# Chapter 11
# Performance Evaluation

The previous chapters pretended that performance evaluation in machine learning is a fairly straightforward matter. All it takes is to apply the induced classifier to a set of examples whose classes are known, and then count the number of errors the classifier has made. In reality, things are not as simple. Error rate rarely paints the whole picture, and there are situations in which it can even be misleading. This is why the conscientious engineer wants to be acquainted with other criteria to assess the classifiers' performance. This knowledge will enable her to choose the one that is best in capturing the behavioral aspects of interest.

So much for the evaluation of classifiers. Somewhat different is the question is how to compare the suitability of alternative machine-learning techniques for induction in a given domain. Dividing the set of pre-classified examples randomly into two subsets (one for induction, the other for testing) may not be the best thing to do, especially if the training set is small; random division may then result in subsets that do not represent the given domain properly. To obtain more reliable results, repeated random runs are necessary.

This chapter addresses both issues, explaining alternative criteria to quantify the performance of classifiers, and then discussing some strategies commonly used in experimental evaluation of machine-learning algorithms. The question of statistical evaluation of the results is relegated to the next chapter.

## 11.1 Basic Performance Criteria

Let us begin with formal definitions of error rate and classification accuracy. After this, we will take a look at the consequences of the decision to refuse to classify an example if the evidence favoring the winning class is weak.



M. Kubat, *An Introduction to Machine Learning*,
DOI 10.1007/978-3-319-63913-0_11

**Table 11.1** The basic quantities used in the definitions of performance criteria

| | | Labels returned by the classifier | |
|---|---|---|---|
| | | `pos` | `neg` |
| True labels: | `pos` | $N_{TP}$ | $N_{FN}$ |
| | `neg` | $N_{FP}$ | $N_{TN}$ |

For instance, $N_{FP}$ is the number of *false positives*: negative examples misclassified by the classifier as positive

**Correct and Incorrect Classification** Let us first define four fundamental quantities that will be used throughout this chapter. When testing a classifier on an example whose real class is known, we can encounter only the following four different outcomes: (1) the example is positive and the classifier correctly recognizes it as such (*true positive*); (2) the example is negative and the classifier correctly recognizes it as such (*true negative*); (3) the example is positive, but the classifier labels it as negative (*false negative*); and (4) the example is negative, but the classifier labels it as positive (*false positive*).

When applying the classifier to an entire set of examples (whose real classes are known), each of these four outcomes will occur a different number of times—and these numbers are then employed in the performance criteria defined below. The symbols representing the four outcomes are summarized in Table 11.1. Specifically, $N_{TP}$ is the number of *true positives*, $N_{TN}$ is the number of *true negatives*, $N_{FP}$ is the number of *false positives*, and $N_{FN}$ is the number of *false negatives*. In the entire example set, $T$, only these four categories are possible; therefore, the size of the set, $|T|$, equals the sum, $|T| = N_{FP} + N_{FN} + N_{TP} + N_{TN}$.

Note that the number of correct classifications is the number of *true positives* plus the number of *true negatives*, $N_{TP} + N_{TN}$; and the number of errors is the number of *false positives* plus the number of *false negatives*, $N_{FP} + N_{FN}$.

**Error Rate and Classification Accuracy** A classifier's *error rate*, $E$, is the frequency of errors made by the classifier over a given set of examples. It is calculated by dividing the number of errors, $N_{FP} + N_{FN}$, by the total number of examples, $N_{TP} + N_{TN} + N_{FP} + N_{FN}$.

$$E = \frac{N_{FP} + N_{FN}}{N_{FP} + N_{FN} + N_{TP} + N_{TN}} \tag{11.1}$$

Sometimes, the engineer prefers to work with the opposite quantity, *classification accuracy*, $Acc$: the frequency of correct classifications made by the classifier over a given set of examples. Classification accuracy is calculated by dividing the number of correct classifications, $N_{TP} + N_{TN}$, by the total number of examples. Note that $Acc = 1 - E$.

$$Acc = \frac{N_{TP} + N_{TN}}{N_{FP} + N_{FN} + N_{TP} + N_{TN}} \tag{11.2}$$

**Rejecting an Example** When discussing the problem of optical character recognition, Sect. 8.1, suggested that the classifier should sometimes be allowed to refuse to classify an example if the evidence supporting the winning class is not strong enough. The motivation is quite simple: in some domains, the penalty for misclassification can be much higher than the penalty for not making any classification at all.

An illustrative example is not difficult to find. Thus the consequence of a classifier's refusal to return the precise value of the ZIP code is that the decision where the letter should be sent will have to be made by a human operator. To be sure, this manual processing is more expensive than automatic processing, but not excessively so. On the other hand, an incorrect value returned by the classifier results in the letter being sent to a wrong destination, which can cause a serious delay in delivery. This latter cost is often much higher than the cost of "manual" reading. Similarly, an incorrect medical diagnosis is often more expensive than no diagnosis at all; lack of knowledge can be remedied by additional tests, but a wrong diagnosis may result in choosing a treatment that does more harm than good.

This is why the classifier should sometimes refuse to classify an example if the evidence favoring either class is insufficient. In some machine-learning paradigms, the term *insufficient evidence* is easy to define. Suppose, for instance, that, in a 7-NN classifier, four neighbors favor the `pos` class, and the remaining three favor the `neg` class. The final count being four versus three, the situation seems "too close to call." More generally, the engineer may define a threshold for the minimum difference between the number of votes favoring the winning class and the number of votes favoring the runner-up class.

In bayesian classifiers, the technique is easily implemented, too. If the difference between the probabilities of the two most strongly supported classes falls short of a user-specified minimum, the example is rejected as too ambiguous to classify. Something similar can be done also in neural networks: compare the signals returned by the corresponding output neurons—and refuse to classify if there is no clear-cut winner.

In other classifiers, such as decision trees, implementation of the rejection mechanism is not so straightforward, and is only made possible by the implementation of "additional tricks."

**Advantages and Disadvantages of a Rejection to Classify** The classifier that occasionally refuses to make a decision about an example's class is of course less likely to go wrong. No wonder that its error rate will be lower. Indeed, the more examples are rejected, the lower the error rate. But the caution should not be exaggerated. It may look like a good thing that the error rate is reduced almost to zero. But if this low rate is achieved only thanks to the refusal to classify almost all examples, the classifier becomes impractical. Which of these two aspects (low error rate versus rare classifications) plays a more important role will depend on the concrete circumstances of the given application.

Figure 11.1 graphically illustrates the essence of the trade-off involved in decisions of this kind. The horizontal axis represents a parameter capable of

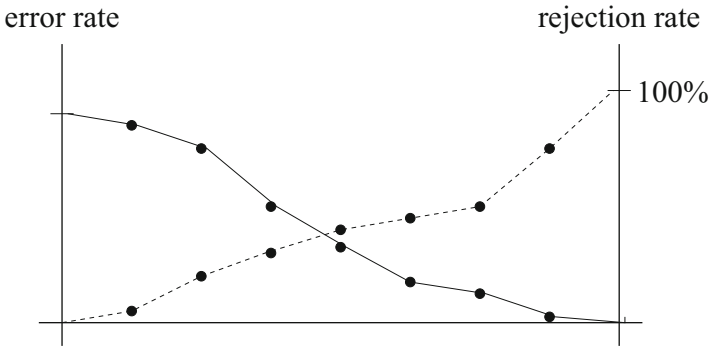


**Fig. 11.1** Error rate can be increased by allowing the classifier to refuse to classify an example if available evidence fails to "convince" it. At the extreme, the error rate drops to $E = 0$ because all examples are rejected

adjusting the rejection rate. As we move from left to right, the rejection rate increases, whereas the error rate goes down until, at the extreme, all examples are rejected. At this point, the zero error rate is a poor consolation for having a classifier that never does anything. The lesson is clear. Occasional rejection of unclear examples makes a lot of sense, but the principle should be handled with care.

## What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- Define the terms *false negative, false positive, true negative*, and *true positive*.
- Specify the formulas for classification accuracy and error rate. How are the two criteria mutually interrelated?
- When should a classifier reject an example (i.e., when should it refuse to classify)? How would you implement this behavior in diverse types of classifiers?
- How does rejection rate relate to error rate? What is the trade-off between the two? Under what circumstances can refusal to classify be useful, and under what circumstances does it hurt?

## 11.2 Precision and Recall

In some applications, negative examples outnumber positive ones by a wide margin. When this happens, error rate offers a misleading picture of classification performance. To see why, just consider the case where only 2% of all examples

are positive, and all the remaining 98% are negative. A "classifier" that returns the negative class for any example in the set will be correct 98% of the time—which may look like a remarkable feat. And yet, the reader will agree, a classifier that never recognizes a positive example is useless.

**Imbalanced Classes Revisited** This observation is worth keeping in mind because domains with imbalanced classes are quite common. We encountered some of them in Chaps. 8 and 10, and other applications can be found. Thus in automated information retrieval, the user may want to find a scientific document dealing with, say, "performance evaluation of classifiers." No matter how attractive the topic may appear to this particular person, papers dealing with it will represent only a small fraction of the millions of documents available in the digital library. Likewise, patients suffering from a specific medical disorder are in the entire population relatively rare. And the same goes for any undertaking that seeks to recognize a rare event such as a default on mortgage payments or a fraudulent use of a credit card. A seasoned engineer will go so far as to say that the majority of realistic applications are in some degree marked by the phenomenon of imbalanced classes.

In domains of this kind, error rate and classification accuracy will hardly tell us anything reasonable about the classifier's practical utility. Rather than averaging the performance over both (or all) classes, we need criteria that focus on a class which, while important, is represented by only a few examples. Let us take a quick look at some of them.

**Precision** By this we mean the percentage of true positives, $N_{TP}$, among all examples that the classifier has labeled as positive: $N_{TP} + N_{FP}$. The value is thus obtained by the following formula:

$$Pr = \frac{N_{TP}}{N_{TP} + N_{FP}} \tag{11.3}$$

Put another way, *precision* is the probability that the classifier is right when labeling an example as positive.

**Recall** By this we mean the probability that a positive example will be correctly recognized as such (by the classifier). The value is therefore obtained by dividing the number of true positives, $N_{TP}$, by the number of positives in the given set: $N_{TP}+N_{FN}$. Here is the formula:

$$Re = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{11.4}$$

Note that the last two formulas differ only in the denominator. This makes sense. Whereas *precision* is the frequency of true positives among all examples deemed positive by the classifier, *recall* is the frequency of the same true positives among all positive examples in the set.

**Table 11.2** Illustration of the two criteria: *precision* and *recall*

Suppose a classifier has been induced. Evaluation on a testing set gave the results summarized in this table:

| | | Labels returned by the classifier | |
|---|---|---|---|
| | | `pos` | `neg` |
| True labels: | `pos` | 20 | 50 |
| | `neg` | 30 | 900 |

From here, the following values of precision, recall, and accuracy are obtained:

$$\text{precision} = \frac{20}{50} = 0.40; \quad \text{recall} = \frac{20}{70} = 0.29; \quad \text{accuracy} = \frac{920}{1000} = 0.92$$

Suppose the classifier's parameters were modified with the intention to improve its behavior on positive examples. After the modification, evaluation on a testing set gave the results summarized in the table below.

| | | Labels returned by the classifier | |
|---|---|---|---|
| | | `pos` | `neg` |
| True labels: | `pos` | 30 | 70 |
| | `neg` | 20 | 880 |

From here, the following values of precision, recall, and accuracy are obtained.

$$\text{precision} = \frac{30}{50} = 0.60; \quad \text{recall} = \frac{30}{100} = 0.30; \quad \text{accuracy} = \frac{910}{1000} = 0.91$$

The reader can see that precision has considerably improved, while recall remained virtually unaffected. Note that classification accuracy has not improved, either.

**Illustration of the Two Criteria** Table 11.2 illustrates the behavior of the two criteria in a simple domain with an imbalanced representation of two classes, `pos` and `neg`. The induced classifier, while exhibiting an impressive classification accuracy, suffers from poor *precision* and *recall*. Specifically, *precision* of $Pr = 0.40$ means that of the 50 examples labeled as positive by the classifier, only 20 are indeed positive, the remaining 30 being nothing but false positives. With *recall*, things are even worse: out of the 70 positive examples in the testing set, only 20 are correctly identified as such by the classifier.

Suppose that the engineer decides to improve the situation by modifying some of the classifier's internal parameters, and suppose that this modification results in an increased number of true positives (from $N_{TP} = 20$ to $N_{TP} = 30$) and also a drop in the number of false positives (from $N_{FP} = 30$ to $N_{FP} = 20$). On the other hand, the number of false negatives has gone up, too: from $N_{FN} = 50$ to $N_{FN} = 70$. The calculations in Table 11.2 indicate that *recall* was thus barely affected, but *precision*

has improved, from $Pr = 0.40$ to $Pr = 0.60$. Classification accuracy remained virtually unchanged—actually, it has even gone down a bit, in spite of the improved *precision*.

**When High Precision Matters** In some domains, *precision* is more important than *recall*. For instance, when you purchase something from an e-commerce web site, their recommender system often reacts with a message to the effect that, "Customers who have bought X buy also Y." The obvious intention is to cajole you into buying Y as well.

Recommender systems are sometimes created with the help of machine learning techniques applied to the company's historical data.[1] When evaluating their performance, the engineer wants to achieve high *precision*. The customers better be happy about the recommended merchandise, or else they will ignore the recommendations in the future.

The value of *recall* is here unimportant. The list offered on the web site has to be of limited size, and so it does not matter much that the system identifies only a small percentage of all items that the customers may like.

**When High Recall Matters** In other domains, by contrast, *recall* is more important. This is often the case in medical diagnosis. A patient suffering from $X$, and properly diagnosed as such, represents a true positive. A patient suffering from $X$ but *not* diagnosed as such represents a false negative, something the doctor wants to avoid—which means that $N_{FN}$ should be small. In the definition of *recall*, $Re = \frac{N_{TP}}{N_{TP}+N_{FN}}$, the number of false negatives appears in the denominator; consequently, a small value of $N_{FN}$ implies a high value of *recall*.

**ROC Curves** In many classifiers, tweaking certain parameters can modify the values of $N_{FP}$ and $N_{FN}$, thus affecting (at least to some extent) the classifier's behavior, for instance, by improving *recall* at the cost of worsened *precision* or vice versa. This possibility can be useful in domains where the user has an idea as to which of these two quantities is more important.

The reader will find it easy to suggest various ways to do the "tweaking." Thus in the $k$-NN classifier, the engineer may request that an example be labeled as negative unless some very strong evidence supports the positive label. For instance, if four out of seven nearest neighbors are positive, the classifier can be instructed still to return the negative label (in spite of the small majority recommending the positive class). In this way, the number of false positives can be reduced, though this often means to the price in terms of an increased number of false negatives. Even stronger reduction in $N_{FP}$ (and an increase in $N_{FN}$) can be achieved by requesting that any example be deemed negative unless at least five (or six) of the seven nearest neighbors are positive.

[1] The concrete techniques employed to this end are somewhat more advanced than those discussed in this textbook, and are thus not treated here.

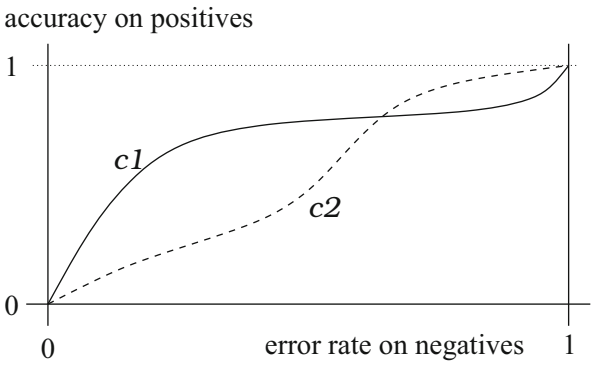


**Fig. 11.2** Example of ROC curves for two classifiers, $c1$ and $c2$. The parameters of the classifiers can be used to modify the numbers of false positives and false negatives

Something similar is easy to implement in Bayesian classifiers and in neural networks. The idea is the same: label the example with the preferred class unless strong evidence indicates that this decision is incorrect.

The behavior of the classifier under different values of its parameters can be visualized by the so-called ROC curve, a graph where the horizontal axis represents error rate on the negative examples, and the vertical axis represents classification accuracy on the positive examples. Example ROC curves for two classifiers, $c1$ and $c2$, are shown in Fig. 11.2. Ideally, we would like to reach the upper-left corner that represents zero error rate on the negatives and 100% accuracy on the positives. For various reasons, this is rarely possible.

An important question to ask is which of the two, $c1$ or $c2$ is better. This can only be answered based on the specific needs of the concrete application. All we can say by just looking at the graph is that $c1$ outperforms $c2$ on the positives in the region with low error rate on negatives. As the error rate on the negative examples increases, $c2$ outperforms $c1$ on the positive examples. Again, whether this is good or bad can only be decided by the user. See the comments concerning the question when to prefer *precision* and when *recall*.

## What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- In what kind of data would we rather evaluate the classifier's performance by the quantities known as *precision* and *recall* (instead of, say, error rate)?
- What formulas help us calculate the concrete values of these criteria? How do the two formulas differ?
- Under what circumstances do we prefer high *precision*, and under what circumstances do we place more emphasis on high *recall*?

- Explain the nature of an ROC curve. What extra information does the curve convey about a classifier's behavior? How does the ROC curve help the user in the choice between two alternative classifiers?

## 11.3 Other Ways to Measure Performance

Apart from error rate, classification accuracy, *precision*, and *recall*, other criteria are sometimes used, each reflecting a somewhat different aspect of the classifier's behavior. Let us take at least a cursory look at some of the most important ones.

**Combining *Precision* and *Recall* in $F_\beta$** Using two different (and sometimes contradictory) performance criteria can be awkward; it is in the human nature to want to quantify things by a single number. Especially in the case of *precision* and *recall*, attempts have been made somehow to combine the two in one quantity. The best-known such formula, $F_\beta$, is defined as follows:

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re} \tag{11.5}$$

The parameter, $\beta \in [0, \infty)$, enables the user to weigh the relative importance of the two criteria. If $\beta > 1$, then more weight is given to *recall*. If $\beta < 1$, then more weight is apportioned to *precision*. It would be easy to show that $F_\beta$ converges to *recall* when $\beta \to \infty$, and to *precision* when $\beta = 0$.

Quite often, the engineer does not really know which of the two, *precision* or *recall*, is more important, and by how much. In that event, she prefers to work with the neutral value of the parameter, $\beta = 1$:

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{11.6}$$

**A Numeric Example** Suppose that an evaluation of a classifier on a testing set resulted in the values summarized in the upper part of Table 11.2. For these, the table established the values of *precision* and *recall*, respectively, as $Pr = 0.40$ and $Re = 0.29$. Using these numbers, we will calculate $F_\beta$ for the following concrete settings of the parameter: $\beta = 0.2$, $\beta = 1$, and $\beta = 5$.

$$F_{0.2} = \frac{(0.2^2 + 1) \times 0.4 \times 0.29}{0.2^2 \times 0.4 + 0.29} = \frac{0.121}{0.306} = 0.39$$

$$F_1 = \frac{(1^2 + 1) \times 0.4 \times 0.29}{0.4 + 0.29} = \frac{0.232}{0.330} = 0.70$$

$$F_5 = \frac{(5^2 + 1) \times 0.4 \times 0.29}{5^2 \times 0.4 + 0.29} = \frac{3.02}{10.29} = 0.29$$

**Sensitivity and Specificity** The choice of the concrete criterion is often influenced by the given application field—with its specific needs and deep-rooted traditions that should not be ignored. Thus the medical domain has become accustomed to assessing performance of their "classifiers" (not necessarily developed by machine learning) by *sensitivity* and *specificity*. In essence, these quantities are nothing but *recall* measured on the positive and negative examples, respectively. Let us be more concrete:

*Sensitivity* is *recall* measured on positive examples:

$$Se = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{11.7}$$

*Specificity* is *recall* measured on negative examples:

$$Sp = \frac{N_{TN}}{N_{TN} + N_{FP}} \tag{11.8}$$

In the machine-learning literature, evaluation in terms of *sensitivity* and *specificity* is rare, but it is still good to be aware of these two criteria. After all, we may be asked to use them when applying machine learning to medical data—and quite often it is the customer who is the ultimate boss.

**Gmean** When inducing a classifier in a domain with imbalanced class representation, the engineer sometimes wants to achieve similar performance on both classes, `pos` and `neg`. In this event, the geometric mean, `gmean`, of the two classification accuracies (on the positive examples and on the negative examples) is used:

$$\texttt{gmean} = \sqrt{acc_{\texttt{pos}} \times acc_{\texttt{neg}}} = \sqrt{\frac{N_{TP}}{N_{TP} + N_{FN}} \times \frac{N_{TN}}{N_{TN} + N_{FP}}} \tag{11.9}$$

Note that *gmean* is actually the square root of the product of two numbers: *recall* on positive examples and *recall* on negative examples—or, in other words, the product of *sensitivity* and *specificity*.

Perhaps the most important aspect of *gmean* is that it depends not only on the concrete values of the two terms under the square root symbol, $acc_{\texttt{pos}}$ and $acc_{\texttt{neg}}$, but also on how close the two values are to each other. A simple numeric example will convince us that this is indeed the case.

Thus the arithmetic average of 0.75 and 0.75 is $(0.75 + 0.75)/2 = 0/75$; also the arithmetic average of 0.55 and 0.95 is $(0.55 + 0.95)/2 = 0.75$. However, the geometric means of the first pair is $\sqrt{0.75 \times 0.75} = 0.75$ whereas the geometric means of the second pair is $\sqrt{0.55 \times 0.95} = 0.72$, a smaller number. We can see that the *gmean* is indeed smaller when the two numbers are different; and the more different they are, the lower the value of *gmean*.

**Another Numeric Example** Again, suppose that the evaluation of a classifier on a testing set resulted in the values summarized in the upper part of Table 11.2. The values of *sensitivity*, *specificity*, and *gmean* are calculated as follows:

$$Se = \frac{20}{50 + 20} = 0.29$$

$$Sp = \frac{900}{900 + 30} = 0.97$$

$$\texttt{gmean} = \sqrt{\frac{20}{50 + 20} \times \frac{900}{900 + 30}} = \sqrt{0.29 \times 0.97} = 0.53$$

**Cost Functions** In the world of machine-learning applications, not all errors carry the same penalty. False positives may be more costly than false negatives or vice versa. Comparisons of alternative classifiers are further complicated by the fact that, in some domains, the costs of the two types of error cannot even be expressed in the same (or at least comparable) units.

Consider the oil spill-recognition problem discussed in Chap. 8. A false positive here means that a "lookalike" is incorrectly taken for an oil spill (false positive). When this happens, a plane is unnecessarily sent to the given location to verify the case. The costs incurred by this error are those associated with the flight. By contrast, a false negative means that a potential environmental hazard has gone undetected, something difficult to express in monetary terms.

## What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- Explain how $F_\beta$ combines *precision* and *recall*. Write down the formula and discuss how different values of $\beta$ give more weight either to *precision* or *recall*. Which value of $\beta$ gives equal weight to both?
- Give the formulas for the calculation of *sensitivity* and *specificity*. Explain their nature, and show their relation to *recall*. When are the two criteria used?
- Give the formula for the calculation of *gmean*. Explain the nature of this quantity, and show its relation to *recall*. When is this criterion used?
- Under what circumstances can the costs associated with false positives be different from the costs associated with false negatives? Suggest a situation where the comparison of the costs is almost impossible.

## 11.4 Learning Curves and Computational Costs

The first four sections of this chapter dealt with the problem of performance evaluation of the induced classifiers. Let us now turn our attention to the evaluation of the learning algorithm itself. How efficient is the given induction technique computationally? And how good are the classifiers it induces? Will better results be achieved if we choose some other machine-learning framework?

In this section, we will give some thought to the costs of learning—in terms of the number of examples needed for successful induction, as well as in terms of the computational time consumed. The other aspect, the ability to induce a tool with high classification performance will be addressed in the following section.

**The Learning Curve** When evaluating a human subject's ability to learn how to solve a certain problem, psychologists rely on a *learning curve*, a notion that machine learning has borrowed for its own purposes.

From our perspective, the learning curve simply shows how the classification performance of the induced classifier depends on the size of the training set. Two such curves are shown in Fig. 11.3. The horizontal axis represents the number of training examples; and the vertical axis represents the classification accuracy of the classifier induced from these examples. Usually, though not always, this classification accuracy is evaluated on independent testing examples.

Most of the time, a larger training set means higher classification performance—at least until the moment when no further improvement is possible. Ideally, we would like to achieve maximum performance from the smallest possible training set. For one thing, training examples can be expensive to obtain, and their source can be limited no matter how much we are willing to spend on them. For another, the more examples we use, the higher the computational costs on induction.

**Comparing Learners with Different Learning Curves** Figure 11.3 shows the learning curves of two learners, $l_1$ and $l_2$. The reader can see that the learning curve of the former, $l_1$, rises very quickly, only to level off at a point beyond which virtually no improvement is possible—the limitation may be imposed by an

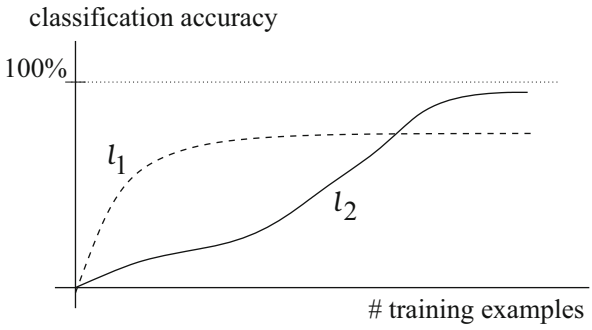


**Fig. 11.3** A learning curve shows how the achieved classification accuracy depends on the number of examples used during learning

incorrect bias (see Sect. 10.1). By contrast, the learning curve of the second learner, $l_2$, does not grow so fast, but in the end achieves higher levels of accuracy than $l_1$.

Which of the two curves indicates a preferable learner depends on the circumstances of the given application. When the source of the training examples is limited, the first learner is clearly more appropriate. If the examples are abundant, we will prefer the other learner, assuming of course that the computational costs are not prohibitive.

**Computational Costs** There are two aspects to computational costs. First is the time needed for the induction of the classifier from the available data. The second is the time it takes to classify a set of examples with the classifier thus induced.

Along these lines, the techniques described in this book cover a fairly broad spectrum. As for induction costs, the cheapest is the basic version of the $k$-NN classifier: the only "computation" involved is to store the training examples.[2] On the other hand, the $k$-NN classifier is expensive in terms of the classification costs. For instance, if we have a million training examples, each described by ten thousand attributes, then tens of billions of arithmetic operations will have to be carried out to classify a single example. When asked to classify millions of examples, even a very fast computer will take quite some time.

The situation is different in the case of decision trees. These are cheap when used to classify examples: usually only a moderate number of single-attribute tests are needed. However, induction of decision trees can take a lot of time if many training examples described by many attributes are available.

Induction costs and classification costs of the other classifiers vary, and the engineer is well advised to consider these costs when choosing the most appropriate machine-learning paradigm for the given application. Just as important is a solid understanding of how these costs depend on the number of the training examples, on the number of attributes describing them, and sometimes also on the required accuracy (e.g., in the case of neural networks).

## What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- What are the two main aspects of costs associated with a given machine-learning paradigm?
- What does the learning curve tell us about a machine learning algorithm's behavior? What shape of the learning curve do we want in the ideal case? What should we expect in reality?
- Under what circumstances is a steeper learning curve with lower maximum better than a flatter curve with a higher maximum?

[2] Some computation will be necessary if we decide to remove noisy or redundant examples.

## 11.5 Methodologies of Experimental Evaluation

The reader understands that different domains will benefit from different induction techniques. The choice is usually not difficult to make. Some knowledge of the available training data often helps us choose the most appropriate paradigm; for instance, if a high percentage of the attributes are suspected to be irrelevant, then decision trees are likely to be more successful than a nearest-neighbor classifier.

However, the success of a given technique also depends on the values of various parameters. Although even here certain time-tested rules of thumb can help, the best parameter setting is usually found only by experimentation.

**Baseline Approach and Its Limitations** The basic scenario is simple. The set of pre-classified examples is divided into two subsets, one used for training, the other for testing. The training-testing session is repeated for different parameter settings, and the one that results in the highest performance is then chosen.

This, however, can only be done if a great many pre-classified examples are available. In domains where examples are scarce, or expensive to obtain, a random division into a pair of the training and testing sets will lack objectivity. Either of the two sets can, by mere chance, misrepresent the given domain adequately. Statisticians are telling us that both the training set and the testing set should have more or less the same distribution of examples. In small sets, of course, this cannot be guaranteed.

**Random Subsampling** When the set of pre-classified examples is small, the engineer usually prefers to repeat the training-testing procedure several times. In each run, the set of examples is randomly divided into two parts, one for training, the other for testing. The measured performances are recorded, and then averaged. Care has to be taken that the individual data-splits are mutually independent.

Once the procedure has been repeated $N$ times (typically, $N = 10$ or $N = 5$), the results are reported in terms of the average classification accuracy and the standard deviation, say, $84.2 \pm 0.6$. For the calculation of averages and standard deviations, the formulas from Chap. 2 are used: the average, $\mu$, is obtained by Eq. (2.12); the standard deviation, $\sigma$, is obtained as the square root of variance, $\sigma^2$, which is calculated using Eq. (2.13).

***N*-Fold Crossvalidation** For more advanced statistical evaluation, experienced experimenters often prefer the so-called $N$-fold crossvalidation. The principle is shown in Fig. 11.4. To begin with, the set of pre-classified examples is divided into $N$ equally sized (or almost equally-sized) subsets which the machine-learning jargon sometimes (not quite correctly) refers to as "folds."

$N$-fold crossvalidation then runs $N$ experiments. In each, one of the $N$ subsets is removed so as to be used only for testing (this guarantees that, in each run, a different testing set is used). The training is then carried out on the union of the remaining $N-1$ subsets. Again, the results are averaged, and the standard deviation calculated.

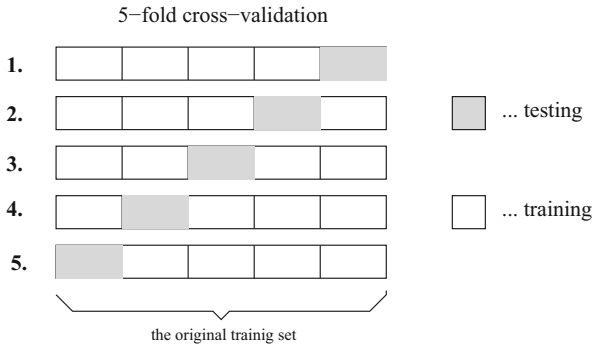


**Fig. 11.4** *N*-fold cross-validation divides the training set into *N* equally sized subsets. In each of the *N* experimental runs, it withholds a different subset for testing, inducing the classifier from the union of the remaining $N - 1$ subsets

The advantage of *N*-fold crossvalidation as compared to random subsampling is that the testing sets are disjoint (non-overlapping), which is deemed advantageous for certain types of statistical evaluations (see Chap. 12).

**Stratified Approaches** Suppose you are dealing with a domain with 60 positive and 940 negative examples. If you rely on *N*-fold crossvalidation with $N = 10$, each of the "folds" is likely to contain a different number of positive examples. On average, there will be six positives in each fold, but the concrete numbers can vary significantly. In a domain where one of the classes is relatively rare, it can even happen that some of the "folds" will not contain any single positive example at all. The situation is typically encountered in domains with heavily imbalanced class representation.

In this event, the experienced experimenter prefers a so-called *stratified* approach to *N*-fold crossvalidation. The idea is to make sure that each of the *N* "folds" contains (approximately) the same representation of the examples from the individual classes. For instance, when using the fivefold crossvalidation in a domain with 60 and 940 positive and negative examples, respectively, each fold should consist of 200 examples of which 12 are positive.

The same principle is often used in the random-subsampling approach—which, admittedly, is in its stratified version no longer totally random. Again, the idea is to make sure that each training set, and each testing set, has about the same representation of each class.

**5 × 2 Crossvalidation (5 × 2cv)** There is yet another approach to experimental evaluation of machine-learning techniques, the so-called $5 \times 2$ crossvalidation, sometimes abbreviated as $5 \times 2$cv. This may actually be the most popular methodology in machine learning. The principle is built around a combination of random subsampling and twofold crossvalidation.

To be more specific, $5 \times 2$cv divides the set of pre-classified examples into two equally sized parts, $T_1$ and $T_2$. Next, it uses $T_1$ for training and $T_2$ for testing, and then the other way round: $T_2$ for training and $T_1$ for testing. The procedure is then repeated five times, each time with a different random division into two subsets. All

**Table 11.3** The algorithm for 5 × 2 cross-validation (5 × 2 CV)

Let $T$ be the original set of pre-classified examples.

1. Divide $T$ randomly into two equally-sized subsets. Repeat the division five times. The result is five pairs of subsets denoted as $T_{i1}$ and $T_{i2}$ (for $i = 1, \ldots 5$).
2. For each of these pairs, use $T_{i1}$ for training and $T_{i2}$ for testing, and then the other way round.
3. For the ten training/testing sessions thus obtained, calculate the mean value and the standard deviation of the chosen performance criterion.

in all, ten learning/testing sessions are thus created. The principle is summarized by the pseudocode in Table 11.3.

Again, many experimenters prefer to work with the stratified version of this methodology, making sure that the representation of the individual classes is about the same in each of the ten parts used in the experiments.

**The No-Free-Lunch Theorem** It would be foolish to expect some machine-learning technique to be a holy grail, a mechanism to be preferred under all circumstances. Nothing like this exists. The reader by now understands that each paradigm has its advantages that make it succeed in some domains—and shortcomings that make it fail miserably in others. Only systematic experiments can tell the engineer which type of classifier, and which induction algorithm, to select for the task at hand. The truth of the matter is that no machine-learning approach will outperform all other machine-learning approaches under all circumstances.

Mathematicians have been able to prove the validity of this statement by a rigorous proof. The result is known under the (somewhat fancy) name of "no-free-lunch theorem."

## What Have You Learned?

To make sure you understand this topic, try to answer the following questions. If you have problems, return to the corresponding place in the preceding text.

- What is the difference between $N$-fold cross-validation and random subsampling? Why do we sometimes prefer to employ the stratified versions of these methodologies?
- Explain the principle of the 5×2 cross-validation (5×2 cv), including its stratified version.
- What does the so-called no-free-lunch theorem tell us?

## 11.6 Summary and Historical Remarks

- The basic criterion to measure classification performance is error rate, $E$, defined as the percentage of misclassified examples in the given set. The complementary quantity is classification accuracy, $Acc = 1 - E$.
- When the evidence for any class is not sufficiently strong, the classifier should better reject the example to avoid the danger of a costly misclassification. Rejection rate then becomes yet another important criterion for the evaluation of classification performance. Higher rejection rate usually means lower error rate; beyond a certain point, however, the classifier's utility will degrade.
- Criteria for measuring classification performance can be defined by the counts (denoted as $N_{TP}, N_{TN}, N_{FP}, N_{FN},$) of true positives, true negatives, false positives, and false negatives, respectively.
- In domains with imbalanced class representation, error rate can be a misleading criterion. A better picture is offered by the use of *precision* ($Pr = \frac{N_{TP}}{N_{TP}+N_{FP}}$) and *recall* ($Re = \frac{N_{TP}}{N_{TP}+N_{FN}}$).
- Sometimes, *precision* and *recall* are combined in a single criterion, $F_\beta$, that is defined by the following formula:

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re}$$

  The value of the user-set parameter $\beta$ determines the relative importance of *precision* ($\beta < 1$) or *recall* ($\beta > 1$). When the two are deemed equally important, we use $\beta = 1$, obtaining the following:

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$$

- Less common criteria for classification performance include *sensitivity*, *specificity*, and *gmean*.
- In domains where an example can belong to more than one class at the same time, the performance is often evaluated by an average taken over the performances measured along the individual classes. Two alternative methods of averaging are used: *micro-averaging* and *macro-averaging*.
- Another important aspect of a machine-learning technique is how many training examples are needed if a certain classification performance is to be reached. The situation is sometimes visualized by means of a *learning curve*. Also worth the engineer's attention are the computational costs associated with induction and with classification.
- When comparing alternative machine-learning techniques in domains with limited numbers of pre-classified examples, engineers rely on methodologies known as random subsampling, $N$-fold cross-validation, and the $5 \times 2$ cross-validation. The stratified versions of these techniques make sure that each training set (and testing set) has the same proportion of examples for each class.

**Historical Remarks** Most of the performance criteria discussed in this chapter are well established in the statistical literature, and have been used for such a long time that it is difficult to trace their origin. The exception is the relatively recent *gmean* that was proposed to this end by Kubat et al. [51].

The idea to refuse to classify examples where the $k$-NN classifier cannot rely on a significant majority was put forward by Hellman [36] and later analyzed by Louizou and Maybank [56]. The principle of $5 \times 2$ cross-validation was suggested, and experimentally explored, by Dietterich [22]. The no-free-lunch theorem was published by Wolpert [100].

## 11.7 Solidify Your Knowledge

The exercises are to solidify the acquired knowledge. The suggested thought experiments will help the reader see this chapter's ideas in a different light and provoke independent thinking. Computer assignments will force the readers to pay attention to seemingly insignificant details they might otherwise overlook.

## Exercises

1. Suppose that the evaluation of a classifier on a testing set resulted in the counts summarized in the following table:

| | | Labels returned by the classifier | |
|---|---|---|---|
| | | `pos` | `neg` |
| True labels: | `pos` | 50 | 50 |
| | `neg` | 40 | 850 |

   Calculate the values of *precision, recall, sensitivity, specificity*, and *gmean*.
2. Using the data from the previous question, calculate $F_\beta$ for different values of the parameter: $\beta = 0.5$, $\beta = 1$, and $\beta = 2$.
3. Suppose that an evaluation of a machine-learning technique using fivefold cross-validation resulted in the following error rates measured in the testing sets:
   $E_{11} = 0.14, E_{12} = 0.16, E_{13} = 0.10, E_{14} = 0.15, E_{15} = 0.18$
   $E_{21} = 0.17, E_{22} = 0.15, E_{23} = 0.12, E_{24} = 0.13, E_{25} = 0.20$
   Calculate the mean value of the error rate as well as the standard deviation, $\sigma$, using the formulas from Chap. 2 (do not forget that standard deviation is the square root of variance, $\sigma^2$).

## Give It Some Thought

1. Suggest a domain where *precision* is much more important than *recall*; conversely, suggest a domain where it is the other way round, *recall* being more important than *precision*.

   (Of course, use different examples than those mentioned in this chapter.)
2. What aspects of the given domain is reflected in the pair, *sensitivity* and *specificity*? Suggest circumstances under which these two give a better picture of the classifier's performance than *precision* and *recall*.
3. Suppose that, for a given domain, you have induced two classifiers: one with very high *precision*, the other with high *recall*. What can be gained from the combination of the two classifiers? How would you implement this combination? Under what circumstances will the idea fail?
4. Try to think about the potential advantages and shortcomings of random subsampling in comparison with $N$-fold crossvalidation.

## Computer Assignments

1. Assume that some machine-learning experiment resulted in a table where each row represents a testing example. The first column contains the examples' class labels ("1" or "0" for the positive and negative examples, respectively), and the second column contains the labels suggested by the induced classifier.

   Write a program that calculates *precision*, *recall*, as well as $F_\beta$ for a user-specified $\beta$.

   Write a program that calculates the values of the other performance criteria.
2. Suppose that the training set has the form of a matrix where each row represents an example, each column represents an attribute, and the rightmost column contains the class labels.

   Write a program that divides this set randomly into five pairs of equally sized subsets, as required by the $5 \times 2$ cross-validation technique. Then write another program that creates the subsets in the *stratified* manner where each subset has approximately the same representation of each class.
3. Write a program that accepts two inputs: (1) a set of class labels of multi-label testing examples, and (2) the labels assigned to these examples by a multi-label classifier. The output consists of *micro*-averaged and *macro*-averaged values of *precision* and *recall*.
4. Write a computer program that accepts as input a training set, and outputs $N$ subsets to be used in $N$-fold crossvalidation. Make sure the approach is *stratified*. How will your program have to be modified if you later decide to use the $5 \times 2$ crossvalidation instead of the plain $N$-fold crossvalidation?