7 Estimating the Predictive Accuracy of a Classifier

7.1 Introduction

Any algorithm which assigns a classification to unseen instances is called a *classifier*. A decision tree of the kind described in earlier chapters is one very popular type of classifier, but there are several others, some of which are described elsewhere in this book.

This chapter is concerned with estimating the performance of a classifier of *any* kind but will be illustrated using decision trees generated with attribute selection using information gain, as described in Chapter 5.

Although the data compression referred to in Chapter 4 can sometimes be important, in practice the principal reason for generating a classifier is to enable unseen instances to be classified. However we have already seen that many different classifiers can be generated from a given dataset. Each one is likely to perform differently on a set of unseen instances.

The most obvious criterion to use for estimating the performance of a classifier is *predictive accuracy*, i.e. the proportion of a set of unseen instances that it correctly classifies. This is often seen as the most important criterion but other criteria are also important, for example algorithmic complexity, efficient use of machine resources and comprehensibility.

For most domains of interest the number of possible unseen instances is potentially very large (e.g. all those who might develop an illness, the weather for every possible day in the future or all the possible objects that might appear

M. Bramer, *Principles of Data Mining*, Undergraduate Topics
in Computer Science, DOI 10.1007/978-1-4471-7307-6_7

on a radar display), so it is not possible ever to establish the predictive accuracy beyond dispute. Instead, it is usual to *estimate* the predictive accuracy of a classifier by measuring its accuracy for a sample of data not used when it was generated. There are three main strategies commonly used for this: dividing the data into a training set and a test set, *k-fold cross-validation* and *N-fold* (or *leave-one-out*) *cross-validation*.

7.2 Method 1: Separate Training and Test Sets

For the 'train and test' method the available data is split into two parts called a *training set* and a *test set* (Figure 7.1). First, the training set is used to construct a classifier (decision tree, neural net etc.). The classifier is then used to predict the classification for the instances in the test set. If the test set contains N instances of which C are correctly classified the *predictive accuracy* of the classifier for the test set is $p = C/N$. This can be used as an estimate of its performance on any unseen dataset.

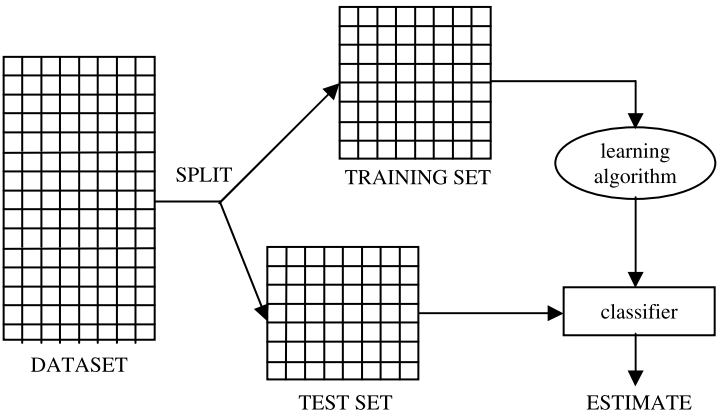

Figure 7.1 Train and Test

NOTE. For some datasets in the UCI Repository (and elsewhere) the data is provided as two separate files, designated as the training set and the test set. In such cases we will consider the two files together as comprising the 'dataset' for that application. In cases where the dataset is only a single file we need to divide it into a training set and a test set before using Method 1. This may be done in many ways, but a random division into two parts in proportions such as 1:1, 2:1, 70:30 or 60:40 would be customary.

7.2.1 Standard Error

It is important to bear in mind that the overall aim is not (just) to classify the instances in the test set but to estimate the predictive accuracy of the classifier for all possible unseen instances, which will generally be many times the number of instances contained in the test set.

If the predictive accuracy calculated for the test set is p and we go on to use the classifier to classify the instances in a different test set, it is very likely that a different value for predictive accuracy would be obtained. All that we can say is that p is an *estimate* of the true predictive accuracy of the classifier for all possible unseen instances.

We cannot determine the true value without collecting all the instances and running the classifier on them, which is usually an impossible task. Instead, we can use statistical methods to find a range of values within which the true value of the predictive accuracy lies, with a given probability or 'confidence level'.

To do this we use the *standard error* associated with the estimated value p. If p is calculated using a test set of N instances the value of its standard error is $\sqrt{p(1-p)/N}$. (The proof of this is outside the scope of this book, but can readily be found in many statistics textbooks.)

The significance of standard error is that it enables us to say that with a specified probability (which we can choose) the true predictive accuracy of the classifier is within so many standard errors above or below the estimated value p. The more certain we wish to be, the greater the number of standard errors. The probability is called the *confidence level*, denoted by CL and the number of standard errors is usually written as Z_{CL}.

Figure 7.2 shows the relationship between commonly used values of CL and Z_{CL}.

Confidence Level (CL)	0.9	0.95	0.99
Z_{CL}	1.64	1.96	2.58

Figure 7.2 Values of Z_{CL} for Certain Confidence Levels

If the predictive accuracy for a test set is p, with standard error S, then using this table we can say that with probability CL (or with a confidence level CL) the true predictive accuracy lies in the interval $p \pm Z_{CL} \times S$.

Example

If the classifications of 80 instances out of a test set of 100 instances were predicted accurately, the predictive accuracy on the test set would be 80/100 = 0.8. The standard error would be $\sqrt{0.8 \times 0.2/100} = \sqrt{0.0016} = 0.04$. We can say that with probability 0.95 the true predictive accuracy lies in the interval $0.8 \pm 1.96 \times 0.04$, i.e. between 0.7216 and 0.8784 (to four decimal places).

Instead of a predictive accuracy of 0.8 (or 80%) we often refer to an *error rate* of 0.2 (or 20%). The standard error for the error rate is the same as that for predictive accuracy.

The value of CL to use when estimating predictive accuracy is a matter of choice, although it is usual to choose a value of at least 0.9. The predictive accuracy of a classifier is often quoted in technical papers as just $p \pm \sqrt{p(1-p)/N}$ without any multiplier Z_{CL}.

7.2.2 Repeated Train and Test

Here the classifier is used to classify k test sets, not just one. If all the test sets are of the same size, N, the predictive accuracy values obtained for the k test sets are then averaged to produce an overall estimate p.

As the total number of instances in the test sets is kN, the standard error of the estimate p is $\sqrt{p(1-p)/kN}$.

If the test sets are not all of the same size the calculations are slightly more complicated.

If there are N_i instances in the ith test set ($1 \leq i \leq k$) and the predictive accuracy calculated for the ith test set is p_i the overall predictive accuracy p is $\sum_{i=1}^{i=k} p_i N_i / T$ where $\sum_{i=1}^{i=k} N_i = T$, i.e. p is the weighted average of the p_i values. The standard error is $\sqrt{p(1-p)/T}$.

7.3 Method 2: *k*-fold Cross-validation

An alternative approach to 'train and test' that is often adopted when the number of instances is small (and which many prefer to use regardless of size) is known as *k-fold cross-validation* (Figure 7.3).

If the dataset comprises N instances, these are divided into k equal parts, k typically being a small number such as 5 or 10. (If N is not exactly divisible by k, the final part will have fewer instances than the other $k-1$ parts.) A

series of k runs is now carried out. Each of the k parts in turn is used as a test set and the other $k-1$ parts are used as a training set.

The total number of instances correctly classified (in all k runs combined) is divided by the total number of instances N to give an overall level of predictive accuracy p, with standard error $\sqrt{p(1-p)/N}$.

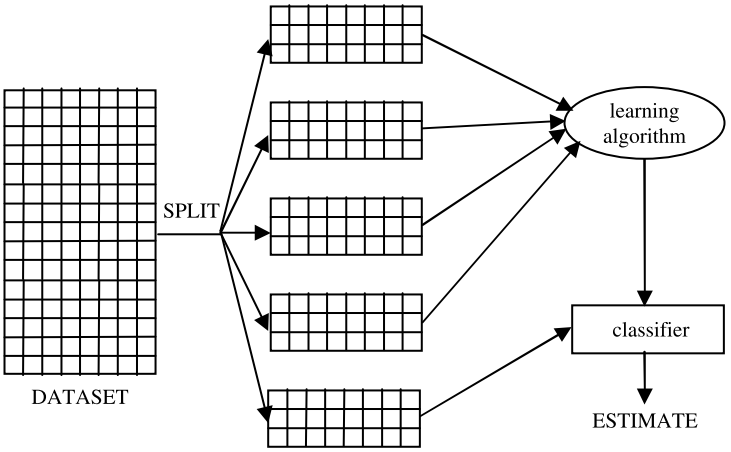

Figure 7.3 k-fold Cross-validation

7.4 Method 3: *N*-fold Cross-validation

N-fold cross-validation is an extreme case of k-fold cross-validation, often known as 'leave-one-out' cross-validation or jack-knifing, where the dataset is divided into as many parts as there are instances, each instance effectively forming a test set of one.

N classifiers are generated, each from $N-1$ instances, and each is used to classify a single test instance. The predictive accuracy p is the total number correctly classified divided by the total number of instances. The standard error is $\sqrt{p(1-p)/N}$.

The large amount of computation involved makes N-fold cross-validation unsuitable for use with large datasets. For other datasets, it is not clear whether any gain in the accuracy of the estimates produced by using N-fold cross-validation justifies the additional computation involved. In practice, the method

is most likely to be of benefit with very small datasets where as much data as possible needs to be used to train the classifier.

7.5 Experimental Results I

In this section we look at experiments to estimate the predictive accuracy of classifiers generated for four datasets.

All the results in this section were obtained using the TDIDT tree induction algorithm, with information gain used for attribute selection.

Basic information about the datasets is given in Figure 7.4 below. Further information about these and most of the other datasets mentioned in this book is given in Appendix B.

Dataset	Description	classes	attributes+		instances	
			categ	cts	training set	test set
vote	Voting in US Congress in 1984	2	16		300	135
pima-indians	Prevalence of Diabetes in Pima Indian Women	2		8	768	
chess	Chess Endgame	2	7		647	
glass	Glass Identification	7		9*	214	

+ categ: categorical; cts: continuous
∗ plus one 'ignore' attribute

Figure 7.4 Four Datasets

The *vote*, *pima-indians* and *glass* datasets are all taken from the UCI Repository. The *chess* dataset was constructed for a well-known series of machine learning experiments [1].

The *vote* dataset has separate training and test sets. The other three datasets were first divided into two parts, with every third instance placed in the test set and the other two placed in the training set in both cases.

The result for the *vote* dataset illustrates the point that TDIDT (along with some but not all other classification algorithms) is sometimes unable to classify an unseen instance (Figure 7.5). The reason for this was discussed in Section 6.7.

Dataset	Test set (instances)	Correctly classified	Incorrectly classified	Unclassified
vote	135	126 (93% ± 2%)	7	2
pima-indians	256	191 (75% ± 3%)	65	
chess	215	214 (99.5% ± 0.5%)	1	
glass	71	50 (70% ± 5%)	21	

Figure 7.5 Train and Test Results for Four Datasets

Unclassified instances can be dealt with by giving the classifier a 'default strategy', such as always allocating them to the largest class, and that will be the approach followed for the remainder of this chapter. It could be argued that it might be better to leave unclassified instances as they are, rather than risk introducing errors by assigning them to a specific class or classes. In practice the number of unclassified instances is generally small and how they are handled makes little difference to the overall predictive accuracy.

Figure 7.6 gives the 'train and test' result for the *vote* dataset modified to incorporate the 'default to largest class' strategy. The difference is slight.

Dataset	Test set (instances)	Correctly classified	Incorrectly classified
vote	135	127 (94% ± 2%)	8

Figure 7.6 Train and Test Results for vote Dataset (Modified)

Figures 7.7 and 7.8 show the results obtained using 10-fold and N-fold Cross-validation for the four datasets.

For the *vote* dataset the 300 instances in the training set are used. For the other two datasets all the available instances are used.

Dataset	Instances	Correctly classified	Incorrectly classified
vote	300	275 (92% ± 2%)	25
pima-indians	768	536 (70% ± 2%)	232
chess	647	645 (99.7% ± 0.2%)	2
glass	214	149 (70% ± 3%)	65

Figure 7.7 10-fold Cross-validation Results for Four Datasets

All the figures given in this section are estimates. The 10-fold cross-validation and N-fold cross-validation results for all four datasets are based

Dataset	Instances	Correctly classified	Incorrectly classified
vote	300	278 (93% ± 2%)	22
pima-indians	768	517 (67% ± 2%)	251
chess	647	646 (99.8% ± 0.2%)	1
glass	214	144 (67% ± 3%)	70

Figure 7.8 N-fold Cross-validation Results for Four Datasets

on considerably more instances than those in the corresponding test sets for the 'train and test' experiments and so are more likely to be reliable.

7.6 Experimental Results II: Datasets with Missing Values

We now look at experiments to estimate the predictive accuracy of a classifier in the case of datasets with missing values. As before we will generate all the classifiers using the TDIDT algorithm, with Information Gain for attribute selection.

Three datasets were used in these experiments, all from the UCI Repository. Basic information about each one is given in Figure 7.9 below.

Dataset	Description	classes	attributes+		instances	
			categ	cts	training set	test set
crx	Credit Card Applications	2	9	6	690 (37)	200 (12)
hypo	Hypothyroid Disorders	5	22	7	2514 (2514)	1258 (371)
labor-ne	Labor Negotiations	2	8	8	40 (39)	17 (17)

+ categ: categorical; cts: continuous

Figure 7.9 Three Datasets with Missing Values

Each dataset has both a training set and a separate test set. In each case, there are missing values in both the training set and the test set. The values in parentheses in the 'training set' and 'test set' columns show the number of instances that have at least one missing value.

The 'train and test' method was used for estimating predictive accuracy.

Two strategies for dealing with missing attribute values were described in Section 2.4. We give results for each of these in turn.

7.6.1 Strategy 1: Discard Instances

This is the simplest strategy: delete all instances where there is at least one missing value and use the remainder. This strategy has the advantage of avoiding introducing any data errors. Its main disadvantage is that discarding data may damage the reliability of the resulting classifier.

A second disadvantage is that the method cannot be used when a high proportion of the instances in the training set have missing values, as is the case for example with both the *hypo* and the *labor-ne* datasets. A final disadvantage is that it is not possible with this strategy to classify any instances in the test set that have missing values.

Together these weaknesses are quite substantial. Although the 'discard instances' strategy may be worth trying when the proportion of missing values is small, it is not recommended in general.

Of the three datasets listed in Figure 7.9, the 'discard instances' strategy can only be applied to *crx*. Doing so gives the possibly surprising result in Figure 7.10.

Dataset	MV strategy	Rules	Test set	
			Correct	Incorrect
crx	Discard Instances	118	188	0

Figure 7.10 Discard Instances Strategy with *crx* Dataset

Clearly discarding the 37 instances with at least one missing value from the training set (5.4%) does not prevent the algorithm constructing a decision tree capable of classifying the 188 instances in the test set that do not have missing values correctly in every case.

7.6.2 Strategy 2: Replace by Most Frequent/Average Value

With this strategy any missing values of a categorical attribute are replaced by its most commonly occurring value in the training set. Any missing values of a continuous attribute are replaced by its average value in the training set.

Figure 7.11 shows the result of applying the 'Most Frequent/Average Value' strategy to the *crx* dataset. As for the 'Discard Instances' strategy all instances in the test set are correctly classified, but this time all 200 instances in the test set are classified, not just the 188 instances in the test set that do not have missing values.

Dataset	MV strategy	Rules	Test set	
			Correct	Incorrect
crx	Discard Instances	118	188	0
crx	Most Frequent/Average Value	139	200	0

Figure 7.11 Comparison of Strategies with *crx* Dataset

With this strategy we can also construct classifiers from the *hypo* and *crx* datasets.

In the case of the *hypo* dataset, we get a decision tree with just 15 rules. The average number of terms per rule is 4.8. When applied to the test data this tree is able to classify correctly 1251 of the 1258 instances in the test set (99%; Figure 7.12). This is a remarkable result with so few rules, especially as there are missing values in every instance in the training set. It gives considerable credence to the belief that using entropy for constructing a decision tree is an effective approach.

Dataset	MV strategy	Rules	Test set	
			Correct	Incorrect
hypo	Most Frequent/Average Value	15	1251	7

Figure 7.12 Most Frequent Value/Average Strategy with *hypo* Dataset

In the case of the *labor-ne* dataset, we obtain a classifier with five rules, which correctly classifies 14 out of the 17 instances in the test set (Figure 7.13).

Dataset	MV strategy	Rules	Test set	
			Correct	Incorrect
labor-ne	Most Frequent/Average Value	5	14	3

Figure 7.13 Most Frequent Value/Average Strategy with *labor-ne* Dataset

7.6.3 Missing Classifications

It is worth noting that for each dataset given in Figure 7.9 the missing values are those of attributes, not classifications. Missing classifications in the training set are a far larger problem than missing attribute values. One possible approach would be to replace them all by the most frequently occurring classification but this is unlikely to prove successful in most cases. The best approach is probably to discard any instances with missing classifications.

7.7 Confusion Matrix

As well as the overall predictive accuracy on unseen instances it is often helpful to see a breakdown of the classifier's performance, i.e. how frequently instances of class X were correctly classified as class X or misclassified as some other class. This information is given in a *confusion matrix*.

The confusion matrix in Figure 7.14 gives the results obtained in 'train and test' mode from the TDIDT algorithm (using information gain for attribute selection) for the *vote* test set, which has two possible classifications: 'republican' and 'democrat'.

Correct classification	Classified as	
	democrat	republican
democrat	81 (97.6%)	2 (2.4%)
republican	6 (11.5%)	46 (88.5%)

Figure 7.14 Example of a Confusion Matrix

The body of the table has one row and column for each possible classification. The rows correspond to the correct classifications. The columns correspond to the predicted classifications.

The value in the ith row and jth column gives the number of instances for which the correct classification is the ith class which are classified as belonging to the jth class. If all the instances were correctly classified, the only non-zero entries would be on the 'leading diagonal' running from top left (i.e. row 1, column 1) down to bottom right.

To demonstrate that the use of a confusion matrix is not restricted to datasets with two classifications, Figure 7.15 shows the results obtained using 10-fold cross-validation with the TDIDT algorithm (using information gain

for attribute section) for the *glass* dataset, which has six classifications: 1, 2, 3, 5, 6 and 7 (there is also a class 4 but it is not used for the training data).

Correct classification	Classified as					
	1	2	3	5	6	7
1	52	10	7	0	0	1
2	15	50	6	2	1	2
3	5	6	6	0	0	0
5	0	2	0	10	0	1
6	0	1	0	0	7	1
7	1	3	0	1	0	24

Figure 7.15 Confusion Matrix for *glass* Dataset

7.7.1 True and False Positives

When a dataset has only two classes, one is often regarded as 'positive' (i.e. the class of principal interest) and the other as 'negative'. In this case the entries in the two rows and columns of the confusion matrix are referred to as *true and false positives* and *true and false negatives* (Figure 7.16).

Correct classification	Classified as	
	+	−
+	true positives	false negatives
−	false positives	true negatives

Figure 7.16 True and False Positives and Negatives

When there are more than two classes, one class is sometimes important enough to be regarded as positive, with all the other classes combined treated as negative. For example we might consider class 1 for the *glass* dataset as the 'positive' class and classes 2, 3, 5, 6 and 7 combined as 'negative'. The confusion matrix given as Figure 7.15 can then be rewritten as shown in Figure 7.17.

Of the 73 instances classified as positive, 52 genuinely are positive (true positives) and the other 21 are really negative (false positives). Of the 141 instances classified as negative, 18 are really positive (false negatives) and the other 123 are genuinely negative (true negatives). With a perfect classifier there would be no false positives or false negatives.

Correct classification	Classified as	
	+	−
+	52	18
−	21	123

Figure 7.17 Revised Confusion Matrix for *glass* Dataset

False positives and false negatives may not be of equal importance, e.g. we may be willing to accept some false positives as long as there are no false negatives or vice versa. We will return to this topic in Chapter 12.

7.8 Chapter Summary

This chapter is concerned with estimating the performance of a classifier (of any kind). Three methods are described for estimating a classifier's predictive accuracy. The first of these is to divide the data available into a training set used for generating the classifier and a test set used for evaluating its performance. The other methods are k-fold cross-validation and its extreme form N-fold (or leave-one-out) cross-validation.

A statistical measure of the accuracy of an estimate formed using any of these methods, known as *standard error* is introduced. Experiments to estimate the predictive accuracy of the classifiers generated for various datasets are described, including datasets with missing attribute values. Finally a tabular way of presenting classifier performance information called a *confusion matrix* is introduced, together with the notion of true and false positive and negative classifications.

7.9 Self-assessment Exercises for Chapter 7

1. Calculate the predictive accuracy and standard error corresponding to the confusion matrices given in Figures 7.14 and 7.15. For each dataset, state the range in which the true value of the predictive accuracy can be expected to lie with probability 0.9, 0.95 and 0.99.

2. Suggest some classification tasks for which either false positive or false negative classifications (or both) would be undesirable. For these tasks, what

proportion of false negative (positive) classifications would you be willing to accept in order to reduce the proportion of false positives (negatives) to zero?

Reference

[1] Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert systems in the micro-electronic age* (pp. 168–201). Edinburgh: Edinburgh University Press.