ارزیابی طبقه‌بندی‌ها

Classification Assessment

CLASSIFICATION PERFORMANCE MEASURES

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.

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

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

CLASSIFICATION PERFORMANCE MEASURES

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 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, wh
density contours (corresponding to one and two standard d
in Figure 22.1. The classifier misclassifies 8 out of the 30 te

$$Error\ Rate = 8/30 = 0.267$$

$$Accuracy = 22/30 = 0.733$$

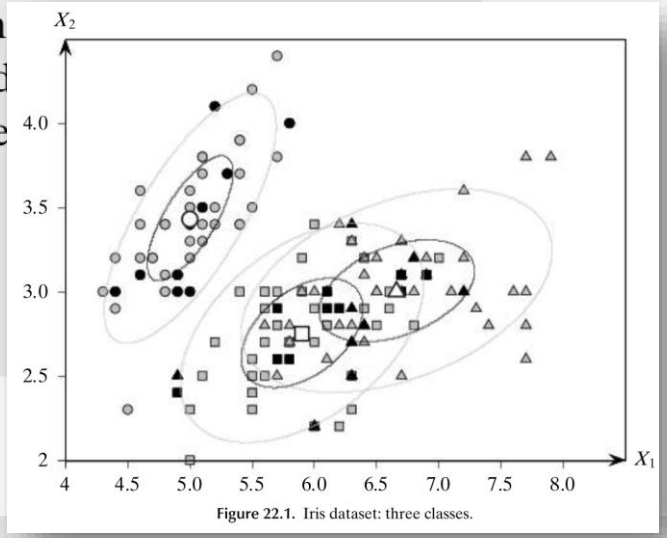


Figure 22.1. Iris dataset: three classes.

CLASSIFICATION PERFORMANCE MEASURES

Contingency Table–based Measures

$$\mathbf{D}_j = \{\mathbf{x}_i^T \mid y_i = c_j\} \quad \text{and} \quad n_i = |\mathbf{D}_i| \qquad \mathbf{R}_j = \{\mathbf{x}_i^T \mid \hat{y}_i = c_j\} \quad \text{and} \quad m_j = |\mathbf{R}_j|$$

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

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

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

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

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

CLASSIFICATION PERFORMANCE MEASURES

Coverage/Recall

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

F-measure

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

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

CLASSIFICATION PERFORMANCE MEASURES

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.

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:

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

CLASSIFICATION PERFORMANCE MEASURES

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

CLASSIFICATION PERFORMANCE MEASURES

Binary Classification: Positive and Negative Class

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)

When there are only $k = 2$ classes, we call class c_1 the positive class and c_2 the negative class.

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

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

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

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

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

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

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

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

CLASSIFICATION PERFORMANCE MEASURES

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

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

Class-specific Precision

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

Sensitivity: True Positive Rate

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

Specificity: True Negative Rate

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

False Negative Rate

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

False Positive Rate

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

CLASSIFICATION PERFORMANCE MEASURES

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.

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

$$Error\ Rate = 10/30 = 0.33$$

$$Accuracy = 20/30 = 0.67$$

CLASSIFICATION PERFORMANCE MEASURES

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

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

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

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

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

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

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

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.

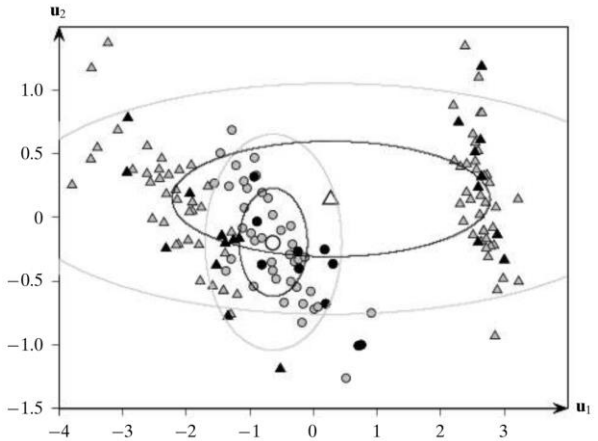


Figure 22.2. Iris principal component dataset: training and testing sets.

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)	$FN = 3$	$TN = 13$	$m_2 = 16$
	$n_1 = 10$	$n_2 = 20$	$n = 30$

CLASSIFICATION PERFORMANCE MEASURES

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:

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

Initially, we classify all points as negative.

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

CLASSIFICATION PERFORMANCE MEASURES

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 (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.

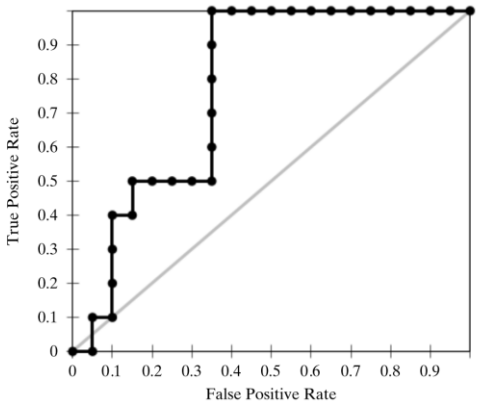


Figure 22.3. ROC plot for Iris principal components dataset. The ROC curves for the naive Bayes (black) and random (gray) classifiers are shown.

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

CLASSIFICATION PERFORMANCE MEASURES

Algorithm 22.1: ROC Curve and Area under the Curve

ROC-CURVE(D, M):

$n_1 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_1\}\right|$ // size of positive class
$n_2 \leftarrow \left|\{\mathbf{x}_i \in \mathbf{D} \mid y_i = c_2\}\right|$ // size of negative class
// classify, score, and sort all test points
$L \leftarrow$ sort the set $\{(S(\mathbf{x}_i), y_i) : \mathbf{x}_i \in \mathbf{D}\}$ by decreasing scores
$FP \leftarrow TP \leftarrow 0$
$FP_{prev} \leftarrow TP_{prev} \leftarrow 0$
$AUC \leftarrow 0$
$\rho \leftarrow \infty$
foreach $(S(\mathbf{x}_i), y_i) \in L$ **do**
  if $\rho > S(\mathbf{x}_i)$ **then**
    plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$
    $AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$
    $\rho \leftarrow S(\mathbf{x}_i)$
    $FP_{prev} \leftarrow FP$
    $TP_{prev} \leftarrow TP$
  if $y_i = c_1$ **then** $TP \leftarrow TP + 1$
  else $FP \leftarrow FP + 1$
plot point $\left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)$
$AUC \leftarrow AUC + \text{TRAPEZOID-AREA}\left(\left(\frac{FP_{prev}}{n_2}, \frac{TP_{prev}}{n_1}\right), \left(\frac{FP}{n_2}, \frac{TP}{n_1}\right)\right)$

TRAPEZOID-AREA($(x_1, y_1), (x_2, y_2)$):

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

CLASSIFICATION PERFORMANCE MEASURES

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:

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

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.

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					

CLASSIFICATION PERFORMANCE MEASURES

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 $n_2 = 2$.

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

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).

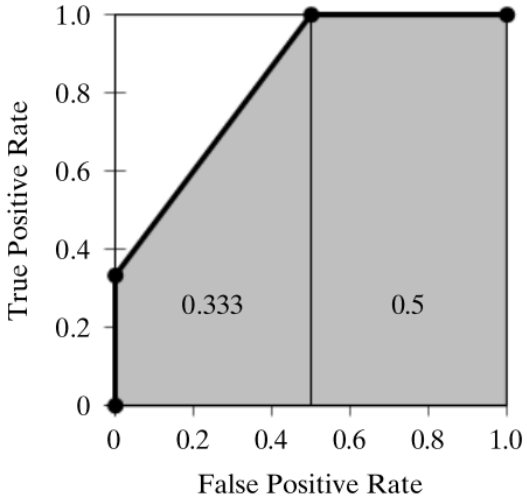


Figure 22.4. ROC plot and AUC: trapezoid region.

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

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

CLASSIFICATION PERFORMANCE MEASURES

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.

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).

CLASSIFIER EVALUATION

Evaluate a classifier M using some performance measure θ.

The training set is used to learn the model M, and the testing set is used to evaluate the measure θ.

D is itself a d-dimensional multivariate random sample drawn from the true (unknown) joint probability density function $f(x)$ 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[θ] from 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 D into K equal-sized parts, called folds, namely $D_1, D_2, \ldots, D_K$. Each fold D_i is, in turn, treated as the testing set, with the remaining folds comprising the training set $D \backslash D_i = \bigcup_{j \neq i} D_j$.

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

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

Algorithm 22.2: K-fold Cross-Validation

```
CROSS-VALIDATION(K, D):
1 D ← randomly shuffle D
2 {D_1, D_2, ..., D_K} ← partition D in K equal parts
3 for i ∈ [1, K] do
4     M_i ← train classifier on D \ D_i
5     θ_i ← assess M_i on D_i
6 μ̂_θ = (1/K) Σ_{i=1}^{K} θ_i
7 σ̂_θ^2 = (1/K) Σ_{i=1}^{K} (θ_i − μ̂_θ)^2
8 return μ̂_θ, σ̂_θ^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 five-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. (22.16) and (22.17), the mean and variance for the error rate are as follows:

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

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 five-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

Instead of partitioning the input dataset D into disjoint folds, the bootstrap method draws K random samples of size n with replacement from D. Each sample D_i is thus the same size as D, and has several repeated points.

Algorithm 22.3: Bootstrap Resampling Method

BOOTSTRAP-RESAMPLING(K, D):

1 **for** $i \in [1, K]$ **do**
2 $\quad \mathbf{D}_i \leftarrow$ sample of size n with replacement from $\mathbf{D}$
3 $\quad M_i \leftarrow$ train classifier on $\mathbf{D}_i$
4 $\quad \theta_i \leftarrow$ 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** $\hat{\mu}_\theta, \hat{\sigma}_\theta^2$

Bootstrap Resampling

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 $q = 1 - \frac{1}{n}$. The probability that x_j is not selected even after n tries is given as

$$P(x_j \notin D_i) = q^n = \left(1 - \frac{1}{n}\right)^n \simeq e^{-1} = 0.368 \Rightarrow P(x_j \in D_i) = 1 - 0.368 = 0.632$$

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. The expected value and variance of the error rate are

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

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 $\hat{\sigma}^2_\theta = 0.00833$.

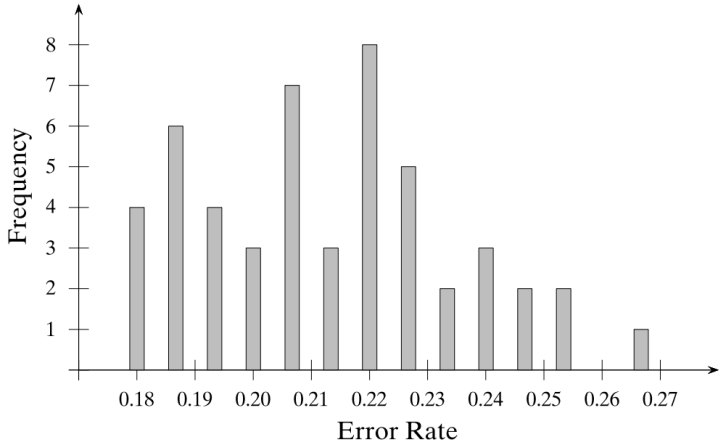


Figure 22.5. Sampling distribution of error rates.

Confidence Intervals

Using the central limit theorem, regardless of the distribution of the individual random variables and just let θ_i be a sequence of IID random variables.

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

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

$$Z_K = \frac{\hat{\mu} - E[\hat{\mu}]}{std(\hat{\mu})} = \frac{\hat{\mu} - \mu}{\frac{\sigma}{\sqrt{K}}} = \sqrt{K}\left(\frac{\hat{\mu} - \mu}{\sigma}\right) \qquad \lim_{K\to\infty} P(Z_K \le x) = \Phi(x)$$

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

Example 22.9. Consider Example 22.7, where we applied five-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:

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

Let $1 - \alpha = 0.95$ be the confidence level, so that $\alpha = 0.05$ is he significance level. 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 $1 - \alpha = 0.99$ (or $\alpha = 0.01$), 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

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

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

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.

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

Example 22.10. Consider Example 22.9. For five-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 $1 - \alpha = 0.95$ (or $\alpha = 0.05$), we use the quantile function for the Student's t-distribution to obtain $t_{\alpha/2} = 2.776$. Thus,

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

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

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

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 in Example 22.9.

Comparing Classifiers: Paired t-Test

To test for a significant difference in the classification performance of two alternative classifiers, 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.

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

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

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

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

Given a desired confidence level $1-\alpha$ (or significance level α), we conclude that

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

CLASSIFIER EVALUATION

Algorithm 22.4: Paired t-Test via Cross-Validation

PAIRED t-TEST($1-\alpha$, K, $\mathbf{D}$):

1 $\mathbf{D} \leftarrow$ randomly shuffle $\mathbf{D}$

2 $\{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_K\} \leftarrow$ partition $\mathbf{D}$ in K equal parts

3 **for** $i \in [1, K]$ **do**

4 $\quad M_i^A, M_i^B \leftarrow$ train the two different classifiers on $\mathbf{D} \setminus \mathbf{D}_i$

5 $\quad \theta_i^A, \theta_i^B \leftarrow$ assess M_i^A and M_i^B on $\mathbf{D}_i$

6 $\quad \delta_i \leftarrow \theta_i^A - \theta_i^B$

7 $\hat{\mu}_\delta \leftarrow \frac{1}{K}\sum_{i=1}^{K} \delta_i$ // mean

8 $\hat{\sigma}_\delta^2 \leftarrow \frac{1}{K}\sum_{i=1}^{K} (\delta_i - \hat{\mu}_\delta)^2$ // variance

9 $Z_\delta^* \leftarrow \frac{\sqrt{K} \cdot \hat{\mu}_\delta}{\hat{\sigma}_\delta}$ // test statistic value

10 $t_{\alpha/2} \leftarrow T_{K-1}^{-1}(1-\alpha/2)$ // compute critical value

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

12 $\quad$ Accept H_0; both classifiers have similar performance

13 **else**

14 $\quad$ 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 $1 - \alpha = 0.95$ (or $\alpha = 0.05$) and $K - 1 = 4$ degrees of freedom, we have $t_{\alpha/2} = 2.776$. Because

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

we cannot reject the null hypothesis. 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.

BIAS-VARIANCE DECOMPOSITION

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.

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

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

Expected Loss

Let $M(x) = c_i$, 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,
$c_i = \underset{y}{\operatorname{argmax}} P(y|\boldsymbol{x})$.

BIAS-VARIANCE DECOMPOSITION

Bias and Variance

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

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

BIAS-VARIANCE DECOMPOSITION

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

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 $var(y|x)$ over all test points x. It contributes a fixed cost to the loss independent of the model, and can thus be ignored when comparing different classifiers.

BIAS-VARIANCE DECOMPOSITION

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.

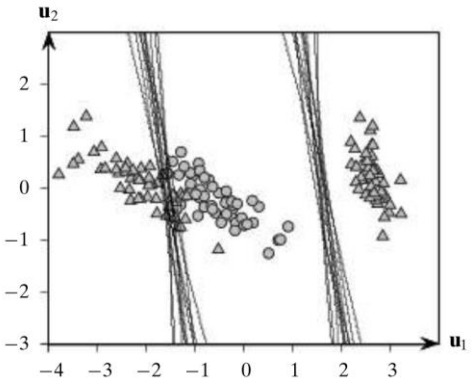
Ideally, we seek a balance between these opposing trends, that is, we prefer a classifier with an acceptable bias (reflecting domain or dataset specific assumptions) and as low a variance as possible.

BIAS-VARIANCE DECOMPOSITION

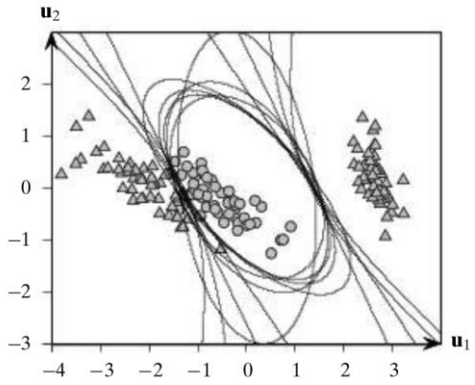
Example 22.12. Figure 22.7 illustrates the trade-off between bias and variance, using the Iris principal components dataset, which has $n = 150$ points and $k = 2$ classes ($c_1 = +1$, and $c_2 = -1$). We construct $K = 10$ training datasets via bootstrap sampling, and use them to train SVM classifiers 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 22.7(a), 22.7(b), and 22.7(c) show that the variance of the SVM model increases as we increase C, as seen from the varying decision boundaries. Figure 22.7(d) 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 $C = 1$.

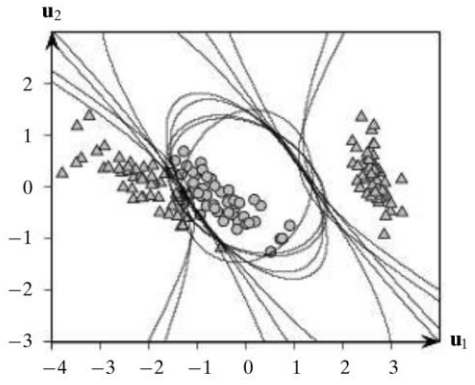
BIAS-VARIANCE DECOMPOSITION



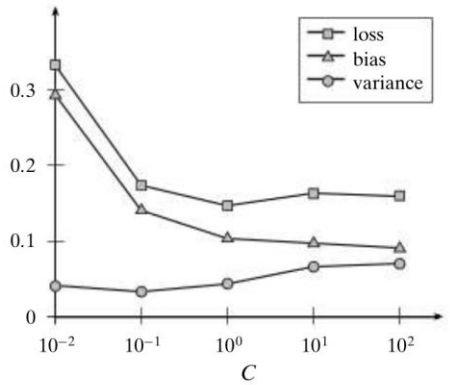
(a) $C = 0.01$



(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.

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.

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 $\boldsymbol{D}$ to create slightly different training sets $D_t, t = 1,2, \ldots, K$. Different base classifiers M_t are learned, with M_t trained on $\boldsymbol{D_t}$. Given any test point x, it is first classified using each of the K base classifiers, Mt. Let the number of classifiers that predict the class of x as cj be given as

$$v_j(\mathbf{x}) = \left| \{ M_t(\mathbf{x}) = c_j \mid t = 1, \ldots, K \} \right|$$

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j} \{ v_j(\mathbf{x}) \mid j = 1, \ldots, k \}$$

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 22.8(a) 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 22.8(b) 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).

ENSEMBLE CLASSIFIERS

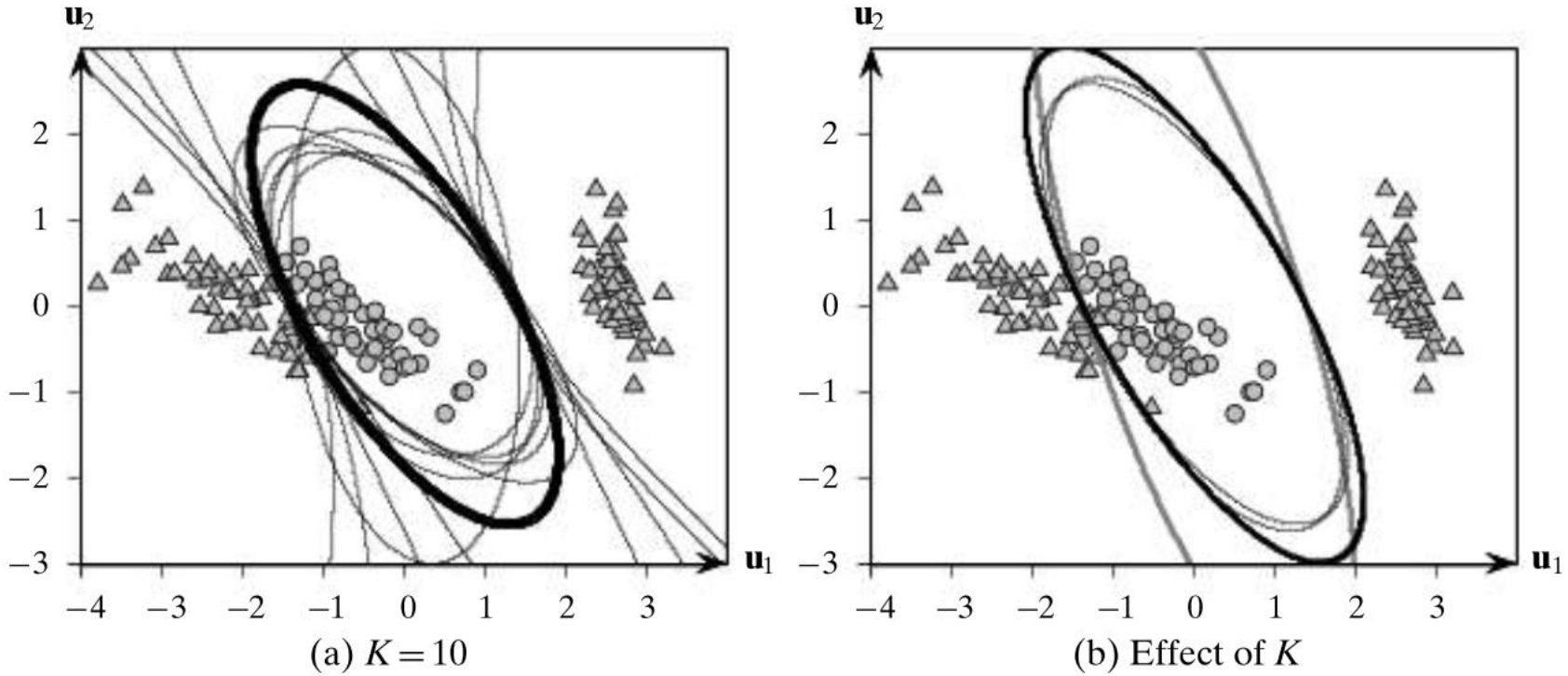


Figure 22.8. Bagging: combined classifiers. (a) uses $K = 10$ bootstrap samples. (b) shows average decision boundary for different values of K.

Random Forest: Bagging Decision Trees

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, as in bagging.

In general, decision trees can model complex decision boundaries between classes, and therefore have a low bias but high variance.

What we require is to have a diversity of trees, so that when we average their decisions, we will see much more of the variance reducing effects of bagging.

Random Forest: Bagging Decision Trees

Instead of evaluating all the d attributes to find the best split point, it samples $p \leq d$ attributes at random, and evaluates split points for only those attributes. A typical value of p is the square root of the number of attributes, i.e., $p = \sqrt{d}$, though this can be tuned for different datasets. The K decision trees $M_1, M_2, \cdots, M_K$ comprise the random forest model $\boldsymbol{M^K}$, which predicts the class of a test point x by majority voting as in bagging:

$$\mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j} \left\{ v_j(\mathbf{x}) \,\middle|\, j = 1, \ldots, k \right\} \qquad v_j(\mathbf{x}) = \left| \left\{ M_t(\mathbf{x}) = c_j \,\middle|\, t = 1, \ldots, K \right\} \right|$$

Notice that if $p = d$ then the random forest approach is equivalent to bagging over decision tree models.

Given bootstrap sample $\boldsymbol{D_t}$, any point in $\boldsymbol{D \backslash D_t}$ is called an **out-of-bag** point for classifier M_t, since it was not used to train $\boldsymbol{M_t}$. One of the side-benefits of the bootstrap approach is that we can compute the out-of-bag error rate for the random forest by considering the prediction of each model $\boldsymbol{M_t}$ over its out-of-bag points.

Random Forest: Bagging Decision Trees

Algorithm 22.5: Random Forest Algorithm

RANDOMFOREST($\mathbf{D}, K, p, \eta, \pi$):

foreach $\mathbf{x}_i \in \mathbf{D}$ **do**
  $v_j(\mathbf{x}_i) \leftarrow 0$, for all $j = 1, 2, \ldots, k$
for $t \in [1, K]$ **do**
  $\mathbf{D}_t \leftarrow$ sample of size n with replacement from $\mathbf{D}$
  $M_t \leftarrow$ DECISIONTREE $(\mathbf{D}_t, \eta, \pi, p)$
  foreach $(\mathbf{x}_i, y_i) \in \mathbf{D} \setminus \mathbf{D}_t$ **do** // `out-of-bag votes`
    $\hat{y}_i \leftarrow M_t(\mathbf{x}_i)$
    if $\hat{y}_i = c_j$ **then** $v_j(\mathbf{x}_i) = v_j(\mathbf{x}_i) + 1$
$\epsilon_{oob} = \frac{1}{n} \cdot \sum_{i=1}^{n} I\Big(y_i \neq \arg\max_{c_j} \big\{v_j(\mathbf{x}_i) | (\mathbf{x}_i, y_i) \in \mathbf{D}\big\}\Big)$ // `OOB error`
return $\{M_1, M_2, \ldots, M_K\}$

ENSEMBLE CLASSIFIERS

Example 22.14 (Random Forest). We illustrate the random forest approach on the Iris principal components dataset comprising $n = 150$ points in 2-dimensional space, as shown in Figure 22.9. The task is to separate `Iris-versicolor` (class c_1; in circles) from the other two Irises (class c_2; in triangles). Since there are only two attributes in this dataset, we pick $p = 1$ attribute at random for each split-point evaluation in a decision tree. Each decision tree is grown using $\eta = 3$, that is the maximum leaf size is 3 (with default minimum purity $\pi = 1.0$). We grow $K = 5$ decision trees on different bootstrap samples. The decision boundary of the random forest is shown in bold in the figure. The error rate on the training data is 2.0%. However, the out-of-bag 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.

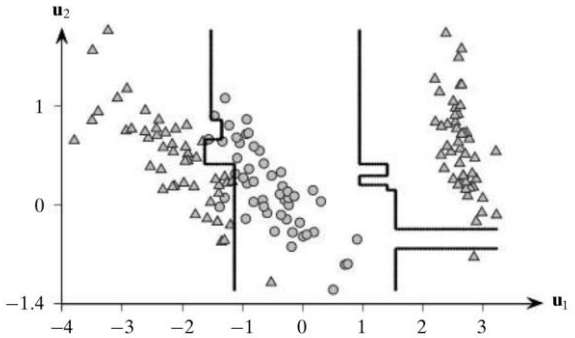


Figure 22.9. Random Forest: Iris principal components dataset ($K = 5$).

Example 22.15 (Random Forest: Varying K). To get a better understanding of the out-of-bag error rate and the number of trees in the random forest, we used the full Iris dataset which has four attributes ($d = 4$), and has three classes ($k = 3$), denoting the three Iris types. We used $p = \sqrt{d} = 2$, so that each decision tree samples only two of the attributes for evaluating each split point. We set $\eta = 3$ and $\pi = 1.0$.

Table 22.6 shows the out-of-bag error ϵ_{oob} and training error ϵ for the random forest model with different number of trees, ranging from $K = 1$ to $K = 10$. Whereas we get low training error rates with only a few trees, we can see that the out-of-bag error decreases rapidly as we increase the number of trees. For this dataset around 9 or 10 trees are sufficient to get low out-of-bag error rates.

Table 22.6. Random Forest: Iris data, varying K

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

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 $\boldsymbol{D_1}$, we train the base classifier $\boldsymbol{M_1}$, 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 $\boldsymbol{M_2}$, we obtain its training error rate. To construct 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 so on. Thus, boosting has more of a bias reducing effect. Each of the weak learners is likely to have high bias, 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

Initially all points have equal weights, that is,

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

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 $\boldsymbol{w_i^{t-1}}$.

Next, we train the classifier $\boldsymbol{M^t}$ using $\boldsymbol{D_t}$, and compute its weighted error rate ϵ_t on the entire input dataset $\boldsymbol{D}$:

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

where $\boldsymbol{I}$ is an indicator function that is 1 when its argument is true.

If the predicted class matches the true class, that is, if $\boldsymbol{M_t(x_i) = y_i}$, then the weight for point $\boldsymbol{x_i}$ remains unchanged. Otherwise, the weight for each point $\boldsymbol{x_i \in D}$ is updated.

Adaptive Boosting: AdaBoost

The weight α_t for the tth classifier is then set as $\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

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

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

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

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 $\boldsymbol{x}$ is obtained via weighted majority voting.

$$v_j(\mathbf{x}) = \sum_{t=1}^{K} \alpha_t \cdot I\left(M_t(\mathbf{x}) = c_j\right) \qquad \mathbf{M}^K(\mathbf{x}) = \arg\max_{c_j}\left\{v_j(\mathbf{x}) \,\middle|\, j = 1, .., k\right\}$$

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.

Algorithm 22.6: Adaptive Boosting Algorithm: AdaBoost

ADABOOST(K, D):

1 $\mathbf{w}^0 \leftarrow \left(\frac{1}{n}\right) \cdot \mathbf{1} \in \mathbb{R}^n$

2 $t \leftarrow 1$

3 **while** $t \leq K$ **do**

4 $\quad \mathbf{D}_t \leftarrow$ weighted resampling with replacement from $\mathbf{D}$ using $\mathbf{w}^{t-1}$

5 $\quad M_t \leftarrow$ train classifier on $\mathbf{D}_t$

6 $\quad \epsilon_t \leftarrow \sum_{i=1}^{n} w_i^{t-1} \cdot I\big(M_t(\mathbf{x}_i) \neq y_i\big)$ // weighted error rate on D

7 $\quad$ **if** $\epsilon_t = 0$ **then break**

8 $\quad$ **else if** $\epsilon_t < 0.5$ **then**

9 $\quad\quad \alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ // classifier weight

10 $\quad\quad$ **for** $i \in [1, n]$ **do**

$\quad\quad\quad$ // update point weights

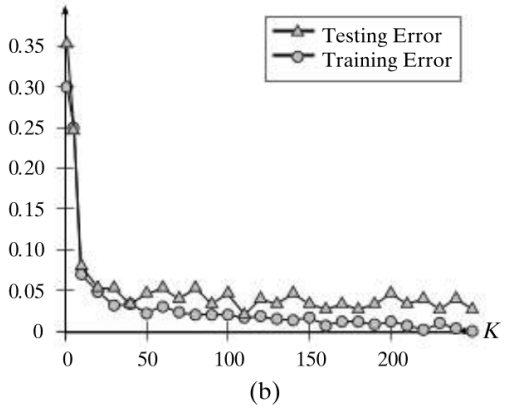
11 $\quad\quad\quad w_i^t = \begin{cases} w_i^{t-1} & \text{if } M_t(\mathbf{x}_i) = y_i \\ w_i^{t-1}\left(\frac{1-\epsilon_t}{\epsilon_t}\right) & \text{if } M_t(\mathbf{x}_i) \neq y_i \end{cases}$

12 $\quad\quad \mathbf{w}^t = \frac{\mathbf{w}^t}{\mathbf{1}^T\mathbf{w}^t}$ // normalize weights

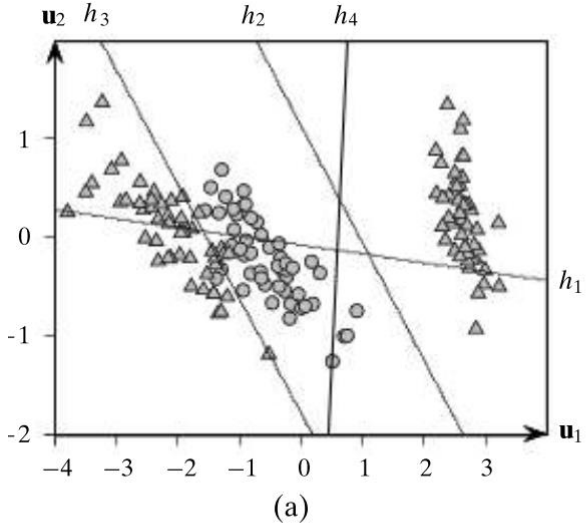
13 $\quad\quad t \leftarrow t + 1$

14 **return** $\{M_1, M_2, \ldots, M_K\}$

ENSEMBLE CLASSIFIERS



(b)



(a)

Example 22.16. Figure 22.10(a) 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 five-fold cross-validation, and plot the average testing and training error rates as a function of K in Figure 22.10(b). We can see that, as the number of base 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.

Figure 22.10. (a) Boosting SVMs with linear kernel. (b) Average testing and training error: five-fold cross-validation.

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 $\boldsymbol{D}$. However, the base classifiers should differ from or be complementary to each other as much as possible so that they perform well on different subsets of the input space. 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.

We create the training dataset, $\boldsymbol{Z}$, for C as follows: For each point $\boldsymbol{x}_i$ in $\boldsymbol{D}$, we create the point $z_i \in R^K$ that records the predicted class from each of the base classifiers. That is,

$$\mathbf{z}_i = \big(M_1(\mathbf{x}_i), M_2(\mathbf{x}_i), \ldots, M_K(\mathbf{x}_i)\big)^T$$

The pairs $(\boldsymbol{z}_i, y_i)$ for $i = 1,2,\cdots, n$ comprise the training dataset $\boldsymbol{Z}$ used to train C. The algorithm returns both the set of base classifiers and the combiner model.

Stacking

Algorithm 22.7: Stacking Algorithm

```
STACKING(K, M, C, D):
// Train base classifiers
for t ∈ [1, K] do
    M_t ← train tth base classifier on D
// Train combiner model C on Z
Z ← ∅
foreach (x_i, y_i) ∈ D do
    z_i ← (M_1(x_i), M_2(x_i), ..., M_K(x_i))^T
    Z ← Z ∪ {(z_i, y_i)}
C ← train combiner classifier on Z
return (C, M_1, M_2, ..., M_K)
```

Example 22.17 (Stacking). We apply stacking on the Iris principal components dataset. We use three base classifiers, namely SVM with a linear kernel (with regularization constant $C = 1$), random forests (with number of trees $K = 5$ and number of random attributes $p = 1$), and naive Bayes. The combiner classifier is an SVM with a Gaussian kernel (with regularization constant $C = 1$ and spread parameter $\sigma^2 = 0.2$).

We trained the data on a random subset of 100 points, and tested on the remaining 50 points. Figure 22.11 shows the decision boundaries for each of the three base classifiers and the combiner model: linear SVM boundary is the line in light gray, naive Bayes boundary is the ellipse comprising the gray squares, random forest boundary is shown via plusses ('+'), and finally the boundary of the stacking classifier is shown as the thicker black lines. The stacked model results in a much better accuracy compared to the base classifiers, as shown in Table 22.7.

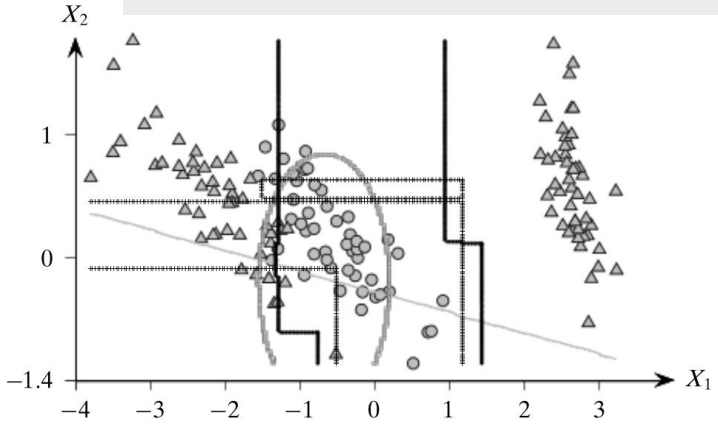


Figure 22.11. Stacking: Iris PC dataset.

Table 22.7. Stacking versus other classifiers

Classifier	Test Accuracy
Linear SVM	0.68
Random Forest	0.82
Naive Bayes	0.74
Stacking	0.92