Chapter 22: Classification Assessment
A classifier is a model or function $M$ that predicts the class label $\hat{y}$ for a given input example $x$:

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

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

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

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

Finally, the model can be deployed to predict the class for future points whose class we typically do not know.
Let $D$ 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 $x_i \in D$, let $y_i$ denote its true class, and let $\hat{y}_i = M(x_i)$ denote its predicted class.

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

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

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

**Accuracy:** The accuracy of a classifier is the fraction of correct predictions over the testing set:

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

Accuracy gives an estimate of the probability of a correct prediction; thus, the higher the accuracy, the better the classifier.
Iris Data: Full Bayes Classifier

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

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

\[
\text{Error Rate} = \frac{8}{30} = 0.27
\]

\[
\text{Accuracy} = \frac{22}{30} = 0.73
\]
Let $\mathcal{D} = \{D_1, D_2, \ldots, D_k\}$ denote a partitioning of the testing points based on their true class labels, where $D_j = \{x_i \in \mathcal{D} | y_i = c_j\}$. Let $n_i = |D_i|$ denote the size of true class $c_i$.

Let $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ denote a partitioning of the testing points based on the predicted labels, that is, $R_j = \{x_i \in \mathcal{D} | \hat{y}_i = c_j\}$. Let $m_j = |R_j|$ denote the size of the predicted class $c_j$.

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

$$
N(i, j) = n_{ij} = |R_i \cap D_j| = \left| \{x_a \in \mathcal{D} | \hat{y}_a = c_i \text{ and } y_a = c_j\} \right|
$$

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.
The class-specific accuracy or precision of the classifier $M$ for class $c_i$ is given as the fraction of correct predictions over all points predicted to be in class $c_i$
\[
acc_i = prec_i = \frac{n_{ii}}{m_i}
\]
where $m_i$ is the number of examples predicted as $c_i$ by classifier $M$. The higher the accuracy on class $c_i$ the better the classifier.

The overall precision or accuracy of the classifier is the weighted average of class-specific accuracies:
\[
\text{Accuracy} = \text{Precision} = \sum_{i=1}^{k} \left( \frac{m_i}{n} \right) acc_i = \frac{1}{n} \sum_{i=1}^{k} n_{ii}
\]

The class-specific coverage or recall of $M$ for class $c_i$ is the fraction of correct predictions over all points in class $c_i$:
\[
\text{coverage}_i = \text{recall}_i = \frac{n_{ii}}{n_i}
\]
The higher the coverage the better the classifier.
The class-specific $F$-measure tries to balance the precision and recall values, by computing their harmonic mean for class $c_i$:

$$F_i = \frac{2}{\frac{1}{\text{prec}_i} + \frac{1}{\text{recall}_i}} = \frac{2 \cdot \text{prec}_i \cdot \text{recall}_i}{\text{prec}_i + \text{recall}_i} = \frac{2 \ n_{ii}}{n_i + m_i}$$

The higher the $F_i$ value the better the classifier.

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

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

For a perfect classifier, the maximum value of the $F$-measure is 1.
Contingency Table for Iris: Full Bayes Classifier

<table>
<thead>
<tr>
<th>Predicted</th>
<th>True</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa (c₁)</td>
<td>Iris-versicolor (c₂)</td>
<td>Iris-virginica (c₃)</td>
</tr>
<tr>
<td>Iris-setosa (c₁)</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Iris-versicolor (c₂)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Iris-virginica (c₃)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>(n₁ = 10)</td>
<td>(n₂ = 10)</td>
<td>(n₃ = 10)</td>
</tr>
</tbody>
</table>

The class-specific precision and recall values are:

\[
\begin{align*}
prec_1 &= \frac{n_{11}}{m_1} = \frac{10}{10} = 1.0 \\
recall_1 &= \frac{n_{11}}{n_1} = \frac{10}{10} = 1.0 \\
prec_2 &= \frac{n_{22}}{m_2} = \frac{7}{12} = 0.583 \\
recall_2 &= \frac{n_{22}}{n_2} = \frac{7}{10} = 0.7 \\
prec_3 &= \frac{n_{33}}{m_3} = \frac{5}{8} = 0.625 \\
recall_3 &= \frac{n_{33}}{n_3} = \frac{5}{10} = 0.5
\end{align*}
\]

The overall accuracy and F-measure is

\[
\begin{align*}
Accuracy &= \frac{(n_{11} + n_{22} + n_{33})}{n} = \frac{(10 + 7 + 5)}{30} = \frac{22}{30} = 0.733 \\
F &= \frac{1}{3} (1.0 + 0.636 + 0.556) = \frac{2.192}{3} = 0.731
\end{align*}
\]
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 $2 \times 2$ confusion matrix are:

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>True Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive ($c_1$)</td>
<td>Positive ($c_1$)</td>
</tr>
<tr>
<td></td>
<td>True Positive ($TP$)</td>
</tr>
<tr>
<td>Negative ($c_2$)</td>
<td>False Positive ($FP$)</td>
</tr>
<tr>
<td></td>
<td>False Negative ($FN$)</td>
</tr>
<tr>
<td></td>
<td>True Negative ($TN$)</td>
</tr>
</tbody>
</table>
Binary Classification: Positive and Negative Class

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

  \[ TP = n_{11} = \left| \{ x_i \mid \hat{y}_i = y_i = c_1 \} \right| \]

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

  \[ FP = n_{12} = \left| \{ x_i \mid \hat{y}_i = c_1 \text{ and } y_i = c_2 \} \right| \]

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

  \[ FN = n_{21} = \left| \{ x_i \mid \hat{y}_i = c_2 \text{ and } y_i = c_1 \} \right| \]

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

  \[ TN = n_{22} = \left| \{ x_i \mid \hat{y}_i = y_i = c_2 \} \right| \]
Binary Classification: Assessment Measures

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

\[
\text{Error Rate} = \frac{FP + FN}{n}
\]

**Accuracy:** The accuracy is the fraction of correct predictions:

\[
\text{Accuracy} = \frac{TP + TN}{n}
\]

The precision for the positive and negative class is given as

\[
\text{prec}_P = \frac{TP}{TP + FP} = \frac{TP}{m_1}
\]

\[
\text{prec}_N = \frac{TN}{TN + FN} = \frac{TN}{m_2}
\]

where \( m_i = |R_i| \) is the number of points predicted by \( M \) as having class \( c_i \).
Binary Classification: Assessment Measures

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

\[
TPR = \text{recall}_P = \frac{TP}{TP + FN} = \frac{TP}{n_1}
\]

where \(n_1\) is the size of the positive class.

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

\[
TNR = \text{specificity} = \text{recall}_N = \frac{TN}{FP + TN} = \frac{TN}{n_2}
\]

where \(n_2\) is the size of the negative class.

**False Negative Rate:** Defined as

\[
FNR = \frac{FN}{TP + FN} = \frac{FN}{n_1} = 1 - \text{sensitivity}
\]

**False Positive Rate:** Defined as

\[
FPR = \frac{FP}{FP + TN} = \frac{FP}{n_2} = 1 - \text{specificity}
\]
Iris Principal Components Data: Naive Bayes Classifier

Iris-versicolor (class $c_1$; in circles) and other two Irises (class $c_2$; in triangles). Training data in grey and testing data in black. Class means in white and density contours.
Iris Principal Components Data: Assessment Measures

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicted</td>
</tr>
<tr>
<td></td>
<td>Positive ($c_1$)</td>
</tr>
<tr>
<td>Positive ($c_1$)</td>
<td>$TP = 7$</td>
</tr>
<tr>
<td>Negative ($c_2$)</td>
<td>$FN = 3$</td>
</tr>
<tr>
<td></td>
<td>$n_1 = 10$</td>
</tr>
<tr>
<td></td>
<td>Negative ($c_2$)</td>
</tr>
<tr>
<td></td>
<td>$FP = 7$</td>
</tr>
<tr>
<td></td>
<td>$m_1 = 14$</td>
</tr>
<tr>
<td></td>
<td>$TN = 13$</td>
</tr>
<tr>
<td></td>
<td>$m_2 = 16$</td>
</tr>
<tr>
<td></td>
<td>$n_2 = 20$</td>
</tr>
<tr>
<td></td>
<td>$n = 30$</td>
</tr>
</tbody>
</table>

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

$$\text{Error Rate} = \frac{10}{30} = 0.33 \quad \text{Accuracy} = \frac{20}{30} = 0.67$$

Other performance measures:

$$\text{prec}_P = \frac{TP}{TP + FP} = \frac{7}{14} = 0.5$$
$$\text{prec}_N = \frac{TN}{TN + FN} = \frac{13}{16} = 0.8125$$

$$\text{sensitivity} = \frac{TP}{TP + FN} = \frac{7}{10} = 0.7$$
$$\text{specificity} = \frac{TN}{TN + FP} = \frac{13}{20} = 0.65$$

$$\text{FNR} = 1 - \text{sensitivity} = 0.3$$
$$\text{FPR} = 1 - \text{specificity} = 0.35$$
Receiver Operating Characteristic (ROC) analysis is a popular strategy for assessing the performance of classifiers when there are two classes.

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

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

ROC analysis plots the performance of the classifier over all possible values of the threshold parameter $\rho$.

In particular, for each value of $\rho$, 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 $S(x_i)$ denote the real-valued score for the positive class output by a classifier $M$ for the point $x_i$. Let the maximum and minimum score thresholds observed on testing dataset $D$ be as follows:

$$ \rho_{\text{min}} = \min_i \{ S(x_i) \} \quad \quad \rho_{\text{max}} = \max_i \{ S(x_i) \} $$

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

<table>
<thead>
<tr>
<th>Predicted</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pos</td>
</tr>
<tr>
<td>Pos</td>
<td>0</td>
</tr>
<tr>
<td>Neg</td>
<td>$FN$</td>
</tr>
</tbody>
</table>

This results in $TPR$ and $FPR$ rates of zero, which correspond to the point $(0, 0)$ at the lower left corner in the ROC plot.
Next, for each distinct value of \( \rho \) in the range \([\rho_{\text{min}}, \rho_{\text{max}}]\), we tabulate the set of positive points:

\[
R_1(\rho) = \{x_i \in D : S(x_i) > \rho\}
\]

and we compute the corresponding true and false positive rates, to obtain a new point in the ROC plot.

Finally, in the last step, we classify all points as positive. Both \( FN \) and \( TN \) are thus zero, as per the confusion matrix:

<table>
<thead>
<tr>
<th>Predicted</th>
<th>True</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pos</td>
<td>Neg</td>
</tr>
<tr>
<td>Pos</td>
<td>( TP )</td>
<td>( FP )</td>
</tr>
<tr>
<td>Neg</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 the confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pos</td>
</tr>
<tr>
<td>Pos</td>
<td>(TP)</td>
</tr>
<tr>
<td>Neg</td>
<td>0</td>
</tr>
</tbody>
</table>

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

**Area Under ROC Curve:** The area under the ROC curve, abbreviated AUC, can be used as a measure of classifier performance. The AUC value is essentially the probability that the classifier will rank a random positive test case higher than a random negative test instance.
### ROC: Different Cases for $2 \times 2$ Confusion Matrix

<table>
<thead>
<tr>
<th>Predicted</th>
<th>True</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Pos</td>
<td>0</td>
</tr>
<tr>
<td>Neg</td>
<td>Neg</td>
<td>FN</td>
</tr>
</tbody>
</table>

(a) Initial: all negative

<table>
<thead>
<tr>
<th>Predicted</th>
<th>True</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Pos</td>
<td>TP</td>
</tr>
<tr>
<td>Neg</td>
<td>Neg</td>
<td>FP</td>
</tr>
</tbody>
</table>

(b) Final: all positive

<table>
<thead>
<tr>
<th>Predicted</th>
<th>True</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Pos</td>
<td>0</td>
</tr>
<tr>
<td>Neg</td>
<td>Neg</td>
<td>0</td>
</tr>
</tbody>
</table>

(c) Ideal classifier
A random classifier corresponds to a diagonal line in the ROC plot.

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

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

The diagonal line thus represents the performance of a random classifier, over all possible positive class prediction thresholds \(r\).
The ROC/AUC takes as input the testing set $D$, and the classifier $M$.

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

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

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

As each test point is examined, the true and false positive values are adjusted based on the true class $y_i$ for the test point $x_i$. If $y_1 = c_1$, we increment the true positives, otherwise, we increment the false positives.
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, $FP_{\text{prev}}$ and $TP_{\text{prev}}$, for the previous score threshold $\rho$.

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

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

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

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

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

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

\[ \text{TRAPEZOID-AREA}((x_1, y_1), (x_2, y_2)):\]

1. \( b \leftarrow |x_2 - x_1| \) // base of trapezoid
2. \( h \leftarrow \frac{1}{2}(y_2 + y_1) \) // average height of trapezoid
3. return \((b \cdot h)\)
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 $x_i$ is therefore $S(x_i) = P(c_1|x_i)$. The sorted scores (in decreasing order) along with the true class labels are as follows:

<table>
<thead>
<tr>
<th>$S(x_i)$</th>
<th>0.93</th>
<th>0.82</th>
<th>0.80</th>
<th>0.77</th>
<th>0.74</th>
<th>0.71</th>
<th>0.69</th>
<th>0.67</th>
<th>0.66</th>
<th>0.61</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_i$</td>
<td>$c_2$</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$c_1$</td>
<td>$c_1$</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$c_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S(x_i)$</th>
<th>0.59</th>
<th>0.55</th>
<th>0.55</th>
<th>0.53</th>
<th>0.47</th>
<th>0.30</th>
<th>0.26</th>
<th>0.11</th>
<th>0.04</th>
<th>2.97e-03</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_i$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_1$</td>
<td>$c_1$</td>
<td>$c_1$</td>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S(x_i)$</th>
<th>1.28e-03</th>
<th>2.55e-07</th>
<th>6.99e-08</th>
<th>3.11e-08</th>
<th>3.109e-08</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_i$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S(x_i)$</th>
<th>1.53e-08</th>
<th>9.76e-09</th>
<th>2.08e-09</th>
<th>1.95e-09</th>
<th>7.83e-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_i$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
<td>$c_2$</td>
</tr>
</tbody>
</table>
AUC for naive Bayes is 0.775, whereas the AUC for the random classifier (ROC plot in grey) is 0.5.
ROC Plot and AUC: Trapezoid Region

![ROC Curve with Trapezoid Region]

- **False Positive Rate**
- **True Positive Rate**

At the point (0.333, 0.5), the area under the curve is 0.333. The total area under the curve is 0.5.
Consider a classifier \( M \), and some performance measure \( \theta \). Typically, the input dataset \( D \) 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 \( \theta \).

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

Also \( D \) 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 \( E[\theta] \) 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 $D \setminus D_i = \bigcup_{j \neq i} D_j$.

After training the model $M_i$ on $D \setminus D_i$, we assess its performance on the testing set $D_i$ to obtain the $i$-th estimate $\theta_i$.

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

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

and its variance as

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

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

**CROSS-VALIDATION**($K$, $D$):
1. $D \leftarrow$ randomly shuffle $D$
2. $\{D_1, D_2, \ldots, D_K\} \leftarrow$ partition $D$ in $K$ equal parts
3. **foreach** $i \in [1, K]$ **do**
   4. $M_i \leftarrow$ train classifier on $D \setminus D_i$
   5. $\theta_i \leftarrow$ assess $M_i$ on $D_i$
6. $\hat{\mu}_\theta = \frac{1}{K} \sum_{i=1}^{K} \theta_i$
7. $\hat{\sigma}^2_\theta = \frac{1}{K} \sum_{i=1}^{K} (\theta_i - \hat{\mu}_\theta)^2$
8. **return** $\hat{\mu}_\theta, \hat{\sigma}^2_\theta$
Bootstrap Resampling

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.

The probability that a particular point $x_j$ is not selected even after $n$ tries is given as

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

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

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

However, the estimated mean and variance of $\theta$ will be somewhat optimistic owing to the fairly large overlap between the training and testing datasets (63.2%).
**Bootstrap Resampling Algorithm**

**BOOTSTRAP-RESAMPLING**(\(K, D\)):

1. for \(i \in [1, K]\) do
2. \(D_i \leftarrow \text{sample of size } n \text{ with replacement from } D\)
3. \(M_i \leftarrow \text{train classifier on } D_i\)
4. \(\theta_i \leftarrow \text{assess } M_i \text{ on } D\)
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 \(\hat{\mu}_\theta, \hat{\sigma}_\theta^2\)
We apply bootstrap sampling to estimate the error rate for the full Bayes classifier, using $K = 50$ samples. The sampling distribution of error rates is:

![Bar chart showing the frequency of error rates.]

The expected value and variance of the error rate are

\[ \hat{\mu}_\theta = 0.213 \quad \hat{\sigma}_\theta^2 = 4.815 \times 10^{-4} \]
Confidence Intervals

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

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

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

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

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

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

The variance of \( \hat{\mu} \) is given as

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

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

\[ \text{std}(\hat{\mu}) = \sqrt{\text{var}(\hat{\mu})} = \frac{\sigma}{\sqrt{K}} \]

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

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

\( Z_K \) specifies the deviation of the estimated mean from the true mean in terms of its standard deviation.
Confidence Intervals

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

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

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

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

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

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

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

Note that

\[-z_{\alpha/2} \leq Z_K \leq z_{\alpha/2}\]

implies that

\[
\left(\hat{\mu} - z_{\alpha/2} \frac{\sigma}{\sqrt{K}}\right) \leq \mu \leq \left(\hat{\mu} + z_{\alpha/2} \frac{\sigma}{\sqrt{K}}\right)
\]

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

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

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

In general we do not know the true variance $\sigma^2$. However, we can replace $\sigma^2$ by the sample variance

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

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

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

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

and thus, we have

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

The interval $\left( \hat{\mu} - \frac{\alpha}{2} \frac{\hat{\sigma}}{\sqrt{K}}, \hat{\mu} + \frac{\alpha}{2} \frac{\hat{\sigma}}{\sqrt{K}} \right)$ is the $\alpha$% confidence interval for $\mu$. 
Confidence Intervals: Small Sample Size

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

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

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

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

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

The $\alpha\%$ confidence interval for the true mean $\mu$ is thus

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

Note the dependence of the interval on both $\alpha$ and the sample size $K$. 
Student's $t$ Distribution: $K$ Degrees of Freedom

$$f(x|0, 1), \quad t(10), \quad t(4), \quad t(1)$$

$$t(10), \quad t(4), \quad t(1)$$

$$t(10), \quad t(4), \quad t(1)$$
Iris 2D Data: Confidence Intervals

We apply 5-fold cross-validation \((K = 5)\) to assess the error rate of the full Bayes classifier on the Iris 2D data. The estimated expected value and variance for the error rate are as follows:

\[
\hat{\mu}_\theta = 0.233 \quad \hat{\sigma}^2_\theta = 0.00833 \quad \hat{\sigma}_\theta = \sqrt{0.00833} = 0.0913
\]

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, we have

\[
z_{\alpha/2} \frac{\hat{\sigma}_\theta}{\sqrt{K}} = \frac{1.96 \times 0.0913}{\sqrt{5}} = 0.08,
\]

and the confidence interval is

\[
P\left(\mu \in (0.233 - 0.08, 0.233 + 0.08)\right) = P\left(\mu \in (0.153, 0.313)\right) = 0.95
\]
Due to the small sample size \((K = 5)\), we can get a better confidence interval by using the \(t\) distribution. For \(K - 1 = 4\) degrees of freedom, for \(\alpha = 0.95\), we get \(t_{\alpha/2, K-1} = 2.776\). Thus,

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

The 95% confidence interval is therefore

\[
(0.233 - 0.113, 0.233 + 0.113) = (0.12, 0.346)
\]

which is much wider than the overly optimistic confidence interval \((0.153, 0.313)\) obtained for the large sample case.
Comparing Classifiers: Paired $t$-Test

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

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

Let $\theta^A_1, \theta^A_2, \ldots, \theta^A_K$ and $\theta^B_1, \theta^B_2, \ldots, \theta^B_K$ 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 $\delta_i$ as the difference in their performance on the $i$th dataset:

$$\delta_i = \theta^A_i - \theta^B_i$$

The expected difference and the variance estimates are given as:

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

$$\hat{\sigma}_\delta^2 = \frac{1}{K} \sum_{i=1}^{K} (\delta_i - \hat{\mu}_\delta)^2$$
Comparing Classifiers: Paired $t$-Test

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

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

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

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

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

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

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

Given a desired confidence level $\alpha$, we conclude that

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

Put another way, if $Z_\delta^* \not\in \left( -t_{\alpha/2, K-1}, t_{\alpha/2, K-1} \right)$, then we may reject the null hypothesis with $\alpha\%$ confidence.
**Paired t-Test via Cross-Validation**

**Paired t-Test** $(\alpha, K, D)$:

1. $D \leftarrow$ randomly shuffle $D$
2. $\{D_1, D_2, \ldots, D_K\} \leftarrow$ partition $D$ in $K$ equal parts
3. **foreach** $i \in [1, K]$ **do**
   4. $M^A_i, M^B_i \leftarrow$ train the two different classifiers on $D \setminus D_i$
   5. $\theta^A_i, \theta^B_i \leftarrow$ assess $M^A_i$ and $M^B_i$ on $D_i$
   6. $\delta_i = \theta^A_i - \theta^B_i$

7. $\hat{\mu}_\delta = \frac{1}{K} \sum_{i=1}^{K} \delta_i$
8. $\hat{\sigma}^2_\delta = \frac{1}{K} \sum_{i=1}^{K} (\delta_i - \hat{\mu}_\delta)^2$
9. $Z^*_\delta = \frac{\sqrt{K} \hat{\mu}_\delta}{\hat{\sigma}_\delta}$
10. **if** $Z^*_\delta \in \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$ **then**
    11. Accept $H_0$; both classifiers have similar performance
12. **else**
    13. Reject $H_0$; classifiers have significantly different performance
Bias-Variance Decomposition

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

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

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

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

$$L(y, M(x)) = (y - M(x))^2$$

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 \( x \), the goal of learning a classification model can be cast as minimizing the expected loss:

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

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

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

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

Thus, to minimize the expected loss we should choose \( c_i \) as the class that maximizes the posterior probability, that is, \( c_i = \arg \max_y P(y|x) \). By definition the error rate is simply an estimate of the expected zero–one loss; this choice thus minimizes the error rate.
Bias and Variance

The expected loss for the squared loss function offers important insight into the classification problem because it can be decomposed into bias and variance terms.

Intuitively, the bias of a classifier refers to the systematic deviation of its predicted decision boundary from the true decision boundary, whereas the variance of a classifier refers to the deviation among the learned decision boundaries over different training sets.

Because $M$ depends on the training set, given a test point $x$, we denote its predicted value as $M(x, D)$. Consider the expected square loss:

$$
E_y \left[ L(y, M(x, D)) \mid x, D \right] = E_y \left[ (y - M(x, D))^2 \mid x, D \right]
$$

$$
= E_y \left[ (y - E_y[y|x])^2 \mid x, D \right] + \left( M(x, D) - E_y[y|x] \right)^2
$$

The first term is simply the variance of $y$ given $x$. The second term is the squared error between the predicted value $M(x, D)$ and the expected value $E_y[y|x]$. 
Bias and Variance

The squared error depends on the training set. We can eliminate this dependence by averaging over all possible training tests of size $n$. The average or expected squared error for a given test point $x$ over all training sets is then given as

$$E_D\left[ (M(x, D) - E_y[y|x])^2 \right] = E_D\left[ (M(x, D) - E_D[M(x, D)])^2 \right] + \left( E_D[M(x, D)] - E_y[y|x] \right)^2$$

The expected squared loss over all test points $x$ and over all training sets $D$ of size $n$ yields the following decomposition:

$$E_{x,D,y}\left[ (y - M(x, D))^2 \right] = E_{x,y}\left[ (y - E_y[y|x])^2 \right] + E_{x,D}\left[ (M(x, D) - E_D[M(x, D)])^2 \right]$$

$$+ E_x\left[ (E_D[M(x, D)] - E_y[y|x])^2 \right]$$

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 $\text{var}(y|x)$ 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.

The classifier specific loss can then be attributed to the variance and bias terms. Bias indicates whether the model $M$ is correct or incorrect.

If the decision boundary is nonlinear, and we use a linear classifier, then it is likely to have high bias. A nonlinear (or a more complex) classifier is more likely to capture the correct decision boundary, and is thus likely to have a low bias.

The complex classifier is not necessarily better, since we also have to consider the variance term, which measures the inconsistency of the classifier decisions. A complex classifier induces a more complex decision boundary and thus may be prone to *overfitting*, and thus may be susceptible to small changes in training set, which may result in high variance.

In general, the expected loss can be attributed to high bias or high variance, with typically a trade-off between the two.
Iris PC Data: Iris-versicolor (class $c_1$; in circles) and other two Irises (class $c_2$; in triangles). $K = 10$ Bootstrap samples, trained via SVMs, varying the regularization constant $C$ from $10^{-2}$ to $10^2$. A small value of $C$ emphasizes the margin, whereas a large value of $C$ tries to minimize the slack terms. The decision boundaries over the 10 samples were as follows:

(a) $C = 0.01$

(b) $C = 1$
Bias-variance Decomposition: SVM Quadratic Kernels

Variance of the SVM model increases as we increase $C$, as seen from the varying decision boundaries. The figure on the right plots 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 $C = 1$. 

(c) $C = 100$

(d) Bias-Variance
A classifier is called *unstable* if small perturbations in the training set result in large changes in the prediction or decision boundary.

High variance classifiers are inherently unstable, since they tend to overfit the data. On the other hand, high bias methods typically underfit the data, and usually have low variance.

In either case, the aim of learning is to reduce classification error by reducing the variance or bias, ideally both.

Ensemble methods create a *combined classifier* using the output of multiple *base classifiers*, which are trained on different data subsets. Depending on how the training sets are selected, and on the stability of the base classifiers, ensemble classifiers can help reduce the variance and the bias, leading to a better overall performance.
**Bagging**

Bagging stands for *Bootstrap Aggregation*. It is an ensemble classification method that employs multiple bootstrap samples (with replacement) from the input training data $\mathbf{D}$ 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

$$v_j(\mathbf{x}) = \left| \{ M_i(\mathbf{x}) = c_j \ | i = 1, \ldots, K \} \right|$$

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

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

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.
Bagging: Combined SVM Classifiers

SVM classifiers are trained on $K = 10$ bootstrap samples using $C = 1$. The combined (average) classifier is shown in bold.

The worst training performance is obtained for $K = 3$ (in thick gray) and the best for $K = 8$ (in thick black).
In *Boosting* the main idea is to carefully select the samples to *boost* the performance on hard to classify instances.

Starting from an initial training sample $D_1$, we train the base classifier $M_1$, and obtain its training error rate.

To construct the next sample $D_2$, we select the misclassified instances with higher probability, and after training $M_2$, 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.

Finally, the combined classifier is obtained via weighted voting over the output of the $K$ base classifiers $M_1, M_2, \ldots, M_K$. 
Boosting is most beneficial when the base classifiers are weak, that is, have an error rate that is slightly less than that for a random classifier.

The idea is that whereas $M_1$ may not be particularly good on all test instances, by design $M_2$ may help classify some cases where $M_1$ fails, and $M_3$ may help classify instances where $M_1$ and $M_2$ fail, and so on. Thus, boosting has more of a bias reducing effect.

Each of the weak learners is likely to have high bias (it is only slightly better than random guessing), but the final combined classifier can have much lower bias, since different weak learners learn to classify instances in different regions of the input space.
Adaptive Boosting: AdaBoost

The boosting process is repeated $K$ times. Let $t$ denote the iteration and let $\alpha_t$ denote the weight for the $t$th classifier $M_t$.

Let $w_t^i$ denote the weight for $x_i$, with $w^t = (w_1^t, w_2^t, \ldots, w_n^t)^T$ being the weight vector over all the points for the $t$th iteration.

$w$ is a probability vector, whose elements sum to one. Initially all points have equal weights, that is,

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

During iteration $t$, the training sample $D_t$ is obtained via weighted resampling using the distribution $w^{t-1}$, that is, we draw a sample of size $n$ with replacement, such that the $i$th point is chosen according to its probability $w_i^{t-1}$.

Using $D_t$ we train the classifier $M_t$, and compute its weighted error rate $\epsilon_t$ on the entire input dataset $D$:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{t-1} \cdot I(M_t(x_i) \neq y_i)$$
Adaptive Boosting: AdaBoost

The weight for the $t$th classifier is then set as

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

The weight for each point $x_i \in D$ is updated as

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

If the predicted class matches the true class, that is, if $M_t(x_i) = y_i$, then the weight for point $x_i$ remains unchanged.

If the point is misclassified, that is, $M_t(x_i) \neq y_i$, then

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

Thus, if the error rate $\epsilon_t$ is small, then there is a greater weight increment for $x_i$. The intuition is that a point that is misclassified by a good classifier (with a low error rate) should be more likely to be selected for the next training dataset.
Adaptive Boosting: AdaBoost

For boosting we require that a base classifier has an error rate at least slightly better than random guessing, that is, $\epsilon_t < 0.5$. If the error rate $\epsilon_t \geq 0.5$, then the boosting method discards the classifier, and tries another data sample.

**Combined Classifier:** Given the set of boosted classifiers, $M_1, M_2, \ldots, M_K$, along with their weights $\alpha_1, \alpha_2, \ldots, \alpha_K$, the class for a test case $x$ is obtained via weighted majority voting.

Let $v_j(x)$ denote the weighted vote for class $c_j$ over the $K$ classifiers, given as

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

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

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

\texttt{AdaBoost}(K, D):
1 \hspace{1em} \texttt{w}^0 \leftarrow \left(\frac{1}{n}\right) \cdot 1 \in \mathbb{R}^n
2 \hspace{1em} t \leftarrow 1
3 \textbf{while } t \leq K \textbf{ do}
4 \hspace{1em} D_t \leftarrow \text{weighted resampling with replacement from } D \text{ using } \texttt{w}^{t-1}
5 \hspace{1em} M_t \leftarrow \text{train classifier on } D_t
6 \hspace{1em} \epsilon_t \leftarrow \sum_{i=1}^{n} w_i^{t-1} \cdot I(M_t(x_i) \neq y_i) \qquad \text{// weighted error rate on } D
7 \hspace{1em} \textbf{if } \epsilon_t = 0 \textbf{ then break}
8 \hspace{1em} \textbf{else if } \epsilon_t < 0.5 \textbf{ then}
9 \hspace{2em} \alpha_t = \ln \left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \qquad \text{// classifier weight}
10 \hspace{2em} \textbf{foreach } i \in [1, n] \textbf{ do}
11 \hspace{3em} \text{// update point weights}
12 \hspace{4em} w_i^t = \begin{cases} w_i^{t-1} & \text{if } M_t(x_i) = y_i \\ w_i^{t-1} \left(\frac{1 - \epsilon_t}{\epsilon_t}\right) & \text{if } M_t(x_i) \neq y_i \end{cases}
13 \hspace{2em} \texttt{w}^t = \frac{w^t}{1/\texttt{w}^t} \quad \text{// normalize weights}
14 \hspace{1em} t \leftarrow t + 1
15 \textbf{return } \{M_1, M_2, \ldots, M_K\}
Iris PC Data: Hyperplane learnt in $t$th iteration is $h_t$. We can observe that the first three hyperplanes $h_1$, $h_2$ and $h_3$ already capture the essential features of the nonlinear decision boundary. Further reduction in the training error is obtained by increasing the number of boosting steps $K$. 

Boosting SVMs: Linear Kernel ($C = 1$)