (triangles)



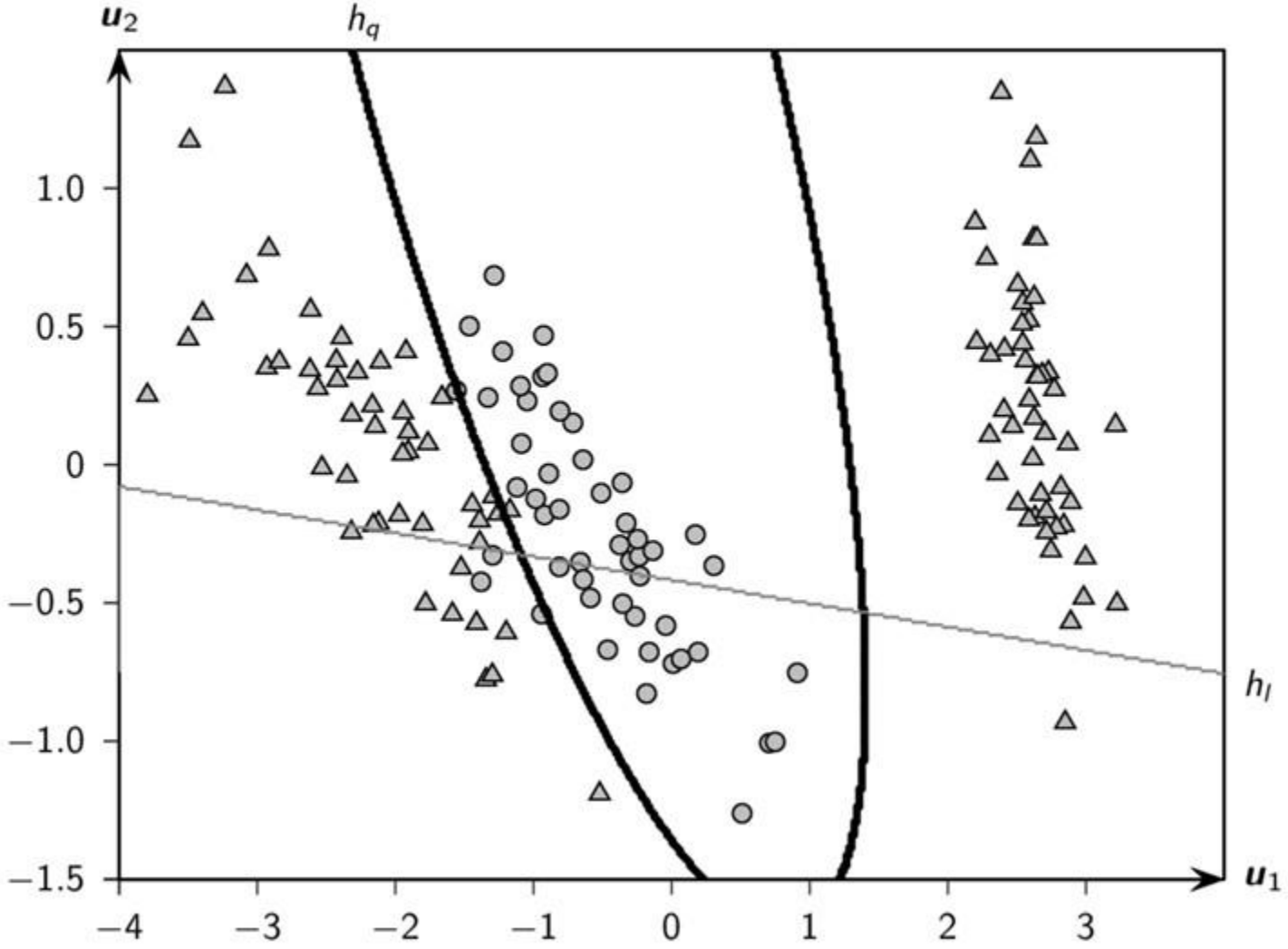
Hyperplane h_{10} uses $C = 10$ and h_{1000} uses $C = 1000$:

$$h_{10}(\boldsymbol{x}): \quad 2.74x_1 - 3.74x_2 - 3.09 = 0$$
$$h_{1000}(\boldsymbol{x}): \quad 8.56x_1 - 7.14x_2 - 23.12 = 0$$

h_{10} has a larger margin, but also a larger slack; h_{1000} has a smaller margin, but it minimizes the slack.

SVM Dual Algorithm: Quadratic versus Linear Kernel

c_1: `Iris-versicolor` (circles) and c_2: other types of Iris flowers (triangles)



Primal Solution: Newton Optimization

Consider the primal optimization function for soft margin SVMs. With $\boldsymbol{w}, \boldsymbol{x}_i \in \mathbb{R}^{d+1}$, we have to minimize the objective function:

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i)^k$$

subject to the linear constraints:

$$y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i\right) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i = 1, \ldots, n$$

Rearranging the above, we obtain an expression for ξ_i

$$\xi_i \geq 1 - y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i\right) \text{ and } \xi_i \geq 0, \text{ which implies that}$$
$$\xi_i = \max\left\{0, 1 - y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i\right)\right\}$$

Primal Solution: Newton Optimization, Quadratic Loss

The objective function can be rewritten as

$$J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \max\left\{0, 1 - y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i\right)\right\}^k$$
$$= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{y_i(\boldsymbol{w}^T\boldsymbol{x}_i)<1} \left(1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i)\right)^k$$

For quadratic loss, we have $k = 2$ and the gradient or the rate of change of the objective function at $\boldsymbol{w}$ is given as the partial derivative of $J(\boldsymbol{w})$ with respect to $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}} = \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \boldsymbol{w} - 2C\boldsymbol{v} + 2C\boldsymbol{S}\boldsymbol{w}$$

where the vector $\boldsymbol{v}$ and the matrix $\boldsymbol{S}$ are given as

$$\boldsymbol{v} = \sum_{y_i(\boldsymbol{w}^T\boldsymbol{x}_i)<1} y_i\boldsymbol{x}_i \qquad \boldsymbol{S} = \sum_{y_i(\boldsymbol{w}^T\boldsymbol{x}_i)<1} \boldsymbol{x}_i\boldsymbol{x}_i^T$$

Primal Solution: Newton Optimization, Quadratic Loss

The *Hessian matrix* is defined as the matrix of second-order partial derivatives of $J(\boldsymbol{w})$ with respect to $\boldsymbol{w}$, which is given as

$$\boldsymbol{H}_{\boldsymbol{w}} = \frac{\partial \nabla_{\boldsymbol{w}}}{\partial \boldsymbol{w}} = \boldsymbol{I} + 2C\boldsymbol{S}$$

Because we want to minimize the objective function $J(\boldsymbol{w})$, we should move in the direction opposite to the gradient. The Newton optimization update rule for $\boldsymbol{w}$ is given as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \boldsymbol{H}_{\boldsymbol{w}_t}^{-1} \nabla_{\boldsymbol{w}_t}$$

where $\eta_t > 0$ is a scalar value denoting the step size at iteration t.

Primal SVM Algorithm

SVM-Primal $(\boldsymbol{D}, C, \epsilon)$:

1 **foreach** $\boldsymbol{x}_i \in \boldsymbol{D}$ **do**

2 $\quad \boldsymbol{x}_i \leftarrow \begin{pmatrix} \boldsymbol{x}_i \\ 1 \end{pmatrix}$ // map to $\mathbb{R}^{d+1}$

3 $t \leftarrow 0$

4 $\boldsymbol{w}_0 \leftarrow (0, \ldots, 0)^T$ // initialize $\boldsymbol{w}_t \in \mathbb{R}^{d+1}$

5 **repeat**

6 $\quad \boldsymbol{v} \leftarrow \sum_{y_i(\boldsymbol{w}_t^T \boldsymbol{x}_i) < 1} y_i \boldsymbol{x}_i$

7 $\quad \boldsymbol{S} \leftarrow \sum_{y_i(\boldsymbol{w}_t^T \boldsymbol{x}_i) < 1} \boldsymbol{x}_i \boldsymbol{x}_i^T$

8 $\quad \nabla \leftarrow (\boldsymbol{I} + 2C\boldsymbol{S})\boldsymbol{w}_t - 2C\boldsymbol{v}$ // gradient

9

10 $\quad \boldsymbol{H} \leftarrow \boldsymbol{I} + 2C\boldsymbol{S}$ // Hessian

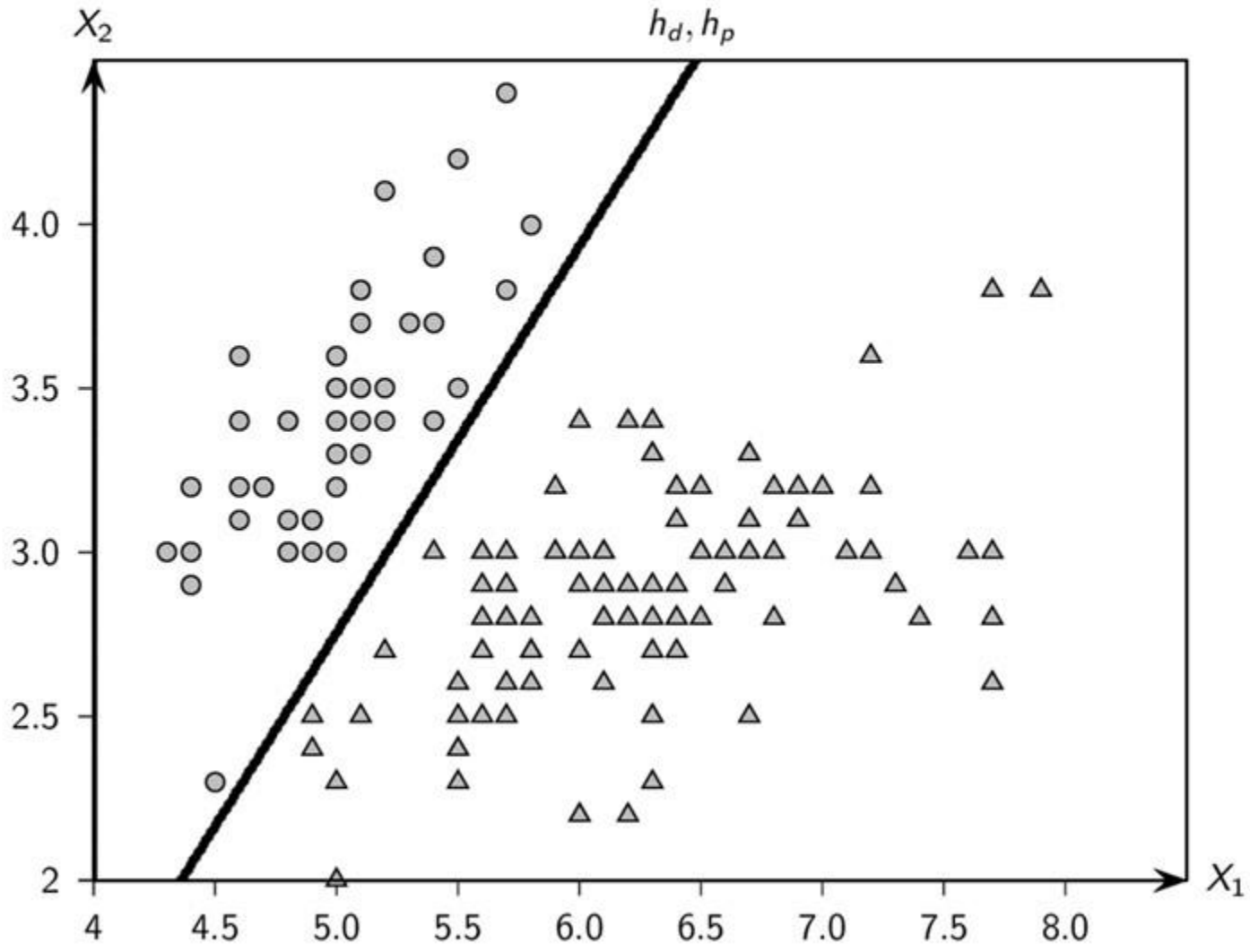
11

12 $\quad \boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta_t \boldsymbol{H}^{-1} \nabla$ // Newton update rule

13 **until** $\|\boldsymbol{w}_t - \boldsymbol{w}_{t-1}\| \leq \epsilon$

SVMs: Dual and Primal Solutions

c_1: `Iris-setosa` (circles) and c_2: other types of Iris flowers (triangles)



SVM Primal Kernel Algorithm: Newton Optimization

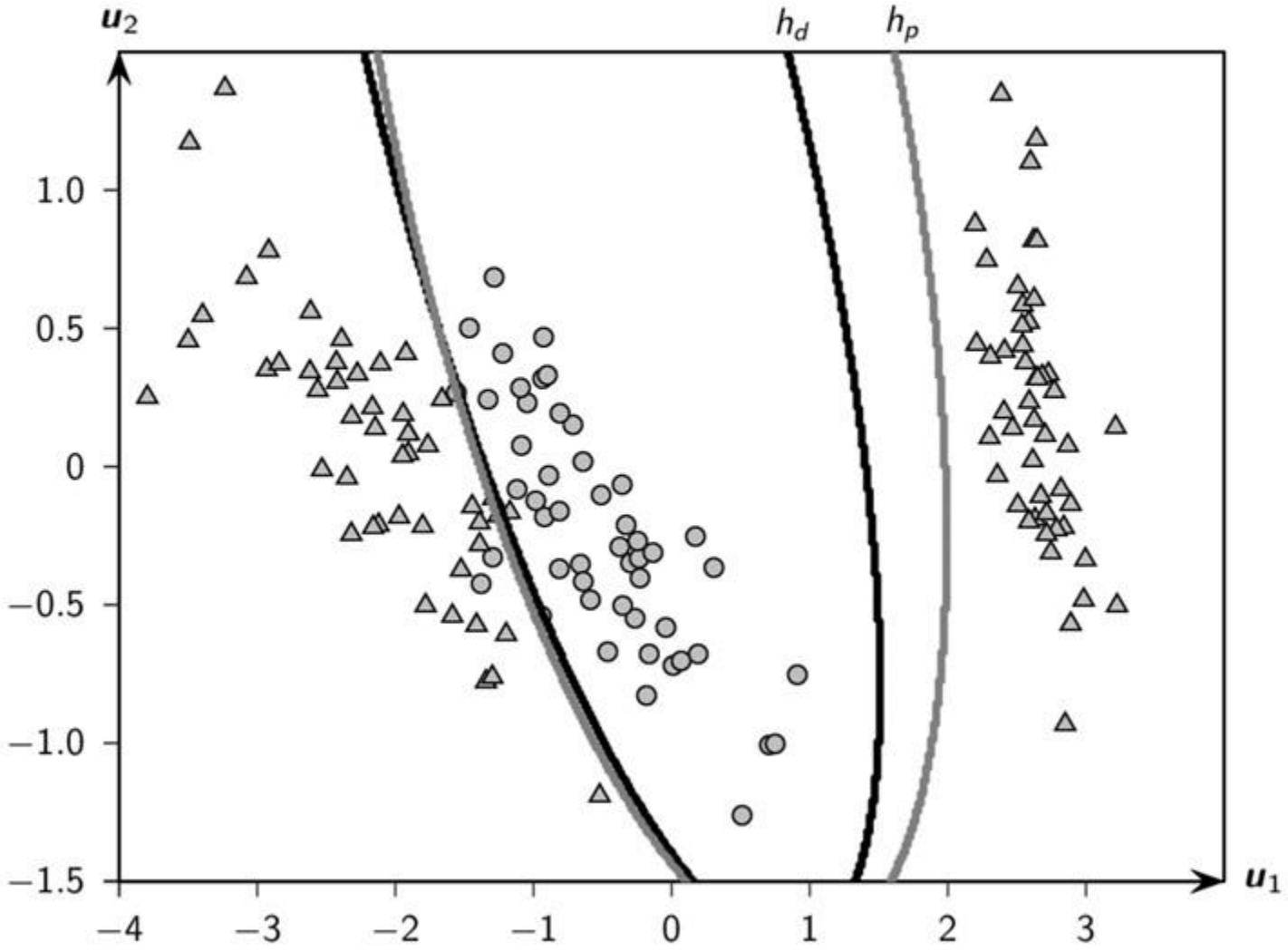
The linear soft margin primal algorithm, with quadratic loss, can easily be extended to work on any kernel matrix $\boldsymbol{K}$:

SVM-Primal-Kernel $(\boldsymbol{D}, K, C, \epsilon)$**:**

foreach $\boldsymbol{x}_i \in \boldsymbol{D}$ **do**
  $\boldsymbol{x}_i \leftarrow \begin{pmatrix} \boldsymbol{x}_i \\ 1 \end{pmatrix}$ // map to $\mathbb{R}^{d+1}$
$\boldsymbol{K} \leftarrow \{K(\boldsymbol{x}_i, \boldsymbol{x}_j)\}_{i,j=1,\ldots,n}$ // compute kernel matrix
$t \leftarrow 0$
$\boldsymbol{\beta}_0 \leftarrow (0, \ldots, 0)^T$ // initialize $\boldsymbol{\beta}_t \in \mathbb{R}^n$
repeat
  $\boldsymbol{v} \leftarrow \sum_{y_i(\boldsymbol{K}_i^T \boldsymbol{\beta}_t) < 1} y_i \boldsymbol{K}_i$
  $\boldsymbol{S} \leftarrow \sum_{y_i(\boldsymbol{K}_i^T \boldsymbol{\beta}_t) < 1} \boldsymbol{K}_i \boldsymbol{K}_i^T$
  $\nabla \leftarrow (\boldsymbol{K} + 2C\boldsymbol{S})\boldsymbol{\beta}_t - 2C\boldsymbol{v}$ // gradient
  $\boldsymbol{H} \leftarrow \boldsymbol{K} + 2C\boldsymbol{S}$ // Hessian
  $\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta_t \boldsymbol{H}^{-1} \nabla$ // Newton update rule
  $t \leftarrow t + 1$
until $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}_{t-1}\| < \epsilon$

SVM Quadratic Kernel: Dual and Primal Solutions

c_1: `Iris-versicolor` (circles) and c_2: other types of Iris flowers (triangles)



Data mining and Machine learning

Part 22. Classification Assessment

Classification Assessment

A classifier is a model or function M that predicts the class label $\hat{y}$ for a given input example $\boldsymbol{x}$:

$$\hat{y} = M(\boldsymbol{x})$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$ is a point in d-dimensional space and $\hat{y} \in \{c_1, c_2, \ldots, c_k\}$ is its predicted class.

To build the classification model M we need a *training set* of points along with their known classes.

Once the model M has been trained, we assess its performance over a separate *testing set* of points for which we know the true classes.

Finally, the model can be deployed to predict the class for future points whose class we typically do not know.

Classification Performance Measures

Let $\boldsymbol{D}$ be the testing set comprising n points in a d-dimensional space, let $\{c_1, c_2, \ldots, c_k\}$ denote the set of k class labels, and let M be a classifier. For $\boldsymbol{x}_i \in \boldsymbol{D}$, let y_i denote its true class, and let $\hat{y}_i = M(\boldsymbol{x}_i)$ denote its predicted class.

Error Rate: The error rate is the fraction of incorrect predictions for the classifier over the testing set, defined as

$$Error\ Rate = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

where I is an indicator function. Error rate is an estimate of the probability of misclassification. The lower the error rate the better the classifier.

Accuracy: The accuracy of a classifier is the fraction of correct predictions:

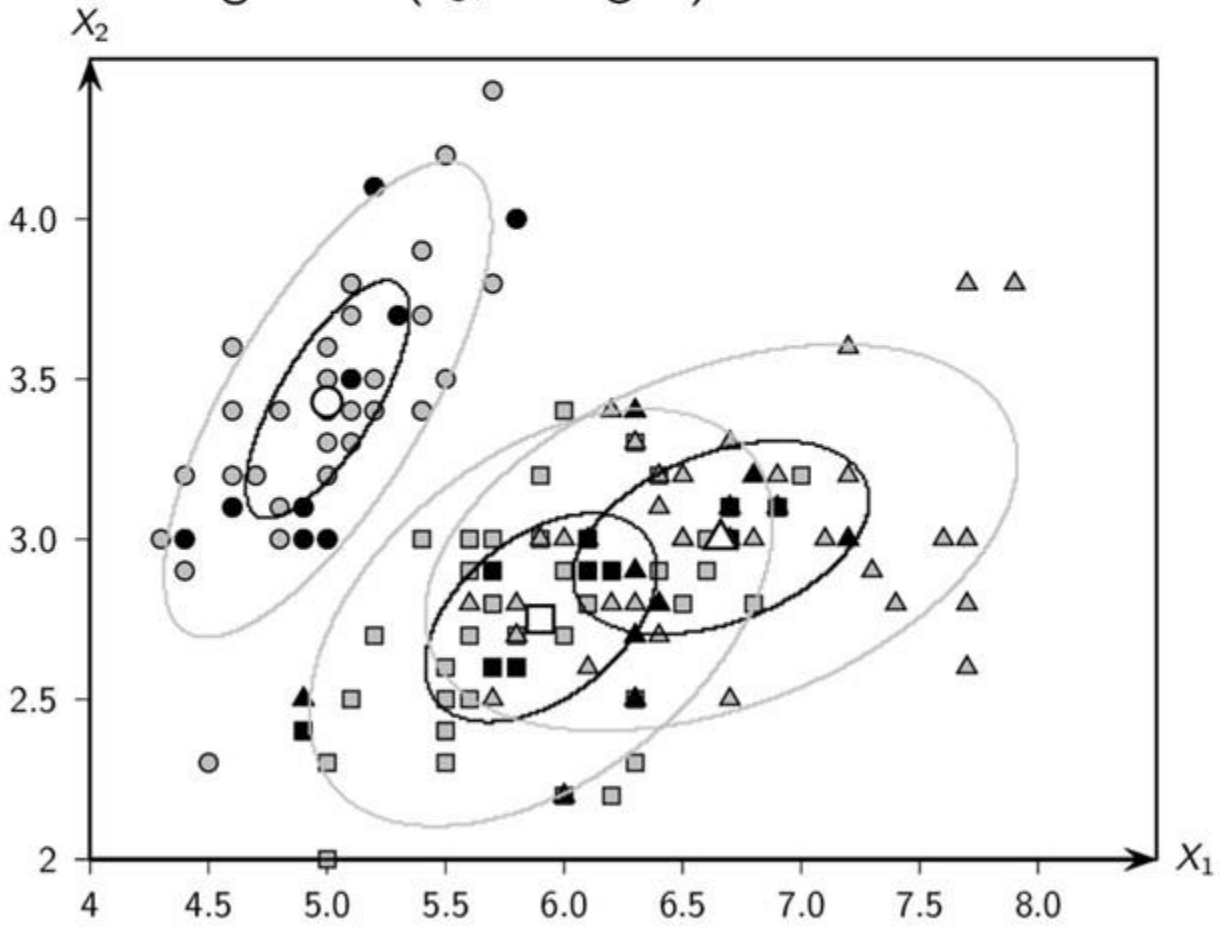
$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i) = 1 - Error\ Rate$$

Accuracy gives an estimate of the probability of a correct prediction; thus, the higher the accuracy, the better the classifier.

Iris Data: Full Bayes Classifier

Training data in grey. Testing data in black.

Three Classes: `Iris-setosa` (c_1; circles), `Iris-versicolor` (c_2; squares) and `Iris-virginica` (c_3; triangles)



Mean (in white) and density contours (1 and 2 standard deviations) shown for each class. The classifier misclassifies 8 out of the 30 test cases. Thus, we have

$$Error\ Rate = \frac{08}{30} = 0.27$$

$$Accuracy = \frac{22}{30} = 0.73$$

Contingency Table–based Measures

Let $\mathcal{D} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_k\}$ denote a partitioning of the testing points based on their true class labels, where $\boldsymbol{D}_j = \{\boldsymbol{x}_i \in \boldsymbol{D} \,|\, y_i = c_j\}$. Let $n_i = |\boldsymbol{D}_i|$ denote the size of true class c_i.

Let $\mathcal{R} = \{\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_k\}$ denote a partitioning of the testing points based on the predicted labels, that is, $\boldsymbol{R}_j = \{\boldsymbol{x}_i \in \boldsymbol{D} \,|\, \hat{y}_i = c_j\}$. Let $m_j = |\boldsymbol{R}_j|$ denote the size of the predicted class c_j.

$\mathcal{R}$ and $\mathcal{D}$ induce a $k \times k$ contingency table $\boldsymbol{N}$, also called a *confusion matrix*, defined as follows:

$$\boldsymbol{N}(i,j) = n_{ij} = |\boldsymbol{R}_i \cap \boldsymbol{D}_j| = \left|\left\{\boldsymbol{x}_a \in \boldsymbol{D} \,|\, \hat{y}_a = c_i \text{ and } y_a = c_j\right\}\right|$$

where $1 \leq i, j \leq k$. The count n_{ij} denotes the number of points with predicted class c_i whose true label is c_j. Thus, n_{ii} (for $1 \leq i \leq k$) denotes the number of cases where the classifier agrees on the true label c_i. The remaining counts n_{ij}, with $i \neq j$, are cases where the classifier and true labels disagree.

Accuracy/Precision and Coverage/Recall

The class-specific *accuracy* or *precision* of the classifier M for class c_i is given as the fraction of correct predictions over all points predicted to be in class c_i

$$acc_i = prec_i = \frac{n_{ii}}{m_i}$$

where m_i is the number of examples predicted as c_i by classifier M. The higher the accuracy on class c_i the better the classifier. The overall precision or accuracy of the classifier is the weighted average of class-specific accuracies:

$$Accuracy = Precision = \sum_{i=1}^{k} \left(\frac{m_i}{n}\right) acc_i = \frac{1}{n} \sum_{i=1}^{k} n_{ii}$$

The class-specific *coverage* or *recall* of M for class c_i is the fraction of correct predictions over all points in class c_i:

$$coverage_i = recall_i = \frac{n_{ii}}{n_i}$$

The higher the coverage the better the classifier.

F-measure

The *class-specific F-measure* tries to balance the precision and recall values, by computing their harmonic mean for class c_i:

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot prec_i \cdot recall_i}{prec_i + recall_i} = \frac{2\, n_{ii}}{n_i + m_i}$$

The higher the F_i value the better the classifier.

The overall *F-measure* for the classifier M is the mean of the class-specific values:

$$F = \frac{1}{k} \sum_{i=1}^{r} F_i$$

For a perfect classifier, the maximum value of the F-measure is 1.

Contingency Table for Iris: Full Bayes Classifier

	True			
Predicted	Iris-setosa (c_1)	Iris-versicolor (c_2)	Iris-virginica(c_3)	
Iris-setosa (c_1)	10	0	0	$m_1 = 10$
Iris-versicolor (c_2)	0	7	5	$m_2 = 12$
Iris-virginica (c_3)	0	3	5	$m_3 = 8$
	$n_1 = 10$	$n_2 = 10$	$n_3 = 10$	$n = 30$

The class-specific precison, recall and F-measure values are:

$$prec_1 = \frac{n_{11}}{m_1} = \frac{10}{10} = 1.0 \qquad recall_1 = \frac{n_{11}}{n_1} = \frac{10}{10} = 1.0 \qquad F_1 = \frac{2 \cdot n_{11}}{(n_1 + m_1)} = \frac{20}{20} = 1.0$$

$$prec_2 = \frac{n_{22}}{m_2} = \frac{7}{12} = 0.583 \qquad recall_2 = \frac{n_{22}}{n_2} = \frac{7}{10} = 0.7 \qquad F_2 = \frac{2 \cdot n_{22}}{(n_2 + m_2)} = \frac{14}{22} = 0.636$$

$$prec_3 = \frac{n_{33}}{m_3} = \frac{5}{8} = 0.625 \qquad recall_3 = \frac{n_{33}}{n_3} = \frac{5}{10} = 0.5 \qquad F_3 = \frac{2 \cdot n_{33}}{(n_3 + m_3)} = \frac{10}{18} = 0.556$$

The overall accuracy and F-measure is

$$Accuracy = \frac{(n_{11} + n_{22} + n_{33})}{n} = \frac{(10+7+5)}{30} = 22/30 = 0.733$$

$$F = \frac{1}{3}(1.0 + 0.636 + 0.556) = \frac{2.192}{3} = 0.731$$

Binary Classification: Positive and Negative Class

When there are only $k = 2$ classes, we call class c_1 the positive class and c_2 the negative class. The entries of the resulting 2×2 confusion matrix are

	True Class	
Predicted Class	Positive (c_1)	Negative (c_2)
Positive (c_1)	True Positive (TP)	False Positive (FP)
Negative (c_2)	False Negative (FN)	True Negative (TN)

Binary Classification: Positive and Negative Class

- *True Positives (TP):* The number of points that the classifier correctly predicts as positive:

$$TP = n_{11} = \left|\{\boldsymbol{x}_i \,|\hat{y}_i = y_i = c_1\}\right|$$

- *False Positives (FP):* The number of points the classifier predicts to be positive, which in fact belong to the negative class:

$$FP = n_{12} = \left|\{\boldsymbol{x}_i \,|\hat{y}_i = c_1 \text{ and } y_i = c_2\}\right|$$

- *False Negatives (FN):* The number of points the classifier predicts to be in the negative class, which in fact belong to the positive class:

$$FN = n_{21} = \left|\{\boldsymbol{x}_i \,|\hat{y}_i = c_2 \text{ and } y_i = c_1\}\right|$$

- *True Negatives (TN):* The number of points that the classifier correctly predicts as negative:

$$TN = n_{22} = \left|\{\boldsymbol{x}_i \,|\hat{y}_i = y_i = c_2\}\right|$$

Binary Classification: Assessment Measures

Error Rate: The error rate for the binary classification case is given as the fraction of mistakes (or false predictions):

$$Error\ Rate = \frac{FP + FN}{n}$$

Accuracy: The accuracy is the fraction of correct predictions:

$$Accuracy = \frac{TP + TN}{n}$$

The precision for the positive and negative class is given as

$$prec_P = \frac{TP}{TP + FP} = \frac{TP}{m_1}$$
$$prec_N = \frac{TN}{TN + FN} = \frac{TN}{m_2}$$

where $m_i = |\boldsymbol{R}_i|$ is the number of points predicted by M as having class c_i.

Binary Classification: Assessment Measures

Sensitivity or True Positive Rate: The fraction of correct predictions with respect to all points in the positive class, i.e., the recall for the positive class

$$TPR = sensitivity = recall_P = \frac{TP}{TP + FN} = \frac{TP}{n_1}$$

where n_1 is the size of the positive class.

Specificity or True Negative Rate: The recall for the negative class:

$$TNR = specificity = recall_N = \frac{TN}{FP + TN} = \frac{TN}{n_2}$$

where n_2 is the size of the negative class.

Binary Classification: Assessment Measures

False Negative Rate: Defined as

$$FNR = \frac{FN}{TP + FN} = \frac{FN}{n_1} = 1 - sensitivity$$

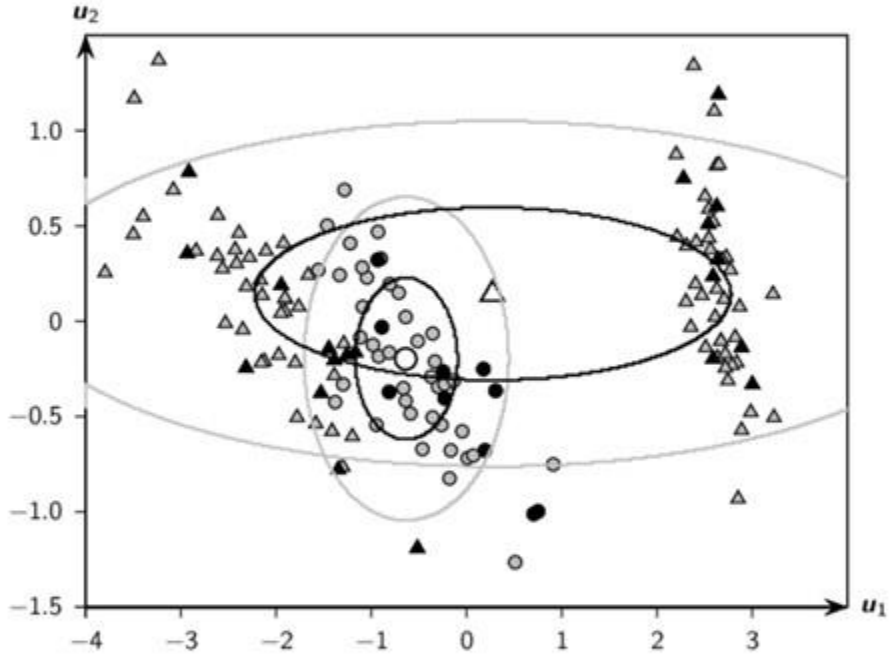
False Positive Rate: Defined as

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{n_2} = 1 - specif icity$$

Iris Principal Components Data: Naive Bayes Classifier

`Iris-versicolor` (c_1 - circles) and other two Irises (c_2 - triangles).

Training data (80%) in grey and testing data (20%) in black.



$$\hat{P}(c_1) = 40/120 = 0.33$$

$$\hat{\mu}_1 = \begin{pmatrix} -0.641 & -0.204 \end{pmatrix}^T$$

$$\hat{\Sigma}_1 = \begin{pmatrix} 0.29 & 0 \\ 0 & 0.18 \end{pmatrix}$$

$$\hat{P}(c_2) = 80/120 = 0.67$$

$$\hat{\mu}_2 = \begin{pmatrix} 0.27 & 0.14 \end{pmatrix}^T$$

$$\hat{\Sigma}_2 = \begin{pmatrix} 6.14 & 0 \\ 0 & 0.206 \end{pmatrix}$$

The mean (in white) and the contour plot of the normal distribution for each class are shown; the contours are shown for one and two standard deviations along each axis.

Iris PC Data: Assessment Measures

	True		
Predicted	Positive (c_1)	Negative (c_2)	
Positive (c_1)	$TP = 7$	$FP = 7$	$m_1 = 14$
Negative (c_2)	$FN = 3$	$TN = 13$	$m_2 = 16$
	$n_1 = 10$	$n_2 = 20$	$n = 30$

The naive Bayes classifier misclassified 10 out of the 30 test instances, resulting in an error rate and accuracy of

$$Error\ Rate = 10/30 = 0.33 \qquad Accuracy = 20/30 = 0.67$$

Other performance measures:

$$prec_P = \frac{TP}{TP + FP} = \frac{7}{14} = 0.5 \qquad prec_N = \frac{TN}{TN + FN} = \frac{13}{16} = 0.812$$

$$recall_P(sensitivity) = \frac{TP}{TP + FN} = \frac{7}{10} = 0.7 \qquad recall_N = specif\,icity = \frac{TN}{TN + FP} = \frac{13}{20} = 0.65$$

$$FNR = 1 - sensitivity = 0.3 \qquad FPR = 1 - specif\,icity = 0.35$$

ROC Analysis

Receiver Operating Characteristic (ROC) analysis is a popular strategy for assessing the performance of classifiers when there are two classes.

ROC analysis requires that a classifier output a score value for the positive class for each point in the testing set. These scores can then be used to order points in decreasing order.

Typically, a binary classifier chooses some positive score threshold ρ, and classifies all points with score above ρ as positive, with the remaining points classified as negative.

ROC analysis plots the performance of the classifier over all possible values of the threshold parameter ρ.

In particular, for each value of ρ, it plots the false positive rate (1-specificity) on the x-axis versus the true positive rate (sensitivity) on the y-axis. The resulting plot is called the *ROC curve* or *ROC plot* for the classifier.

ROC Analysis

Let $S(\boldsymbol{x}_i)$ denote the real-valued score for the positive class output by a classifier M for the point $\boldsymbol{x}_i$. Let the maximum and minimum score thresholds observed on testing dataset $\boldsymbol{D}$ be as follows:

$$\rho^{\min} = \min_i\{S(\boldsymbol{x}_i)\} \qquad \rho^{\max} = \max_i\{S(\boldsymbol{x}_i)\}$$

Initially, we classify all points as negative. Both TP and FP are thus initially zero, as given in the confusion matrix:

	True	
Predicted	Pos	Neg
Pos	0	0
Neg	FN	TN

This results in TPR and FPR rates of zero, which correspond to the point $(0, 0)$ at the lower left corner in the ROC plot.

ROC Analysis

Next, for each distinct value of ρ in the range $[\rho^{\min}, \rho^{\max}]$, we tabulate the set of positive points:

$$\boldsymbol{R}_1(\rho) = \{\boldsymbol{x}_i \in \boldsymbol{D} : S(\boldsymbol{x}_i) > \rho\}$$

and we compute the corresponding true and false positive rates, to obtain a new point in the ROC plot.
Finally, in the last step, we classify all points as positive. Both *FN* and *TN* are thus zero, as per the confusion matrix

	True	
Predicted	Pos	Neg
Pos	*TP*	*FP*
Neg	0	0

resulting in *TPR* and *FPR* values of 1. This results in the point $(1,1)$ at the top right-hand corner in the ROC plot.

ROC Analysis

An ideal classifier corresponds to the top left point $(0,1)$, which corresponds to the case $FPR = 0$ and $TPR = 1$, that is, the classifier has no false positives, and identifies all true positives (as a consequence, it also correctly predicts all the points in the negative class). This case is shown in the confusion matrix:

	True	
Predicted	Pos	Neg
Pos	TP	0
Neg	0	TN

A classifier with a curve closer to the ideal case, that is, closer to the upper left corner, is a better classifier.

Area Under ROC Curve: The area under the ROC curve, abbreviated AUC, can be used as a measure of classifier performance. The AUC value is essentially the probability that the classifier will rank a random positive test case higher than a random negative test instance.

ROC: Different Cases for 2×2 Confusion Matrix

	True	
Predicted	Pos	Neg
Pos	0	0
Neg	*FN*	*TN*

(a) Initial: all negative

	True	
Predicted	Pos	Neg
Pos	*TP*	*FP*
Neg	0	0

(b) Final: all positive

	True	
Predicted	Pos	Neg
Pos	*TP*	0
Neg	0	*TN*

(c) Ideal classifier

Random Classifier

A random classifier corresponds to a diagonal line in the ROC plot.

Consider a classifier that randomly guesses the class of a point as positive half the time, and negative the other half. We then expect that half of the true positives and true negatives will be identified correctly, resulting in the point $(TPR, FPR) = (0.5, 0.5)$ for the ROC plot.

In general, any fixed probability of prediction, say r, for the positive class, yields the point (r, r) in ROC space.

The diagonal line thus represents the performance of a random classifier, over all possible positive class prediction thresholds r.

ROC/AUC Algorithm

The ROC/AUC takes as input the testing set $\boldsymbol{D}$, and the classifier M.

The first step is to predict the score $S(\boldsymbol{x}_i)$ for the positive class (c_1) for each test point $\boldsymbol{x}_i \in \boldsymbol{D}$. Next, we sort the $(S(\boldsymbol{x}_i), y_i)$ pairs, that is, the score and the true class pairs, in decreasing order of the scores

Initially, we set the positive score threshold $\rho = \infty$. We then examine each pair $(S(\boldsymbol{x}_i), y_i)$ in sorted order, and for each distinct value of the score, we set $\rho = S(\boldsymbol{x}_i)$ and plot the point

$$(FPR, TPR) = \left(\frac{FP}{n_2}, \frac{TP}{n_1} \right)$$

As each test point is examined, the true and false positive values are adjusted based on the true class y_i for the test point $\boldsymbol{x}_i$. If $y_1 = c_1$, we increment the true positives, otherwise, we increment the false positives

ROC/AUC Algorithm

The AUC value is computed as each new point is added to the ROC plot. The algorithm maintains the previous values of the false and true positives, FP_{prev} and TP_{prev}, for the previous score threshold ρ.

Given the current FP and TP values, we compute the area under the curve defined by the four points

$$(x_1, y_1) = \left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1} \right) \qquad (x_2, y_2) = \left(\frac{FP}{n_2}, \frac{TP}{n_1} \right)$$

$$(x_1, 0) = \left(\frac{FP_{prev}}{n_2}, 0 \right) \qquad (x_2, 0) = \left(\frac{FP}{n_2}, 0 \right)$$

These four points define a trapezoid, whenever $x_2 > x_1$ and $y_2 > y_1$, otherwise, they define a rectangle (which may be degenerate, with zero area).

The area under the trapezoid is given as $b \cdot h$, where $b = |x_2 - x_1|$ is the length of the base of the trapezoid and $h = \frac{1}{2}(y_2 + y_1)$ is the average height of the trapezoid.

Algorithm ROC-Curve

ROC-Curve(D, M):

1 $n_1 \leftarrow \left|\{\boldsymbol{x}_i \in \boldsymbol{D} | y_i = c_1\}\right|$ // size of positive class
2 $n_2 \leftarrow \left|\{\boldsymbol{x}_i \in \boldsymbol{D} | y_i = c_2\}\right|$ // size of negative class
// classify, score, and sort all test points
3 $L \leftarrow$ sort the set $\{(S(\boldsymbol{x}_i), y_i) : \boldsymbol{x}_i \in \boldsymbol{D}\}$ by decreasing scores
4 $FP \leftarrow TP \leftarrow 0$
5 $FP_{prev} \leftarrow TP_{prev} \leftarrow 0$
6 $AUC \leftarrow 0$
7 $\rho \leftarrow \infty$
8 **foreach** $(S(\boldsymbol{x}_i), y_i) \in L$ **do**
9 **if** $\rho > S(\boldsymbol{x}_i)$ **then**
10 plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$
11 $AUC \leftarrow AUC + \text{Trapezoid-Area}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$
12 $\rho \leftarrow S(\boldsymbol{x}_i)$
13 $FP_{prev} \leftarrow FP$
14 $TP_{prev} \leftarrow TP$
15 **if** $y_i = c_1$ **then** $TP \leftarrow TP + 1$
16 **else** $FP \leftarrow FP + 1$
17 plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$
18 $AUC \leftarrow AUC + \text{Trapezoid-Area}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$

Algorithm Trapezoid-Area

Trapezoid-Area$((x_1, y_1), (x_2, y_2))$**:**

1 $b \leftarrow |x_2 - x_1|$ `// base of trapezoid`
2 $h \leftarrow \frac{1}{2}(y_2 + y_1)$ `// average height of trapezoid`
3 **return** $(b \cdot h)$

Iris PC Data: ROC Analysis

We use the naive Bayes classifier to compute the probability that each test point belongs to the positive class (c_1; `iris-versicolor`).

The score of the classifier for test point $\boldsymbol{x}_i$ is therefore $S(\boldsymbol{x}_i) = P(c_1|\boldsymbol{x}_i)$. The sorted scores (in decreasing order) along with the true class labels are as follows:

$S(\boldsymbol{x}_i)$	0.93	0.82	0.80	0.77	0.74	0.71	0.69	0.67	0.66	0.61
y_i	c_2	c_1	c_2	c_1	c_1	c_1	c_2	c_1	c_2	c_2

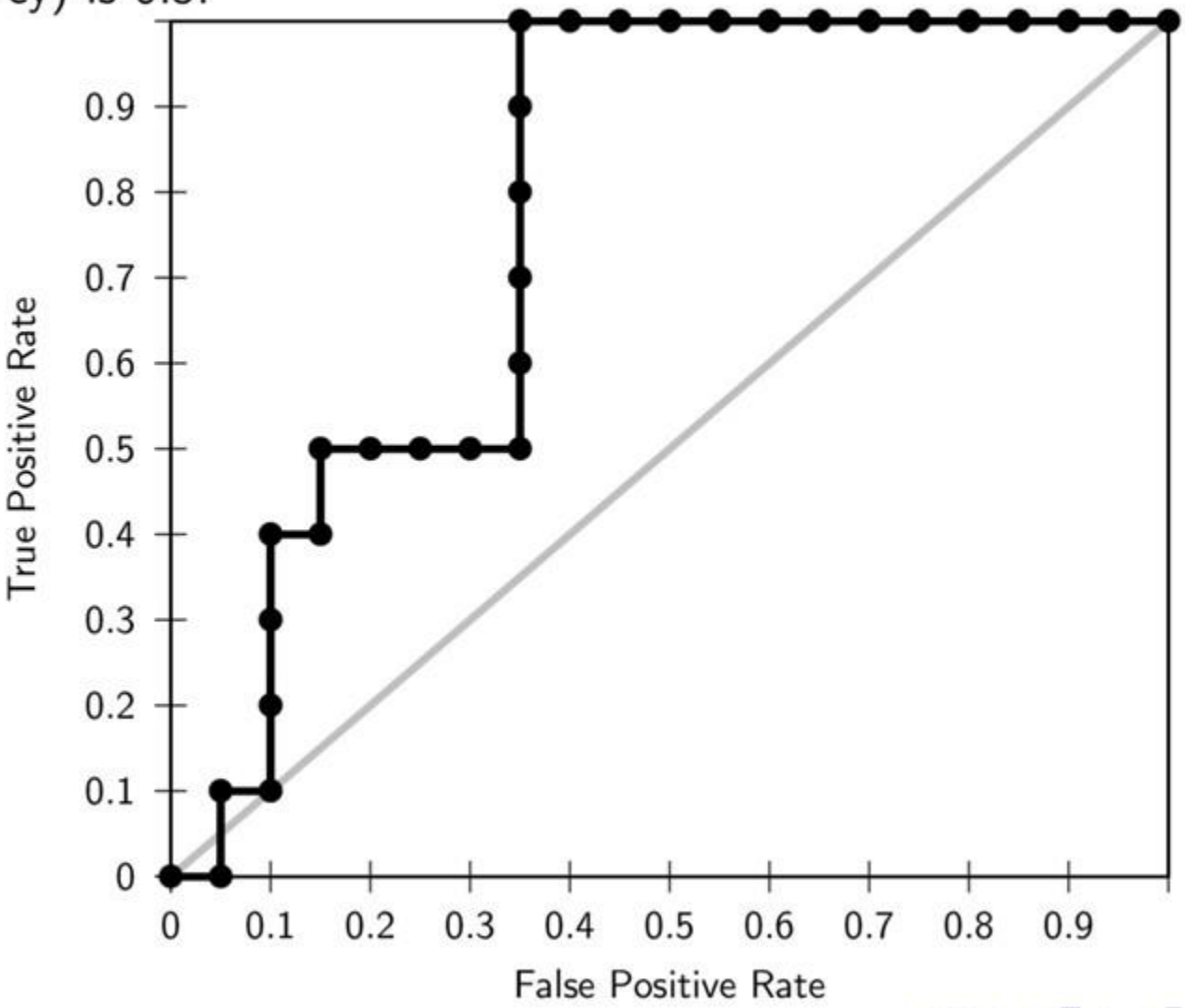
$S(\boldsymbol{x}_i)$	0.59	0.55	0.55	0.53	0.47	0.30	0.26	0.11	0.04	2.97e-03
y_i	c_2	c_2	c_1	c_1	c_1	c_1	c_1	c_2	c_2	c_2

$S(\boldsymbol{x}_i)$	1.28e-03	2.55e-07	6.99e-08	3.11e-08	3.109e-08
y_i	c_2	c_2	c_2	c_2	c_2

$S(\boldsymbol{x}_i)$	1.53e-08	9.76e-09	2.08e-09	1.95e-09	7.83e-10
y_i	c_2	c_2	c_2	c_2	c_2

ROC Plot for Iris PC Data

AUC for naive Bayes is 0.775, whereas the AUC for the random classifier (ROC plot in grey) is 0.5.



ROC Plot and AUC: Trapezoid Region

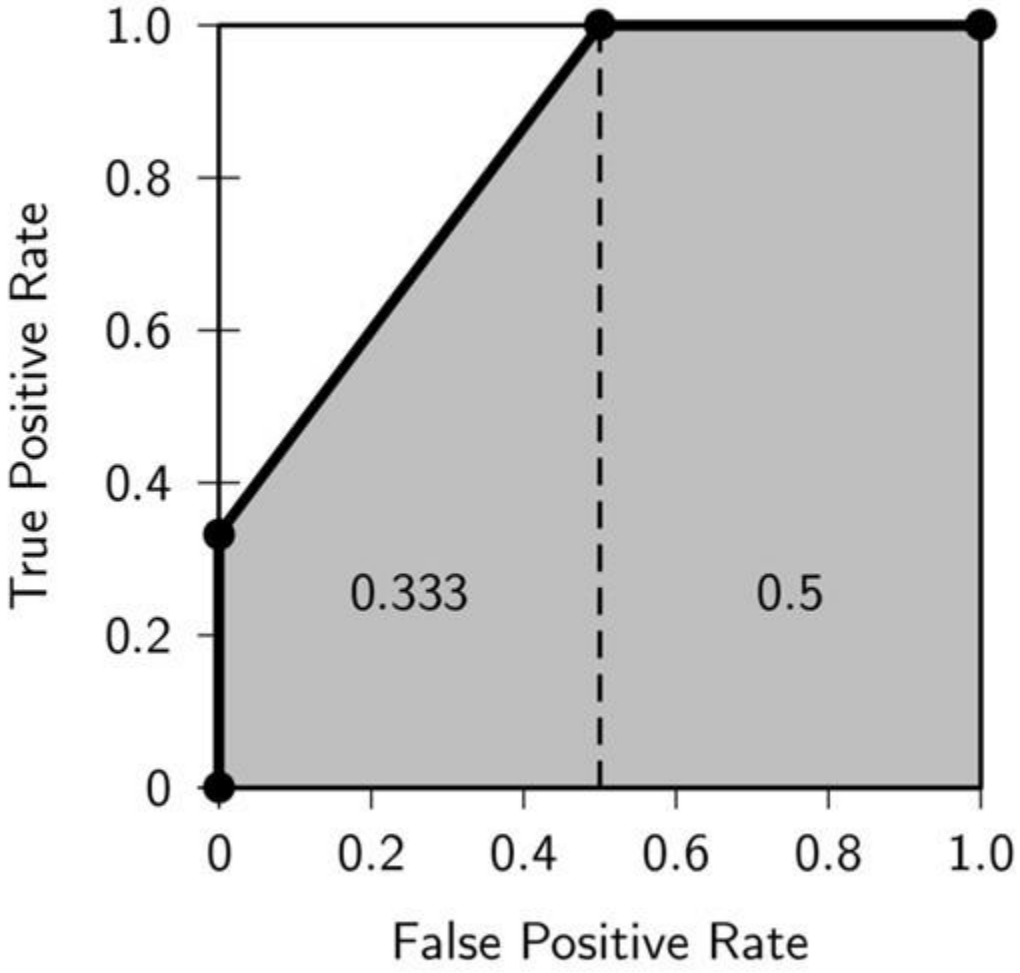
Consider the following sorted scores, along with the true class, for some testing dataset with $n = 5$, $n_1 = 3$ and $n_2 = 2$.

$$(0.9, c_1), (0.8, c_2), (0.8, c_1), (0.8, c_1), (0.1, c_2)$$

Algorithm yields the following points that are added to the ROC plot, along with the running AUC:

ρ	FP	TP	(FPR, TPR)	AUC
∞	0	0	$(0, 0)$	0
0.9	0	1	$(0, 0.333)$	0
0.8	1	3	$(0.5, 1)$	0.333
0.1	2	3	$(1, 1)$	0.833

ROC Plot and AUC: Trapezoid Region



Classifier Evaluation

Consider a classifier M, and some performance measure θ. Typically, the input dataset $\boldsymbol{D}$ is randomly split into a disjoint training set and testing set. The training set is used to learn the model M, and the testing set is used to evaluate the measure θ.

How confident can we be about the classification performance? The results may be due to an artifact of the random split.

Also $\boldsymbol{D}$ is itself a d-dimensional multivariate random sample drawn from the true (unknown) joint probability density function $f(\boldsymbol{x})$ that represents the population of interest. Ideally, we would like to know the expected value $E[\theta]$ of the performance measure over all possible testing sets drawn from f. However, because f is unknown, we have to estimate $E[\theta]$ from $\boldsymbol{D}$.

Cross-validation and resampling are two common approaches to compute the expected value and variance of a given performance measure.

K-fold Cross-Validation

Cross-validation divides the dataset $\boldsymbol{D}$ into K equal-sized parts, called *folds*, namely $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_K$.

Each fold $\boldsymbol{D}_i$ is, in turn, treated as the testing set, with the remaining folds comprising the training set $\boldsymbol{D} \setminus \boldsymbol{D}_i = \bigcup_{j \neq i} \boldsymbol{D}_j$.

After training the model M_i on $\boldsymbol{D} \setminus \boldsymbol{D}_i$, we assess its performance on the testing set $\boldsymbol{D}_i$ to obtain the i-th estimate θ_i.

The expected value of the performance measure can then be estimated as

$$\hat{\mu}_\theta = E[\theta] = \frac{1}{K} \sum_{i=1}^{K} \theta_i$$

and its variance as

$$\hat{\sigma}^2_\theta = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$$

Usually K is chosen to be 5 or 10. The special case, when $K = n$, is called *leave-one-out* cross-validation.

K-fold Cross-Validation Algorithm

Cross-Validation(K, $\boldsymbol{D}$):

1 $\boldsymbol{D} \leftarrow$ randomly shuffle $\boldsymbol{D}$
2 $\{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_K\} \leftarrow$ partition $\boldsymbol{D}$ in K equal parts
3 **foreach** $i \in [1, K]$ **do**
4 $\quad M_i \leftarrow$ train classifier on $\boldsymbol{D} \setminus \boldsymbol{D}_i$
5 $\quad \theta_i \leftarrow$ assess M_i on $\boldsymbol{D}_i$
6 $\hat{\mu}_\theta = \frac{1}{K} \sum_{i=1}^{K} \theta_i$
7 $\hat{\sigma}^2_\theta = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$
8 **return** $\hat{\mu}_\theta, \hat{\sigma}^2_\theta$

2D Iris Dataset: K-fold Cross-Validation

Consider the 2D Iris dataset with $k = 3$ classes. We assess the error rate of the full Bayes classifier via 5-fold cross-validation, obtaining the following error rates when testing on each fold:

$$\theta_1 = 0.267 \qquad \theta_2 = 0.133 \qquad \theta_3 = 0.233 \qquad \theta_4 = 0.367 \qquad \theta_5 = 0.167$$

The mean and variance for the error rate are as follows:

$$\hat{\mu}_\theta = \frac{1.167}{5} = 0.233 \qquad \hat{\sigma}^2_\theta = 0.00833$$

Performing ten 5-fold cross-validation runs for the Iris dataset results in the mean of the expected error rate as 0.232, and the mean of the variance as 0.00521, with the variance in both these estimates being less than 10^{-3}.

Bootstrap Resampling

The bootstrap method draws K random samples of size n *with replacement* from $\boldsymbol{D}$.Each sample $\boldsymbol{D}_i$ is thus the same size as $\boldsymbol{D}$, and has several repeated points.

The probability that a particular point $\boldsymbol{x}_j$ is not selected even after n tries is given as

$$P(\boldsymbol{x}_j \notin \boldsymbol{D}_i) = q^n = \left(1 - \frac{1}{n}\right)^n \simeq e^{-1} = 0.368$$

which implies that each bootstrap sample contains approximately 63.2% of the points from $\boldsymbol{D}$.

The bootstrap samples can be used to evaluate the classifier by training it on each of samples $\boldsymbol{D}_i$ and then using the full input dataset $\boldsymbol{D}$ as the testing set.

However, the estimated mean and variance of θ will be somewhat optimistic owing to the fairly large overlap between the training and testing datasets (63.2%).

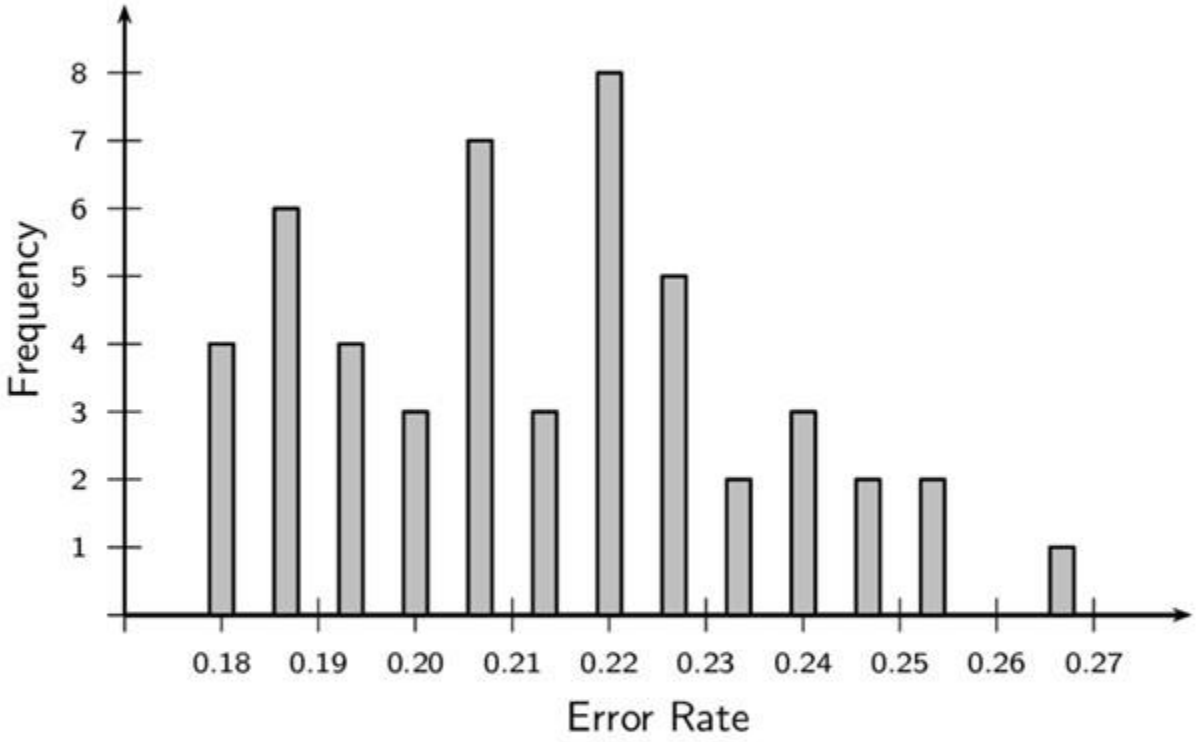
Bootstrap Resampling Algorithm

Bootstrap-Resampling(K, $\boldsymbol{D}$):

1 **for** $i \in [1, K]$ **do**
2 $\quad \boldsymbol{D}_i \leftarrow$ sample of size n with replacement from $\boldsymbol{D}$
3 $\quad M_i \leftarrow$ train classifier on $\boldsymbol{D}_i$
4 $\quad \theta_i \leftarrow$ assess M_i on $\boldsymbol{D}$
5 $\hat{\mu}_\theta = \frac{1}{K} \sum_{i=1}^{K} \theta_i$
6 $\hat{\sigma}^2_\theta = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$
7 **return** $\hat{\mu}_\theta, \hat{\sigma}^2_\theta$

Iris 2D Data: Bootstrap Resampling using Error Rate

We apply bootstrap sampling to estimate the error rate for the full Bayes classifier, using $K = 50$ samples. The sampling distribution of error rates is:



The expected value and variance of the error rate are

$$\hat{\mu}_\theta = 0.213 \qquad \hat{\sigma}^2_\theta = 4.815 \times 10^{-4}$$

Confidence Intervals

We would like to derive confidence bounds on how much the estimated mean and variance may deviate from the true value.

To answer this question we make use of the central limit theorem, which states that the sum of a large number of independent and identically distributed (IID) random variables has approximately a normal distribution, regardless of the distribution of the individual random variables.

Let $\theta_1, \theta_2, \ldots, \theta_K$ be a sequence of IID random variables, representing, for example, the error rate or some other performance measure over the K-folds in cross-validation or K bootstrap samples.

Assume that each θ_i has a finite mean $E[\theta_i] = \mu$ and finite variance $var(\theta_i) = \sigma^2$.

Let $\hat{\mu}$ denote the sample mean:

$$\hat{\mu} = \frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)$$

Confidence Intervals

By linearity of expectation, we have

$$E[\hat{\mu}] = E\left[\frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)\right] = \frac{1}{K}\sum_{i=1}^{K} E[\theta_i] = \frac{1}{K}(K\mu) = \mu$$

The variance of $\hat{\mu}$ is given as

$$var(\hat{\mu}) = var\left(\frac{1}{K}(\theta_1 + \theta_2 + \cdots + \theta_K)\right) = \frac{1}{K^2}\sum_{i=1}^{K} var(\theta_i) = \frac{1}{K^2}(K\sigma^2) = \frac{\sigma^2}{K}$$

Thus, the standard deviation of $\hat{\mu}$ is given as

$$std(\hat{\mu}) = \sqrt{var(\hat{\mu})} = \frac{\sigma}{\sqrt{K}}$$

We are interested in the distribution of the z-score of $\hat{\mu}$, which is itself a random variable

$$Z_K = \frac{\hat{\mu} - E[\hat{\mu}]}{std(\hat{\mu})} = \frac{\hat{\mu} - \mu}{\frac{\sigma}{\sqrt{K}}} = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\sigma}\right)$$

Z_K specifies the deviation of the estimated mean from the true mean in terms of its standard deviation.

Confidence Intervals

The central limit theorem states that as the sample size increases, the random variable Z_K *converges in distribution* to the standard normal distribution (which has mean 0 and variance 1). That is, as $K \to \infty$, for any $x \in \mathbb{R}$, we have

$$\lim_{K \to \infty} P(Z_K \leq x) = \Phi(x)$$

where $\Phi(x)$ is the cumulative distribution function for the standard normal density function $f(x|0,1)$.

Let $z_{\alpha/2}$ denote the z-score value that encompasses $\alpha/2$ of the probability mass for a standard normal distribution, that is,

$$P(0 \leq Z_K \leq z_{\alpha/2}) = \Phi(z_{\alpha/2}) - \Phi(0) = \alpha/2$$

then, because the normal distribution is symmetric about the mean, we have

$$\lim_{K \to \infty} P(-z_{\alpha/2} \leq Z_K \leq z_{\alpha/2}) = 2 \cdot P(0 \leq Z_K \leq z_{\alpha/2}) = \alpha$$

Confidence Intervals

Note that

$$-z_{\alpha/2} \leq Z_K \leq z_{\alpha/2} \text{ implies that } \left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) \leq \mu \leq \left(\hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)$$

We obtain bounds on the value of the true mean μ in terms of the estimated value $\hat{\mu}$:

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \leq \mu \leq \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) = 1 - \alpha$$

Thus, for any given *level of confidence* α, we can compute the probability that the true mean μ lies in the $\alpha\%$ confidence interval $\left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}}, \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)$.

Confidence Intervals: Unknown Variance

The true variance σ^2 is usually unknown, but we may use the sample variance:

$$\hat{\sigma}^2 = \frac{1}{K}\sum_{i=1}^{K}(\theta_i - \hat{\mu})^2$$

because $\hat{\sigma}^2$ is a *consistent* estimator for σ^2, that is, as $K \to \infty$, $\hat{\sigma}^2$ converges with probability 1, also called *converges almost surely*, to σ^2.

The central limit theorem then states that the random variable Z_K^* defined below converges in distribution to the standard normal distribution:

$$Z_K^* = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\hat{\sigma}}\right)$$

and thus, we have

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}} \leq \mu \leq \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right) = 1 - \alpha$$

$\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}, \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right)$ is the 100(1-α)% confidence interval for μ.

2D Iris Data: Confidence Intervals

Consider the 5-fold cross-validation ($K = 5$) to assess the full Bayes classifier:

$$\hat{\mu}_\theta = 0.233 \qquad \hat{\sigma}^2_\theta = 0.00833 \qquad \hat{\sigma}_\theta = \sqrt{0.00833} = 0.0913$$

Let $1 - \alpha = 0.95$ be the confidence level and $\alpha = 0.05$ the significance level. The standard normal distribution has 95% of the probability density within $z_{\alpha/2} = 1.96$ standard deviations from the mean.

$$P\left(\mu \in \left(\hat{\mu}_\theta - z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}}, \hat{\mu}_\theta + z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}}\right)\right) = 0.95$$

Because $z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = \frac{1.96 \times 0.0913}{\sqrt{5}} = 0.08$, we have

$$P\Big(\mu \in (0.233 - 0.08, 0.233 + 0.08)\Big) = P\Big(\mu \in (0.153, 0.313)\Big) = 0.95$$

With 95% confidence, the true expected error rate lies in the interval $(0.153, 0.313)$. If $\alpha = 0.01$, then $z_{\alpha/2} = 2.58$, $z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = \frac{2.58 \times 0.0913}{\sqrt{5}} = 0.105$, and the interval becomes $(0.128, 0.338)$.

Confidence Intervals: Small Sample Size

The confidence interval applies only when the sample size $K \to \infty$. However, in practice for K-fold cross-validation or bootstrap resampling K is small.

In the small sample case, instead of the normal density to derive the confidence interval, we use the Student's t distribution.

In particular, we choose the value $t_{\alpha/2,K-1}$ such that the cumulative t distribution function with $K-1$ degrees of freedom encompasses $\alpha/2$ of the probability mass, that is,

$$P(0 \leq Z_K^* \leq t_{\alpha/2,K-1}) = T_{K-1}(t_{\alpha/2}) - T_{K-1}(0) = \alpha/2$$

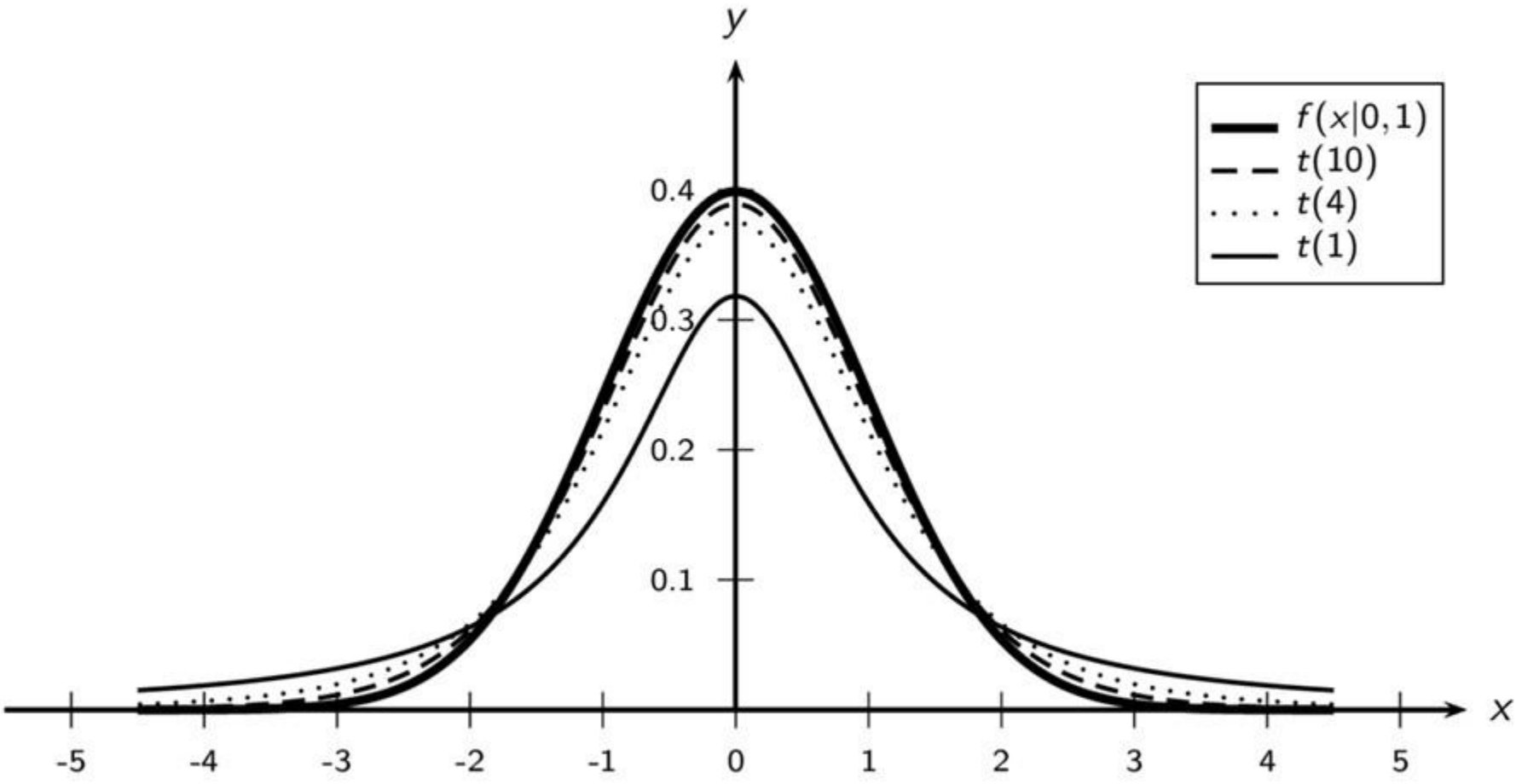
where T_{K-1} is the cumulative distribution function for the Student's t distribution with $K-1$ degrees of freedom.

The 100(1-α%) confidence interval for the true mean μ is thus

$$\left(\hat{\mu} - t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}} \leq \mu \leq \hat{\mu} + t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}}\right)$$

Note the dependence of the interval on both α and the sample size K.

Student's t Distribution: K Degrees of Freedom



Iris 2D Data: Small Sample Confidence Intervals

Due to the small sample size ($K = 5$), we can get a better confidence interval by using the t distribution. For $K - 1 = 4$ degrees of freedom, for $\alpha = 0.95$, we get $t_{\alpha/2,K-1} = 2.776$. Thus,

$$t_{\alpha/2,K-1}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = 2.776 \times \frac{0.0913}{\sqrt{5}} = 0.113$$

The 95% confidence interval is therefore

$$(0.233 - 0.113, 0.233 + 0.113) = (0.12, 0.346)$$

which is much wider than the overly optimistic confidence interval $(0.153, 0.313)$ obtained for the large sample case.
For $1 - \alpha = 0.99$, $t_{\alpha/2} = 4.604$, $t_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = 4.604 \times \frac{0.0913}{\sqrt{5}} = 0.188$ and the 99% confidence interval is

$$(0.233 - 0.188, 0.233 + 0.188) = (0.045, 0.421)$$

This is also much wider than the 99% confidence interval $(0.128, 0.338)$ obtained for the large sample case.

Comparing Classifiers: Paired t-Test

How can we test for a significant difference in the classification performance of two alternative classifiers, M^A and M^B on a given dataset $\boldsymbol{D}$.

We can apply K-fold cross-validation (or bootstrap resampling) and tabulate their performance over each of the K folds, with identical folds for both classifiers. That is, we perform a *paired test*, with both classifiers trained and tested on the same data.

Let $\theta_1^A, \theta_2^A, \ldots, \theta_K^A$ and $\theta_1^B, \theta_2^B, \ldots, \theta_K^B$ denote the performance values for M_A and M_B, respectively. To determine if the two classifiers have different or similar performance, define the random variable δ_i as the difference in their performance on the ith dataset:

$$\delta_i = \theta_i^A - \theta_i^B$$

The expected difference and the variance estimates are given as:

$$\hat{\mu}_\delta = \frac{1}{K}\sum_{i=1}^{K}\delta_i \qquad \hat{\sigma}_\delta^2 = \frac{1}{K}\sum_{i=1}^{K}(\delta_i - \hat{\mu}_\delta)^2$$

Comparing Classifiers: Paired t-Test

The null hypothesis H_0 is that the performance of M^A and M^B is the same. The alternative hypothesis H_a is that they are not the same, that is:

$$H_0: \quad \mu_\delta = 0 \qquad\qquad H_a: \quad \mu_\delta \neq 0$$

Define the z-score random variable for the estimated expected difference as

$$Z_\delta^* = \sqrt{K}\left(\frac{\hat{\mu}_\delta - \mu_\delta}{\hat{\sigma}_\delta}\right)$$

Z_δ^* follows a t distribution with $K-1$ degrees of freedom. However, under the null hypothesis we have $\mu_\delta = 0$, and thus

$$Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta} \sim t_{K-1}$$

i.e., Z_δ^* follows the t distribution with $K-1$ degrees of freedom.

Given a desired confidence level α, we conclude that

$$P\left(-t_{\alpha/2,K-1} \leq Z_\delta^* \leq t_{\alpha/2,K-1}\right) = \alpha$$

Put another way, if $Z_\delta^* \notin \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$, then we may reject the null hypothesis with α% confidence.

Paired t-Test via Cross-Validation

Paired t-Test(α, K, $\boldsymbol{D}$):

$\boldsymbol{D} \leftarrow$ randomly shuffle $\boldsymbol{D}$

$\{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_K\} \leftarrow$ partition $\boldsymbol{D}$ in K equal parts

foreach $i \in [1, K]$ **do**

  $M_i^A, M_i^B \leftarrow$ train the two different classifiers on $\boldsymbol{D} \setminus \boldsymbol{D}_i$

  $\theta_i^A, \theta_i^B \leftarrow$ assess M_i^A and M_i^B on $\boldsymbol{D}_i$

  $\delta_i = \theta_i^A - \theta_i^B$

$\hat{\mu}_\delta = \frac{1}{K} \sum_{i=1}^{K} \delta_i$

$\hat{\sigma}_\delta^2 = \frac{1}{K} \sum_{i=1}^{K} (\delta_i - \hat{\mu}_\delta)^2$

$Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta}$

if $Z_\delta^* \in \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$ **then**

  Accept H_0; both classifiers have similar performance

else

  Reject H_0; classifiers have significantly different performance

2D Iris Dataset: Paired t-Test

We compare, via error rate, the naive Bayes (M^A) with the full Bayes (M^B) classifier via cross-validation using $K = 5$.

i	1	2	3	4	5
θ_i^A	0.233	0.267	0.1	0.4	0.3
θ_i^B	0.2	0.2	0.167	0.333	0.233
δ_i	0.033	0.067	−0.067	0.067	0.067

The estimated expected difference and variance of the differences are

$$\hat{\mu}_\delta = \frac{0.167}{5} = 0.033 \qquad \hat{\sigma}_\delta^2 = 0.00333 \qquad \hat{\sigma}_\delta = \sqrt{0.00333} = 0.0577$$

The z-score value is given as $Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta} = \frac{\sqrt{5} \times 0.033}{0.0577} = 1.28$ For $1 - \alpha = 0.95$ (or $\alpha = 0.05$) and $4(K-1)$ degrees of freedom, $t_{\alpha/2} = 2.776$.

Because $Z_\delta^* = 1.28 \in (-2.776, 2.776) = \left(-t_{\alpha/2}, t_{\alpha/2}\right)$, we cannot reject the null hypothesis, that is, there is no significant difference between the naive and full Bayes classifier for this dataset.

Bias-Variance Decomposition

In many applications there may be costs associated with making wrong predictions. A *loss function* specifies the cost or penalty of predicting the class to be $\hat{y} = M(\boldsymbol{x})$, when the true class is y.

A commonly used loss function for classification is the *zero-one loss*, defined as

$$L(y, M(\boldsymbol{x})) = I(M(\boldsymbol{x}) \neq y) = \begin{cases} 0 & \text{if } M(\boldsymbol{x}) = y \\ 1 & \text{if } M(\boldsymbol{x}) \neq y \end{cases}$$

Thus, zero-one loss assigns a cost of zero if the prediction is correct, and one otherwise.

Another commonly used loss function is the *squared loss*, defined as

$$L(y, M(\boldsymbol{x})) = (y - M(\boldsymbol{x}))^2$$

where we assume that the classes are discrete valued, and not categorical.

Expected Loss

An ideal or optimal classifier is the one that minimizes the loss function. Because the true class is not known for a test case $\boldsymbol{x}$, the goal of learning a classification model can be cast as minimizing the expected loss:

$$E_y\left[L(y, M(\boldsymbol{x}))\,|\boldsymbol{x}\right] = \sum_y L(y, M(\boldsymbol{x})) \cdot P(y|\boldsymbol{x})$$

where $P(y|\boldsymbol{x})$ is the conditional probability of class y given test point $\boldsymbol{x}$, and E_y denotes that the expectation is taken over the different class values y.

Minimizing the expected zero–one loss corresponds to minimizing the error rate. Let $M(\boldsymbol{x}) = c_i$, then we have

$$E_y\left[L(y, M(\boldsymbol{x}))\,|\boldsymbol{x}\right] = \sum_y I(y \neq c_i) \cdot P(y|\boldsymbol{x}) = \sum_{y \neq c_i} P(y|\boldsymbol{x}) = 1 - P(c_i|\boldsymbol{x})$$

To minimize E_y, we should choose $c_i = \arg\max_y P(y|\boldsymbol{x})$.

The error rate is an estimate of the expected zero–one loss, which minimizes the error rate.

Bias and Variance

The expected loss for the squared loss function offers important insight into the classification problem because it can be decomposed into bias and variance terms.

Intuitively, the *bias* of a classifier refers to the systematic deviation of its predicted decision boundary from the true decision boundary, whereas the *variance* of a classifier refers to the deviation among the learned decision boundaries over different training sets.

Because M depends on the training set, given a test point $\boldsymbol{x}$, we denote its predicted value as $M(\boldsymbol{x}, \boldsymbol{D})$. Consider the expected square loss:

$$
\begin{aligned}
E_y\Big[L(y, M(\boldsymbol{x}, \boldsymbol{D}))\,|\boldsymbol{x}, \boldsymbol{D}\Big] &= E_y\Big[(y - M(\boldsymbol{x}, \boldsymbol{D}))^2 | \boldsymbol{x}, \boldsymbol{D}\Big] \\
&= \underbrace{E_y\Big[(y - E_y[y|\boldsymbol{x}])^2\,|\boldsymbol{x}, \boldsymbol{D}\Big]}_{var(y|\boldsymbol{x})} + \underbrace{\Big(M(\boldsymbol{x}, \boldsymbol{D}) - E_y[y|\boldsymbol{x}]\Big)^2}_{\textit{squared-error}}
\end{aligned}
$$

The first term is simply the variance of y given $\boldsymbol{x}$. The second term is the squared error between the predicted value $M(\boldsymbol{x}, \boldsymbol{D})$ and the expected value $E_y[y|\boldsymbol{x}]$.

Bias and Variance

The squared error depends on the training set. We can eliminate this dependence by averaging over all possible training tests of size n. The average or expected squared error for a given test point $\boldsymbol{x}$ over all training sets is then given as

$$E_{\boldsymbol{D}}\left[\left(M(\boldsymbol{x},\boldsymbol{D})-E_y[y|\boldsymbol{x}]\right)^2\right]=\underbrace{E_{\boldsymbol{D}}\left[\left(M(\boldsymbol{x},\boldsymbol{D})-E_{\boldsymbol{D}}[M(\boldsymbol{x},\boldsymbol{D})]\right)^2\right]}_{\text{variance}}+\underbrace{\left(E_{\boldsymbol{D}}[M(\boldsymbol{x},\boldsymbol{D})]-E_y[y|\boldsymbol{x}]\right)^2}_{\text{bias}}$$

The expected squared loss over all test points $\boldsymbol{x}$ and over all training sets $\boldsymbol{D}$ of size n yields the following decomposition:

$$E_{\boldsymbol{x},\boldsymbol{D},y}\left[\left(y-M(\boldsymbol{x},\boldsymbol{D})\right)^2\right]=\underbrace{E_{\boldsymbol{x},y}\left[\left(y-E_y[y|\boldsymbol{x}]\right)^2\right]}_{\text{noise}}+\underbrace{E_{\boldsymbol{x},\boldsymbol{D}}\left[\left(M(\boldsymbol{x},\boldsymbol{D})-E_{\boldsymbol{D}}[M(\boldsymbol{x},\boldsymbol{D})]\right)^2\right]}_{\textit{average variance}}+\underbrace{E_{\boldsymbol{x}}\left[\left(E_{\boldsymbol{D}}[M(\boldsymbol{x},\boldsymbol{D})]-E_y[y|\boldsymbol{x}]\right)^2\right]}_{\textit{average bias}}$$

The expected square loss over all test points and training sets can be decomposed into three terms: noise, average bias, and average variance.

Bias and Variance

The noise term is the average variance $var(y|\boldsymbol{x})$ over all test points $\boldsymbol{x}$. It contributes a fixed cost to the loss independent of the model, and can thus be ignored when comparing different classifiers.

The classifier specific loss can then be attributed to the variance and bias terms. Bias indicates whether the model M is correct or incorrect.

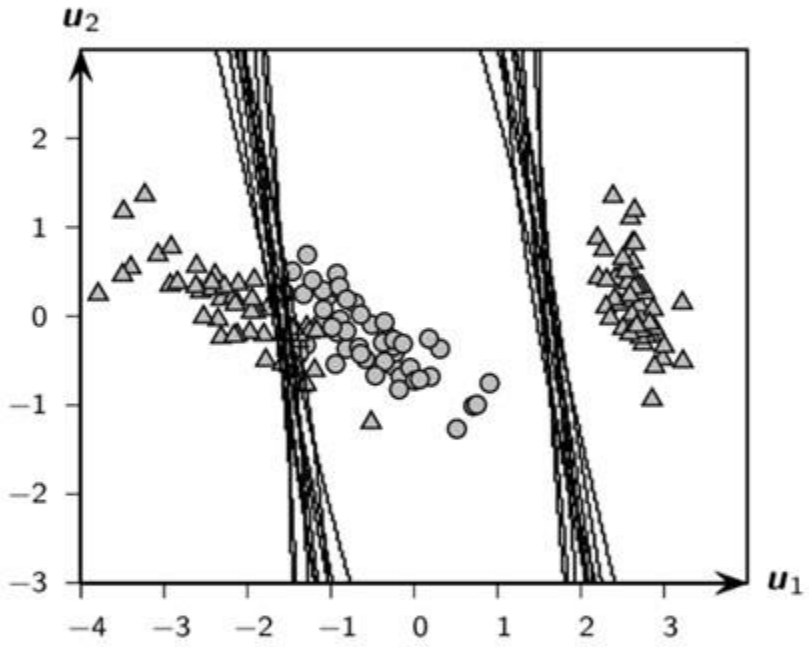
If the decision boundary is nonlinear, and we use a linear classifier, then it is likely to have high bias. A nonlinear (or a more complex) classifier is more likely to capture the correct decision boundary, and is thus likely to have a low bias.

The complex classifier is not necessarily better, since we also have to consider the variance term, which measures the inconsistency of the classifier decisions. A complex classifier induces a more complex decision boundary and thus may be prone to *overfitting*, and thus may be susceptible to small changes in training set, which may result in high variance.

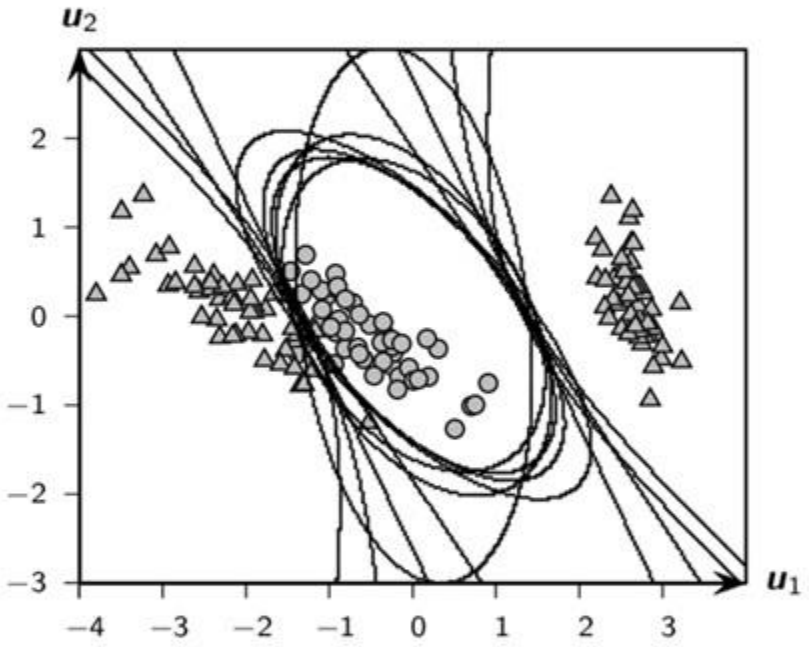
In general, the expected loss can be attributed to high bias or high variance, typically a trade-off between the two. We prefer a classifier with an acceptable bias and as low a variance as possible.

Bias-variance Decomposition: SVM Quadratic Kernels

Iris PC Data: `Iris-versicolor` (c_1 -circles) and other two Irises (c_2 - triangles). $K = 10$ Bootstrap samples, trained via SVMs, varying the regularization constant C from 10^{-2} to 10^2. The decision boundaries over the 10 samples were as follows:



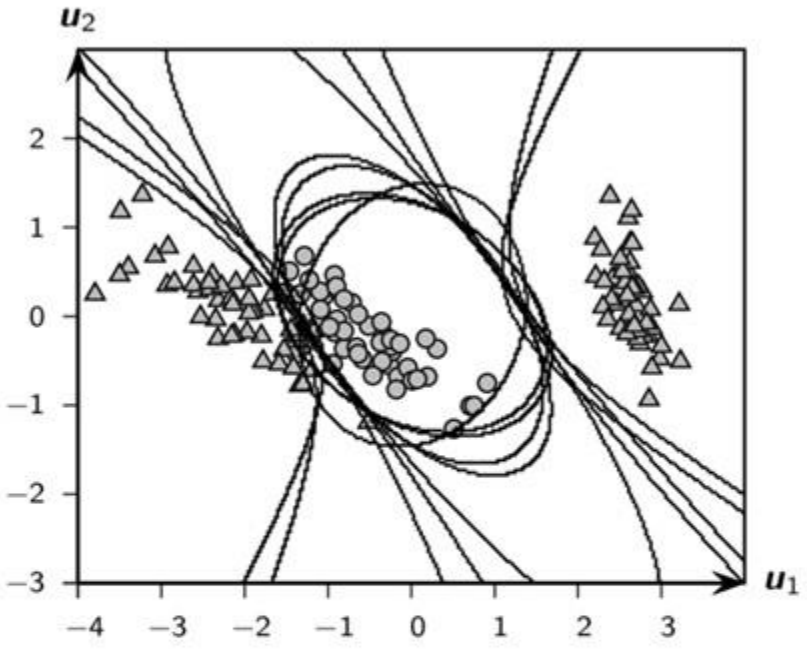
(a) $C = 0.01$



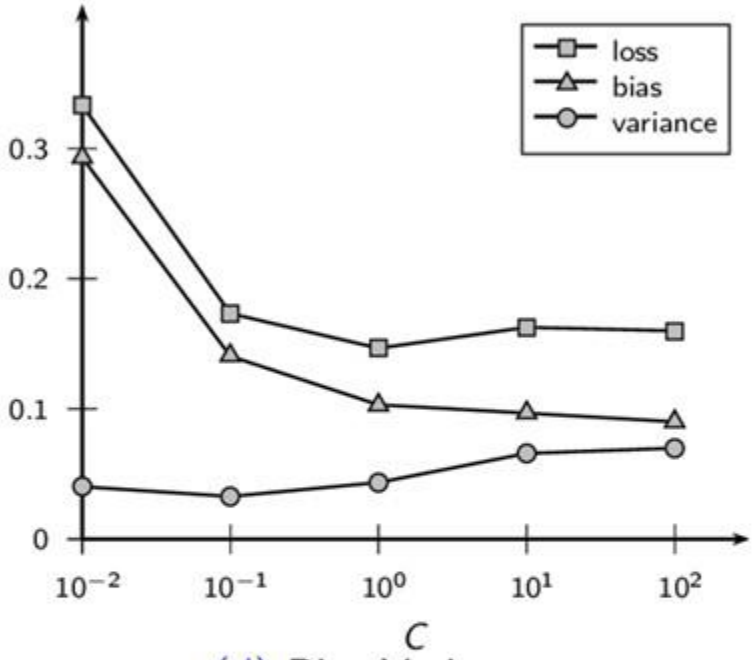
(b) $C = 1$

A small value of C emphasizes the margin, whereas a large value of C tries to minimize the slack terms.

Bias-variance Decomposition: SVM Quadratic Kernels



(c) $C = 100$



(d) Bias-Variance

Variance of the SVM model increases as we increase C.

The bias-variance tradeoff is visible, since as the bias reduces, the variance increases.

The lowest expected loss is obtained when $C = 1$.

Ensemble Classifiers

A classifier is called *unstable* if small perturbations in the training set result in large changes in the prediction or decision boundary.

High variance classifiers are inherently unstable, since they tend to overfit the data. On the other hand, high bias methods typically underfit the data, and usually have low variance.

In either case, the aim of learning is to reduce classification error by reducing the variance or bias, ideally both.

Ensemble methods create a *combined classifier* using the output of multiple *base classifiers*, which are trained on different data subsets. Depending on how the training sets are selected, and on the stability of the base classifiers, ensemble classifiers can help reduce the variance and the bias, leading to a better overall performance.

Bagging

Bagging stands for *Bootstrap Aggregation*. It is an ensemble classification method that employs multiple bootstrap samples (with replacement) from the input training data $\boldsymbol{D}$ to create slightly different training sets $\boldsymbol{D}_i$, $i = 1, 2, \ldots, K$. Different base classifiers M_i are learned, with M_i trained on $\boldsymbol{D}_i$.

Given any test point $\boldsymbol{x}$, it is first classified using each of the K base classifiers, M_i. Let the number of classifiers that predict the class of $\boldsymbol{x}$ as c_j be given as

$$v_j(\boldsymbol{x}) = \Big| \big\{ M_i(\boldsymbol{x}) = c_j \,\big|\, i = 1, \ldots, K \big\} \Big|$$

The combined classifier, denoted $\boldsymbol{M}^K$, predicts the class of a test point $\boldsymbol{x}$ by *majority voting* among the k classes:

$$\boldsymbol{M}^K(\boldsymbol{x}) = \arg\max_{c_j} \Big\{ v_j(\boldsymbol{x}) \,\big|\, j = 1, \ldots, k \Big\}$$

Bagging can help reduce the variance, especially if the base classifiers are unstable, due to the averaging effect of majority voting. It does not, in general, have much effect on the bias.

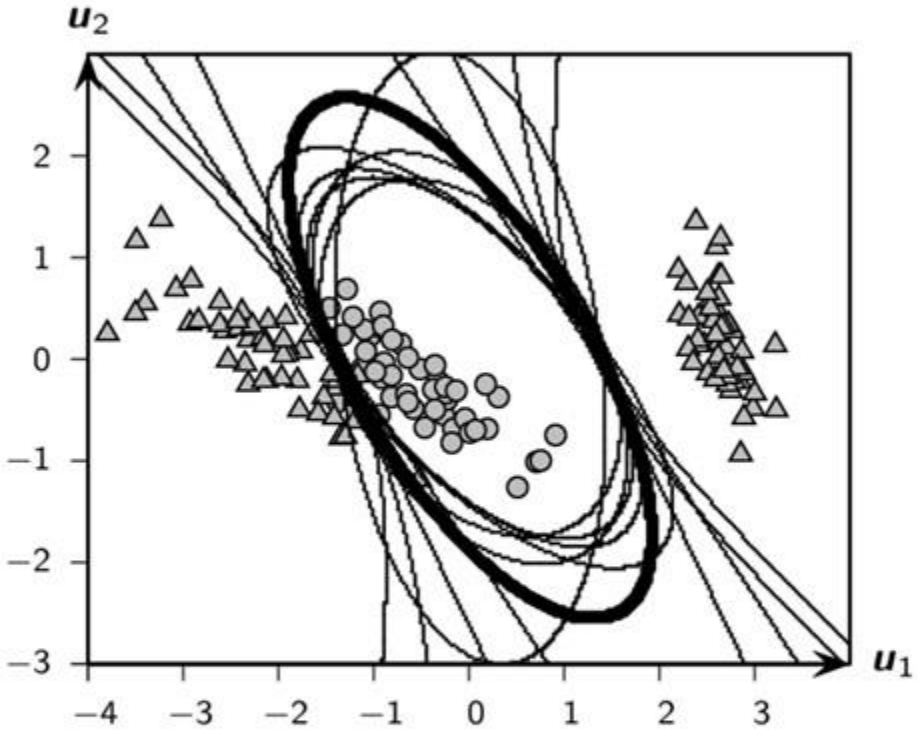
Bagging: Combined SVM Classifiers

SVM classifiers are trained on $K = 10$ bootstrap samples of the Iris PCA dataset using $C = 1$.

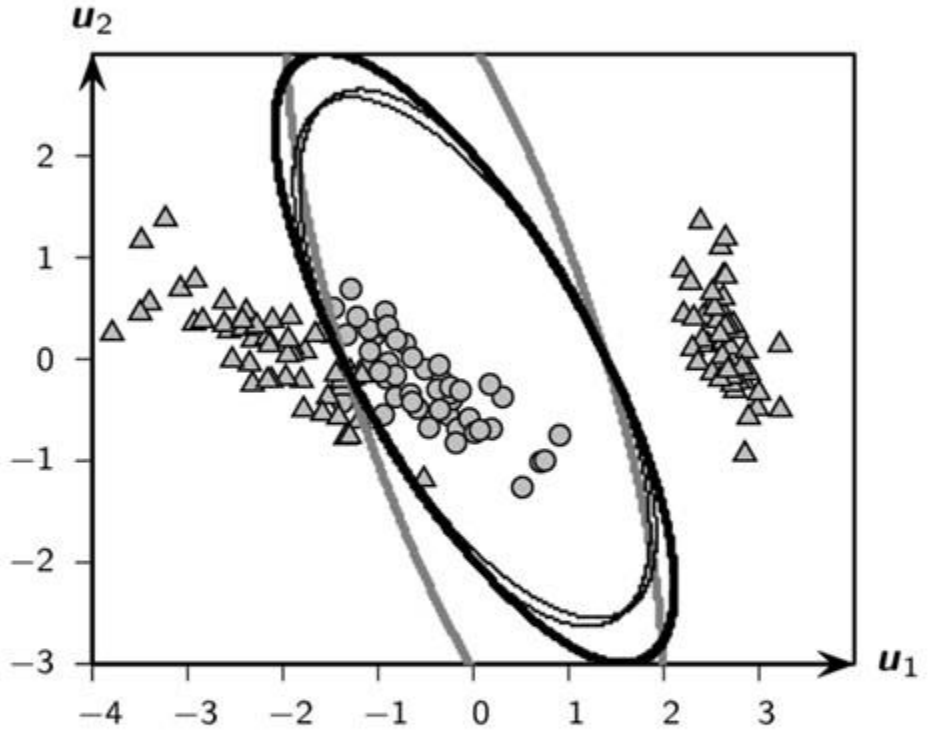
K	Zero–one loss	Squared loss
3	0.047	0.187
5	0.04	0.16
8	0.02	0.10
10	0.027	0.113
15	0.027	0.107

Bagging: Combined SVM Classifiers

The combined (average) classifier is shown in bold.



(a) $K = 10$



(b) Effect of K

The worst training performance is obtained for $K = 3$ (in thick gray) and the best for $K = 8$ (in thick black).

Random Forest

A *random forest* is an ensemble of K classifiers, $M_1, \ldots M_K$, where each classifier is a decision tree created from a different bootstrap sample, and by sampling a random subset of the attributes at each internal node in the decision tree, typically $\sqrt{(}d)$.

Bagging only would generate similar decision trees. The attribute sampling reduces the trees correlation.

The K decision trees $M_1, M_2, \cdots, M_K$ predict the class of $\boldsymbol{x}$ by majority voting:

$$\boldsymbol{M}^K(\boldsymbol{x}) = \arg\max_{c_j} \left\{ v_j(\boldsymbol{x}) \,|j = 1, \ldots, k \right\}$$

where v_j is the number of trees that predict the class of $\boldsymbol{x}$ as c_j.

Notice that if $p = d$ then the random forest approach is equivalent to bagging over decision tree models.

Random Forest Algorithm

RandomForest$(\boldsymbol{D}, K, p, \eta, \pi)$**:**

foreach $\boldsymbol{x}_i \in \boldsymbol{D}$ **do**
  $v_j(\boldsymbol{x}_i) \leftarrow 0$, for all $j = 1, 2, \ldots, k$
for $t \in [1, K]$ **do**
  $\boldsymbol{D}_t \leftarrow$ sample of size n with replacement from $\boldsymbol{D}$
  $M_t \leftarrow$ DecisionTree $(\boldsymbol{D}_t, \eta, \pi, p)$
  foreach $(\boldsymbol{x}_i, y_i) \in \boldsymbol{D} \setminus \boldsymbol{D}_t$ **do** `// out-of-bag votes`
    $\hat{y}_i \leftarrow M_t(\boldsymbol{x}_i)$
    if $\hat{y}_i = c_j$ **then** $v_j(\boldsymbol{x}_i) = v_j(\boldsymbol{x}_i) + 1$
$\epsilon_{oob} = \frac{1}{n} \cdot \sum_{i=1}^{n} I\left(y_i \neq \arg\max_{c_j}\left\{v_j(\boldsymbol{x}_i) | (\boldsymbol{x}_i, y_i) \in \boldsymbol{D}\right\}\right)$ `// OOB error`
return $\{M_1, M_2, \ldots, M_K\}$

Random Forest - Out of bag estimation

Given $\boldsymbol{D}_t$, any point in $\boldsymbol{D} \setminus \boldsymbol{D}_t$ is called an *out-of-bag* point for M_t.

The *out-of-bag* error rate for each M_t may be calculated by considering the prediction over its out-of-bag points.

The out-of-bag (OOB) error for the random forest is given as:

$$\epsilon_{oob} = \frac{1}{n} \cdot \sum_{i=1}^{n} I\left(y_i \neq max_{c_j}\left\{v_j(\boldsymbol{x}_i) | (\boldsymbol{x}_i, y_i) \in \boldsymbol{D}\right\}\right)$$

I is an indicator function that returns 1 if its argument is true, and 0 otherwise.

We check whether the majority OOB class for $\boldsymbol{x}_i \in \boldsymbol{D}$ matches y_i.

The out-of-bag error rate is the fraction of mismatched points w.r.t. y_i.

The out-of-bag error rate approximates the cross-validation error rate quite well.

Iris PC Dataset: Random Forest

The task is to separate `Iris-versicolor` (c_1 - circles) from the other two Irises (c_2 - triangles).

Since $d = 2$, we pick $p = 1$ attribute for each split-point evaluation.

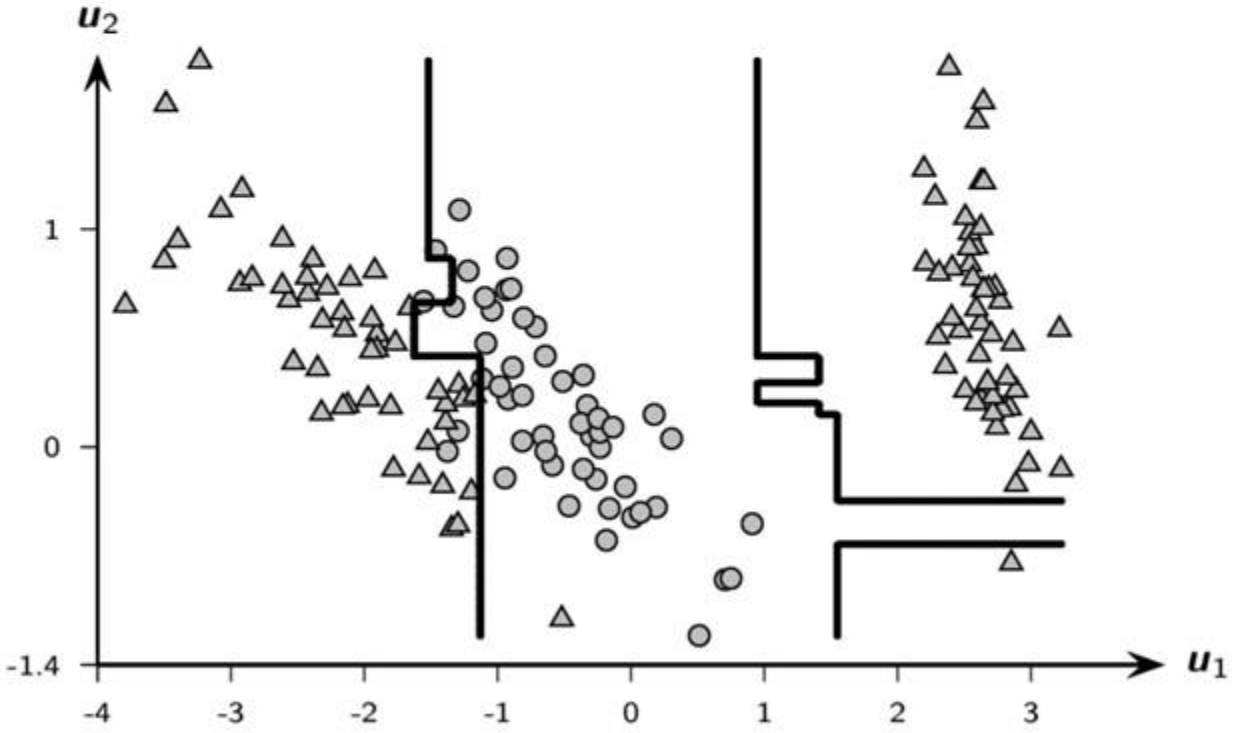
We use $\eta = 3$ (maximum leaf size) and minimum purity $\pi = 1.0$.

We grow $K = 5$ decision trees on different bootstrap samples.

Iris PC Dataset: Random Forest

Decision boundary is shown in bold.

The training error rate is 2.0%, but OOB error rate is 49.33%, which is overly pessimistic in this case, since the dataset has only two attributes, and we use only one attribute to evaluate each split point.



Random Forest: Varying K

We used the full Iris dataset which has four attributes ($d = 4$), and three classes ($k = 3$).

We employed $p = \sqrt{d} = 2$, $\eta = 3$ and $\pi = 1.0$.

K	ϵ_{oob}	ϵ
1	0.4333	0.0267
2	0.2933	0.0267
3	0.1867	0.0267
4	0.1200	0.0400
5	0.1133	0.0333
6	0.1067	0.0400
7	0.0733	0.0333
8	0.0600	0.0267
9	0.0467	0.0267
10	0.0467	0.0267

We can see that the OOB error decreases as we increase the number of trees.

Boosting

In *Boosting* the main idea is to carefully select the samples to *boost* the performance on hard to classify instances.

Starting from an initial training sample $\boldsymbol{D}_1$, we train the base classifier M_1, and obtain its training error rate.

To construct the next sample $\boldsymbol{D}_2$, we select the misclassified instances with higher probability, and after training M_2, we obtain its training error rate.

To construct $\boldsymbol{D}_3$, those instances that are hard to classify by M_1 or M_2, have a higher probability of being selected. This process is repeated for K iterations.

Finally, the combined classifier is obtained via weighted voting over the output of the K base classifiers $M_1, M_2, \ldots, M_K$.

Boosting

Boosting is most beneficial when the base classifiers are *weak*, that is, have an error rate that is slightly less than that for a random classifier.

The idea is that whereas M_1 may not be particularly good on all test instances, by design M_2 may help classify some cases where M_1 fails, and M_3 may help classify instances where M_1 and M_2 fail, and so on. Thus, boosting has more of a bias reducing effect.

Each of the weak learners is likely to have high bias (it is only slightly better than random guessing), but the final combined classifier can have much lower bias, since different weak learners learn to classify instances in different regions of the input space.

Adaptive Boosting: AdaBoost

AdaBoost repeats the boosting process K times. Let t denote the iteration and let α_t denote the weight for the tth classifier M_t.

Let w_i^t denote the weight for $\boldsymbol{x}_i$, with $\boldsymbol{w}^t = (w_1^t, w_2^t, \ldots, w_n^t)^T$ being the weight vector over all the points for the tth iteration.

$\boldsymbol{w}$ is a probability vector, whose elements sum to one. Initially all points have equal weights, that is,

$$\boldsymbol{w}^0 = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)^T = \frac{1}{n}\mathbf{1}$$

During iteration t, the training sample $\boldsymbol{D}_t$ is obtained via weighted resampling using the distribution $\boldsymbol{w}^{t-1}$, that is, we draw a sample of size n with replacement, such that the ith point is chosen according to its probability w_i^{t-1}.

Using $\boldsymbol{D}_t$ we train the classifier M_t, and compute its weighted error rate ϵ_t on the entire input dataset $\boldsymbol{D}$:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{t-1} \cdot I\big(M_t(\boldsymbol{x}_i) \neq y_i\big)$$

Adaptive Boosting: AdaBoost

The weight for the tth classifier is then set as

$$\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

The weight for each point $\boldsymbol{x}_i \in \boldsymbol{D}$ is updated as

$$w_i^t = w_i^{t-1} \cdot \exp\Big\{\alpha_t \cdot I\big(M_t(\boldsymbol{x}_i) \neq y_i\big)\Big\}$$

If the predicted class matches the true class, that is, if $M_t(\boldsymbol{x}_i) = y_i$, then the weight for point $\boldsymbol{x}_i$ remains unchanged.

If the point is misclassified, that is, $M_t(\boldsymbol{x}_i) \neq y_i$, then

$$w_i^t = w_i^{t-1} \cdot \exp\{\alpha_t\} = w_i^{t-1} \exp\left\{\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\right\} = w_i^{t-1}\left(\frac{1}{\epsilon_t} - 1\right)$$

Thus, if the error rate ϵ_t is small, then there is a greater weight increment for $\boldsymbol{x}_i$. The intuition is that a point that is misclassified by a good classifier (with a low error rate) should be more likely to be selected for the next training dataset.

Adaptive Boosting: AdaBoost

For boosting we require that a base classifier has an error rate at least slightly better than random guessing, that is, $\epsilon_t < 0.5$. If the error rate $\epsilon_t \geq 0.5$, then the boosting method discards the classifier, and tries another data sample.

Combined Classifier: Given the set of boosted classifiers, $M_1, M_2, \ldots, M_K$, along with their weights $\alpha_1, \alpha_2, \ldots, \alpha_K$, the class for a test case $\boldsymbol{x}$ is obtained via weighted majority voting.

Let $v_j(\boldsymbol{x})$ denote the weighted vote for class c_j over the K classifiers, given as

$$v_j(\boldsymbol{x}) = \sum_{t=1}^{K} \alpha_t \cdot I\big(M_t(\boldsymbol{x}) = c_j\big)$$

Because $I(M_t(\boldsymbol{x}) = c_j)$ is 1 only when $M_t(\boldsymbol{x}) = c_j$, the variable $v_j(\boldsymbol{x})$ simply obtains the tally for class c_j among the K base classifiers, taking into account the classifier weights. The combined classifier, denoted $\boldsymbol{M}^K$, then predicts the class for $\boldsymbol{x}$ as follows:

$$\boldsymbol{M}^K(\boldsymbol{x}) = \arg\max_{c_j} \Big\{ v_j(\boldsymbol{x}) \,\big|\, j = 1, .., k \Big\}$$

AdaBoost Algorithm

AdaBoost(K, $\boldsymbol{D}$):

$\boldsymbol{w}^0 \leftarrow \left(\frac{1}{n}\right) \cdot 1 \in \mathbb{R}^n$

$t \leftarrow 1$

while $t \leq K$ **do**

  $\boldsymbol{D}_t \leftarrow$ weighted resampling with replacement from $\boldsymbol{D}$ using $\boldsymbol{w}^{t-1}$

  $M_t \leftarrow$ train classifier on $\boldsymbol{D}_t$

  $\epsilon_t \leftarrow \sum_{i=1}^{n} w_i^{t-1} \cdot I\left(M_t(\boldsymbol{x}_i) \neq y_i\right)$ `// weighted error rate on` $\boldsymbol{D}$

  if $\epsilon_t = 0$ **then break**

  else if $\epsilon_t < 0.5$ **then**

    $\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ `// classifier weight`

    foreach $i \in [1, n]$ **do**

      `// update point weights`

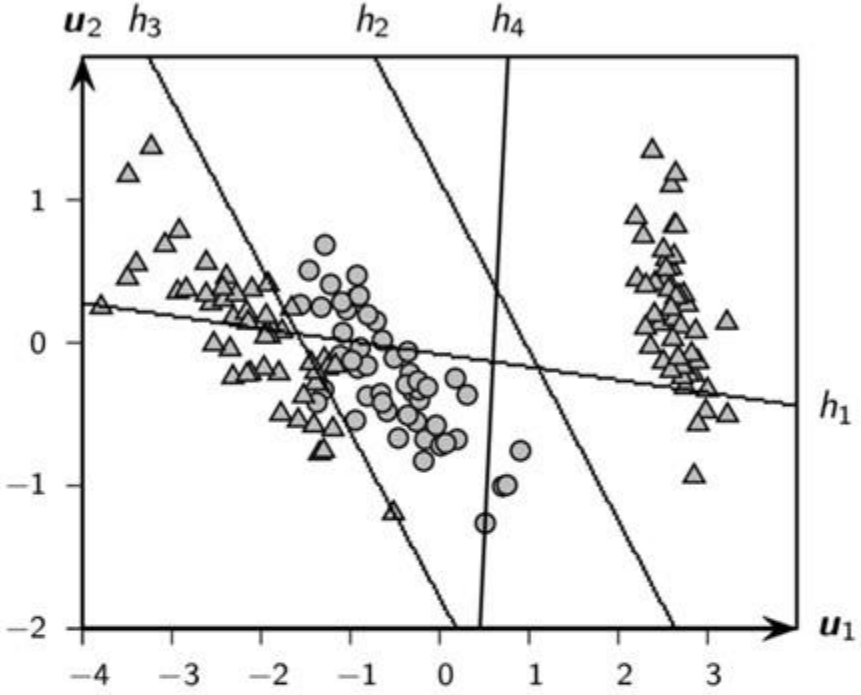
      $$w_i^t = \begin{cases} w_i^{t-1} & \text{if } M_t(\boldsymbol{x}_i) = y_i \\ w_i^{t-1}\left(\frac{1-\epsilon_t}{\epsilon_t}\right) & \text{if } M_t(\boldsymbol{x}_i) \neq y_i \end{cases}$$

    $\boldsymbol{w}^t = \frac{\boldsymbol{w}^t}{\mathbf{1}^T \boldsymbol{w}^t}$ `// normalize weights`

    $t \leftarrow t + 1$

return $\{M_1, M_2, \ldots, M_K\}$

Boosting SVMs: Linear Kernel ($C = 1$)



The regularization constant $C = 1$.

h_t is learned in iteration t.

No h_t discriminates between classes well:

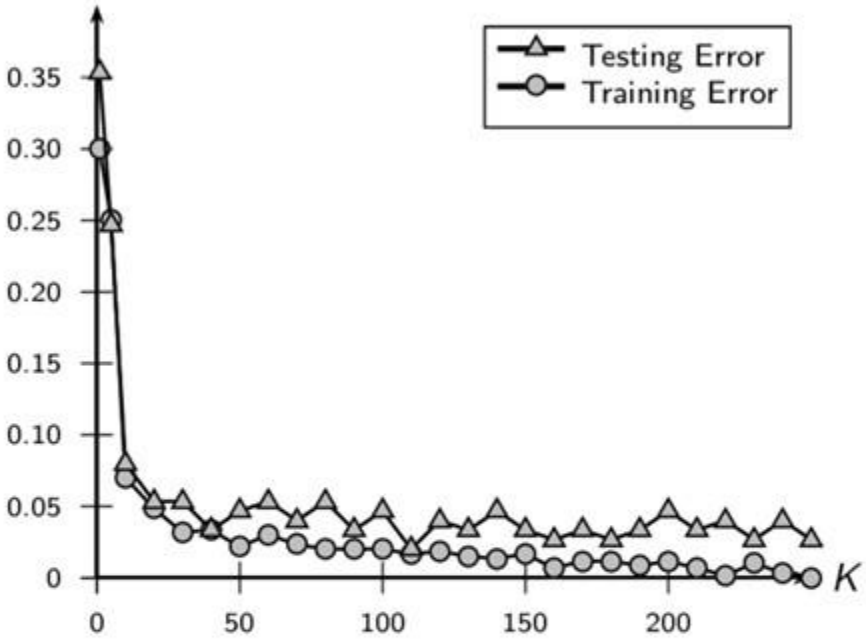
M_t	ϵ_t	α_t
h_1	0.280	0.944
h_2	0.305	0.826
h_3	0.174	1.559
h_4	0.282	0.935

When we combine the decisions from hyperplanes weighted by α_t, we observe a marked drop in the error rate:

combined model	M^1	M^2	M^3	M^4
training error rate	0.280	0.253	0.073	0.047

Boosting SVMs: Linear Kernel ($C = 1$)

Five-fold cross-validation using independent testing data.



As the number of base classifiers K increases, both error rates reduce.

However, while the training error essentially goes to 0, the testing error does not reduce beyond 0.02 ($K = 110$).

This example illustrates the effectiveness of boosting in reducing the bias.

Bagging can be considered as a special case of AdaBoost, where $w^t = \frac{1}{n}1$, and $\alpha_t = 1$ for all K iterations.

The weighted resampling defaults to regular resampling with replacement, and majority voting predicts the class.

Stacking

Stacking or stacked generalization is an ensemble technique where we employ two layers of classifiers.

The first layer is composed of K base classifiers which are trained independently on the entire training data $\boldsymbol{D}$.

The second layer comprises a combiner classifier C that is trained on the predicted classes from the base classifiers, so that it automatically learns how to combine the outputs of the base classifiers to make the final prediction for a given input.

For example, the combiner classifier may learn to ignore the output of a base classifiers for an input that lies in a region of the input space where the base classifier has poor performance. It can also learn to correct the prediction in cases where most base classifiers do not predict the outcome correctly.

Stacking is a strategy for estimating and correcting the biases of the set of base classifiers.

Stacking

Stacking$(K, \boldsymbol{M}, C, \boldsymbol{D})$:

// Train base classifiers

for $t \in [1, K]$ **do**

  $M_t \leftarrow$ train tth base classifier on $\boldsymbol{D}$

// Train combiner model C on $\boldsymbol{Z}$

$\boldsymbol{Z} \leftarrow \emptyset$

foreach $(\boldsymbol{x}_i, y_i) \in \boldsymbol{D}$ **do**

  $\boldsymbol{z}_i \leftarrow \left(M_1(\boldsymbol{x}_i), M_2(\boldsymbol{x}_i), \ldots, M_K(\boldsymbol{x}_i)\right)^T$

  $\boldsymbol{Z} \leftarrow \boldsymbol{Z} \cup \{(\boldsymbol{z}_i, y_i)\}$

$C \leftarrow$ train combiner classifier on $\boldsymbol{Z}$

return $(C, M_1, M_2, \ldots, M_K)$

Iris 2D PCA - Stacking

We use three base classifiers:

- SVM with a linear kernel (with regularization constant $C = 1$),
- random forests (with $K = 5$ trees and $p = 1$ random attributes), and
- naive Bayes.

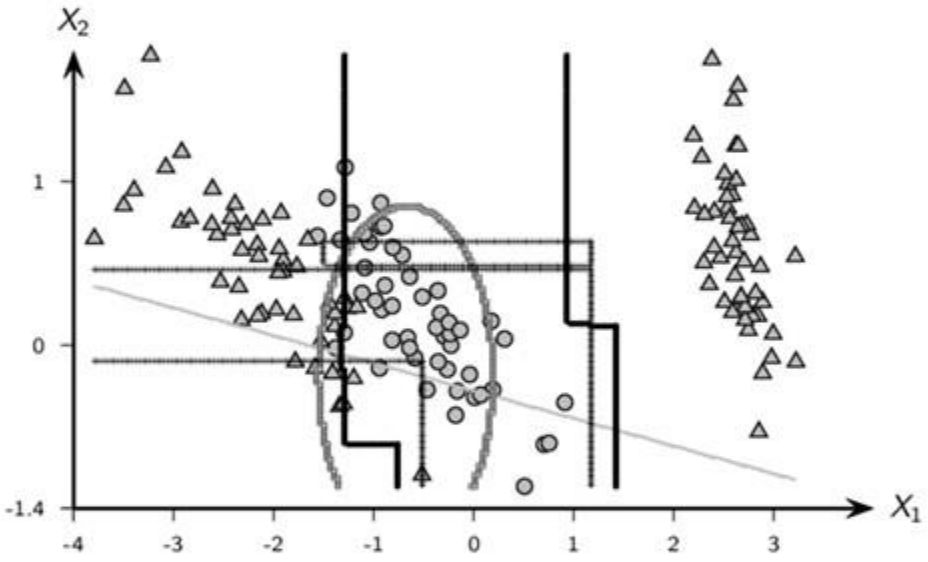
The combiner classifier is an SVM with a Gaussian kernel (with regularization constant $C = 1$ and spread parameter $\sigma^2 = 0.2$).

Training uses an 100-point random subset.

Testing uses the remaining 50 points.

Classifier	Test Accuracy
Linear SVM	0.68
Random Forest	0.82
Naive Bayes	0.74
Stacking	0.92

Iris 2D PCA - Stacking



Plot shows the decision boundaries for each of the three base classifiers:

- linear SVM boundary is the line in light gray,
- naive Bayes boundary is the ellipse comprising the gray circles, and
- random forest boundary is shown via plusses ('+').

The boundary of the stacking classifier is shown as the thicker black lines.

Q&A