CHAPTER 22 Classification Assessment

We have seen different classifiers in the preceding chapters, such as decision trees, full and naive Bayes classifiers, nearest neighbors classifier, support vector machines, and so on. In general, we may think of the classifier as a model or function M that predicts the class label $\hat{y}$ for a given input example $\mathbf{x}$:

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

where $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$ 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. Different classifiers are obtained depending on the assumptions used to build the model M. For instance, support vector machines use the maximum margin hyperplane to construct M. On the other hand, the Bayes classifier directly computes the posterior probability $P(c_j|\mathbf{x})$ for each class c_j, and predicts the class of $\mathbf{x}$ as the one with the maximum posterior probability, $\hat{y} = \arg\max_{c_j} \left\{P(c_j|\mathbf{x})\right\}$. 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.

In this chapter we look at methods to assess a classifier, and to compare multiple classifiers. We start by defining metrics of classifier accuracy. We then discuss how to determine bounds on the expected error. We finally discuss how to assess the performance of classifiers and compare them.

22.1 CLASSIFICATION PERFORMANCE MEASURES

Let $\mathbf{D}$ 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 $\mathbf{x}_i \in \mathbf{D}$, let y_i denote its true class, and let $\hat{y}_i = M(\mathbf{x}_i)$ denote its predicted class.

Error Rate

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

$$Error\ Rate = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \tag{22.1}$$

where I is an indicator function that has the value 1 when its argument is true, and 0 otherwise. 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 over the testing set:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} I(y_i = \hat{y}_i) = 1 - Error\ Rate \tag{22.2}$$

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

Example 22.1. Figure 22.1 shows the 2-dimensional Iris dataset, with the two attributes being `sepal length` and `sepal width`. It has 150 points, and has three equal-sized classes: `Iris-setosa` (c_1; circles), `Iris-versicolor` (c_2; squares) and `Iris-virginica` (c_3; triangles). The dataset is partitioned into training and testing sets, in the ratio 80:20. Thus, the training set has 120 points (shown in light gray), and

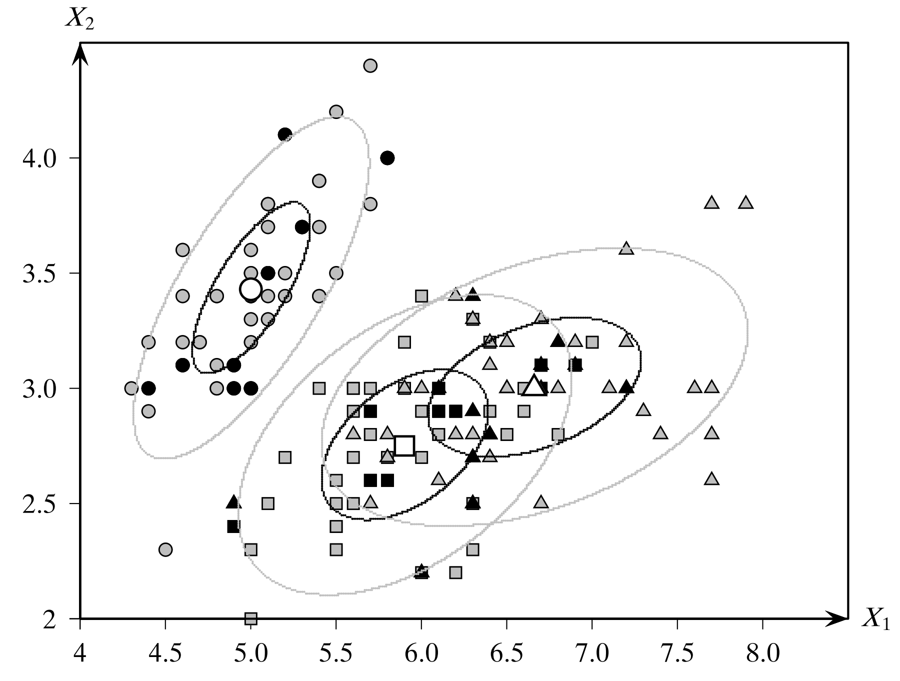


Figure 22.1. Iris dataset: three classes.

the testing set $\mathbf{D}$ has $n = 30$ points (shown in black). One can see that whereas c_1 is well separated from the other classes, c_2 and c_3 are not easy to separate. In fact, some points are labeled as both c_2 and c_3 (e.g., the point $(6, 2.2)^T$ appears twice, labeled as c_2 and c_3).

We classify the test points using the full Bayes classifier (see Chapter 18). Each class is modeled using a single normal distribution, whose mean (in white) and density contours (corresponding to one and two standard deviations) are also plotted in Figure 22.1. The classifier misclassifies 8 out of the 30 test cases. Thus, we have

$$\begin{aligned} \textit{Error Rate} &= 8/30 = 0.267 \\ \textit{Accuracy} &= 22/30 = 0.733 \end{aligned}$$

22.1.1 Contingency Table–based Measures

The error rate (and, thus also the accuracy) is a global measure in that it does not explicitly consider the classes that contribute to the error. More informative measures can be obtained by tabulating the class specific agreement and disagreement between the true and predicted labels over the testing set. Let $\mathcal{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_k\}$ denote a partitioning of the testing points based on their true class labels, where

$$\mathbf{D}_j = \{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_j\}$$

Let $n_i = |\mathbf{D}_i|$ denote the size of true class c_i.

Let $\mathcal{R} = \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_k\}$ denote a partitioning of the testing points based on the predicted labels, that is,

$$\mathbf{R}_j = \{\mathbf{x}_i \in \mathbf{D} \mid \hat{y}_i = c_j\}$$

Let $m_j = |\mathbf{R}_j|$ denote the size of the predicted class c_j.

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

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

where $1 \le i, j \le k$. The count n_{ij} denotes the number of points with predicted class c_i whose true label is c_j. Thus, n_{ii} (for $1 \le i \le k$) 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

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 the class-specific accuracy:

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

This is identical to the expression in Eq. (22.2).

Coverage/Recall

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:

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

where n_i is the number of points in class c_i. The higher the coverage the better the classifier.

F-measure

Often there is a trade-off between the precision and recall of a classifier. For example, it is easy to make $recall_i = 1$, by predicting all testing points to be in class c_i. However, in this case $prec_i$ will be low. On the other hand, we can make $prec_i$ very high by predicting only a few points as c_i, for instance, for those predictions where M has the most confidence, but in this case $recall_i$ will be low. Ideally, we would like both precision and recall to be high.

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.

Example 22.2. Consider the 2-dimensional Iris dataset shown in Figure 22.1. In Example 22.1 we saw that the error rate was 26.7%. However, the error rate measure does not give much information about the classes or instances that are more difficult to classify. From the class-specific normal distribution in the figure, it is clear that the Bayes classifier should perform well for c_1, but it is likely to have problems discriminating some test cases that lie close to the decision boundary between c_2 and c_3. This information is better captured by the confusion matrix obtained on the testing set, as shown in Table 22.1. We can observe that all 10 points in c_1 are classified correctly. However, only 7 out of the 10 for c_2 and 5 out of the 10 for c_3 are classified correctly.

Table 22.1. Contingency table for Iris dataset: testing set

	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	$m_3 = 8$
	$n_1 = 10$	$n_2 = 10$	$n_3 = 10$	$n = 30$

From the confusion matrix we can compute the class-specific precision (or accuracy) values:

$$prec_1 = \frac{n_{11}}{m_1} = 10/10 = 1.0$$

$$prec_2 = \frac{n_{22}}{m_2} = 7/12 = 0.583$$

$$prec_3 = \frac{n_{33}}{m_3} = 5/8 = 0.625$$

The overall accuracy tallies with that reported in Example 22.1:

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

The class-specific recall (or coverage) values are given as

$$recall_1 = \frac{n_{11}}{n_1} = 10/10 = 1.0$$

$$recall_2 = \frac{n_{22}}{n_2} = 7/10 = 0.7$$

$$recall_3 = \frac{n_{33}}{n_3} = 5/10 = 0.5$$

From these we can compute the class-specific F-measure values:

$$F_1 = \frac{2 \cdot n_{11}}{(n_1 + m_1)} = 20/20 = 1.0$$

$$F_2 = \frac{2 \cdot n_{22}}{(n_2 + m_2)} = 14/22 = 0.636$$

$$F_3 = \frac{2 \cdot n_{33}}{(n_3 + m_3)} = 10/18 = 0.556$$

Thus, the overall F-measure for the classifier is

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

Table 22.2. Confusion matrix for two classes

	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)

22.1.2 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, shown in Table 22.2, are given special names, as follows:

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

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

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

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

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

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

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

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

Error Rate

The error rate [Eq. (22.1)] for the binary classification case is given as the fraction of mistakes (or false predictions):

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

Accuracy

The accuracy [Eq. (22.2)] is the fraction of correct predictions:

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

The above are global measures of classifier performance. We can obtain class-specific measures as follows.

Class-specific Precision

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

Sensitivity: True Positive Rate

The true positive rate, also called *sensitivity*, is the fraction of correct predictions with respect to all points in the positive class, that is, it is simply the recall for the positive class

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

where n_1 is the size of the positive class.

Specificity: True Negative Rate

The true negative rate, also called *specificity*, is simply 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.

False Negative Rate

The false negative rate is defined as

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

False Positive Rate

The false positive rate is defined as

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

Example 22.3. Consider the Iris dataset projected onto its first two principal components, as shown in Figure 22.2. The task is to separate `Iris-versicolor` (class c_1; in circles) from the other two Irises (class c_2; in triangles). The points from class c_1 lie in-between the points from class c_2, making this is a hard problem for (linear) classification. The dataset has been randomly split into 80% training (in gray) and 20% testing points (in black). Thus, the training set has 120 points and the testing set has $n = 30$ points.

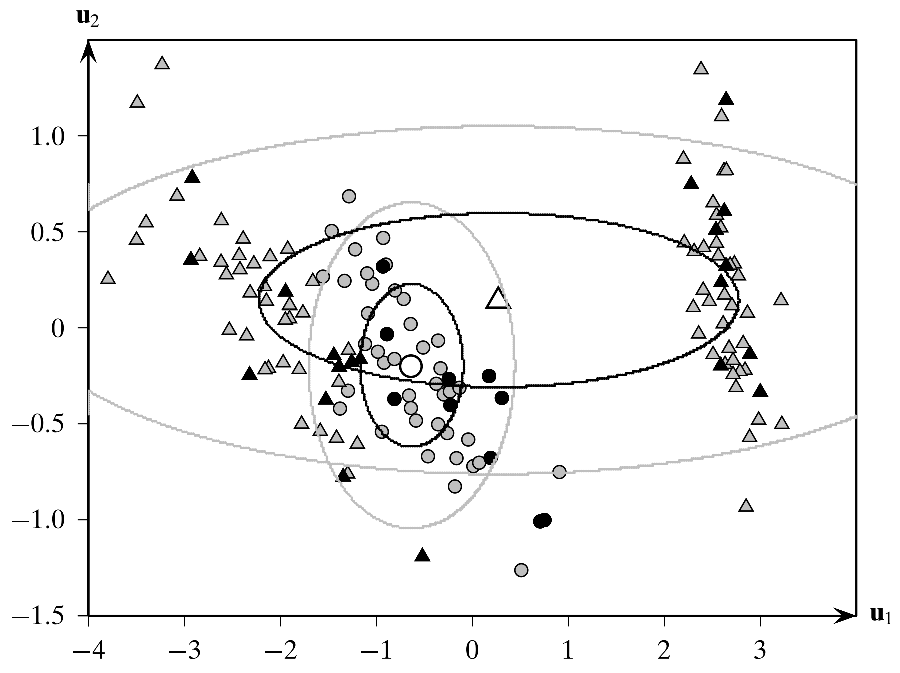


Figure 22.2. Iris principal component dataset: training and testing sets.

Applying the naive Bayes classifier (with one normal per class) on the training set yields the following estimates for the mean, covariance matrix and prior probability for each class:

$$\hat{P}(c_1) = 40/120 = 0.33 \qquad \hat{P}(c_2) = 80/120 = 0.67$$

$$\hat{\mu}_1 = \begin{pmatrix}-0.641 & -0.204\end{pmatrix}^T \qquad \hat{\mu}_2 = \begin{pmatrix}0.27 & 0.14\end{pmatrix}^T$$

$$\widehat{\mathbf{\Sigma}}_1 = \begin{pmatrix}0.29 & 0\\ 0 & 0.18\end{pmatrix} \qquad \widehat{\mathbf{\Sigma}}_2 = \begin{pmatrix}6.14 & 0\\ 0 & 0.206\end{pmatrix}$$

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

For each of the 30 testing points, we classify them using the above parameter estimates (see Chapter 18). The naive Bayes classifier misclassified 10 out of the 30 test instances, resulting in an error rate and accuracy of

$$\textit{Error Rate} = 10/30 = 0.33$$

$$\textit{Accuracy} = 20/30 = 0.67$$

The confusion matrix for this binary classification problem is shown in Table 22.3. From this table, we can compute the various performance measures:

$$prec_P = \frac{TP}{TP+FP} = \frac{7}{14} = 0.5$$

Table 22.3. Iris PC dataset: contingency table for binary classification

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

$$prec_N = \frac{TN}{TN+FN} = \frac{13}{16} = 0.8125$$

$$recall_P = sensitivity = TPR = \frac{TP}{TP+FN} = \frac{7}{10} = 0.7$$

$$recall_N = specificity = TNR = \frac{TN}{TN+FP} = \frac{13}{20} = 0.65$$

$$FNR = 1 - sensitivity = 1 - 0.7 = 0.3$$

$$FPR = 1 - specificity = 1 - 0.65 = 0.35$$

We can observe that the precision for the positive class is rather low. The true positive rate is also low, and the false positive rate is relatively high. Thus, the naive Bayes classifier is not particularly effective on this testing dataset.

22.1.3 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. For instance, we can use the posterior probability $P(c_1|\mathbf{x}_i)$ as the score, for example, for the Bayes classifiers. For SVM classifiers, we can use the signed distance from the hyperplane as the score because large positive distances are high confidence predictions for c_1, and large negative distances are very low confidence predictions for c_1 (they are, in fact, high confidence predictions for the negative class c_2).

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. However, such a threshold is likely to be somewhat arbitrary. Instead, 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.

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

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

Table 22.4. 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

Initially, we classify all points as negative. Both *TP* and *FP* are thus initially zero (as shown in Table 22.4a), resulting in *TPR* and *FPR* rates of zero, which correspond to the point $(0, 0)$ at the lower left corner in the ROC plot. Next, for each distinct value of ρ in the range $[\rho^{\min}, \rho^{\max}]$, we tabulate the set of positive points:

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

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 shown in Table 22.4b), 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. 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 Table 22.4c. As such, a ROC curve indicates the extent to which the classifier ranks positive instances higher than the negative instances. An ideal classifier should score all positive points higher than any negative point. Thus, 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. Because the total area of the plot is 1, the AUC lies in the interval $[0, 1]$ – the higher the better. 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/AUC Algorithm

Algorithm 22.1 shows the steps for plotting a ROC curve, and for computing the area under the curve. It takes as input the testing set $\mathbf{D}$, and the classifier M. The first step is to predict the score $S(\mathbf{x}_i)$ for the positive class (c_1) for each test point $\mathbf{x}_i \in \mathbf{D}$. Next, we sort the $(S(\mathbf{x}_i), y_i)$ pairs, that is, the score and the true class pairs, in decreasing order of the scores (line 3). Initially, we set the positive score threshold $\rho = \infty$ (line 7). The for loop (line 8) examines each pair $(S(\mathbf{x}_i), y_i)$ in sorted order, and for each distinct value of the score, it sets $\rho = S(\mathbf{x}_i)$ and plots the point

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

ALGORITHM 22.1. ROC Curve and Area under the Curve

```
ROC-CURVE(D, M):
1  $n_1 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_1\}\right|$ // size of positive class
2  $n_2 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_2\}\right|$ // size of negative class
   // classify, score, and sort all test points
3  $L \leftarrow$ sort the set $\{(S(\mathbf{x}_i), y_i) : \mathbf{x}_i \in \mathbf{D}\}$ 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 $(S(\mathbf{x}_i), y_i) \in L$ do
9      if $\rho > S(\mathbf{x}_i)$ 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         $\rho \leftarrow S(\mathbf{x}_i)$
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 $AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$

TRAPEZOID-AREA($(x_1, y_1), (x_2, y_2)$):
19 $b \leftarrow |x_2 - x_1|$ // base of trapezoid
20 $h \leftarrow \frac{1}{2}(y_2 + y_1)$ // average height of trapezoid
21 return $(b \cdot h)$
```

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 $\mathbf{x}_i$. If $y_1 = c_1$, we increment the true positives, otherwise, we increment the false positives (lines 15-16). At the end of the for loop we plot the final point in the ROC curve (line 17).

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 function

Table 22.5. Sorted scores and true classes

$S(\mathbf{x}_i)$	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
$S(\mathbf{x}_i)$	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

$S(\mathbf{x}_i)$	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
$S(\mathbf{x}_i)$	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

TRAPEZOID-AREA computes the area under the trapezoid, which 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.

Example 22.4. Consider the binary classification problem from Example 22.3 for the Iris principal components dataset. The test dataset $\mathbf{D}$ has $n = 30$ points, with $n_1 = 10$ points in the positive class and $n_2 = 20$ points in the negative class.

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 $\mathbf{x}_i$ is therefore $S(\mathbf{x}_i) = P(c_1|\mathbf{x}_i)$. The sorted scores (in decreasing order) along with the true class labels are shown in Table 22.5.

The ROC curve for the test dataset is shown in Figure 22.3. Consider the positive score threshold $\rho = 0.71$. If we classify all points with a score above this value as positive, then we have the following counts for the true and false positives:

$$TP = 3 \qquad\qquad FP = 2$$

The false positive rate is therefore $\frac{FP}{n_2} = 2/20 = 0.1$, and the true positive rate is $\frac{TP}{n_1} = 3/10 = 0.3$. This corresponds to the point $(0.1, 0.3)$ in the ROC curve. Other points on the ROC curve are obtained in a similar manner as shown in Figure 22.3. The total area under the curve is 0.775.

Example 22.5 (AUC). To see why we need to account for trapezoids when computing the AUC, consider the following sorted scores, along with the true class, for some testing dataset with $n = 5$, $n_1 = 3$ and $n_2 = 2$.

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

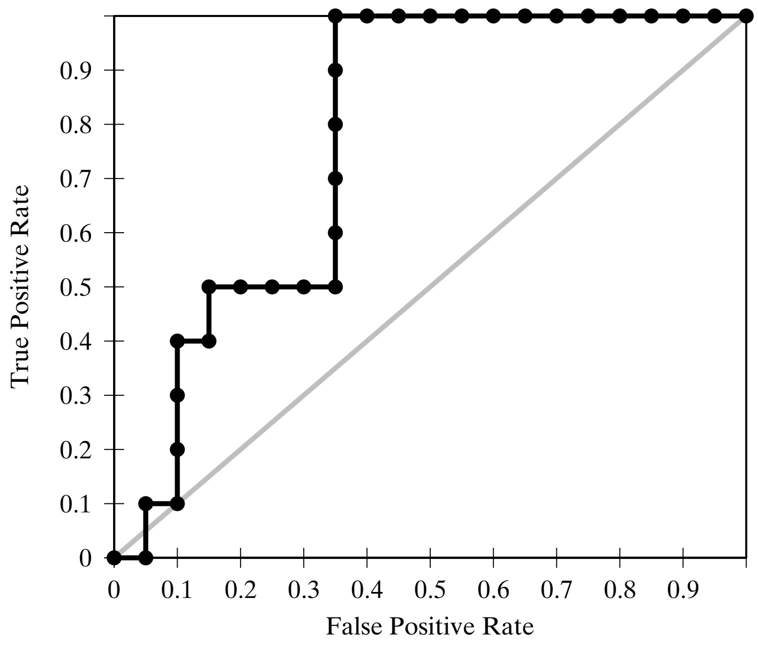


Figure 22.3. ROC plot for Iris principal components dataset. The ROC curves for the naive Bayes (black) and random (gray) classifiers are shown.

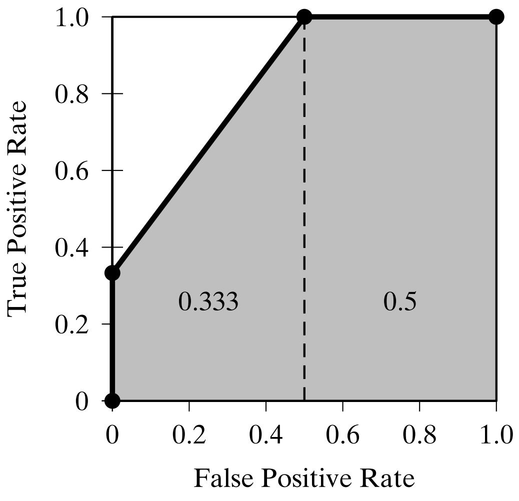


Figure 22.4. ROC plot and AUC: trapezoid region.

Algorithm 22.1 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

Figure 22.4 shows the ROC plot, with the shaded region representing the AUC. We can observe that a trapezoid is obtained whenever there is at least one positive and one negative point with the same score. The total AUC is 0.833, obtained as the sum of the trapezoidal region on the left (0.333) and the rectangular region on the right (0.5).

Random Classifier

It is interesting to note that a random classifier corresponds to a diagonal line in the ROC plot. To see this think of 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. If, on the other hand, the classifier guesses the class of a point as positive 90% of the time and as negative 10% of the time, then we expect 90% of the true positives and 10% of the true negatives to be labeled correctly, resulting in $TPR = 0.9$ and $FPR = 1 - TNR = 1 - 0.1 = 0.9$, that is, we get the point $(0.9, 0.9)$ in 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. If follows that if the ROC curve for any classifier is below the diagonal, it indicates performance worse than random guessing. For such cases, inverting the class assignment will produce a better classifier. As a consequence of the diagonal ROC curve, the AUC value for a random classifier is 0.5. Thus, if any classifier has an AUC value less than 0.5, that also indicates performance worse than random.

Example 22.6. In addition to the ROC curve for the naive Bayes classifier, Figure 22.3 also shows the ROC plot for the random classifier (the diagonal line in gray). We can see that the ROC curve for the naive Bayes classifier is much better than random. Its AUC value is 0.775, which is much better than the 0.5 AUC for a random classifier. However, at the very beginning naive Bayes performs worse than the random classifier because the highest scored point is from the negative class. As such, the ROC curve should be considered as a discrete approximation of a smooth curve that would be obtained for a very large (infinite) testing dataset.

Class Imbalance

It is worth remarking that ROC curves are insensitive to class skew. This is because the TPR, interpreted as the probability of predicting a positive point as positive, and the FPR, interpreted as the probability of predicting a negative point as positive, do not depend on the ratio of the positive to negative class size. This is a desirable property, since the ROC curve will essentially remain the same whether the classes are balanced (have relatively the same number of points) or skewed (when one class has many more points than the other).

22.2 CLASSIFIER EVALUATION

In this section we discuss how to evaluate a classifier M using some performance measure θ. Typically, the input dataset $\mathbf{D}$ 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. However, how confident can we be about the classification performance? The results may be due to an artifact of the random split, for example, by random chance the testing set may have particularly easy (or hard) to classify points, leading to good (or poor) classifier performance. As such, a fixed, pre-defined partitioning of the dataset is not a good strategy for evaluating classifiers. Also note that, in general, $\mathbf{D}$ is itself a d-dimensional multivariate random sample drawn from the true (unknown) joint probability density function $f(\mathbf{x})$ 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 $\mathbf{D}$. Cross-validation and resampling are two common approaches to compute the expected value and variance of a given performance measure; we discuss these methods in the following sections.

22.2.1 K-fold Cross-Validation

Cross-validation divides the dataset $\mathbf{D}$ into K equal-sized parts, called *folds*, namely $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K$. Each fold $\mathbf{D}_i$ is, in turn, treated as the testing set, with the remaining folds comprising the training set $\mathbf{D} \setminus \mathbf{D}_i = \bigcup_{j \neq i} \mathbf{D}_j$. After training the model M_i on $\mathbf{D} \setminus \mathbf{D}_i$, we assess its performance on the testing set $\mathbf{D}_i$ to obtain the i-th estimate θ_i. The expected value of the performance measure can then be estimated as

$$\hat{\mu}_\theta = E[\theta] = \frac{1}{K}\sum_{i=1}^{K}\theta_i \tag{22.3}$$

and its variance as

$$\hat{\sigma}_\theta^2 = \frac{1}{K}\sum_{i=1}^{K}(\theta_i - \hat{\mu}_\theta)^2 \tag{22.4}$$

Algorithm 22.2 shows the pseudo-code for K-fold cross-validation. After randomly shuffling the dataset $\mathbf{D}$, we partition it into K equal folds (except for possibly the last one). Next, each fold $\mathbf{D}_i$ is used as the testing set on which we assess the performance θ_i of the classifier M_i trained on $\mathbf{D} \setminus \mathbf{D}_i$. The estimated mean and variance of θ can then be reported. Note that the K-fold cross-validation can be repeated multiple times; the initial random shuffling ensures that the folds are different each time.

Usually K is chosen to be 5 or 10. The special case, when $K = n$, is called *leave-one-out* cross-validation, where the testing set comprises a single point and the remaining data is used for training purposes.

ALGORITHM 22.2. *K*-fold Cross-Validation

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

Example 22.7. Consider the 2-dimensional Iris dataset from Example 22.1 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:

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

Using Eqs. (22.3) and (22.4), the mean and variance for the error rate are as follows:

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

We can repeat the whole cross-validation approach multiple times, with a different permutation of the input points, and then we can compute the mean of the average error rate, and mean of the variance. 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}.

22.2.2 Bootstrap Resampling

Another approach to estimate the expected performance of a classifier is to use the bootstrap resampling method. Instead of partitioning the input dataset $\mathbf{D}$ into disjoint folds, the bootstrap method draws K random samples of size n *with replacement* from $\mathbf{D}$. Each sample $\mathbf{D}_i$ is thus the same size as $\mathbf{D}$, and has several repeated points. Consider the probability that a point $\mathbf{x}_j \in \mathbf{D}$ is not selected for the ith bootstrap sample $\mathbf{D}_i$. Due to sampling with replacement, the probability that a given point is selected is given as $p = \frac{1}{n}$, and thus the probability that it is not selected is

$$q = 1 - p = \left(1 - \frac{1}{n}\right)$$

Because $\mathbf{D}_i$ has n points, the probability that $\mathbf{x}_j$ is not selected even after n tries is given as

$$P(\mathbf{x}_j \not\in \mathbf{D}_i) = q^n = \left(1 - \frac{1}{n}\right)^n \simeq e^{-1} = 0.368$$

ALGORITHM 22.3. Bootstrap Resampling Method

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

On the other hand, the probability that $\mathbf{x}_j \in \mathbf{D}$ is given as

$$P(\mathbf{x}_j \in \mathbf{D}_i) = 1 - P(\mathbf{x}_j \notin \mathbf{D}_i) = 1 - 0.368 = 0.632$$

This means that each bootstrap sample contains approximately 63.2% of the points from $\mathbf{D}$.

The bootstrap samples can be used to evaluate the classifier by training it on each of samples $\mathbf{D}_i$ and then using the full input dataset $\mathbf{D}$ as the testing set, as shown in Algorithm 22.3. The expected value and variance of the performance measure θ can be obtained using Eqs. (22.3) and (22.4). However, it should be borne in mind that the estimates will be somewhat optimistic owing to the fairly large overlap between the training and testing datasets (63.2%). The cross-validation approach does not suffer from this limitation because it keeps the training and testing sets disjoint.

Example 22.8. We continue with the Iris dataset from Example 22.7. However, we now apply bootstrap sampling to estimate the error rate for the full Bayes classifier, using $K = 50$ samples. The sampling distribution of error rates is shown in Figure 22.5.

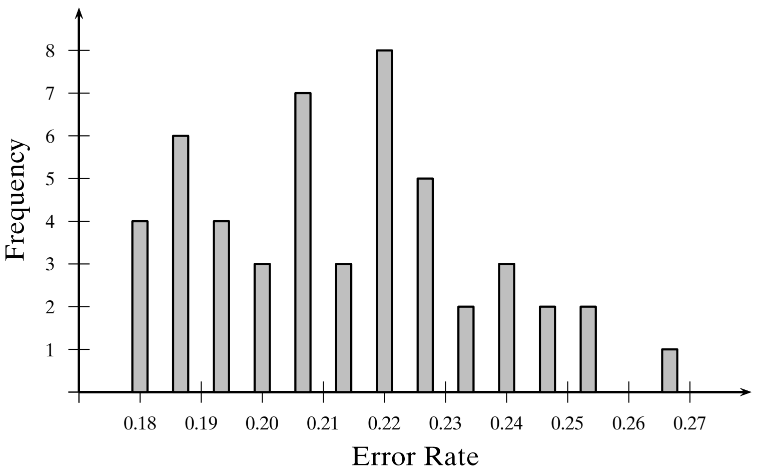


Figure 22.5. Sampling distribution of error rates.

The expected value and variance of the error rate are

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

Due to the overlap between the training and testing sets, the estimates are more optimistic (i.e., lower) compared to those obtained via cross-validation in Example 22.7, where we had $\hat{\mu}_\theta = 0.233$ and $\hat{\sigma}_\theta^2 = 0.00833$.

22.2.3 Confidence Intervals

Having estimated the expected value and variance for a chosen performance measure, we would like to derive confidence bounds on how much the estimate 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. More formally, let $\theta_1, \theta_2, \dots, \theta_K$ 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:

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

By linearity of expectation, we have

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

Utilizing the linearity of variance for independent random variables, and noting that $var(aX) = a^2 \cdot var(X)$ for $a \in \mathbb{R}$, the variance of $\hat{\mu}$ is given as

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

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. 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

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

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,

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

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

$$\lim_{K\to\infty} P(-z_{\alpha/2} \le Z_K \le z_{\alpha/2}) = 2 \cdot P(0 \le Z_K \le z_{\alpha/2}) = \alpha \tag{22.5}$$

Note that

$$\begin{aligned}
-z_{\alpha/2} \le Z_K \le z_{\alpha/2} &\implies -z_{\alpha/2} \le \sqrt{K}\left(\frac{\hat{\mu}-\mu}{\sigma}\right) \le z_{\alpha/2} \\
&\implies -z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \le \hat{\mu}-\mu \le z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \\
&\implies \left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) \le \mu \le \left(\hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)
\end{aligned}$$

Substituting the above into Eq. (22.5) we obtain bounds on the value of the true mean μ in terms of the estimated value $\hat{\mu}$, that is,

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}} \le \mu \le \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right) = \alpha \tag{22.6}$$

Thus, for any given level of confidence α, we can compute the probability that the true mean μ lies in the α% confidence interval $\left(\hat{\mu} - z_{\alpha/2}\frac{\sigma}{\sqrt{K}}, \hat{\mu} + z_{\alpha/2}\frac{\sigma}{\sqrt{K}}\right)$. In other words, even though we do not know the true mean μ, we can obtain a high-confidence estimate of the interval within which it must lie (e.g., by setting $\alpha = 0.95$ or $\alpha = 0.99$).

Unknown Variance

The analysis above assumes that we know the true variance σ^2, which is generally not the case. However, we can replace σ^2 by the sample variance

$$\hat{\sigma}^2 = \frac{1}{K}\sum_{i=1}^{K}(\theta_i - \hat{\mu})^2 \tag{22.7}$$

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:

$$Z_K^* = \sqrt{K}\left(\frac{\hat{\mu}-\mu}{\hat{\sigma}}\right) \tag{22.8}$$

and thus, we have

$$\lim_{K\to\infty} P\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right) = \alpha \tag{22.9}$$

In other words, we say that $\left(\hat{\mu} - z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}},\ \hat{\mu} + z_{\alpha/2}\frac{\hat{\sigma}}{\sqrt{K}}\right)$ is the $\alpha\%$ confidence interval for μ.

Example 22.9. Consider Example 22.7, where we applied 5-fold cross-validation ($K = 5$) to assess the error rate of the full Bayes classifier. The estimated expected value and variance for the error rate were as follows:

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

Let $\alpha = 0.95$ be the confidence value. It is known that the standard normal distribution has 95% of the probability density within $z_{\alpha/2} = 1.96$ standard deviations from the mean. Thus, in the limit of large sample size, we have

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

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

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

Put differently, with 95% confidence, the true expected error rate lies in the interval $(0.153, 0.313)$.

If we want greater confidence, for example, for $\alpha = 0.99$, then the corresponding z-score value is $z_{\alpha/2} = 2.58$, and thus $z_{\alpha/2}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = \frac{2.58\times 0.0913}{\sqrt{5}} = 0.105$. The 99% confidence interval for μ is therefore wider $(0.128, 0.338)$.

Nevertheless, $K = 5$ is not a large sample size, and thus the above confidence intervals are not that reliable.

Small Sample Size

The confidence interval in Eq. (22.9) applies only when the sample size $K \to \infty$. We would like to obtain more precise confidence intervals for small samples. Consider the random variables V_i, for $i = 1, \ldots, K$, defined as

$$V_i = \frac{\theta_i - \hat{\mu}}{\sigma}$$

Further, consider the sum of their squares:

$$S = \sum_{i=1}^{K} V_i^2 = \sum_{i=1}^{K}\left(\frac{\theta_i - \hat{\mu}}{\sigma}\right)^2 = \frac{1}{\sigma^2}\sum_{i=1}^{K}(\theta_i - \hat{\mu})^2 = \frac{K\hat{\sigma}^2}{\sigma^2} \tag{22.10}$$

The last step follows from the definition of sample variance in Eq. (22.7).

If we assume that the V_i's are IID with the standard normal distribution, then the sum S follows a chi-squared distribution with $K - 1$ degrees of freedom, denoted

$\chi^2(K-1)$, since S is the sum of the squares of K random variables V_i. There are only $K-1$ degrees of freedom because each V_i depends on $\hat{\mu}$ and the sum of the θ_i's is thus fixed.

Consider the random variable Z_K^* in Eq. (22.8). We have,

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

Dividing the numerator and denominator in the expression above by $\sigma/\sqrt{K}$, we get

$$Z_K^* = \left(\frac{\hat{\mu}-\mu}{\sigma/\sqrt{K}}\Big/\frac{\hat{\sigma}/\sqrt{K}}{\sigma/\sqrt{K}}\right) = \left(\frac{\frac{\hat{\mu}-\mu}{\sigma/\sqrt{K}}}{\hat{\sigma}/\sigma}\right) = \frac{Z_K}{\sqrt{S/K}} \tag{22.11}$$

The last step follows from Eq. (22.10) because

$$S = \frac{K\hat{\sigma}^2}{\sigma^2} \text{ implies that } \frac{\hat{\sigma}}{\sigma} = \sqrt{S/K}$$

Assuming that Z_K follows a standard normal distribution, and noting that S follows a chi-squared distribution with $K-1$ degrees of freedom, then the distribution of Z_K^* is precisely the Student's t distribution with $K-1$ degrees of freedom. Thus, in the small sample case, instead of using the standard normal density to derive the confidence interval, we use the 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,

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

where T_{K-1} is the cumulative distribution function for the Student's t distribution with $K-1$ degrees of freedom. Because the t distribution is symmetric about the mean, we have

$$P\left(\hat{\mu} - t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}} \le \mu \le \hat{\mu} + t_{\alpha/2,K-1}\frac{\hat{\sigma}}{\sqrt{K}}\right) = \alpha \tag{22.12}$$

The α% confidence interval for the true mean μ is thus

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

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

Figure 22.6 shows the t distribution density function for different values of K. It also shows the standard normal density function. We can observe that the t distribution has more probability concentrated in its tails compared to the standard normal distribution. Further, as K increases, the t distribution very rapidly converges in distribution to the standard normal distribution, consistent with the large sample case. Thus, for large samples, we may use the usual $z_{\alpha/2}$ threshold.

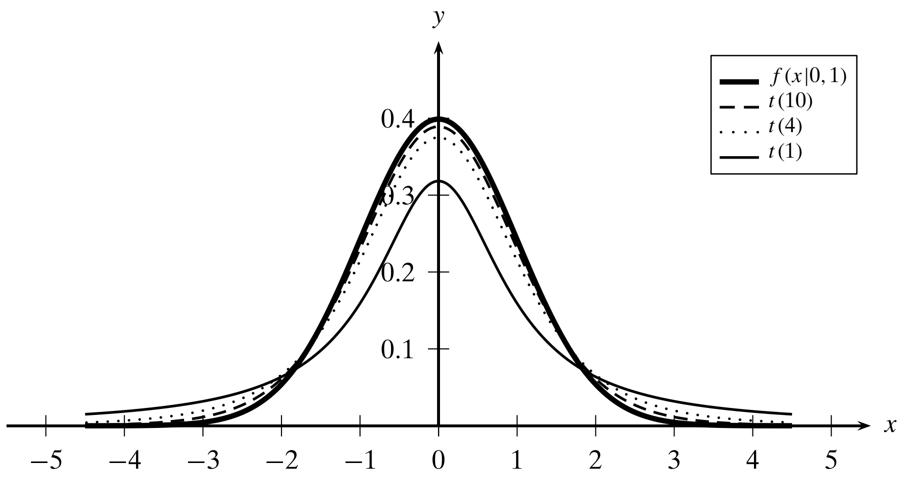


Figure 22.6. Student's t distribution: K degrees of freedom. The thick solid line is standard normal distribution.

Example 22.10. Consider Example 22.9. For 5-fold cross-validation, the estimated mean error rate is $\hat{\mu}_\theta = 0.233$, and the estimated variance is $\hat{\sigma}_\theta = 0.0913$.

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 use the quantile function for the Student's t-distribution to obtain $t_{\alpha/2,K-1} = 2.776$. 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 in Example 22.9.

For $\alpha = 0.99$, we have $t_{\alpha/2,K-1} = 4.604$, and thus

$$t_{\alpha/2,K-1}\frac{\hat{\sigma}_\theta}{\sqrt{K}} = 4.604 \times \frac{0.0913}{\sqrt{5}} = 0.188$$

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 in Example 22.9.

22.2.4 Comparing Classifiers: Paired t-Test

In this section we look at a method that allows us to test for a significant difference in the classification performance of two alternative classifiers, M^A and M^B. We want to assess which of them has a superior classification performance on a given dataset $\mathbf{D}$.

Following the evaluation methodology above, 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:

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

Now consider the estimates for the expected difference and the variance of the differences:

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

We can set up a hypothesis testing framework to determine if there is a statistically significant difference between the performance of M^A and M^B. The null hypothesis H_0 is that their performance is the same, that is, the true expected difference is zero, whereas the alternative hypothesis H_a is that they are not the same, that is, the true expected difference μ_δ is not zero:

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

Let us define the z-score random variable for the estimated expected difference as

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

Following a similar argument as in Eq. (22.11), Z_δ^* follows a t distribution with $K-1$ degrees of freedom. However, under the null hypothesis we have $\mu_\delta = 0$, and thus

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

where the notation $Z_\delta^* \sim t_{K-1}$ means that Z_δ^* follows the t distribution with $K-1$ degrees of freedom.

Given a desired confidence level α, we conclude that

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

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. In this case, we conclude that there is a significant difference between the performance of M^A and M^B. On the other hand, if Z_δ^* does lie in the above confidence interval, then we accept the null hypothesis that both M^A and M^B have essentially the same performance. The pseudo-code for the paired t-test is shown in Algorithm 22.4.

ALGORITHM 22.4. Paired *t*-Test via Cross-Validation

```
PAIRED t-TEST($\alpha$, $K$, $\mathbf{D}$):
1  $\mathbf{D} \leftarrow$ randomly shuffle $\mathbf{D}$
2  $\{\mathbf{D}_1, \mathbf{D}_2, \dots, \mathbf{D}_K\} \leftarrow$ partition $\mathbf{D}$ in $K$ equal parts
3  foreach $i \in [1, K]$ do
4      $M_i^A, M_i^B \leftarrow$ train the two different classifiers on $\mathbf{D} \setminus \mathbf{D}_i$
5      $\theta_i^A, \theta_i^B \leftarrow$ assess $M_i^A$ and $M_i^B$ on $\mathbf{D}_i$
6      $\delta_i = \theta_i^A - \theta_i^B$
7  $\hat{\mu}_\delta = \frac{1}{K}\sum_{i=1}^K \delta_i$
8  $\hat{\sigma}_\delta^2 = \frac{1}{K}\sum_{i=1}^K (\delta_i - \hat{\mu}_\delta)^2$
9  $Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta}$
10 if $Z_\delta^* \in \big(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\big)$ then
11     Accept $H_0$; both classifiers have similar performance
12 else
13     Reject $H_0$; classifiers have significantly different performance
```

Example 22.11. Consider the 2-dimensional Iris dataset from Example 22.1, with $k = 3$ classes. We compare the naive Bayes (M^A) with the full Bayes (M^B) classifier via cross-validation using $K = 5$ folds. Using error rate as the performance measure, we obtain the following values for the error rates and their difference over each of the K folds:

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

$$Z_\delta^* = \frac{\sqrt{K}\hat{\mu}_\delta}{\hat{\sigma}_\delta} = \frac{\sqrt{5} \times 0.033}{0.0577} = 1.28$$

From Example 22.10, for $\alpha = 0.95$ and $K - 1 = 4$ degrees of freedom, we have $t_{\alpha/2,K-1} = 2.776$. Because

$$Z_\delta^* = 1.28 \in (-2.776, 2.776) = \big(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\big)$$

we cannot reject the null hypothesis. Instead, we accept the null hypothesis that $\mu_\delta = 0$, that is, there is no significant difference between the naive and full Bayes classifier for this dataset.

22.3 BIAS-VARIANCE DECOMPOSITION

Given a training set $\mathbf{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, comprising n points $\mathbf{x}_i \in \mathbb{R}^d$, with their corresponding classes y_i, a learned classification model M predicts the class for a given test point $\mathbf{x}$. The various performance measures we described above mainly focus on minimizing the prediction error by tabulating the fraction of misclassified points. However, 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 $\hat{y} = M(\mathbf{x})$, when the true class is y. A commonly used loss function for classification is the *zero-one loss*, defined as

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

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

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

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 $\mathbf{x}$, the goal of learning a classification model can be cast as minimizing the expected loss:

$$E_y[L(y, M(\mathbf{x}))\,|\mathbf{x}] = \sum_y L(y, M(\mathbf{x})) \cdot P(y|\mathbf{x}) \tag{22.13}$$

where $P(y|\mathbf{x})$ is the conditional probability of class y given test point $\mathbf{x}$, 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. This can be seen by expanding Eq. (22.13) with zero–one loss. Let $M(\mathbf{x}) = c_i$, then we have

$$\begin{aligned} E_y[L(y, M(\mathbf{x}))\,|\mathbf{x}] &= E_y[I(y \neq M(\mathbf{x}))\,|\mathbf{x}] \\ &= \sum_y I(y \neq c_i) \cdot P(y|\mathbf{x}) \\ &= \sum_{y \neq c_i} P(y|\mathbf{x}) \\ &= 1 - P(c_i|\mathbf{x}) \end{aligned}$$

Thus, to minimize the expected loss we should choose c_i as the class that maximizes the posterior probability, that is, $c_i = \arg\max_y P(y|\mathbf{x})$. Because by definition [Eq. (22.1)], the error rate is simply an estimate of the expected zero–one loss, this choice also 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. More formally, because M depends on the training set, given a test point $\mathbf{x}$, we denote its predicted value as $M(\mathbf{x},\mathbf{D})$. Consider the expected square loss:

$$\begin{aligned}
&E_y\Big[L\big(y, M(\mathbf{x},\mathbf{D})\big)\,|\mathbf{x},\mathbf{D}\Big]\\
&\quad= E_y\Big[\big(y - M(\mathbf{x},\mathbf{D})\big)^2|\mathbf{x},\mathbf{D}\Big]\\
&\quad= E_y\Big[\big(y \underbrace{-E_y[y|\mathbf{x}] + E_y[y|\mathbf{x}]}_{\text{add and subtract same term}} - M(\mathbf{x},\mathbf{D})\big)^2\,|\mathbf{x},\mathbf{D}\Big]\\
&\quad= E_y\Big[\big(y - E_y[y|\mathbf{x}]\big)^2\,|\mathbf{x},\mathbf{D}\Big] + E_y\Big[\big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2\,|\mathbf{x},\mathbf{D}\Big]\\
&\qquad\quad + E_y\Big[2\big(y - E_y[y|\mathbf{x}]\big)\cdot\big(E_y[y|\mathbf{x}] - M(\mathbf{x},\mathbf{D})\big)\,|\mathbf{x},\mathbf{D}\Big]\\
&\quad= E_y\Big[\big(y - E_y[y|\mathbf{x}]\big)^2\,|\mathbf{x},\mathbf{D}\Big] + \big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2\\
&\qquad\quad + 2\big(E_y[y|\mathbf{x}] - M(\mathbf{x},\mathbf{D})\big)\cdot\underbrace{\big(E_y[y|\mathbf{x}] - E_y[y|\mathbf{x}]\big)}_{0}\\
&\quad= \underbrace{E_y\Big[\big(y - E_y[y|\mathbf{x}]\big)^2\,|\mathbf{x},\mathbf{D}\Big]}_{var(y|\mathbf{x})} + \underbrace{\Big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\Big)^2}_{squared\text{-}error}
\end{aligned} \tag{22.14}$$

Above, we made use of the fact that for any random variables X and Y, and for any constant a, we have $E[X+Y] = E[X] + E[Y]$, $E[aX] = aE[X]$, and $E[a] = a$. The first term in Eq. (22.14) is simply the variance of y given $\mathbf{x}$. The second term is the squared error between the predicted value $M(\mathbf{x},\mathbf{D})$ and the expected value $E_y[y|\mathbf{x}]$. Because this term 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 $\mathbf{x}$ over all training sets is then given as

$$\begin{aligned}
&E_{\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2\Big]\\
&\quad= E_{\mathbf{D}}\Big[\Big(M(\mathbf{x},\mathbf{D}) \underbrace{-E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] + E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]}_{\text{add and subtract same term}} - E_y[y|\mathbf{x}]\Big)^2\Big]\\
&\quad= E_{\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)^2\Big] + E_{\mathbf{D}}\Big[\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\big)^2\Big]\\
&\qquad\quad + 2\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\big)\cdot\underbrace{\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)}_{0}\\
&\quad= \underbrace{E_{\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)^2\Big]}_{variance} + \underbrace{\Big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\Big)^2}_{bias}
\end{aligned} \tag{22.15}$$

This means that the expected squared error for a given test point can be decomposed into bias and variance terms. Combining Eqs. (22.14) and (22.15) the expected squared loss over all test points $\mathbf{x}$ and over all training sets $\mathbf{D}$ of size n yields the following decomposition into noise, variance and bias terms:

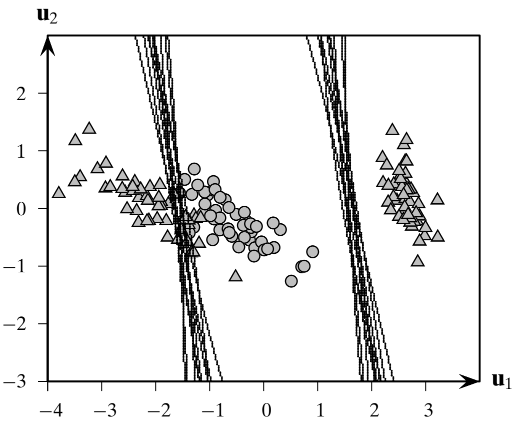
$$\begin{aligned}
&E_{\mathbf{x},\mathbf{D},y}\Big[\big(y - M(\mathbf{x},\mathbf{D})\big)^2\Big]\\
&\quad= E_{\mathbf{x},\mathbf{D},y}\Big[\big(y - E_y[y|\mathbf{x}]\big)^2 \,|\mathbf{x},\mathbf{D}\Big] + E_{\mathbf{x},\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_y[y|\mathbf{x}]\big)^2\Big]\\
&\quad= \underbrace{E_{\mathbf{x},y}\Big[\big(y - E_y[y|\mathbf{x}]\big)^2\Big]}_{noise} + \underbrace{E_{\mathbf{x},\mathbf{D}}\Big[\big(M(\mathbf{x},\mathbf{D}) - E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})]\big)^2\Big]}_{average\ variance}\\
&\qquad + \underbrace{E_{\mathbf{x}}\Big[\big(E_{\mathbf{D}}[M(\mathbf{x},\mathbf{D})] - E_y[y|\mathbf{x}]\big)^2\Big]}_{average\ bias}
\end{aligned} \tag{22.16}$$

Thus, the expected square loss over all test points and training sets can be decomposed into three terms: noise, average bias, and average variance. The noise term is the average variance $var(y|\mathbf{x})$ over all test points $\mathbf{x}$. 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. In general, bias indicates whether the model M is correct or incorrect. It also reflects our assumptions about the domain in terms of the decision boundary. For example, if the decision boundary is nonlinear, and we use a linear classifier, then it is likely to have high bias, that is, it will be consistently incorrect over different training sets. On the other hand, 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. Nevertheless, this does not necessarily mean that a complex classifier will be a better one, 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*, that is, it may try to model all the small nuances in the training data, and thus may be susceptible to small changes in training set, which may result in high variance.

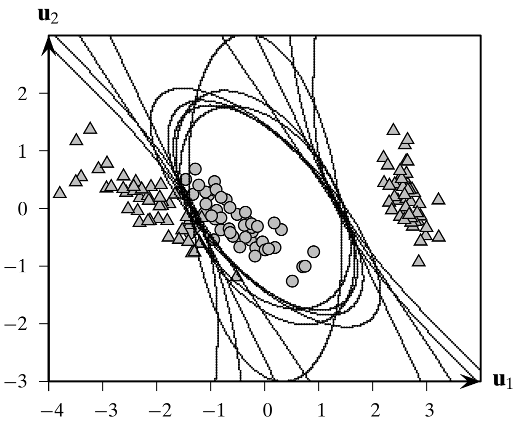
In general, the expected loss can be attributed to high bias or high variance, with typically a trade-off between these two terms. Ideally, we seek a balance between these opposing trends, that is, we prefer a classifier with an acceptable bias (reflecting domain or dataset specific assumptions) and as low a variance as possible.

Example 22.12. Figure 22.7 illustrates the trade-off between bias and variance, using the Iris principal components dataset, which has $n = 150$ points and $k = 2$ classes ($c_1 = +1$, and $c_2 = -1$). We construct $K = 10$ training datasets via bootstrap sampling, and use them to train SVM classifiers using a quadratic (homogeneous) kernel, varying the regularization constant C from 10^{-2} to 10^2.

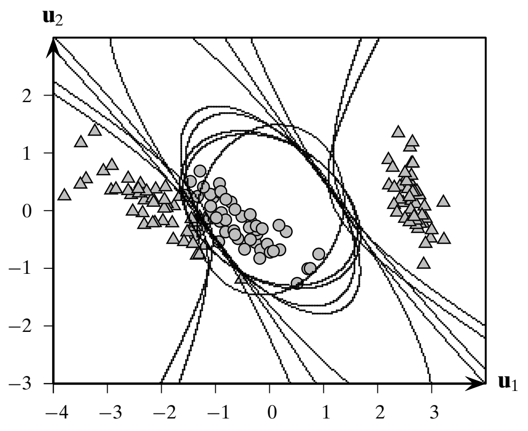
Recall that C controls the weight placed on the slack variables, as opposed to the margin of the hyperplane (see Section 21.3). A small value of C emphasizes the margin, whereas a large value of C tries to minimize the slack terms. Figures 22.7a, 22.7b, and 22.7c show that the variance of the SVM model increases



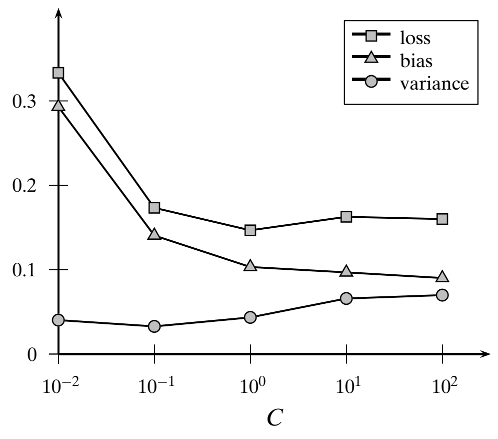
(a) $C = 0.01$



(b) $C = 1$



(c) $C = 100$



(d) Bias-Variance

Figure 22.7. Bias-variance decomposition: SVM quadratic kernels. Decision boundaries plotted for $K = 10$ bootstrap samples.

as we increase C, as seen from the varying decision boundaries. Figure 22.7d plots the average variance and average bias for different values of C, as well as the expected loss. The bias-variance tradeoff is clearly visible, since as the bias reduces, the variance increases. The lowest expected loss is obtained when $C = 1$.

22.3.1 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, which stands for *Bootstrap Aggregation*, is an ensemble classification method that employs multiple bootstrap samples (with replacement) from the input training data $\mathbf{D}$ to create slightly different training sets $\mathbf{D}_i$, $i = 1, 2, \ldots, K$. Different base classifiers M_i are learned, with M_i trained on $\mathbf{D}_i$. Given any test point $\mathbf{x}$, it is first classified using each of the K base classifiers, M_i. Let the number of classifiers that predict the class of $\mathbf{x}$ as c_j be given as

$$v_j(\mathbf{x}) = \Big|\big\{M_i(\mathbf{x}) = c_j \;|\, i = 1, \ldots, K\big\}\Big|$$

The combined classifier, denoted $\mathbf{M}^K$, predicts the class of a test point $\mathbf{x}$ by *majority voting* among the k classes:

$$\mathbf{M}^K(\mathbf{x}) = \underset{c_j}{\arg\max}\Big\{v_j(\mathbf{x}) \;|\, j = 1, \ldots, k\Big\}$$

For binary classification, assuming that the classes are given as $\{+1, -1\}$, the combined classifier $\mathbf{M}^K$ can be expressed more simply as

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{K} M_i(\mathbf{x})\right)$$

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.

Example 22.13. Figure 22.8a shows the averaging effect of bagging for the Iris principal components dataset from Example 22.12. The figure shows the SVM decision boundaries for the quadratic kernel using $C = 1$. The base SVM classifiers are trained on $K = 10$ bootstrap samples. The combined (average) classifier is shown in bold.

Figure 22.8b shows the combined classifiers obtained for different values of K, keeping $C = 1$. The zero–one and squared loss for selected values of K are shown below

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

The worst training performance is obtained for $K = 3$ (in thick gray) and the best for $K = 8$ (in thick black).

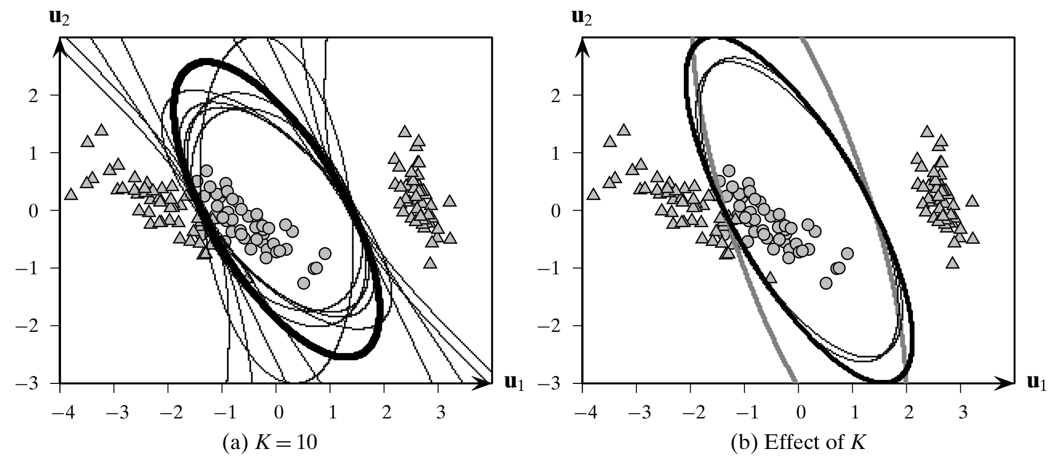


Figure 22.8. Bagging: combined classifiers. (a) uses $K = 10$ bootstrap samples. (b) shows average decision boundary for different values of K.

Boosting

Boosting is another ensemble technique that trains the base classifiers on different samples. However, the main idea is to carefully select the samples to *boost* the performance on hard to classify instances. Starting from an initial training sample $\mathbf{D}_1$, we train the base classifier M_1, and obtain its training error rate. To construct the next sample $\mathbf{D}_2$, we select the misclassified instances with higher probability, and after training M_2, we obtain its training error rate. To construct $\mathbf{D}_3$, 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. Thus, unlike bagging that uses independent random samples from the input dataset, boosting employs weighted or biased samples to construct the different training sets, with the current sample depending on the previous ones. Finally, the combined classifier is obtained via weighted voting over the output of the K base classifiers $M_1, M_2, \dots, M_K$.

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. Several variants of boosting can be obtained based on how the instance weights are computed for sampling, how the base classifiers are combined, and so on. We discuss *Adaptive Boosting (AdaBoost)*, which is one of the most popular variants.

Adaptive Boosting: AdaBoost Let $\mathbf{D}$ be the input training set, comprising n points $\mathbf{x}_i \in \mathbb{R}^d$. The boosting process will be repeated 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 $\mathbf{x}_i$, with $\mathbf{w}^t = (w_1^t, w_2^t, \dots, w_n^t)^T$ being the weight vector over all the points for the tth iteration.

ALGORITHM 22.5. Adaptive Boosting Algorithm: AdaBoost

```
ADABOOST($K$, $\mathbf{D}$):
1  $\mathbf{w}^0 \leftarrow \left(\frac{1}{n}\right) \cdot \mathbf{1} \in \mathbb{R}^n$
2  $t \leftarrow 1$
3  while $t \le K$ do
5      $\mathbf{D}_t \leftarrow$ weighted resampling with replacement from $\mathbf{D}$ using $\mathbf{w}^{t-1}$
6      $M_t \leftarrow$ train classifier on $\mathbf{D}_t$
7      $\epsilon_t \leftarrow \sum_{i=1}^n w_i^{t-1} \cdot I\big(M_t(\mathbf{x}_i) \ne y_i\big)$ // weighted error rate on $\mathbf{D}$
8      if $\epsilon_t = 0$ then break
9      else if $\epsilon_t < 0.5$ then
10         $\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ // classifier weight
11         foreach $i \in [1, n]$ do
               // update point weights
12             $w_i^t = \begin{cases} w_i^{t-1} & \text{if } M_t(\mathbf{x}_i) = y_i \\ w_i^{t-1}\left(\frac{1-\epsilon_t}{\epsilon_t}\right) & \text{if } M_t(\mathbf{x}_i) \ne y_i \end{cases}$
14         $\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t}$ // normalize weights
15         $t \leftarrow t+1$
16 return $\{M_1, M_2, \ldots, M_K\}$
```

In fact, $\mathbf{w}$ is a probability vector, whose elements sum to one. Initially all points have equal weights, that is,

$$\mathbf{w}^0 = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)^T = \frac{1}{n}\mathbf{1}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the n-dimensional vector of all 1's.

The pseudo-code for AdaBoost is shown in Algorithm 22.5. During iteration t, the training sample $\mathbf{D}_t$ is obtained via weighted resampling using the distribution $\mathbf{w}^{t-1}$, 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}. Next, we train the classifier M_t using $\mathbf{D}_t$, and compute its weighted error rate ϵ_t on the entire input dataset $\mathbf{D}$:

$$\epsilon_t = \sum_{i=1}^n w_i^{t-1} \cdot I\big(M_t(\mathbf{x}_i) \ne y_i\big)$$

where I is an indicator function that is 1 when its argument is true, that is, when M_t misclassifies $\mathbf{x}_i$, and is 0 otherwise.

The weight for the tth classifier is then set as

$$\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

and the weight for each point $\mathbf{x}_i \in \mathbf{D}$ is updated based on whether the point is misclassified or not

$$w_i^t = w_i^{t-1} \cdot \exp\Big\{\alpha_t \cdot I\big(M_t(\mathbf{x}_i) \neq y_i\big)\Big\}$$

Thus, if the predicted class matches the true class, that is, if $M_t(\mathbf{x}_i) = y_i$, then $I(M_t(\mathbf{x}_i) \neq y_i) = 0$, and the weight for point $\mathbf{x}_i$ remains unchanged. On the other hand, if the point is misclassified, that is, $M_t(\mathbf{x}_i) \neq y_i$, then we have $I(M_t(\mathbf{x}_i) \neq y_i) = 1$, and

$$w_i^t = w_i^{t-1} \cdot \exp\{\alpha_t\} = w_i^{t-1} \exp\left\{\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\right\} = w_i^{t-1}\left(\frac{1}{\epsilon_t} - 1\right)$$

We can observe that if the error rate ϵ_t is small, then there is a greater weight increment for $\mathbf{x}_i$. 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. On the other hand, if the error rate of the base classifier is close to 0.5, then there is only a small change in the weight, since a bad classifier (with a high error rate) is expected to misclassify many instances. Note that for a binary class problem, an error rate of 0.5 corresponds to a random classifier, that is, one that makes a random guess. Thus, 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 returns to line 5 to try another data sample. Alternatively, one can simply invert the predictions for binary classification. It is worth emphasizing that for a multi-class problem (with $k > 2$), the requirement that $\epsilon_t < 0.5$ is a significantly stronger requirement than for the binary ($k = 2$) class problem because in the multiclass case a random classifier is expected to have an error rate of $\frac{k-1}{k}$. Note also that if the error rate of the base classifier $\epsilon_t = 0$, then we can stop the boosting iterations.

Once the point weights have been updated, we re-normalize the weights so that $\mathbf{w}^t$ is a probability vector (line 14):

$$\mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t} = \frac{1}{\sum_{j=1}^n w_j^t}\left(w_1^t, w_2^t, \ldots, w_n^t\right)^T$$

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 $\mathbf{x}$ is obtained via weighted majority voting. Let $v_j(\mathbf{x})$ denote the weighted vote for class c_j over the K classifiers, given as

$$v_j(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t \cdot I\big(M_t(\mathbf{x}) = c_j\big)$$

Because $I(M_t(\mathbf{x}) = c_j)$ is 1 only when $M_t(\mathbf{x}) = c_j$, the variable $v_j(\mathbf{x})$ simply obtains the tally for class c_j among the K base classifiers, taking into account the classifier weights. The combined classifier, denoted $\mathbf{M}^K$, then predicts the class for $\mathbf{x}$ as follows:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j}\Big\{v_j(\mathbf{x}) \,\big|\, j = 1, .., k\Big\}$$

In the case of binary classification, with classes $\{+1, -1\}$, the combined classifier $\mathbf{M}^K$ can be expressed more simply as

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{K} \alpha_t M_t(\mathbf{x})\right)$$

Example 22.14. Figure 22.9a illustrates the boosting approach on the Iris principal components dataset, using linear SVMs as the base classifiers. The regularization constant was set to $C = 1$. The hyperplane learned in iteration t is denoted h_t, thus, the classifier model is given as $M_t(\mathbf{x}) = \text{sign}(h_t(\mathbf{x}))$. As such, no individual linear hyperplane can discriminate between the classes very well, as seen from their error rates on the training set:

M_t	h_1	h_2	h_3	h_4
ϵ_t	0.280	0.305	0.174	0.282
α_t	0.944	0.826	1.559	0.935

However, when we combine the decisions from successive hyperplanes weighted by α_t, we observe a marked drop in the error rate for the combined classifier $\mathbf{M}^K(\mathbf{x})$ as K increases:

combined model	$\mathbf{M}^1$	$\mathbf{M}^2$	$\mathbf{M}^3$	$\mathbf{M}^4$
training error rate	0.280	0.253	0.073	0.047

We can see, for example, that the combined classifier $\mathbf{M}^3$, comprising h_1, h_2 and h_3, has already captured the essential features of the nonlinear decision boundary between the two classes, yielding an error rate of 7.3%. Further reduction in the training error is obtained by increasing the number of boosting steps.

To assess the performance of the combined classifier on independent testing data, we employ 5-fold cross-validation, and plot the average testing and training error rates as a function of K in Figure 22.9b. We can see that as the number of base

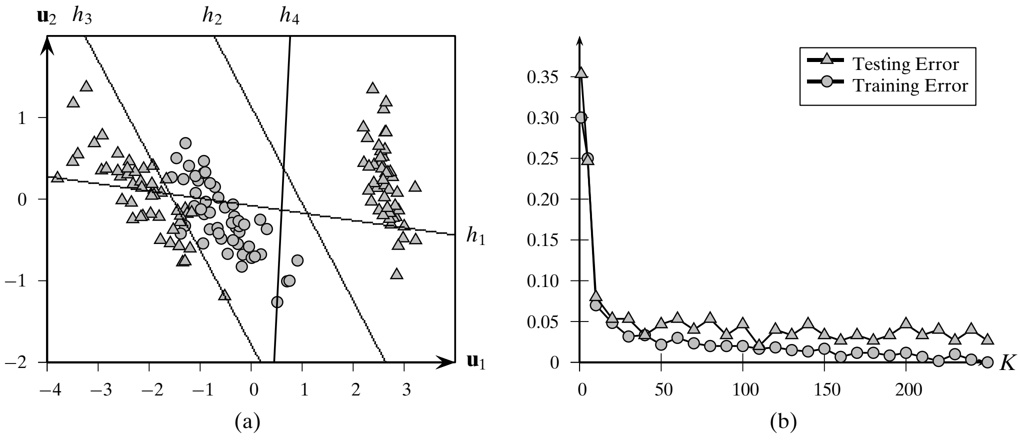


Figure 22.9. (a) Boosting SVMs with linear kernel. (b) Average testing and training error: 5-fold cross-validation.

classifiers K increases, both the training and testing error rates reduce. However, while the training error essentially goes to 0, the testing error does not reduce beyond 0.02, which happens at $K = 110$. This example illustrates the effectiveness of boosting in reducing the bias.

Bagging as a Special Case of AdaBoost: Bagging can be considered as a special case of AdaBoost, where $w^t = \frac{1}{n}\mathbf{1}$, and $\alpha_t = 1$ for all K iterations. In this case, the weighted resampling defaults to regular resampling with replacement, and the predicted class for a test case also defaults to simple majority voting.

22.4 FURTHER READING

The application of ROC analysis to classifier performance was introduced in Provost and Fawcett (1997), with an excellent introduction to ROC analysis given in Fawcett (2006). For an in-depth description of the bootstrap, cross-validation, and other methods for assessing classification accuracy see Efron and Tibshirani (1993). For many datasets simple rules, like one-level decision trees, can yield good classification performance; see Holte (1993) for details. For a recent review and comparison of classifiers over multiple datasets see Demšar (2006). A discussion of bias, variance, and zero–one loss for classification appears in Friedman (1997), with a unified decomposition of bias and variance for both squared and zero–one loss given in Domingos (2000). The concept of bagging was proposed in Breiman (1996), and that of adaptive boosting in Freund and Schapire (1997). Random forests is a tree-based ensemble approach that can be very effective; see Breiman (2001) for details. For a comprehensive overview on the evaluation of classification algorithms see Japkowicz and Shah (2011).

Breiman, L. (1996). "Bagging predictors." *Machine Learning*, 24 (2): 123–140.

Breiman, L. (2001). "Random forests." *Machine Learning*, 45 (1): 5–32.

Demšar, J. (2006). "Statistical comparisons of classifiers over multiple data sets." *The Journal of Machine Learning Research*, 7: 1–30.

Domingos, P. (2000). "A unified bias-variance decomposition for zero-one and squared loss." *In Proceedings of the National Conference on Artificial Intelligence*, 564–569.

Efron, B. and Tibshirani, R. (1993). *An Introduction to the Bootstrap,* vol. 57. Boca Raton, FL: Chapman & Hall/CRC.

Fawcett, T. (2006). "An introduction to ROC analysis." *Pattern Recognition Letters*, 27 (8): 861–874.

Freund, Y. and Schapire, R. E. (1997). "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of Computer and System Sciences*, 55 (1): 119–139.

Friedman, J. H. (1997). "On bias, variance, 0/1-loss, and the curse-of-dimensionality." *Data Mining and Knowledge Discovery*, 1 (1): 55–77.

Holte, R. C. (1993). "Very simple classification rules perform well on most commonly used datasets." *Machine Learning*, 11 (1): 63–90.

Japkowicz, N. and Shah, M. (2011). *Evaluating Learning Algorithms: A Classification Perspective*. New York: Cambridge University Press.

Provost, F. and Fawcett, T. (1997). "Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions." *In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining,* Menlo Park, CA, 43–48.

22.5 EXERCISES

Q1. True or False:

(a) A classification model must have 100% accuracy (overall) on the training dataset.

(b) A classification model must have 100% coverage (overall) on the training dataset.

Q2. Given the training database in Table 22.6a and the testing data in Table 22.6b, answer the following questions:

(a) Build the complete decision tree using binary splits and Gini index as the evaluation measure (see Chapter 19).

(b) Compute the accuracy of the classifier on the test data. Also show the per class accuracy and coverage.

Table 22.6. Data for Q2

X	Y	Z	Class
15	1	A	1
20	3	B	2
25	2	A	1
30	4	A	1
35	2	B	2
25	4	A	1
15	2	B	2
20	3	B	2

(a) Training

X	Y	Z	Class
10	2	A	2
20	1	B	1
30	3	A	2
40	2	B	2
15	1	B	1

(b) Testing

Q3. Show that for binary classification the majority voting for the combined classifier decision in boosting can be expressed as

$$\mathbf{M}^K(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{K} \alpha_t M_t(\mathbf{x})\right)$$

Q4. Consider the 2-dimensional dataset shown in Figure 22.10, with the labeled points belonging to two classes: c_1 (triangles) and c_2 (circles). Assume that the six hyperplanes were learned from different bootstrap samples. Find the error rate for each of the six hyperplanes on the entire dataset. Then, compute the 95% confidence

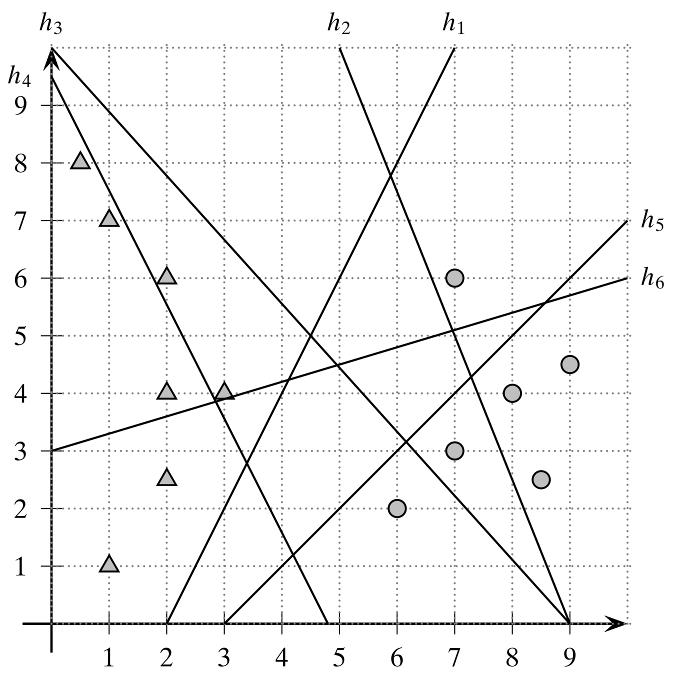


Figure 22.10. For Q4.

Table 22.7. Critical values for t-test

dof	1	2	3	4	5	6
$t_{\alpha/2}$	12.7065	4.3026	3.1824	2.7764	2.5706	2.4469

interval for the expected error rate, using the t-distribution critical values for different degrees of freedom (dof) given in Table 22.7.

Q5. Consider the probabilities $P(+1|x_i)$ for the positive class obtained for some classifier, and given the true class labels y_i

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
y_i	$+1$	-1	$+1$	$+1$	-1	$+1$	-1	$+1$	-1	-1
$P(+1\|x_i)$	0.53	0.86	0.25	0.95	0.87	0.86	0.76	0.94	0.44	0.86

Plot the ROC curve for this classifier.