CHAPTER

# Credibility: Evaluating What's Been Learned

5

Evaluation is the key to making real progress in data mining. There are lots of ways of inferring structure from data: We have encountered many already and will see further refinements, and new methods, in Chapter 6. However, in order to determine which ones to use on a particular problem we need systematic ways to evaluate how different methods work and to compare one with another. But evaluation is not as simple as it might appear at first sight.

What's the problem? We have the training set; surely we can just look at how well different methods do on that. Well, no: As we will see very shortly, performance on the training set is definitely not a good indicator of performance on an independent test set. We need ways of predicting performance bounds in practice, based on experiments with whatever data can be obtained.

When a vast supply of data is available, this is no problem: Just make a model based on a large training set, and try it out on another large test set. But although data mining sometimes involves "big data"—particularly in marketing, sales, and customer support applications—it is often the case that data, quality data, is scarce. The oil slicks mentioned in Chapter 1 (page 23) had to be detected and marked manually—a skilled and labor-intensive process—before being used as training data. Even in the personal loan application data (page 22), there turned out to be only 1000 training examples of the appropriate type. The electricity supply data (page 24) went back 15 years, 5000 days—but only 15 Christmas days and Thanksgivings, and just four February 29s and presidential elections. The electromechanical diagnosis application (page 25) was able to capitalize on 20 years of recorded experience, but this yielded only 300 usable examples of faults. The marketing and sales applications (page 26) certainly involve big data, but many others do not: Training data frequently relies on specialist human expertise—and that is always in short supply.

The question of predicting performance based on limited data is an interesting, and still controversial, one. We will encounter many different techniques, of which one—repeated cross-validation—is probably the method of choice in most practical limited-data situations. Comparing the performance of different machine learning schemes on a given problem is another matter that is not as easy as it sounds: To be sure that apparent differences are not caused by chance effects, statistical tests are needed.

So far we have tacitly assumed that what is being predicted is the ability to classify test instances accurately; however, some situations involve predicting class probabilities rather than the classes themselves, and others involve predicting numeric rather than nominal values. Different methods are needed in each case. Then we look at the question of cost. In most practical data mining situations, the cost of a misclassification error depends on the type of error it is—whether, for example, a positive example was erroneously classified as negative or vice versa. When doing data mining, and evaluating its performance, it is often essential to take these costs into account. Fortunately, there are simple techniques to make most learning schemes cost sensitive without grappling with the algorithm's internals. Finally, the whole notion of evaluation has fascinating philosophical connections. For 2000 years, philosophers have debated the question of how to evaluate scientific theories, and the issues are brought into sharp focus by data mining because what is extracted is essentially a "theory" of the data.

## 5.1 TRAINING AND TESTING

For classification problems, it is natural to measure a classifier's performance in terms of the *error rate*. The classifier predicts the class of each instance: If it is correct, that is counted as a *success*; if not, it is an *error*. The error rate is just the proportion of errors made over a whole set of instances, and it measures the overall performance of the classifier.

Of course, what we are interested in is the likely future performance on new data, not the past performance on old data. We already know the classifications of each instance in the training set, which after all is why we can use it for training. We are not generally interested in learning about those classifications—although we might be if our purpose is data cleansing rather than prediction. So the question is, is the error rate on old data likely to be a good indicator of the error rate on new data? The answer is a resounding *no*—not if the old data was used during the learning process to train the classifier.

This is a surprising fact, and a very important one. The error rate on the training set is *not* likely to be a good indicator of future performance. Why? Because the classifier has been learned from the very same training data, any estimate of performance based on that data will be optimistic, even hopelessly optimistic.

We have already seen an example of this in the labor relations dataset. Figure 1.3(b) (page 18) was generated directly from the training data, and Figure 1.3(a) was obtained from it by a process of pruning. The former is potentially more accurate on the data that was used to train the classifier, but may perform less well on independent test data because it is overfitted to the training data. The first tree will look good according to the error rate on the training data, better than the second tree. But this does not necessarily reflect how they will perform on independent test data.

The error rate on the training data is called the *resubstitution error* because it is calculated by resubstituting the training instances into a classifier that was constructed from them. Although it is not a reliable predictor of the true error rate on new data, it is nevertheless often useful to know.

To predict the performance of a classifier on new data, we need to assess its error rate on a dataset that played no part in the formation of the classifier. This independent dataset is called the *test set*. We assume that both the training data and the test data are representative samples of the underlying problem.

In some cases the test data might be distinct in nature from the training data. Consider, for example, the credit risk problem from Section 1.3 (page 22). Suppose the bank had training data from branches in New York and Florida and wanted to know how well a classifier trained on one of these datasets would perform in a new branch in Nebraska. It should probably use the Florida data as test data for evaluating the New York–trained classifier and the New York data to evaluate the Florida-trained classifier. If the datasets were amalgamated before training, performance on the test data would probably not be a good indicator of performance on future data in a completely different state.

It is important that the test data is *not used in any way* to create the classifier. For example, some learning schemes involve two stages, one to come up with a basic structure and the second to optimize parameters involved in that structure, and separate sets of data may be needed in the two stages. Or you might try out several learning schemes on the training data and then evaluate them—on a fresh dataset, of course—to see which one works best. But none of this data may be used to determine an estimate of the future error rate.

In such situations people often talk about three datasets: the *training* data, the *validation* data, and the *test* data. The training data is used by one or more learning schemes to come up with classifiers. The validation data is used to optimize parameters of those classifier, or to select a particular one. Then the test data is used to calculate the error rate of the final, optimized, method. Each of the three sets must be chosen independently: The validation set must be different from the training set to obtain good performance in the optimization or selection stage, and the test set must be different from both to obtain a reliable estimate of the true error rate.

It may be that once the error rate has been determined, the test data is bundled back into the training data to produce a new classifier for actual use. There is nothing wrong with this: It is just a way of maximizing the amount of data used to generate the classifier that will actually be employed in practice. With well-behaved learning schemes, this should not decrease predictive performance. Also, once the validation data has been used—maybe to determine the best type of learning scheme to use—then it can be bundled back into the training data to retrain that learning scheme, maximizing the use of data.

If lots of data is available, there is no problem: We take a large sample and use it for training; then another, independent large sample of different data and use it for testing. Provided both samples are representative, the error rate on the test set will give a good indication of future performance. Generally, the larger the training sample, the better the classifier, although the returns begin to diminish once a certain volume of training data is exceeded. And the larger the test sample, the more accurate the error estimate. The accuracy of the error estimate can be quantified statistically, as we will see in Section 5.2.

The real problem occurs when there is not a vast supply of data available. In many situations the training data must be classified manually—and so must the test data, of course, to obtain error estimates. This limits the amount of data that can be used for training, validation, and testing, and the problem becomes how to make the most of a limited dataset. From this dataset, a certain amount is held over for testing—this is called the *holdout* procedure—and the remainder used for training (and, if necessary, part of that is set aside for validation). There's a dilemma here: To find a good classifier, we want to use as much of the data as possible for training; to obtain a good error estimate, we want to use as much of it as possible for testing. Sections 5.3 and 5.4 review widely used methods for dealing with this dilemma.

## 5.2 PREDICTING PERFORMANCE

Suppose we measure the error of a classifier on a test set and obtain a certain numerical error rate—say 25%. Actually, in this section we talk about success rate rather than error rate, so this corresponds to a success rate of 75%. Now, this is only an estimate. What can you say about the *true* success rate on the target population? Sure, it's expected to be close to 75%. But how close—within 5 or 10%? It must depend on the size of the test set. Naturally, we would be more confident of the 75% figure if it were based on a test set of 10,000 instances rather than a test set of 100 instances. But how much more confident would we be?

To answer these questions, we need some statistical reasoning. In statistics, a succession of independent events that either succeed or fail is called a *Bernoulli process*. The classic example is coin tossing. Each toss is an independent event. Let's say we always predict heads; but rather than "heads" or "tails," each toss is considered a "success" or a "failure." Let's say the coin is biased, but we don't know what the probability of heads is. Then, if we actually toss the coin 100 times and 75 of the tosses are heads, we have a situation very like the one just described for a classifier with an observed 75% success rate on a test set. What can we say about the true success probability? In other words, imagine that there is a Bernoulli process—a biased coin—with a true (but unknown) success rate of $p$. Suppose that out of $N$ trials, $S$ are successes; thus, the observed success rate is $f = S/N$. The question is, what does this tell you about the true success rate $p$?

The answer to this question is usually expressed as a confidence interval—that is, $p$ lies within a certain specified interval with a certain specified confidence. For example, if $S = 750$ successes are observed out of $N = 1000$ trials, this indicates that the true success rate must be around 75%. But how close to 75%? It turns out that with 80% confidence, the true success rate $p$ lies between 73.2% and 76.7%. If $S = 75$ successes are observed out of $N = 100$ trials, this also indicates that the true success rate must be around 75%. But the experiment is smaller, and so the 80% confidence interval for $p$ is wider, stretching from 69.1 to 80.1%.

These figures are easy to relate to qualitatively, but how are they derived quantitatively? We reason as follows: The mean and variance of a single Bernoulli trial with success rate $p$ are $p$ and $p(1 - p)$, respectively. If $N$ trials are taken from a Bernoulli process, the expected success rate $f = S/N$ is a random variable with the same mean $p$; the variance is reduced by a factor of $N$ to $p(1 - p)/N$. For large $N$, the distribution of this random variable approaches the normal distribution. These are all facts of statistics—we will not go into how they are derived.

The probability that a random variable $X$, with zero mean, lies within a certain confidence range of width $2z$ is

$$\Pr[-z \leq X \leq z] = c$$

For a normal distribution, values of $c$ and corresponding values of $z$ are given in tables printed at the back of most statistical texts. However, the tabulations conventionally take a slightly different form: They give the confidence that $X$ will lie outside the range, and they give it for the upper part of the range only:

$$\Pr[X \geq z]$$

This is called a *one-tailed* probability because it refers only to the upper "tail" of the distribution. Normal distributions are symmetric, so the probabilities for the lower tail

$$\Pr[X \leq -z]$$

are just the same.

Table 5.1 gives an example. Like other tables for the normal distribution, this assumes that the random variable $X$ has a mean of 0 and a variance of 1. Alternatively, you might say that the $z$ figures are measured in *standard deviations from the mean*. Thus, the figure for $\Pr[X \geq z] = 5\%$ implies that there is a 5% chance that $X$ lies more than 1.65 standard deviations above the mean. Because the distribution is symmetric, the chance that $X$ lies more than 1.65 standard deviations from the mean (above or below) is 10%, or

$$\Pr[-1.65 \leq X \leq 1.65] = 90\%$$

Now all we need to do is reduce the random variable $f$ to have zero mean and unit variance. We do this by subtracting the mean $p$ and dividing by the standard deviation $\sqrt{p(1-p)/N}$. This leads to

$$\Pr\left[-z < \frac{f-p}{\sqrt{p(1-p)/N}} < z\right] = c$$

Here is the procedure for finding confidence limits. Given a particular confidence figure $c$, consult Table 5.1 for the corresponding $z$ value. To use the table you will first have to subtract $c$ from 1 and then halve the result, so that for $c = 90\%$ you use the table entry for 5%. Linear interpolation can be used for intermediate confidence levels. Then write the inequality in the preceding expression as an equality and invert it to find an expression for $p$.

The final step involves solving a quadratic equation. Although this is not hard to do, it leads to an unpleasantly formidable expression for the confidence limits:

$$p = \left(f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}\right) \Bigg/ \left(1 + \frac{z^2}{N}\right)$$

The ± in this expression gives two values for $p$ that represent the upper and lower confidence boundaries. Although the formula looks complicated, it is not hard to work out in particular cases.

This result can be used to obtain the values in the numeric example given earlier. Setting $f = 75\%$, $N = 1000$, and $c = 80\%$ (so that $z = 1.28$) leads to the interval [0.732, 0.767] for $p$, and $N = 100$ leads to [0.691, 0.801] for the same level of confidence. Note that the normal distribution assumption is only valid for large $N$ (say, $N > 100$). Thus, $f = 75\%$ and $N = 10$ leads to confidence limits [0.549, 0.881], but these should be taken with a grain of salt.

**Table 5.1** Confidence Limits for the Normal Distribution

| **Pr[$X \geq z$]** | $z$ |
|---|---|
| 0.1% | 3.09 |
| 0.5% | 2.58 |
| 1% | 2.33 |
| 5% | 1.65 |
| 10% | 1.28 |
| 20% | 0.84 |
| 40% | 0.25 |

## 5.3 CROSS-VALIDATION

Now consider what to do when the amount of data for training and testing is limited. The holdout method reserves a certain amount for testing and uses the remainder for training (and sets part of that aside for validation, if required). In practical terms, it is common to hold out one-third of the data for testing and use the remaining two-thirds for training.

Of course, you may be unlucky: The sample used for training (or testing) might not be representative. In general, you cannot tell whether a sample is representative or not. But there is one simple check that might be worthwhile: Each class in the full dataset should be represented in about the right proportion in the training and testing sets. If, by bad luck, all examples with a certain class were omitted from the training set, you could hardly expect a classifier learned from that data to perform well on examples of that class—and the situation would be exacerbated by the fact that the class would necessarily be overrepresented in the test set because none of its instances made it into the training set! Instead, you should ensure that the random sampling is done in a way that guarantees that each class is properly represented in both training and test sets. This procedure is called *stratification*, and we might speak of *stratified holdout*. While it is generally well worth doing, stratification provides only a primitive safeguard against uneven representation in training and test sets.

A more general way to mitigate any bias caused by the particular sample chosen for holdout is to repeat the whole process, training and testing, several times with different random samples. In each iteration a certain proportion, say two-thirds, of the data is randomly selected for training, possibly with stratification, and the remainder is used for testing. The error rates on the different iterations are averaged to yield an overall error rate. This is the *repeated holdout* method of error rate estimation.

In a single holdout procedure, you might consider swapping the roles of the testing and training data—that is, train the system on the test data and test it on the training data—and average the two results, thus reducing the effect of uneven representation in training and test sets. Unfortunately, this is only really plausible with a 50:50 split between training and test data, which is generally not ideal—it is better to use more than half the data for training even at the expense of test data. However, a simple variant forms the basis of an important statistical technique called *cross-validation*. In cross-validation, you decide on a fixed number of *folds*, or partitions, of the data. Suppose we use three. Then the data is split into three approximately equal partitions; each in turn is used for testing and the remainder is used for training. That is, use two-thirds of the data for training and one-third for testing, and repeat the procedure three times so that in the end, every instance has been used exactly once for testing. This is called *threefold cross-validation*, and if stratification is adopted as well—which it often is—it is *stratified threefold cross-validation*.

The standard way of predicting the error rate of a learning technique given a single, fixed sample of data is to use stratified tenfold cross-validation. The data is divided randomly into 10 parts in which the class is represented in approximately the same proportions as in the full dataset. Each part is held out in turn and the learning scheme trained on the remaining nine-tenths; then its error rate is calculated on the holdout set. Thus, the learning procedure is executed a total of 10 times on different training sets (each set has a lot in common with the others). Finally, the 10 error estimates are averaged to yield an overall error estimate.

Why 10? Extensive tests on numerous different datasets, with different learning techniques, have shown that 10 is about the right number of folds to get the best estimate of error, and there is also some theoretical evidence that backs this up. Although these arguments are by no means conclusive, and debate continues to rage in machine learning and data mining circles about what is the best scheme for evaluation, tenfold cross-validation has become the standard method in practical terms. Tests have also shown that the use of stratification improves results slightly. Thus, the standard evaluation technique in situations where only limited data is available is stratified tenfold cross-validation. Note that neither the stratification nor the division into 10 folds has to be exact: It is enough to divide the data into 10 approximately equal sets in which the various class values are represented in approximately the right proportion. Moreover, there is nothing magic about the exact number 10: 5-fold or 20-fold cross-validation is likely to be almost as good.

A single tenfold cross-validation might not be enough to get a reliable error estimate. Different tenfold cross-validation experiments with the same learning scheme and dataset often produce different results because of the effect of random variation in choosing the folds themselves. Stratification reduces the variation, but it certainly does not eliminate it entirely. When seeking an accurate error estimate, it is standard procedure to repeat the cross-validation process 10 times—that is, 10 times tenfold cross-validation—and average the results. This involves invoking the learning algorithm 100 times on datasets that are all nine-tenths the size of the original. Getting a good measure of performance is a computation-intensive undertaking.

## 5.4 OTHER ESTIMATES

Tenfold cross-validation is the standard way of measuring the error rate of a learning scheme on a particular dataset; for reliable results, 10 times tenfold cross-validation. But many other methods are used instead. Two that are particularly prevalent are *leave-one-out* cross-validation and the *bootstrap*.

### Leave-One-Out Cross-Validation

Leave-one-out cross-validation is simply $n$-fold cross-validation, where $n$ is the number of instances in the dataset. Each instance in turn is left out, and the learning scheme is trained on all the remaining instances. It is judged by its correctness on the remaining instance—one or zero for success or failure, respectively. The results of all $n$ judgments, one for each member of the dataset, are averaged, and that average represents the final error estimate.

This procedure is an attractive one for two reasons. First, the greatest possible amount of data is used for training in each case, which presumably increases the chance that the classifier is an accurate one. Second, the procedure is deterministic: No random sampling is involved. There is no point in repeating it 10 times, or repeating it at all: The same result will be obtained each time. Set against this is the high computational cost, because the entire learning procedure must be executed $n$ times and this is usually infeasible for large datasets. Nevertheless, leave-one-out seems to offer a chance of squeezing the maximum out of a small dataset and getting as accurate an estimate as possible.

But there is a disadvantage to leave-one-out cross-validation, apart from the computational expense. By its very nature, it cannot be stratified—worse than that, it *guarantees* a nonstratified sample. Stratification involves getting the correct proportion of examples in each class into the test set, and this is impossible when the test set contains only a single example. A dramatic, although highly artificial, illustration of the problems this might cause is to imagine a completely random dataset that contains exactly the same number of instances of each of two classes. The best that an inducer can do with random data is to predict the majority class, giving a true error rate of 50%. But in each fold of leave-one-out, the opposite class to the test instance is in the majority—and therefore the predictions will always be incorrect, leading to an estimated error rate of 100%!

### The Bootstrap

The second estimation method we describe, the bootstrap, is based on the statistical procedure of sampling *with replacement*. Previously, whenever a sample was taken from the dataset to form a training or test set, it was drawn without replacement. That is, the same instance, once selected, could not be selected again. It is like picking teams for football: You cannot choose the same person twice. But dataset instances are not like people. Most learning schemes *can* use the same instance twice, and it makes a difference in the result of learning if it is present in the training set twice. (Mathematical sticklers will notice that we should not really be talking about "sets" at all if the same object can appear more than once.)

The idea of the bootstrap is to sample the dataset with replacement to form a training set. We will describe a particular variant, mysteriously (but for a reason that will soon become apparent) called the *0.632 bootstrap*. For this, a dataset of $n$ instances is sampled $n$ times, with replacement, to give another dataset of $n$ instances. Because some elements in this second dataset will (almost certainly) be repeated, there must be some instances in the original dataset that have not been picked—we will use these as test instances.

What is the chance that a particular instance will not be picked for the training set? It has a $1/n$ probability of being picked each time and so a $1 - 1/n$ probability of *not* being picked. Multiply these probabilities together for a sufficient number of picking opportunities, $n$, and the result is a figure of

$$\left(1-\frac{1}{n}\right)^n \approx e^{-1} = 0.368$$

where $e$ is the base of natural logarithms, 2.7183 (not the error rate!) This gives the chance of a particular instance not being picked at all. Thus, for a reasonably large dataset, the test set will contain about 36.8% of the instances and the training set will contain about 63.2% of them (now you can see why it's called the *0.632 bootstrap*). Some instances will be repeated in the training set, bringing it up to a total size of $n$, the same as in the original dataset.

The figure obtained by training a learning system on the training set and calculating its error over the test set will be a pessimistic estimate of the true error rate because the training set, although its size is $n$, nevertheless contains only 63% of the instances, which is not a great deal compared, for example, with the 90% used in tenfold cross-validation. To compensate for this, we combine the test-set error rate with the resubstitution error on the instances in the training set. The resubstitution figure, as we warned earlier, gives a very optimistic estimate of the true error and should certainly not be used as an error figure on its own. But the bootstrap procedure combines it with the test error rate to give a final estimate $e$ as follows:

$$e = 0.632 \times e_{\text{test instances}} + 0.368 \times e_{\text{training instances}}$$

Then, the whole bootstrap procedure is repeated several times, with different replacement samples for the training set, and the results are averaged.

The bootstrap procedure may be the best way of estimating the error rate for very small datasets. However, like leave-one-out cross-validation, it has disadvantages that can be illustrated by considering a special, artificial situation. In fact, the very dataset we considered above will do: a completely random dataset with two classes of equal size. The true error rate is 50% for any prediction rule. But a scheme that memorized the training set would give a perfect resubstitution score of 100%, so that $e_{\text{training instances}} = 0$, and the 0.632 bootstrap will mix this in with a weight of 0.368 to give an overall error rate of only 31.6% ($0.632 \times 50\% + 0.368 \times 0\%$), which is misleadingly optimistic.

## 5.5 COMPARING DATA MINING SCHEMES

We often need to compare two different learning schemes on the same problem to see which is the better one to use. It seems simple: Estimate the error using cross-validation (or any other suitable estimation procedure), perhaps repeated several times, and choose the scheme with the smaller estimate. This is quite sufficient in many practical applications: If one scheme has a lower estimated error than another on a particular dataset, the best we can do is to use the former scheme's model. However, it may be that the difference is simply due to estimation error, and in some circumstances it is important to determine whether one scheme is *really* better than another on a particular problem. This is a standard challenge for machine learning researchers. If a new learning algorithm is proposed, its proponents must show that it improves on the state of the art for the problem at hand and demonstrate that the observed improvement is not just a chance effect in the estimation process.

This is a job for a statistical test based on confidence bounds, the kind we met previously when trying to predict true performance from a given test-set error rate. If there were unlimited data, we could use a large amount for training and evaluate performance on a large independent test set, obtaining confidence bounds just as before. However, if the difference turns out to be significant we must ensure that this is not just because of the particular dataset we happened to base the experiment on. What we want to determine is whether one scheme is better or worse than another on average, across all possible training and test datasets that can be drawn from the domain. Because the amount of training data naturally affects performance, all datasets should be the same size. Indeed, the experiment might be repeated with different sizes to obtain a learning curve.

For the moment, assume that the supply of data is unlimited. For definiteness, suppose that cross-validation is being used to obtain the error estimates (other estimators, such as repeated cross-validation, are equally viable). For each learning scheme we can draw several datasets of the same size, obtain an accuracy estimate for each dataset using cross-validation, and compute the mean of the estimates. Each cross-validation experiment yields a different, independent error estimate. What we are interested in is the mean accuracy across all possible datasets of the same size, and whether this mean is greater for one scheme or the other.

From this point of view, we are trying to determine whether the mean of a set of samples—cross-validation estimates for the various datasets that we sampled from the domain—is significantly greater than, or significantly less than, the mean of another. This is a job for a statistical device known as the *t-test*, or *Student's t-test*. Because the same cross-validation experiment can be used for both learning schemes to obtain a matched pair of results for each dataset, a more sensitive version of the *t*-test known as a *paired t-test* can be used.

We need some notation. There is a set of samples $x_1$, $x_2$, …, $x_k$ obtained by successive tenfold cross-validations using one learning scheme, and a second set of samples $y_1$, $y_2$, …, $y_k$ obtained by successive tenfold cross-validations using the other. Each cross-validation estimate is generated using a different dataset, but all datasets are of the same size and from the same domain. We will get best results if exactly the same cross-validation partitions are used for both schemes, so that $x_1$ and $y_1$ are obtained using the same cross-validation split, as are $x_2$ and $y_2$, and so on. Denote the mean of the first set of samples by $\bar{x}$ and the mean of the second set by $\bar{y}$. We are trying to determine whether $\bar{x}$ is significantly different from $\bar{y}$.

If there are enough samples, the mean ($\bar{x}$) of a set of independent samples ($x_1$, $x_2$, …, $x_k$) has a normal (i.e., Gaussian) distribution, regardless of the distribution underlying the samples themselves. Call the true value of the mean $\mu$. If we knew the variance of that normal distribution, so that it could be reduced to have zero mean and unit variance, we could obtain confidence limits on $\mu$ given the mean of the samples ($\bar{x}$). However, the variance is unknown, and the only way we can obtain it is to estimate it from the set of samples.

That is not hard to do. The variance of $\bar{x}$ can be estimated by dividing the variance calculated from the samples $x_1$, $x_2$, …, $x_k$—call it $\sigma_x^2$—by $k$. We can reduce the distribution of $\bar{x}$ to have zero mean and unit variance by using

$$\frac{\bar{x} -}{\sqrt{\sigma_x^2/k}}$$

The fact that we have to *estimate* the variance changes things somewhat. Because the variance is only an estimate, this does *not* have a normal distribution (although it does become normal for large values of $k$). Instead, it has what is called a *Student's distribution with k – 1 degrees of freedom*. What this means in practice is that we have to use a table of confidence intervals for the Student's distribution rather than the confidence table for the normal distribution given earlier. For 9 degrees of freedom (which is the correct number if we are using the average of 10 cross-validations) the appropriate confidence limits are shown in Table 5.2. If you compare them with Table 5.1 you will see that the Student's figures are slightly more conservative—for a given degree of confidence, the interval is slightly wider—and this reflects the additional uncertainty caused by having to estimate the variance. Different tables are needed for different numbers of degrees of freedom, and if there are more than 100 degrees of freedom the confidence limits are very close to those for the normal distribution. Like Table 5.1, the figures in Table 5.2 are for a "one-sided" confidence interval.

To decide whether the means $\bar{x}$ and $\bar{y}$, each an average of the same number $k$ of samples, are the same or not, we consider the differences $d_i$ between corresponding observations, $d_i = x_i - y_i$. This is legitimate because the observations are paired. The mean of this difference is just the difference between the two means, $\bar{d} = \bar{x} - \bar{y}$, and, like the means themselves, it has a Student's distribution with $k - 1$ degrees of freedom. If the means are the same, the difference is zero (this is called the *null hypothesis*); if they're significantly different, the difference will be significantly different from zero. So for a given confidence level, we will check whether the actual difference exceeds the confidence limit.

First, reduce the difference to a zero-mean, unit-variance variable called the *t-statistic*,

$$t = \frac{\bar{d}}{\sqrt{\sigma_d{}^2/k}}$$

where $\sigma_d^2$ is the variance of the difference samples. Then, decide on a confidence level—generally, 5% or 1% is used in practice. From this, the confidence limit z is determined using Table 5.2 if $k$ is 10; if it is not, a confidence table of the Student distribution for the $k$ value in question is used. A two-tailed test is appropriate because we do not know in advance whether the mean of the $x$'s is likely to be greater than that of the $y$'s or vice versa; thus, for a 1% test we use the value corresponding to 0.5% in Table 5.2. If the value of $t$ according to the last formula is greater than $z$, or less than $-z$, we reject the null hypothesis that the means are the same and conclude that there really is a significant difference between the two learning methods on that domain for that dataset size.

Two observations are worth making on this procedure. The first is technical: What if the observations were not paired? That is, what if we were unable, for some reason, to assess the error of each learning scheme on the same datasets? What if the number of datasets for each scheme was not even the same? These conditions could arise if someone else had evaluated one of the schemes and published several different estimates for a particular domain and dataset size—or perhaps just their mean and variance—and we wished to compare this with a different learning scheme. Then it is necessary to use a regular, nonpaired $t$-test. Instead of taking the mean of the difference, $\bar{d}$, we use the difference of the means, $\bar{x} - \bar{y}$. Of course, that's the same thing: The mean of the difference *is* the difference of the means. But the variance of the difference $\bar{d}$ is *not* the same. If the variance of the samples $x_1, x_2, \ldots, x_k$ is $\sigma_x^2$ and the variance of the samples $y_1, y_2, \ldots, y_\ell$ is $\sigma_y^2$,

$$\frac{\sigma_x{}^2}{k} + \frac{\sigma_y{}^2}{\ell}$$

is a good estimate of the variance of the difference of the means. It is this variance (or rather its square root) that should be used as the denominator of the $t$-statistic given previously. The degrees of freedom, necessary for consulting Student's confidence tables, should be taken conservatively to be the minimum of the degrees of freedom of the two samples. Essentially, knowing that the observations are paired allows the use of a better estimate for the variance, which will produce tighter confidence bounds.

The second observation concerns the assumption that there is essentially unlimited data, so that several independent datasets of the right size can be used. In practice, there is usually only a single dataset of limited size. What can be done? We could split the data into subsets (perhaps 10) and perform a cross-validation on each one. However, the overall result will only tell us whether a learning scheme is preferable for that particular size—one-tenth of the original dataset. Alternatively, the original dataset could be reused—for example, with different randomizations of the dataset for each cross-validation. However, the resulting cross-validation estimates will not be independent because they are not based on independent datasets. In practice, this means that a difference may be judged to be significant when in fact it is not. Indeed, just increasing the number of samples *k*—that is, the number of cross-validation runs—will eventually yield an apparently significant difference because the value of the *t*-statistic increases without bound.

Various modifications of the standard *t*-test have been proposed to circumvent this problem, all of them heuristic and somewhat lacking in theoretical justification. One that appears to work well in practice is the *corrected resampled t-test*. Assume for the moment that the repeated holdout method is used instead of cross-validation, repeated *k* times on different random splits of the same dataset to obtain accuracy estimates for two learning schemes. Each time, $n_1$ instances are used for training and $n_2$ for testing, and differences $d_i$ are computed from performance on the test data. The corrected resampled *t*-test uses the modified statistic

$$t = \frac{\bar{d}}{\sqrt{\left(\frac{1}{k} + \frac{n_2}{n_1}\right)\sigma_d^2}}$$

in exactly the same way as the standard *t*-statistic. A closer look at the formula shows that its value cannot be increased simply by increasing *k*. The same modified statistic can be used with repeated cross-validation, which is just a special case of repeated holdout in which the individual test sets for *one* cross-validation do not overlap. For tenfold cross-validation repeated 10 times, $k = 100$, $n_2/n_1 = 0.1/0.9$, and $\sigma_d^2$ is based on 100 differences.

**Table 5.2** Confidence Limits for Student's Distribution with 9 Degrees of Freedom

| **Pr[*X* ≥ *z*]** | ***z*** |
|---|---|
| 0.1% | 4.30 |
| 0.5% | 3.25 |
| 1% | 2.82 |
| 5% | 1.83 |
| 10% | 1.38 |
| 20% | 0.88 |

## 5.6 PREDICTING PROBABILITIES

Throughout this chapter we have tacitly assumed that the goal is to maximize the success rate of the predictions. The outcome for each test instance is either *correct*, if the prediction agrees with the actual value for that instance, or *incorrect*, if it does not. There are no grays: Everything is black or white, correct or incorrect. In many situations, this is the most appropriate perspective. If the learning scheme, when it is actually applied, results in either a correct or an incorrect prediction, success is the right measure to use. This is sometimes called a *0 – 1 loss function*: The "loss" is either 0 if the prediction is correct or 1 if it is not. The use of *loss* is conventional, although a more optimistic terminology might couch the outcome in terms of profit instead.

Other situations are softer-edged. Most learning schemes can associate a probability with each prediction (as the Naïve Bayes scheme does). It might be more natural to take this probability into account when judging correctness. For example, a correct outcome predicted with a probability of 99% should perhaps weigh more heavily than one predicted with a probability of 51%, and, in a two-class situation, perhaps the latter is not all that much better than an *incorrect* outcome predicted with probability 51%. Whether it is appropriate to take prediction probabilities into account depends on the application. If the ultimate application really is just a prediction of the outcome, and no prizes are awarded for a realistic assessment of the likelihood of the prediction, it does not seem appropriate to use probabilities. If the prediction is subject to further processing, however—perhaps involving assessment by a person, or a cost analysis, or maybe even serving as input to a second-level learning process—then it may well be appropriate to take prediction probabilities into account.

## Quadratic Loss Function

Suppose for a single instance there are $k$ possible outcomes, or classes, and for a given instance the learning scheme comes up with a probability vector $p_1$, $p_2$, …, $p_k$ for the classes (where these probabilities sum to 1). The actual outcome for that instance will be one of the possible classes. However, it is convenient to express it as a vector $a_1$, $a_2$, …, $a_k$ whose $i$th component, where $i$ is the actual class, is 1 and all other components are 0. We can express the penalty associated with this situation as a loss function that depends on both the $p$ vector and the $a$ vector.

One criterion that is frequently used to evaluate probabilistic prediction is the *quadratic loss function*:

$$\sum_j (p_j - a_j)^2$$

Note that this is for a single instance: The summation is over possible outputs, not over different instances. Just one of the $a$'s will be 1 and the rest 0, so the sum contains contributions of $p_j^2$ for the incorrect predictions and $(1-p_i)^2$ for the correct one. Consequently, it can be written as

$$1 - 2p_i + \sum_j p_j^2$$

where $i$ is the correct class. When the test set contains several instances, the loss function is summed over them all.

It is an interesting theoretical fact that if you seek to minimize the value of the quadratic loss function in a situation where the actual class is generated probabilistically, the best strategy is to choose for the $p$ vector the actual probabilities of the different outcomes—that is, $p_i$ = Pr[class = $i$]. If the true probabilities are known, they will be the best values for $p$. If they are not, a system that strives to minimize the quadratic loss function will be encouraged to use its best estimate of Pr[class = $i$] as the value for $p_i$.

This is quite easy to see. Denote the true probabilities by $p_1$*, $p_2$*, …, $p_k$* so that $p_i$* = Pr[class = $i$]. The expected value of the quadratic loss function over test instances can be rewritten as

$$\begin{aligned} E\left[\sum_j (p_j - a_j)^2\right] &= \sum_j (E[p_j^2] - 2E[p_j a_j] + E[a_j^2]) \\ &= \sum_j (p_j^2 - 2p_j p_j^* + p_j^*) \\ &= \sum_j ((p_j - p_j^*)^2 + p_j^*(1 - p_j^*)) \end{aligned}$$

The first stage involves bringing the expectation inside the sum and expanding the square. For the second, $p_j$ is just a constant and the expected value of $a_j$ is simply $p_j$*; moreover, because $a_j$ is either 0 or 1, $a_j^2 = a_j$ and its expected value is $p_j$* as well. The third stage is straightforward algebra. To minimize the resulting sum, it is clear that it is best to choose $p_j = p_j$*, so that the squared term disappears and all that remains is a term that is just the variance of the true distribution governing the actual class.

Minimizing the squared error has a long history in prediction problems. In the present context, the quadratic loss function forces the predictor to be honest about choosing its best estimate of the probabilities—or, rather, it gives preference to predictors that are able to make the best guess at the true probabilities. Moreover, the quadratic loss function has some useful theoretical properties that we will not go into here. For all these reasons, it is frequently used as the criterion of success in probabilistic prediction situations.

## Informational Loss Function

Another popular criterion used to evaluate probabilistic prediction is the *informational loss function*,

$$-\log_2 p_i$$

where the $i$th prediction is the correct one. This is in fact identical to the negative of the log-likelihood function that is optimized by logistic regression, described in Section 4.6 (modulo a constant factor, which is determined by the base of the logarithm). It represents the information (in bits) required to express the actual class $i$ with respect to the probability distribution $p_1, p_2, \ldots, p_k$. In other words, if you were given the probability distribution and someone had to communicate to you which class was the one that actually occurred, this is the number of bits they would need to encode the information if they did it as effectively as possible. (Of course, it is always possible to use *more* bits.) Because probabilities are always less than 1, their logarithms are negative, and the minus sign makes the outcome positive. For example, in a two-class situation—heads or tails—with an equal probability of each class, the occurrence of a head would take 1 bit to transmit because $-\log_2 1/2$ is 1.

The expected value of the informational loss function, if the true probabilities are $p_1^*$, $p_2^*$, …, $p_k^*$, is

$$-p_1^*\log_2 p_1 - p_2^*\log_2 p_2 - \ldots - p_k^*\log_2 p_k$$

Like the quadratic loss function, this expression is minimized by choosing $p_j = p_j^*$, in which case the expression becomes the entropy of the true distribution:

$$-p_1^*\log_2 p_1^* - p_2^*\log_2 p_2^* - \ldots - p_k^*\log_2 p_k^*$$

Thus, the informational loss function also rewards honesty in predictors that know the true probabilities, and encourages predictors that do not to put forward their best guess.

One problem with the informational loss function is that if you assign a probability of 0 to an event that actually occurs, the function's value is infinity. This corresponds to losing your shirt when gambling. Prudent predictors operating under the informational loss function do not assign zero probability to any outcome. This does lead to a problem when no information is available about that outcome on which to base a prediction. This is called the *zero-frequency problem*, and various plausible solutions have been proposed, such as the Laplace estimator discussed for Naïve Bayes in Chapter 4 (page 93).

## Discussion

If you are in the business of evaluating predictions of probabilities, which of the two loss functions should you use? That's a good question, and there is no universally agreed-on answer—it's really a matter of taste. They both do the fundamental job expected of a loss function: They give maximum reward to predictors that are capable of predicting the true probabilities accurately. However, there are some objective differences between the two that may help you form an opinion.

The quadratic loss function takes into account not only the probability assigned to the event that actually occurred but also the other probabilities. For example, in a four-class situation, suppose you assigned 40% to the class that actually came up and distributed the remainder among the other three classes. The quadratic loss will depend on how you distributed it because of the sum of the $p_j^2$ that occurs in the expression given earlier for the quadratic loss function. The loss will be smallest if the 60% was distributed evenly among the three classes: An uneven distribution will increase the sum of the squares. The informational loss function, on the other hand, depends solely on the probability assigned to the class that actually occurred. If you're gambling on a particular event coming up, and it does, who cares about potential winnings from other events?

If you assign a very small probability to the class that actually occurs, the information loss function will penalize you massively. The maximum penalty, for a zero probability, is infinite. The quadratic loss function, on the other hand, is milder, being bounded by

$$1+\sum_j p_j^2$$

which can never exceed 2.

Finally, proponents of the informational loss function point to a general theory of performance assessment in learning called the *minimum description length (MDL) principle*. They argue that the size of the structures that a scheme learns can be measured in bits of information, and if the same units are used to measure the loss, the two can be combined in useful and powerful ways. We return to this in Section 5.9.

## 5.7 COUNTING THE COST

The evaluations that have been discussed so far do not take into account the cost of making wrong decisions, wrong classifications. Optimizing the classification rate without considering the cost of the errors often leads to strange results. In one case, machine learning was being used to determine the exact day that each cow in a dairy herd was in estrus, or "in heat." Cows were identified by electronic ear tags, and various attributes were used such as milk volume and chemical composition (recorded automatically by a high-tech milking machine) and milking order—for cows are regular beasts and generally arrive in the milking shed in the same order, except in unusual circumstances such as estrus. In a modern dairy operation it's important to know when a cow is ready: Animals are fertilized by artificial insemination and missing a cycle will delay calving unnecessarily, causing complications down the line. In early experiments, machine learning schemes stubbornly predicted that each cow was *never* in estrus. Like humans, cows have a menstrual cycle of approximately 30 days, so this "null" rule is correct about 97% of the time—an impressive degree of accuracy in any agricultural domain! What was wanted, of course, was rules that predicted the "in estrus" situation more accurately than the "not in estrus" one: The costs of the two kinds of error were different. Evaluation by classification accuracy tacitly assumes equal error costs.

Other examples where errors cost different amounts include loan decisions: The cost of lending to a defaulter is far greater than the lost-business cost of refusing a loan to a nondefaulter. And oil-slick detection: The cost of failing to detect an environment-threatening real slick is far greater than the cost of a false alarm. And load forecasting: The cost of gearing up electricity generators for a storm that doesn't hit is far less than the cost of being caught completely unprepared. And diagnosis:

**Table 5.3** Different Outcomes of a Two-Class Prediction

| | | Predicted Class | |
|---|---|---|---|
| | | *yes* | *no* |
| **Actual Class** | *yes* | true positive | false negative |
| | *no* | false positive | true negative |

The cost of misidentifying problems with a machine that turns out to be free of faults is less than the cost of overlooking problems with one that is about to fail. And promotional mailing: The cost of sending junk mail to a household that doesn't respond is far less than the lost-business cost of not sending it to a household that would have responded. Why—these are all the examples from Chapter 1! In truth, you'd be hard pressed to find an application in which the costs of different kinds of errors were the same.

In the two-class case with classes *yes* and *no*—lend or not lend, mark a suspicious patch as an oil slick or not, and so on—a single prediction has the four different possible outcomes shown in Table 5.3. The *true positives* (TP) and *true negatives* (TN) are correct classifications. A *false positive* (FP) is when the outcome is incorrectly predicted as *yes* (or positive) when it is actually *no* (negative). A *false negative* (FN) is when the outcome is incorrectly predicted as negative when it is actually positive. The *true positive rate* is TP divided by the total number of positives, which is TP + FN; the *false positive rate* is FP divided by the total number of negatives, which is FP + TN. The overall success rate is the number of correct classifications divided by the total number of classifications:

$$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$

Finally, the error rate is 1 minus this.

In multiclass prediction, the result on a test set is often displayed as a two-dimensional *confusion matrix* with a row and column for each class. Each matrix element shows the number of test examples for which the actual class is the row and the predicted class is the column. Good results correspond to large numbers down the main diagonal and small, ideally zero, off-diagonal elements. Table 5.4(a) shows a numeric example with three classes. In this case, the test set has 200 instances (the sum of the nine numbers in the matrix), and 88 + 40 + 12 = 140 of them are predicted correctly, so the success rate is 70%.

But is this a fair measure of overall success? How many agreements would you expect *by chance*? This predictor predicts a total of 120 *a*'s, 60 *b*'s, and 20 *c*'s; what if you had a random predictor that predicted the same total numbers of the three classes? The answer is shown in Table 5.4(b). Its first row divides the 100 *a*'s in the test set into these overall proportions, and the second and third rows do the same

**Table 5.4** Different Outcomes of a Three-Class Prediction: (a) Actual and (b) Expected

| | | **Predicted Class** | | | | | | **Predicted Class** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *a* | *b* | *c* | *Total* | | | *a* | *b* | *c* | *Total* |
| **Actual Class** | *a* | 88 | 10 | 2 | 100 | **Actual Class** | *a* | 60 | 30 | 10 | 100 |
| | *b* | 14 | 40 | 6 | 60 | | *b* | 36 | 18 | 6 | 60 |
| | *c* | 18 | 10 | 12 | 40 | | *c* | 24 | 12 | 4 | 40 |
| | *Total* | 120 | 60 | 20 | | | *Total* | 120 | 60 | 20 | |
| | | **(a)** | | | | | | **(b)** | | | |

thing for the other two classes. Of course, the row and column totals for this matrix are the same as before—the number of instances hasn't changed, and we have ensured that the random predictor predicts the same number of *a*'s, *b*'s, and *c*'s as the actual predictor.

This random predictor gets 60 + 18 + 4 = 82 instances correct. A measure called the *Kappa statistic* takes this expected figure into account by deducting it from the predictor's successes and expressing the result as a proportion of the total for a perfect predictor, to yield 140 – 82 = 58 extra successes out of a possible total of 200 – 82 = 118, or 49.2%. The maximum value of Kappa is 100%, and the expected value for a random predictor with the same column totals is 0. In summary, the Kappa statistic is used to measure the agreement between predicted and observed categorizations of a dataset, while correcting for an agreement that occurs by chance. However, like the plain success rate, it does not take costs into account.

## Cost-Sensitive Classification

If the costs are known, they can be incorporated into a financial analysis of the decision-making process. In the two-class case, in which the confusion matrix is like that of Table 5.3, the two kinds of error—false positives and false negatives—will have different costs; likewise, the two types of correct classification may have different benefits. In the two-class case, costs can be summarized in the form of a 2 × 2 matrix in which the diagonal elements represent the two types of correct classification and the off-diagonal elements represent the two types of error. In the multiclass case this generalizes to a square matrix whose size is the number of classes, and again the diagonal elements represent the cost of correct classification. Table 5.5(a) and (b) shows default cost matrixes for the two- and three-class cases, whose values simply give the number of errors: Misclassification costs are all 1.

Taking the cost matrix into account replaces the success rate by the average cost (or, thinking more positively, profit) per decision. Although we will not do so here, a complete financial analysis of the decision-making process might also take into account the cost of using the machine learning tool—including the cost of gathering the training data—and the cost of using the model, or decision structure, that it

**Table 5.5** Default Cost Matrixes: (a) Two-Class Case and (b) Three-Class Case

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Predicted Class** | | | | | **Predicted Class** | | |
| | | *yes* | *no* | | | *a* | *b* | *c* |
| **Actual Class** | *yes* | 0 | 1 | **Actual Class** | *a* | 0 | 1 | 1 |
| | *no* | 1 | 0 | | *b* | 1 | 0 | 1 |
| | | | | | *c* | 1 | 1 | 0 |
| | | **(a)** | | | | **(b)** | | |

produces—including the cost of determining the attributes for the test instances. If all costs are known, and the projected number of the four different outcomes in the cost matrix can be estimated, say using cross-validation, it is straightforward to perform this kind of financial analysis.

Given a cost matrix, you can calculate the cost of a particular learned model on a given test set just by summing the relevant elements of the cost matrix for the model's prediction for each test instance. Here, costs are ignored when making predictions, but taken into account when evaluating them.

If the model outputs the probability associated with each prediction, it can be adjusted to minimize the expected cost of the predictions. Given a set of predicted probabilities for each outcome on a certain test instance, one normally selects the most likely outcome. Instead, the model could predict the class with the smallest expected misclassification cost. For example, suppose in a three-class situation the model assigns the classes *a, b,* and *c* to a test instance with probabilities $p_a$, $p_b$, and $p_c$, and the cost matrix is that in Table 5.5(b). If it predicts *a*, the expected cost of the prediction is obtained by multiplying the first column of the matrix, [0,1,1], by the probability vector, $[p_a, p_b, p_c]$, yielding $p_b + p_c$, or $1 - p_a$, because the three probabilities sum to 1. Similarly, the costs for predicting the other two classes are $1 - p_b$ and $1 - p_c$. For this cost matrix, choosing the prediction with the lowest expected cost is the same as choosing the one with the greatest probability. For a different cost matrix it might be different.

We have assumed that the learning scheme outputs probabilities, as Naïve Bayes does. Even if they do not normally output probabilities, most classifiers can easily be adapted to compute them. In a decision tree, for example, the probability distribution for a test instance is just the distribution of classes at the corresponding leaf.

## Cost-Sensitive Learning

We have seen how a classifier, built without taking costs into consideration, can be used to make predictions that are sensitive to the cost matrix. In this case, costs are ignored at training time but used at prediction time. An alternative is to do just the opposite: Take the cost matrix into account during the training process and ignore costs at prediction time. In principle, better performance might be obtained if the classifier were tailored by the learning algorithm to the cost matrix.

In the two-class situation, there is a simple and general way to make any learning scheme cost sensitive. The idea is to generate training data with a different proportion of *yes* and *no* instances. Suppose you artificially increase the number of *no* instances by a factor of 10 and use the resulting dataset for training. If the learning scheme is striving to minimize the number of errors, it will come up with a decision structure that is biased toward avoiding errors on the *no* instances because such errors are effectively penalized tenfold. If data with the original proportion of *no* instances is used for testing, fewer errors will be made on these than on *yes* instances—that is, there will be fewer false positives than false negatives—because false positives have been weighted 10 times more heavily than false negatives.

Varying the proportion of instances in the training set is a general technique for building cost-sensitive classifiers.

One way to vary the proportion of training instances is to duplicate instances in the dataset. However, many learning schemes allow instances to be weighted. (As we mentioned in Section 3.2, this is a common technique for handling missing values.) Instance weights are normally initialized to 1. To build cost-sensitive classifiers the weights can be initialized to the relative cost of the two kinds of error, false positives and false negatives.

## Lift Charts

In practice, costs are rarely known with any degree of accuracy, and people will want to ponder various different scenarios. Imagine you're in the direct-mailing business and are contemplating a mass mailout of a promotional offer to 1,000,000 households, most of whom won't respond, of course. Let us say that, based on previous experience, the proportion that normally respond is known to be 0.1% (1000 respondents). Suppose a data mining tool is available that, based on known information about the households, identifies a subset of 100,000 for which the response rate is 0.4% (400 respondents). It may well pay off to restrict the mailout to these 100,000 households; this, of course, depends on the mailing cost compared with the return gained for each response to the offer. In marketing terminology, the increase in response rate, a factor of 4 in this case, is known as the *lift* factor yielded by the learning tool. If you knew the costs, you could determine the payoff implied by a particular lift factor.

But you probably want to evaluate other possibilities too. The same data mining scheme, with different parameter settings, may be able to identify 400,000 households for which the response rate will be 0.2% (800 respondents), corresponding to a lift factor of 2. Again, whether this would be a more profitable target for the mailout can be calculated from the costs involved. It may be necessary to factor in the cost of creating and using the model, including collecting the information that is required to come up with the attribute values. After all, if developing the model is very expensive, a mass mailing may be more cost effective than a targeted one.

Given a learning scheme that outputs probabilities for the predicted class of each member of the set of test instances (as Naïve Bayes does), your job is to find subsets of test instances that have a high proportion of positive instances, higher than in the test set as a whole. To do this, the instances should be sorted in descending order of predicted probability of *yes*. Then, to find a sample of a given size with the greatest possible proportion of positive instances, just read the requisite number of instances off the list, starting at the top. If each test instance's class is known, you can calculate the lift factor by simply counting the number of positive instances that the sample includes, dividing by the sample size to obtain a success proportion, and dividing by the success proportion for the complete test set to determine the lift factor.

**Table 5.6** Data for a Lift Chart

| Rank | Predicted | Actual Class |
|---|---|---|
| 1 | 0.95 | *yes* |
| 2 | 0.93 | *yes* |
| 3 | 0.93 | *no* |
| 4 | 0.88 | *yes* |
| 5 | 0.86 | *yes* |
| 6 | 0.85 | *yes* |
| 7 | 0.82 | *yes* |
| 8 | 0.80 | *yes* |
| 9 | 0.80 | *no* |
| 10 | 0.79 | *yes* |
| 11 | 0.77 | *no* |
| 12 | 0.76 | *yes* |
| 13 | 0.73 | *yes* |
| 14 | 0.65 | *no* |
| 15 | 0.63 | *yes* |
| 16 | 0.58 | *no* |
| 17 | 0.56 | *yes* |
| 18 | 0.49 | *no* |
| 19 | 0.48 | *yes* |
| … | … | … |

Table 5.6 shows an example, for a small dataset that has 150 instances, of which 50 are *yes* responses—an overall success proportion of 33%. The instances have been sorted in descending probability order according to the predicted probability of a *yes* response. The first instance is the one that the learning scheme thinks is the most likely to be positive, the second is the next most likely, and so on. The numeric values of the probabilities are unimportant: Rank is the only thing that matters. With each rank is given the actual class of the instance. Thus, the learning scheme was correct about items 1 and 2—they are indeed positives—but wrong about item 3, which turned out to be negative. Now, if you were seeking the most promising sample of size 10, but only knew the predicted probabilities and not the actual classes, your best bet would be the top 10 ranking instances. Eight of these are positive, so the success proportion for this sample is 80%, corresponding to a lift factor of about 2.4.

If you knew the different costs involved, you could work them out for each sample size and choose the most profitable. But a graphical depiction of the various possibilities will often be far more revealing than presenting a single "optimal" decision. Repeating the operation for different-size samples allows you to plot a lift chart like that of Figure 5.1. The horizontal axis shows the sample size as a proportion of the total possible mailout. The vertical axis shows the number of responses obtained. The lower left and upper right points correspond to no mailout at all, with a response of 0, and a full mailout, with a response of 1000. The diagonal line gives the expected result for different-size random samples. But we do not choose random samples; we choose those instances that, according to the data mining tool, are most likely to generate a positive response. These correspond to the upper line, which is derived by summing the actual responses over the corresponding percentage of the instance list sorted in probability order. The two particular scenarios described previously are marked: a 10% mailout that yields 400 respondents and a 40% one that yields 800.

Where you'd like to be in a lift chart is near the upper left corner: At the very best, 1000 responses from a mailout of just 1000, where you send only to those

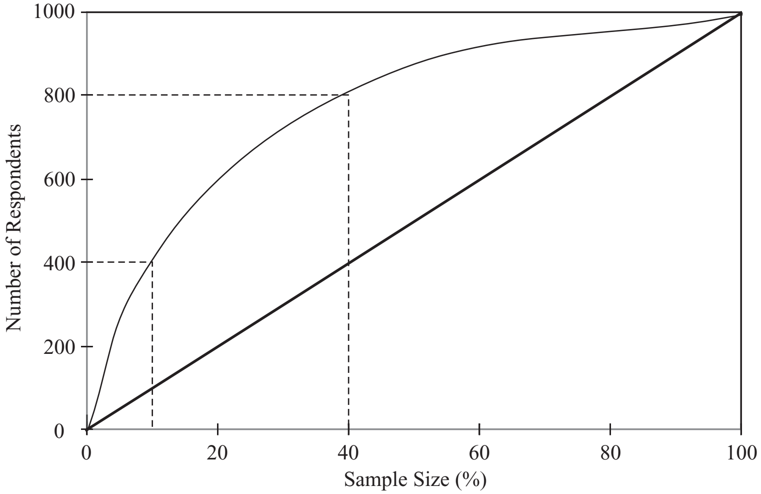


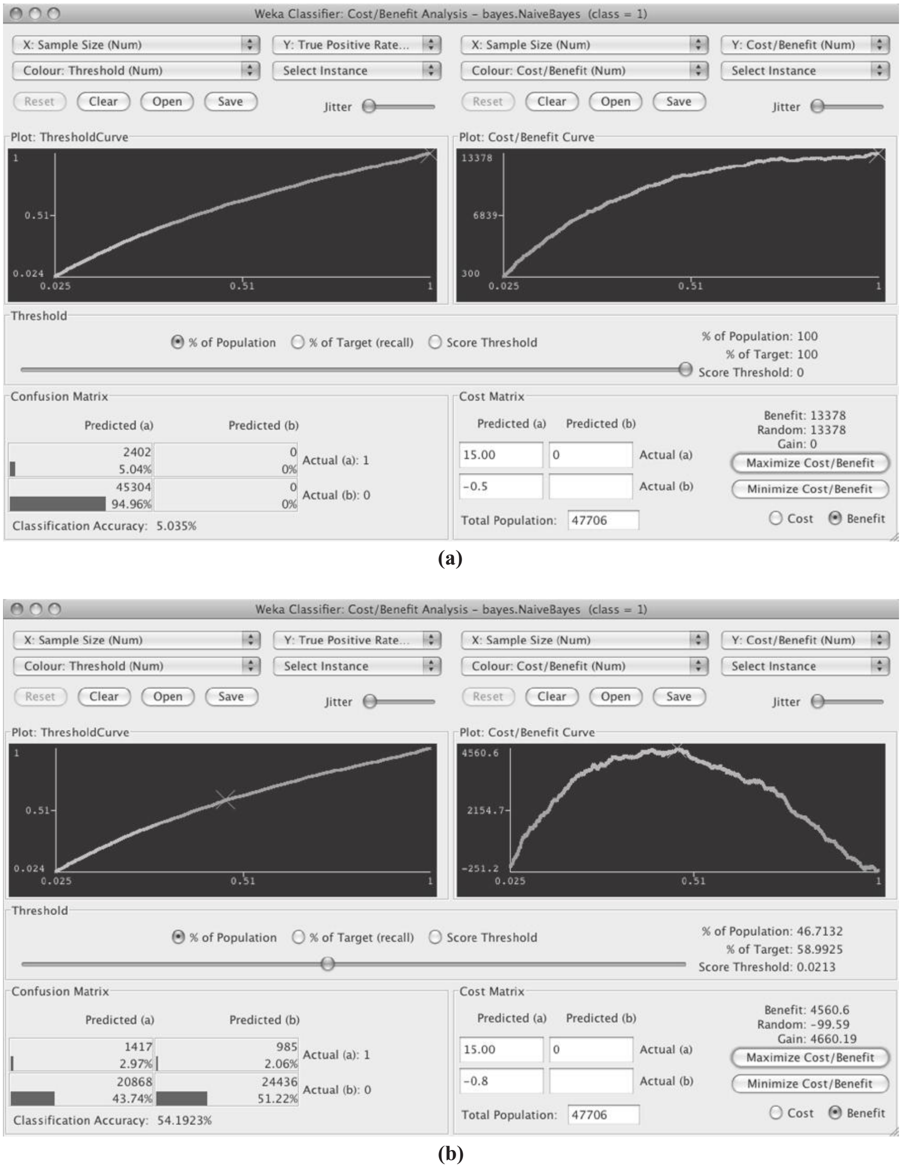
**FIGURE 5.1**

A hypothetical lift chart.

households that will respond and are rewarded with a 100% success rate. Any selection procedure worthy of the name will keep you above the diagonal—otherwise, you'd be seeing a response that is worse than for random sampling. So the operating part of the diagram is the upper triangle, and the farther to the upper left the better.

Figure 5.2(a) shows a visualization that allows various cost scenarios to be explored in an interactive fashion (called the *cost–benefit analyzer*, it forms part of the Weka workbench described in Part III). Here it is displaying results for predictions generated by the Naïve Bayes classifier on a real-world direct-mail data set. In this example, 47,706 instances were used for training and a further 47,706 for testing. The test instances were ranked according to the predicted probability of a response to the mailout. The graphs show a lift chart on the left and the total cost (or benefit), plotted against the sample size, on the right. At the lower left is a confusion matrix; at the lower right is a cost matrix.

Cost or benefit values associated with incorrect or correct classifications can be entered into the matrix and affect the shape of the curve above. The horizontal slider in the middle allows users to vary the percentage of the population that is selected from the ranked list. Alternatively, one can determine the sample size by adjusting the recall level (the proportion of positives to be included in the sample) or by adjusting a threshold on the probability of the positive class, which here corresponds to a response to the mailout. When the slider is moved, a large cross shows the corresponding point on both graphs. The total cost or benefit associated with the selected sample size is shown at the lower right, along with the expected response to a random mailout of the same size.



X: Sample Size (Num)

Y: True Positive Rate...

Colour: Threshold (Num)

Select Instance

X: Sample Size (Num)

Y: Cost/Benefit (Num)

Colour: Cost/Benefit (Num)

Select Instance

Threshold

● % of Population ○ % of Target (recall) ○ Score Threshold

% of Population: 100
% of Target: 100
Score Threshold: 0

Confusion Matrix

| Predicted (a) | Predicted (b) | |
|---|---|---|
| 2402<br>5.04% | 0<br>0% | Actual (a): 1 |
| 45304<br>94.96% | 0<br>0% | Actual (b): 0 |

Classification Accuracy: 5.035%

Cost Matrix

| Predicted (a) | Predicted (b) | |
|---|---|---|
| 15.00 | 0 | Actual (a) |
| -0.5 | | Actual (b) |

Total Population: 47706

Benefit: 13378
Random: 13378
Gain: 0

○ Cost ● Benefit

**(a)**

X: Sample Size (Num)

Y: True Positive Rate...

Colour: Threshold (Num)

Select Instance

X: Sample Size (Num)

Y: Cost/Benefit (Num)

Colour: Cost/Benefit (Num)

Select Instance

Threshold

● % of Population ○ % of Target (recall) ○ Score Threshold

% of Population: 46.7132
% of Target: 58.9925
Score Threshold: 0.0213

Confusion Matrix

| Predicted (a) | Predicted (b) | |
|---|---|---|
| 1417<br>2.97% | 985<br>2.06% | Actual (a): 1 |
| 20868<br>43.74% | 24436<br>51.22% | Actual (b): 0 |

Classification Accuracy: 54.1923%

Cost Matrix

| Predicted (a) | Predicted (b) | |
|---|---|---|
| 15.00 | 0 | Actual (a) |
| -0.8 | | Actual (b) |

Total Population: 47706

Benefit: 4560.6
Random: -99.59
Gain: 4660.19

○ Cost ● Benefit

**(b)**

**FIGURE 5.2**

Analyzing the expected benefit of a mailing campaign when the cost of mailing is (a) $0.50 and (b) $0.80.

In the cost matrix in Figure 5.2(a), a cost of $0.50—the cost of mailing—has been associated with nonrespondents and a benefit of $15.00 with respondents (after deducting the mailing cost). Under these conditions, and using the Naïve Bayes classifier, there is no subset from the ranked list of prospects that yields a greater profit than mailing to the entire population. However, a slightly higher mailing cost changes the situation dramatically, and Figure 5.2(b) shows what happens when it is increased to $0.80. Assuming the same profit of $15.00 per respondent, a maximum profit of $4,560.60 is achieved by mailing to the top 46.7% of the population. In this situation, a random sample of the same size achieves a loss of $99.59.

### ROC Curves

Lift charts are a valuable tool, widely used in marketing. They are closely related to a graphical technique for evaluating data mining schemes known as *ROC curves*, which are used in just the same situation, where the learner is trying to select samples of test instances that have a high proportion of positives. The acronym stands for *receiver operating characteristic*, a term used in signal detection to characterize the tradeoff between hit rate and false-alarm rate over a noisy channel. ROC curves depict the performance of a classifier without regard to class distribution or error costs. They plot the true positive rate on the vertical axis against the true negative rate on the horizontal axis. The former is the number of positives included in the sample, expressed as a percentage of the total number of positives (TP Rate = 100 × TP/(TP + FN)); the latter is the number of negatives included in the sample, expressed as a percentage of the total number of negatives (FP Rate = 100 × FP/(FP + TN)). The vertical axis is the same as the lift chart's except that it is expressed as a percentage. The horizontal axis is slightly different—it is the number of negatives rather than the sample size. However, in direct marketing situations where the proportion of positives is very small anyway (like 0.1%), there is negligible difference between the size of a sample and the number of negatives it contains, so the ROC curve and lift chart look very similar. As with lift charts, the upper left corner is the place to be.

Figure 5.3 shows an example ROC curve—the jagged line—for the sample of test data shown earlier in Table 5.6. You can follow it along with the table. From the origin: Go up two (two positives), along one (one negative), up five (five positives), along two (two negatives), up one, along one, up two, and so on. Each point corresponds to drawing a line at a certain position on the ranked list, counting the *yes's* and *no's* above it, and plotting them vertically and horizontally, respectively. As you go farther down the list, corresponding to a larger sample, the number of positives and negatives both increase.

The jagged ROC line in Figure 5.3 depends intimately on the details of the particular sample of test data. This sample dependence can be reduced by applying cross-validation. For each different number of *no*'s—that is, each position along the horizontal axis—take just enough of the highest-ranked instances to include that number of *no*'s, and count the number of *yes*'s they contain. Finally, average that

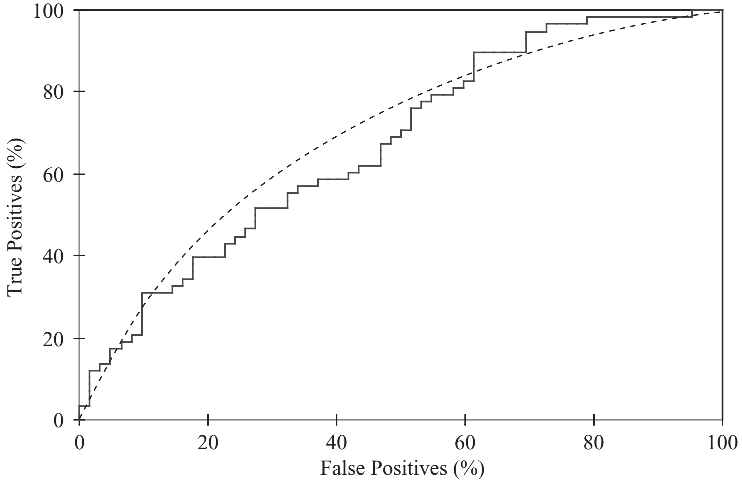


**FIGURE 5.3**

A sample ROC curve.

number over different folds of the cross-validation. The result is a smooth curve like that in Figure 5.3—although in reality such curves do not generally look quite so smooth.

This is just one way of using cross-validation to generate ROC curves. A simpler approach is to collect the predicted probabilities for all the various test sets (of which there are 10 in a tenfold cross-validation), along with the true class labels of the corresponding instances, and generate a single ranked list based on this data. This assumes that the probability estimates from the classifiers built from the different training sets are all based on equally sized random samples of the data. It is not clear which method is preferable. However, the latter method is easier to implement.

If the learning scheme does not allow the instances to be ordered, you can first make it cost-sensitive as described earlier. For each fold of a tenfold cross-validation, weight the instances for a selection of different cost ratios, train the scheme on each weighted set, count the true positives and false positives in the test set, and plot the resulting point on the ROC axes. (It doesn't matter whether the test set is weighted or not because the axes in the ROC diagram are expressed as the percentage of true and false positives.) However, for probabilistic classifiers such as Naïve Bayes it is far more costly than the method described previously because it involves a separate learning problem for every point on the curve.

It is instructive to look at ROC curves obtained using different learning schemes. For example, in Figure 5.4, method A excels if a small, focused sample is sought—that is, if you are working toward the left side of the graph. Clearly, if you aim to cover just 40% of the true positives you should choose method A, which gives a false positive rate of around 5%, rather than method B, which gives more than 20%

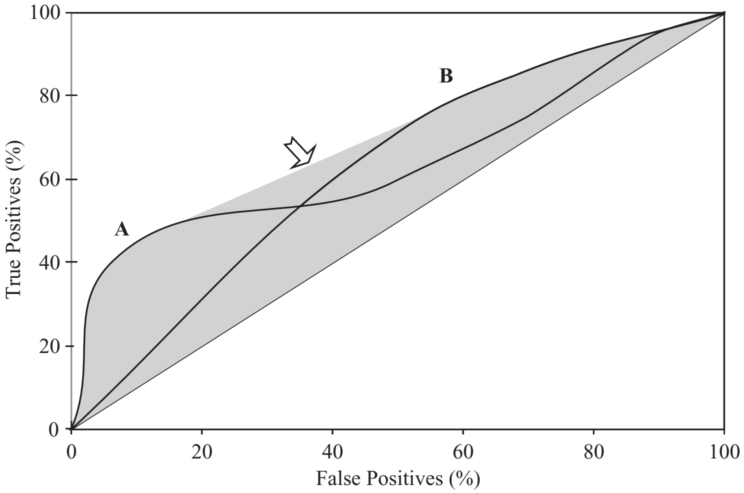


**FIGURE 5.4**

ROC curves for two learning schemes.

false positives. But method B excels if you are planning a large sample: If you are covering 80% of the true positives, B will give a false positive rate of 60% as compared with method A's 80%. The shaded area is called the *convex hull* of the two curves, and you should always operate at a point that lies on the upper boundary of the convex hull.

What about the region in the middle where neither method A nor method B lies on the convex hull? It is a remarkable fact that you can get anywhere in the shaded region by combining methods A and B and using them at random with appropriate probabilities. To see this, choose a particular probability cutoff for method A that gives true and false positive rates of $t_A$ and $f_A$, respectively, and another cutoff for method B that gives $t_B$ and $f_B$. If you use these two schemes at random with probabilities $p$ and $q$, where $p + q = 1$, then you will get true and false positive rates of $p \, . \, t_A + q \, . \, t_B$ and $p \, . f_A + q \, . f_B$. This represents a point lying on the straight line joining the points $(t_A, f_A)$ and $(t_B, f_B)$, and by varying $p$ and $q$ you can trace out the whole line between these two points. By this device, the entire shaded region can be reached. Only if a particular scheme generates a point that lies on the convex hull should it be used alone. Otherwise, it would always be better to use a combination of classifiers corresponding to a point that lies on the convex hull.

## Recall–Precision Curves

People have grappled with the fundamental tradeoff illustrated by lift charts and ROC curves in a wide variety of domains. Information retrieval is a good example. Given a query, a Web search engine produces a list of hits that represent documents supposedly relevant to the query. Compare one system that locates 100 documents, 40 of which are relevant, with another that locates 400 documents, 80 of which are relevant. Which is better? The answer should now be obvious: It depends on the relative cost of false positives, documents returned that aren't relevant, and false negatives, documents that are relevant but aren't returned. Information retrieval researchers define parameters called *recall* and *precision*:

$$\text{Recall} = \frac{\text{number of documents retrieved that are relevant}}{\text{total number of documents that are relevant}}$$

$$\text{Precision} = \frac{\text{number of documents retrieved that are relevant}}{\text{total number of documents that are retrieved}}$$

For example, if the list of *yes*'s and *no*'s in Table 5.6 represented a ranked list of retrieved documents and whether they were relevant or not, and the entire collection contained a total of 40 relevant documents, then "recall at 10" would refer to the recall for the top 10 documents—that is, 8/40 = 20%—while "precision at 10" would be 8/10 = 80%. Information retrieval experts use *recall–precision curves* that plot one against the other, for different numbers of retrieved documents, in just the same way as ROC curves and lift charts—except that, because the axes are different, the curves are hyperbolic in shape and the desired operating point is toward the upper right.

## Discussion

Table 5.7 summarizes the three different ways introduced for evaluating the same basic tradeoff; TP, FP, TN, and FN are the numbers of true positives, false positives, true negatives, and false negatives, respectively. You want to choose a set of instances with a high proportion of *yes* instances and a high coverage of the *yes* instances: You can increase the proportion by (conservatively) using a smaller coverage, or (liberally) increase the coverage at the expense of the proportion. Different techniques give different tradeoffs, and can be plotted as different lines on any of these graphical charts.

People also seek single measures that characterize performance. Two that are used in information retrieval are *three-point average recall*, which gives the average precision obtained at recall values of 20%, 50%, and 80%, and *11-point average recall*, which gives the average precision obtained at recall values of 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%. Also used in information retrieval is the *F-measure*, which is

$$\frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}$$

Different terms are used in different domains. Physicians, for example, talk about the *sensitivity* and *specificity* of diagnostic tests. Sensitivity refers to the proportion

**Table 5.7** Different Measures Used to Evaluate False Positive versus False Negative Tradeoff

| | Domain | Plot | Axes | Explanation of Axes |
|---|---|---|---|---|
| Lift chart | Marketing | TP vs. subset size | TP | number of true positives |
| | | | subset size | $\frac{\mathrm{TP}+\mathrm{FP}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}} \times 100\%$ |
| ROC curve | Communications | TP rate vs. FP rate | TP rate | $tp = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \times 100\%$ |
| | | | FP rate | $fp = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} \times 100\%$ |
| Recall–precision curve | Information retrieval | Recall vs. precision | Recall | same as TP rate of *tp* above |
| | | | precision | $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \times 100\%$ |

of people with disease who have a positive test result—that is, *tp*. Specificity refers to the proportion of people without disease who have a negative test result, which is 1 – *fp*. Sometimes the product of these is used as an overall measure:

$$\text{sensitivity} \times \text{specificity} = tp(1 - fp) = \frac{\text{TP} \times \text{TN}}{(\text{TP} + \text{FN}) \times (\text{FP} + \text{TN})}$$

Finally, of course, there is our old friend the success rate:

$$\frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

To summarize ROC curves in a single quantity, people sometimes use the area under the curve (AUC) because, roughly speaking, the larger the area the better the model. The area also has a nice interpretation as the probability that the classifier ranks a randomly chosen positive instance above a randomly chosen negative one. Although such measures may be useful if costs and class distributions are unknown and one scheme must be chosen to handle all situations, no single number is able to capture the tradeoff. That can only be done by two-dimensional depictions such as lift charts, ROC curves, and recall–precision diagrams.

Several methods are commonly employed for computing the area under the ROC curve. One, corresponding to a geometric interpretation, is to approximate it by fitting several trapezoids under the curve and summing up their area. Another is to compute the probability that the classifier ranks a randomly chosen positive instance above a randomly chosen negative one. This can be accomplished by calculating the Mann–Whitney *U* statistic, or, more specifically, the $\rho$ statistic from the *U* statistic. This value is easily obtained from a list of test instances sorted in descending order of predicted probability of the positive class. For each positive instance, count how many negative ones are ranked below it (increase the count by ½ if positive and negative instances tie in rank). The *U* statistic is simply the total of these counts. The $\rho$ statistic is obtained by dividing *U* by the product of the number of positive and negative instances in the test set—in other words, the *U* value that would result if all positive instances were ranked above the negative ones.

The area under the precision–recall curve (AUPRC) is an alternative summary statistic that is preferred by some practitioners, particularly in the information retrieval area.

## Cost Curves

ROC curves and their relatives are very useful for exploring the tradeoffs among different classifiers over a range of scenarios. However, they are not ideal for evaluating machine learning models in situations with known error costs. For example, it is not easy to read off the expected cost of a classifier for a fixed cost matrix and class distribution. Neither can you easily determine the ranges of applicability of different classifiers. For example, from the crossover point between the two ROC curves in Figure 5.4 it is hard to tell for what cost and class distributions classifier A outperforms classifier B.

*Cost curves* are a different kind of display on which a single classifier corresponds to a straight line that shows how the performance varies as the class distribution changes. Again, they work best in the two-class case, although you can always make a multiclass problem into a two-class one by singling out one class and evaluating it against the remaining ones.

Figure 5.5(a) plots the expected error against the probability of one of the classes. You could imagine adjusting this probability by resampling the test set in a nonuniform way. We denote the two classes by + and –. The diagonals show the performance of two extreme classifiers: One always predicts +, giving an expected error of 1 if the dataset contains no + instances and 0 if all its instances are +; the other always predicts –, giving the opposite performance. The dashed horizontal line shows the performance of the classifier that is always wrong, and the *x*-axis itself represents the classifier that is always correct. In practice, of course, neither of these is realizable. Good classifiers have low error rates, so where you want to be is as close to the bottom of the diagram as possible.

The line marked A represents the error rate of a particular classifier. If you calculate its performance on a certain test set, its false positive rate, *fp*, is its expected error on a subsample of the test set that contains only examples that are negative (*p*[+] = 0), and its false negative rate, *fn*, is the error on a subsample that contains only positive examples, (*p*[+] = 1). These are the values of the intercepts at the left and right, respectively. You can see immediately from the plot that if *p*[+] is smaller than about 0.2, predictor A is outperformed by the extreme classifier that always predicts –, while if it is larger than about 0.65, the other extreme classifier is better.

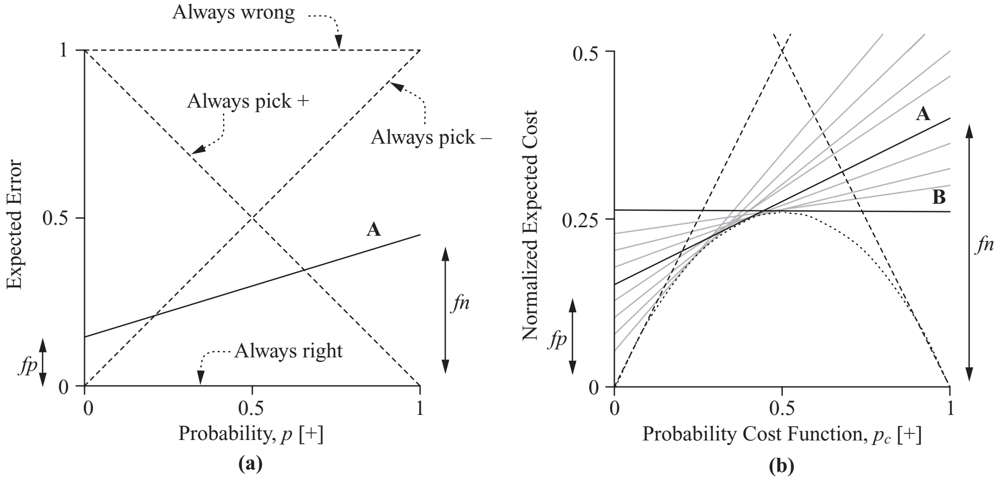


**FIGURE 5.5**

Effect of varying the probability threshold: (a) error curve and (b) cost curve.

So far we have not taken costs into account, or rather we have used the default cost matrix in which all errors cost the same. Cost curves, which do take cost into account, look very similar—very similar indeed—but the axes are different. Figure 5.5(b) shows a cost curve for the same classifier A (note that the vertical scale has been enlarged, for convenience, and ignore the gray lines for now). It plots the expected cost of using A against the probability cost function, which is a distorted version of $p[+]$ that retains the same extremes: 0 when $p[+] = 0$ and 1 when $p[+] = 1$. Denote by $C[+ \mid -]$ the cost of predicting + when the instance is actually –, and the reverse by $C[- \mid +]$. Then the axes of Figure 5.5(b) are

$$\text{Normalized expected cost} = fn \times p_C[+] + fp \times (1 - p_C[+])$$

$$\text{Probability cost function } p_C[+] = \frac{p[+]C[- \mid +]}{p[+]C[- \mid +] + p[-]C[+ \mid -]}$$

We are assuming here that correct predictions have no cost: $C[+ \mid +] = C[- \mid -] = 0$. If that is not the case, the formulas are a little more complex.

The maximum value that the normalized expected cost can have is 1—that is why it is “normalized.” One nice thing about cost curves is that the extreme cost values at the left and right sides of the graph are *fp* and *fn*, just as they are for the error curve, so you can draw the cost curve for any classifier very easily.

Figure 5.5(b) also shows classifier B, whose expected cost remains the same across the range—that is, its false positive and false negative rates are equal. As you can see, it outperforms classifier A if the probability cost function exceeds about 0.45, and knowing the costs we could easily work out what this corresponds to in terms of class distribution. In situations that involve different class distributions, cost curves make it easy to tell when one classifier will outperform another.

In what circumstances might this be useful? To return to our example of predicting when cows will be in estrus, their 30-day cycle, or 1/30 prior probability, is unlikely to vary greatly (barring a genetic cataclysm!). But a particular herd may have different proportions of cows that are likely to reach estrus in any given week, perhaps synchronized with—who knows?—the phase of the moon. Then, different classifiers would be appropriate at different times. In the oil spill example, different batches of data may have different spill probabilities. In these situations cost curves can help to show which classifier to use when.

Each point on a lift chart, ROC curve, or recall–precision curve represents a classifier, typically obtained by using different threshold values for a method such as Naïve Bayes. Cost curves represent each classifier by a straight line, and a suite of classifiers will sweep out a curved envelope whose lower limit shows how well that type of classifier can do if the parameter is well chosen. Figure 5.5(b) indicates this with a few gray lines. If the process were continued, it would sweep out the dotted parabolic curve.

The operating region of classifier B ranges from a probability cost value of about 0.25 to a value of about 0.75. Outside this region, classifier B is outperformed by the trivial classifiers represented by dashed lines. Suppose we decide to use classifier

B within this range and the appropriate trivial classifier below and above it. All points on the parabola are certainly better than this scheme. But how much better? It is hard to answer such questions from an ROC curve, but the cost curve makes them easy. The performance difference is negligible if the probability cost value is around 0.5, and below a value of about 0.2 and above 0.8 it is barely perceptible. The greatest difference occurs at probability cost values of 0.25 and 0.75 and is about 0.04, or 4% of the maximum possible cost figure.

## 5.8 EVALUATING NUMERIC PREDICTION

All the evaluation measures we have described pertain to classification situations rather than numeric prediction situations. The basic principles—using an independent test set rather than the training set for performance evaluation, the holdout method, cross-validation—apply equally well to numeric prediction. But the basic quality measure offered by the error rate is no longer appropriate: Errors are not simply present or absent; they come in different sizes.

Several alternative measures, some of which are summarized in Table 5.8, can be used to evaluate the success of numeric prediction. The predicted values on the test instances are $p_1, p_2, \ldots, p_n$; the actual values are $a_1, a_2, \ldots, a_n$. Notice that $p_i$ means

**Table 5.8** Performance Measures for Numeric Prediction

| | |
|---|---|
| Mean-squared error | $\frac{(p_1-a_1)^2+\quad+(p_n-a_n)^2}{n}$ |
| Root mean-squared error | $\sqrt{\frac{(p_1-a_1)^2+\quad+(p_n-a_n)^2}{n}}$ |
| Mean-absolute error | $\frac{\lvert p_1-a_1\rvert+\quad+\lvert p_n-a_n\rvert}{n}$ |
| Relative-squared error* | $\frac{(p_1-a_1)^2+\quad+(p_n-a_n)^2}{(a_1-\bar{a})^2+\quad+(a_n-\bar{a})^2}$ |
| Root relative-squared error* | $\sqrt{\frac{(p_1-a_1)^2+\quad+(p_n-a_n)^2}{(a_1-\bar{a})^2+\quad+(a_n-\bar{a})^2}}$ |
| Relative-absolute error* | $\frac{\lvert p_1-a_1\rvert+\quad+\lvert p_n-a_n\rvert}{\lvert a_1-\bar{a}\rvert+\quad+\lvert a_n-\bar{a}\rvert}$ |
| Correlation coefficient** | $\frac{S_{PA}}{\sqrt{S_P S_A}}$, where $S_{PA}=\frac{\sum_i (p_i-\bar{p})(a_i-\bar{a})}{n-1}$,<br>$S_P=\frac{\sum_i (p_i-\bar{p})^2}{n-1}$, $S_A=\frac{\sum_i (a_i-\bar{a})^2}{n-1}$ |

**Here, $\bar{a}$ is the mean value over the training data.*
***Here, $\bar{a}$ is the mean value over the test data.*

something very different here from what it meant in the last section: There it was the probability that a particular prediction was in the *i*th class; here it refers to the numerical value of the prediction for the *i*th test instance.

*Mean-squared error* is the principal and most commonly used measure; sometimes the square root is taken to give it the same dimensions as the predicted value itself. Many mathematical techniques (such as linear regression, explained in Chapter 4) use the mean-squared error because it tends to be the easiest measure to manipulate mathematically: It is, as mathematicians say, "well behaved." However, here we are considering it as a performance measure: All the performance measures are easy to calculate, so mean-squared error has no particular advantage. The question is, is it an appropriate measure for the task at hand?

*Mean absolute error* is an alternative: Just average the magnitude of the individual errors without taking account of their sign. Mean-squared error tends to exaggerate the effect of outliers—instances when the prediction error is larger than the others—but absolute error does not have this effect: All sizes of error are treated evenly according to their magnitude.

Sometimes it is the *relative* rather than *absolute* error values that are of importance. For example, if a 10% error is equally important whether it is an error of 50 in a prediction of 500 or an error of 0.2 in a prediction of 2, then averages of absolute error will be meaningless—relative errors are appropriate. This effect would be taken into account by using the relative errors in the mean-squared error calculation or the mean absolute error calculation.

*Relative squared* error in Table 5.8 refers to something quite different. The error is made relative to what it would have been if a simple predictor had been used. The simple predictor in question is just the average of the actual values from the training data, denoted by $\overline{a}$. Thus, relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the default predictor. The root relative squared error is obtained in the obvious way.

The next error measure goes by the glorious name of *relative absolute error* and is just the total absolute error, with the same kind of normalization. In these three relative error measures, the errors are normalized by the error of the simple predictor that predicts average values.

The final measure in Table 5.8 is the correlation coefficient, which measures the statistical correlation between the *a*'s and the *p*'s. The correlation coefficient ranges from 1 for perfectly correlated results, through 0 when there is no correlation, to –1 when the results are perfectly correlated negatively. Of course, negative values should not occur for reasonable prediction methods. Correlation is slightly different from the other measures because it is scale independent in that, if you take a particular set of predictions, the error is unchanged if all the predictions are multiplied by a constant factor and the actual values are left unchanged. This factor appears in every term of $S_{PA}$ in the numerator and in every term of $S_P$ in the denominator, thus canceling out. (This is not true for the relative error figures, despite normalization: If you multiply all the predictions by a large constant, then the difference between the predicted and actual values will change dramatically, as will the percentage errors.) It is also different in that good performance leads to a large value of the correlation coefficient, whereas because the other methods measure error, good performance is indicated by small values.

Which of these measures is appropriate in any given situation is a matter that can only be determined by studying the application itself. What are we trying to minimize? What is the cost of different kinds of error? Often it is not easy to decide. The squared error measures and root-squared error measures weigh large discrepancies much more heavily than small ones, whereas the absolute error measures do not. Taking the square root (root mean-squared error) just reduces the figure to have the same dimensionality as the quantity being predicted. The relative error figures try to compensate for the basic predictability or unpredictability of the output variable: If it tends to lie fairly close to its average value, then you expect prediction to be good and the relative figure compensates for this. Otherwise, if the error figure in one situation is far greater than in another situation, it may be because the quantity in the first situation is inherently more variable and therefore harder to predict, not because the predictor is any worse.

Fortunately, it turns out that in most practical situations the best numerical prediction method is still the best no matter which error measure is used. For example, Table 5.9 shows the result of four different numeric prediction techniques on a given dataset, measured using cross-validation. Method D is the best according to all five metrics: It has the smallest value for each error measure and the largest correlation coefficient. Method C is the second best by all five metrics. The performance of A and B is open to dispute: They have the same correlation coefficient; A is better than B according to mean-squared and relative squared errors, and the reverse is true for absolute and relative absolute error. It is likely that the extra emphasis that the squaring operation gives to outliers accounts for the differences in this case.

When comparing two different learning schemes that involve numeric prediction, the methodology developed in Section 5.5 still applies. The only difference is that success rate is replaced by the appropriate performance measure (e.g., root mean-squared error) when performing the significance test.

**Table 5.9** Performance Measures for Four Numeric Prediction Models

| | A | B | C | D |
|---|---|---|---|---|
| Root mean-squared error | 67.8 | 91.7 | 63.3 | 57.4 |
| Mean absolute error | 41.3 | 38.5 | 33.4 | 29.2 |
| Root relative squared error | 42.2% | 57.2% | 39.4% | 35.8% |
| Relative absolute error | 43.1% | 40.1% | 34.8% | 30.4% |
| Correlation coefficient | 0.88 | 0.88 | 0.89 | 0.91 |

## 5.9 MINIMUM DESCRIPTION LENGTH PRINCIPLE

What is learned by a machine learning scheme is a kind of "theory" of the domain from which the examples are drawn, a theory that is predictive in that it is capable of generating new facts about the domain—in other words, the class of unseen instances. *Theory* is rather a grandiose term: We are using it here only in the sense of a predictive model. Thus, theories might comprise decision trees or sets of rules—they don't have to be any more "theoretical" than that.

There is a long-standing tradition in science that, other things being equal, simple theories are preferable to complex ones. This is known as *Occam's Razor* after the medieval philosopher William of Occam (or Ockham). Occam's Razor shaves philosophical hairs off a theory. The idea is that the best scientific theory is the smallest one that explains all the facts. As Einstein is reputed to have said, "Everything should be made as simple as possible, but no simpler." Of course, quite a lot is hidden in the phrase "other things being equal," and it can be hard to assess objectively whether a particular theory really does "explain" all the facts on which it is based—that's what controversy in science is all about.

In our case, in machine learning, most theories make errors. And if what is learned is a theory, then the errors it makes are like *exceptions* to the theory. One way to ensure that other things *are* equal is to insist that the information embodied in the exceptions is included as part of the theory when its "simplicity" is judged.

Imagine an imperfect theory for which there are a few exceptions. Not all the data is explained by the theory, but most is. What we do is simply adjoin the exceptions to the theory, specifying them explicitly as exceptions. This new theory is larger: That is a price that, quite justifiably, has to be paid for its inability to explain all the data. However, it may be that the simplicity—is it too much to call it *elegance*?—of the original theory is sufficient to outweigh the fact that it does not quite explain everything compared with a large, baroque theory that is more comprehensive and accurate.

For example, even though Kepler's three laws of planetary motion did not at the time account for the known data quite so well as Copernicus' latest refinement of the Ptolemaic theory of epicycles, they had the advantage of being far less complex, and that would have justified any slight apparent inaccuracy. Kepler was well aware of the benefits of having a theory that was compact, despite the fact that his theory violated his own aesthetic sense because it depended on "ovals" rather than pure circular motion. He expressed this in a forceful metaphor: "I have cleared the Augean stables of astronomy of cycles and spirals, and left behind me only a single cartload of dung."

The *minimum description length*, or MDL, principle takes the stance that the best theory for a body of data is one that minimizes the size of the theory plus the amount of information necessary to specify the exceptions relative to the theory—the smallest "cartload of dung." In statistical estimation theory, this has been applied successfully to various parameter-fitting problems. It applies to machine learning as follows: Given a set of instances, a learning scheme infers a theory—be it ever so simple;

unworthy, perhaps, to be called a "theory"—from them. Using a metaphor of communication, imagine that the instances are to be transmitted through a noiseless channel. Any similarity that is detected among them can be exploited to give a more compact coding. According to the MDL principle, the best theory is the one that minimizes the number of bits required to communicate the theory, along with the labels of the examples from which it was made.

Now the connection with the informational loss function introduced in Section 5.6 should be starting to emerge. That function measures the error in terms of the number of bits required to transmit the instances' class labels, given the probabilistic predictions made by the theory. According to the MDL principle, we need to add to this the "size" of the theory in bits, suitably encoded, to obtain an overall figure for complexity. However, the MDL principle refers to the information required to transmit the examples from which the theory was formed—that is, the *training* instances, not a test set. The overfitting problem is avoided because a complex theory that overfits will be penalized relative to a simple one by virtue of the fact that it takes more bits to encode. At one extreme is a very complex, highly overfitted theory that makes no errors on the training set. At the other is a very simple theory—the null theory—which does not help at all when transmitting the training set. And in between are theories of intermediate complexity, which make probabilistic predictions that are imperfect and need to be corrected by transmitting some information about the training set. The MDL principle provides a means of comparing all these possibilities on an equal footing to see which is the best. We have found the holy grail: an evaluation scheme that works on the training set alone and does not need a separate test set. But the devil is in the details, as we will see.

Suppose a learning scheme comes up with a theory $T$, based on a training set $E$ of examples, that requires a certain number of bits L[$T$] to encode, where L is for length. We are only interested in predicting class labels correctly, so we assume that $E$ stands for the collection of class labels in the training set. Given the theory, the training set itself can be encoded in a certain number of bits, L[$E \mid T$]. L[$E \mid T$] is in fact given by the informational loss function summed over all members of the training set. Then the total description length of theory plus training set is

$$\mathrm{L}[T] + \mathrm{L}[E|T]$$

and the MDL principle recommends choosing the theory $T$ that minimizes this sum.

There is a remarkable connection between the MDL principle and basic probability theory. Given a training set $E$, we seek the "most likely" theory $T$—that is, the theory for which the a posteriori probability Pr[$T \mid E$]—the probability after the examples have been seen—is maximized. Bayes' rule of conditional probability (the very same rule that we encountered in Section 4.2) dictates that

$$\Pr[T \mid E] = \frac{\Pr[E|T]\Pr[T]}{\Pr[E]}$$

Taking negative logarithms,

$$-\log \Pr[T \mid E] = -\log \Pr[E \mid T] - \log \Pr[T] + \log \Pr[E]$$

Maximizing the probability is the same as minimizing its negative logarithm. Now (as we saw in Section 5.6) the number of bits required to code something is just the negative logarithm of its probability. Furthermore, the final term, log Pr[*E*], depends solely on the training set and not on the learning method. Thus, choosing the theory that maximizes the probability Pr[*T* | *E*] is tantamount to choosing the theory that minimizes

$$\mathrm{L}[E \mid T] + \mathrm{L}[T]$$

In other words, the MDL principle!

This astonishing correspondence with the notion of maximizing the a posteriori probability of a theory after the training set has been taken into account gives credence to the MDL principle. But it also points out where the problems will sprout when the principle is applied in practice. The difficulty with applying Bayes' rule directly is in finding a suitable prior probability distribution Pr[*T*] for the theory. In the MDL formulation, that translates into finding how to code the theory *T* into bits in the most efficient way. There are many ways of coding things, and they all depend on presuppositions that must be shared by encoder and decoder. If you know in advance that the theory is going to take a certain form, you can use that information to encode it more efficiently. How are you going to actually encode *T*? The devil is in the details.

Encoding *E* with respect to *T* to obtain L[*E* | *T*] seems a little more straightforward: We have already met the informational loss function. But actually, when you encode one member of the training set after another, you are encoding a *sequence* rather than a *set*. It is not necessary to transmit the training set in any particular order, and it ought to be possible to use that fact to reduce the number of bits required. Often, this is simply approximated by subtracting log *n!* (where *n* is the number of elements in *E*), which is the number of bits needed to specify a particular permutation of the training set (and because this is the same for all theories, it doesn't actually affect the comparison between them). But one can imagine using the frequency of the individual errors to reduce the number of bits needed to code them. Of course, the more sophisticated the method that is used to code the errors, the less the need for a theory in the first place—so whether a theory is justified or not depends to some extent on how the errors are coded. The details, the details.

We end this section as we began, on a philosophical note. It is important to appreciate that Occam's Razor, the preference of simple theories over complex ones, has the status of a philosophical position or "axiom" rather than something that can be proven from first principles. While it may seem self-evident to us, this is a function of our education and the times we live in. A preference for simplicity is—or may be—culture specific rather than absolute.

The Greek philosopher Epicurus (who enjoyed good food and wine and supposedly advocated sensual pleasure—in moderation—as the highest good) expressed almost the opposite sentiment. His *principle of multiple explanations* advises that "If more than one theory is consistent with the data, keep them all" on the basis that if several explanations are equally in agreement, it may be possible to achieve a higher degree of precision by using them together—and, anyway, it would be unscientific to discard some arbitrarily. This brings to mind instance-based learning, in which all the evidence is retained to provide robust predictions, and resonates strongly with decision combination methods such as bagging and boosting (described in Chapter 8) that actually do gain predictive power by using multiple explanations together.

## 5.10 APPLYING THE MDL PRINCIPLE TO CLUSTERING

One of the nice things about the minimum description length principle is that, unlike other evaluation criteria, it can be applied under widely different circumstances. Although in some sense equivalent to Bayes' rule in that, as we have seen, devising a coding scheme for theories is tantamount to assigning them a prior probability distribution, schemes for coding are somehow far more tangible and easier to think about in concrete terms than intuitive prior probabilities. To illustrate this we will briefly describe—without entering into coding details—how you might go about applying the MDL principle to clustering.

Clustering seems intrinsically difficult to evaluate. Whereas classification or association learning has an objective criterion of success—predictions made on test cases are either right or wrong—this is not so with clustering. It seems that the only realistic evaluation is whether the result of learning—the clustering—proves useful in the application context. (It is worth pointing out that really this is the case for all types of learning, not just clustering.)

Despite this, clustering can be evaluated from a description-length perspective. Suppose a cluster-learning technique divides the training set $E$ into $k$ clusters. If these clusters are natural ones, it should be possible to use them to encode $E$ more efficiently. The best clustering will support the most efficient encoding.

One way of encoding the instances in $E$ with respect to a given clustering is to start by encoding the cluster centers—the average value of each attribute over all instances in the cluster. Then, for each instance in $E$, transmit which cluster it belongs to (in $\log_2 k$ bits) followed by its attribute values with respect to the cluster center—perhaps as the numeric difference of each attribute value from the center. Couched as it is in terms of averages and differences, this description presupposes numeric attributes and raises thorny questions of how to code numbers efficiently. Nominal attributes can be handled in a similar manner: For each cluster there is a probability distribution for the attribute values, and the distributions are different for different clusters. The coding issue becomes more straightforward: Attribute values are coded with respect to the relevant probability distribution, a standard operation in data compression.

If the data exhibits extremely strong clustering, this technique will result in a smaller description length than simply transmitting the elements of *E* without any clusters. However, if the clustering effect is not so strong, it will likely increase rather than decrease the description length. The overhead of transmitting cluster-specific distributions for attribute values will more than offset the advantage gained by encoding each training instance relative to the cluster it lies in. This is where more sophisticated coding techniques come in. Once the cluster centers have been communicated, it is possible to transmit cluster-specific probability distributions adaptively, in tandem with the relevant instances: The instances themselves help to define the probability distributions, and the probability distributions help to define the instances. We will not venture further into coding techniques here. The point is that the MDL formulation, properly applied, may be flexible enough to support the evaluation of clustering. But actually doing it satisfactorily in practice is not easy.

## 5.11 FURTHER READING

The statistical basis of confidence tests is well covered in most statistics texts, which also give tables of the normal distribution and Student's distribution. (We use an excellent course text by Wild and Seber (1995) that we recommend very strongly if you can get hold of it.) "Student" is the *nom de plume* of a statistician called William Gosset, who obtained a post as a chemist in the Guinness brewery in Dublin, Ireland, in 1899 and invented the *t*-test to handle small samples for quality control in brewing. The corrected resampled *t*-test was proposed by Nadeau and Bengio (2003). Cross-validation is a standard statistical technique, and its application in machine learning has been extensively investigated and compared with the bootstrap by Kohavi (1995a). The bootstrap technique itself is thoroughly covered by Efron and Tibshirani (1993).

The Kappa statistic was introduced by Cohen (1960). Ting (2002) has investigated a heuristic way of generalizing to the multiclass case the algorithm given in Section 5.7 to make two-class learning schemes cost sensitive. Lift charts are described by Berry and Linoff (1997). The use of ROC analysis in signal detection theory is covered by Egan (1975); this work has been extended for visualizing and analyzing the behavior of diagnostic systems (Swets, 1988) and is also used in medicine (Beck and Schultz, 1986). Provost and Fawcett (1997) brought the idea of ROC analysis to the attention of the machine learning and data mining community. Witten et al. (1999b) explain the use of recall and precision in information retrieval systems; the *F*-measure is described by van Rijsbergen (1979). Drummond and Holte (2000) introduced cost curves and investigated their properties.

The MDL principle was formulated by Rissanen (1985). Kepler's discovery of his economical three laws of planetary motion, and his doubts about them, are recounted by Koestler (1964).

Epicurus' principle of multiple explanations is mentioned by Li and Vityani (1992), quoting from Asmis (1984).

This page intentionally left blank