CHAPTER 9

Guide to Using the System

We must finally descend to the level of nitty-gritty detail to discuss the way that C4.5 is used. The advice given by the *Hitch-Hikers' Guide* is relevant here: DON'T PANIC! Although there are numerous options that control how the system behaves, many of these need never concern the typical user.

In the following sections it is often necessary to denote parts of commands and file names that may or may not be present. We follow the usual convention of enclosing this optional material in brackets; remember that these brackets should be omitted when typing commands.

9.1 Files

9.1.1 Filestem

As mentioned in Chapter 1, every task needs a short name, referred to as its *filestem*, that identifies its files. All files read and written by the system are of the form *filestem.extension*, where *extension* characterizes the type of information involved. The filestem used in Chapter 1 was labor-neg; generally, a filestem can be any string of characters that is acceptable as a file name to your operating system. If your environment imposes restrictions on the length of file names, the filestem should be at least nine characters shorter than this limit to allow for the addition of extensions, and shorter still if you intend to use the cross-validation shell script.

9.1.2 Names file

The fundamental file for any task is the *names* file, called *filestem*.names, that provides names for classes, attributes, and attribute values. Each name consists of a string of characters of any length, with some restrictions:

- A name cannot be the single character "?."
- Although any character can appear in a name, the *special characters* comma (,), colon (:), vertical bar (|), and backslash (\) have particular

meanings and must be *escaped* (preceded by a backslash character) if they appear in a name.

- A period may appear in a name provided it is not followed by a space.
- Embedded spaces are also permitted in a name, but multiple *white-space* characters (spaces and tabs) are replaced by a single space.

Otherwise, the form of a name is rather unconstrained—for example, numbers are perfectly acceptable as names.

The names file consists of a series of entries, each starting on a new line and ending with a period. Blank lines, spaces, and tabs may be used to make the file more readable and have no significance (except within a name). In addition, the vertical bar character (|) appearing anywhere on a line causes the rest of that line to be ignored, and can be used to incorporate comments in the file.

The first entry in the names file gives the class names, separated by commas (and don't forget the period at the end of the entry!). There must be at least two class names and their order is not important.

The rest of the file consists of a single entry for each attribute. An attribute entry begins with the attribute name followed by a colon, and then a specification of the values that the attribute can take. Four specifications are possible:

- ignore; causes the value of the attribute to be disregarded.
- continuous; indicates that the attribute has numeric values, either integer or floating point.
- discrete N, where N is a positive integer; specifies that the attribute has discrete values, and there are no more than N of them.
- A list of names separated by commas; also indicates that the attribute has discrete values and specifies them explicitly (preferred to discrete since it enables the data to be checked). As with class names, the order of attribute values is arbitrary.

Each entry is terminated with a period.

This will probably be clearer if you look again at the sample names file in Figure 1–1. The first line contains the class names and is followed by an optional blank line to improve readability. Next come 16 entries, 1 for every attribute. Each attribute name is followed by a colon and then either continuous (if it is a real-valued attribute) or a list of the

possible values it can take if it is a discrete attribute. In these entries, tabs have been used to make the file more intelligible. Some attribute names such as wage increase first year, and some attribute values such as below average, contain embedded spaces, which are significant; all other spaces and tabs are ignored.

9.1.3 Data file

The data file *filestem*.data is used to describe the training cases from which decision trees and/or rulesets are to be constructed. Each line describes one case, providing the values for all the attributes and then the case's class, separated by commas and terminated by a period. The attribute values must appear in the same order that the attributes were given in the names file. The order of cases themselves does not matter.

Names used as attribute or class values obey the same restrictions as for the names file; in particular, embedded special characters must be escaped by preceding them with a backslash. If the value of an attribute, either discrete or continuous, is not known or is not relevant, that value is specified by a question mark. Values of continuous attributes may be given in integer, fixed-point, or floating-point form, so that all the following are acceptable:

1, 1.2, +1.2, −.2, −12E−3, 0.00012

In fact, any numeric value that is acceptable to your local C compiler will probably work.

The data file can also contain embedded comments. The appearance of the character | (unescaped) anywhere on a line causes the remainder of that line to be ignored. It is unwise to commence a comment in the middle of a name.

9.1.4 Test file

For some applications, you may decide to reserve part of the available data as a test set to evaluate the classifier you have produced. If there is one, the test set appears in the file *filestem*.test, in exactly the same format as the data file.

9.1.5 Files generated by the system

The programs C4.5 and C4.5RULES and the cross-validation script described later also generate files with the same filestem as above. As a

user, you will not have to worry about these, other than to make sure that you do not delete or modify them while they are still relevant! The principal file extensions are:

- unpruned, containing the one or more unpruned trees generated by C4.5. Used by C4.5RULES.
- tree, the final pruned tree; if more than one tree is generated via windowing, this is the tree chosen as the best. Used by CONSULT.
- rules, the final ruleset generated by C4.5RULES. Used by CONSULTR.
- to*i*[+*identifier*] and ro*i*[+*identifier*], for $i = 0, 1, 2, \ldots$; output of the decision tree and rules programs on different sections of a cross-validation experiment that has an optional identifier.
- tres[+*identifier*] and rres[+*identifier*]; summary of results for trees and rules on the cross-validation.

9.1.6 Size restrictions

The software does not impose any (practical) limit on the number of classes, attributes, attribute values, or cases in the data and test files. The numbers of classes, attributes, and discrete values per attribute must all be representable by short integers; in most implementations of C, this allows more than 16,000 of each. The number of training and test cases are represented by full integers, normally permitting about 10^9 of each. However, your particular operating system may limit the amount of memory that can be allocated to any one process, affecting the size of problem that can be attempted.

If a large training set is processed on a machine with a small physical memory, the programs will run very slowly as a result of the need to swap pages in and out.

9.2 Running the programs

Each of the programs allows options to be invoked on the command line. Although some of them are rarely used, they are all given here for completeness, together with the default values that are used if the option is not invoked.

9.2.1 Decision tree induction

The command for invoking the program to build a decision tree is c4.5. The options that can be used with this command are:

-f *filestem* (Default: DF)
This option is almost always used to specify the filestem of the task as above. If no filestem is given, the default filestem DF is assumed.

-u (Default: no test set)
This option is invoked when a test file has been prepared.

-s (Default: no grouping)
As described in Chapter 7, this option causes the values of discrete attributes to be grouped for tests. The default is no grouping, in which case tests on discrete attributes have a separate branch for every possible value of the attribute.

-m *weight* (Default: 2)
Near-trivial tests in which almost all the training cases have the same outcome can lead to odd trees with little predictive power. To avoid this undesirable eventuality, C4.5 requires that any test used in the tree must have at least two outcomes with a minimum number of cases (or, to be more precise, the sum of the weights of the cases for at least two of the subsets T_i must attain some minimum). The default minimum is 2, but can be changed by this option; a higher value may be a good idea for tasks where there is a lot of noisy data.

-c *CF* (Default: 25%)
The *CF* value affects decision tree pruning, discussed in Chapter 4. Small values cause more heavy pruning than large values, with a most pronounced effect on smaller data sets. The default value seems to work reasonably well for many tasks, but you might want to alter it to a lower value if you notice that the actual error rate of pruned trees on test cases is much higher than the estimated error rate (indicative of underpruning).

-v *level* (Default: level 0)
The program contains embedded code to show what is going on during the execution of the algorithms. Level 0, the default, generates no expository output of this kind, level 1 a little, and so on. The levels 3 and above can generate an *enormous* amount of output which is meaningful only to people who have an intimate knowledge of the code. The maximum level is 5.

-t *trees* (Default: 10)
This is one of the options that can be used to invoke windowing, discussed in Chapter 6, in which trees are grown iteratively. It specifies the number of trees to be grown in this manner before one is selected as the best. Unless one of the windowing options is specified, the program will grow a single tree in the standard way as described in Chapter 2.

-w *size* (Default: determined by data file)
This option invokes windowing and specifies the number of cases to be included in the initial window. The default is the maximum of 20% of the training cases and twice the square root of the number of training cases.

-i *increment* (Default: determined by window size)
This option activates windowing and gives the maximum number of cases that can be added to the window at each iteration. The default is 20% of the initial window size. Whatever the value of this option, however, at least half of the training cases misclassified by the current rule are added to the next window.

-g (Default: gain ratio criterion)
The criterion used for assessing possible splits, described in Chapter 2, can be changed to the older gain criterion by this option.

-p (Default: hard thresholds)
When this option is invoked, subsidiary cutpoints Z^+ and Z^- are determined for each test on a continuous attribute. The subsidiary cutpoints affect only the way cases are classified using the interactive decision tree interpreter as described in Chapter 8.

Options can appear in any order. For example, the command

```
c4.5 -c 10 -u -f mytask -s
```

would invoke the decision tree program on a task with filestem mytask (so with names file mytask.names and training cases in mytask.data). A single tree would be constructed with grouped-value tests for discrete attributes, pruned using a *CF* value of 10%, and tested on the unseen cases in mytask.test.

9.2.2 Rule induction

The rule induction program, invoked by the command c4.5rules, should only be used after running the decision tree program C4.5, since it reads the unpruned file containing the unpruned tree(s). Four of the six options have the same meaning as for the previous program, namely

-f *filestem* (Default: DF)

-u (Default: no test set)

-v *level* (Default: level 0)

-c *CF* (Default: 25%)

(where the *CF* value is used to prune rules rather than trees). The last two options are:

-F *confidence* (Default: no significance testing)
If this option is used, the significance of each condition in the left-hand side of a rule is checked using Fisher's "exact" test; each condition must be judged significant at the specified confidence level. This will generally have the effect of producing shorter rules than the standard pruning method, with some risk of over-generalization for tasks with little data.

-r *redundancy* (Default: 1.0)
The number of bits required to encode a set of rules, as presented in Chapter 5, can be increased substantially by the presence of irrelevant or redundant attributes. This

option can be used to specify an approximate redundancy factor when the user has reason to believe that there are many redundant attributes. A redundancy value of 2.5, for instance, implies that there are 2.5 times more attributes than are likely to be useful. The -r option is likely to be beneficial only when the user knows the data well and can estimate an appropriate value.

9.2.3 Interactive classification model interpreters

These programs, CONSULT for decision tree models and CONSULTR for production rule models, accept the same input and so can be dealt with together. Naturally enough, the programs should be used only after the appropriate models have been generated by the C4.5 and C4.5RULES programs, and are invoked respectively by

consult [-f *filestem*] [-t]
consultr [-f *filestem*] [-t]

where the -f option gives the filestem, as before. If the -t option appears, the decision tree or ruleset is printed at the start of the consultation session.

These programs request information from the user by prompting for the values of attributes. A prompt consists of an attribute name followed by a colon. If the attribute is discrete, the user can reply with

- "?" indicating the attribute value is not known;
- a single possible value; or
- a set of possible values of the form

$$V_1 : P_1, V_2 : P_2, \ldots, V_k : P_k$$

where the V_i's are possible values and the P_i's are corresponding value probabilities (see Chapter 8). If the V_i's do not cover all possible values and the sum of the P_i's is less than one, the unassigned probability is distributed equally over the remaining values.

Similarly, the user can reply to a query concerning a real-valued attribute with

- "?" indicating the attribute value is not known;
- a single number; or
- an interval consisting of two numbers separated by a hyphen (-). In this latter case, the value of the attribute is treated as being uniformly distributed in the interval.

Each reply must be followed by a carriage return. Any incorrect input will cause a brief error message to appear, after which the user will be prompted again for the same attribute value.

When the classification model has inquired after all relevant information, the conclusion is presented as described in the previous chapter. The interpreter then prompts the user with

Retry, new case or quit [r,n,q]:

The response to this prompt is one of the three characters indicated, with the following effects:

- q: The program exits.
- n: A new case is classified; the queries recommence as above.
- r: The same case is reexamined. At each query, the reply given last time appears in square brackets. This previous information can be left unchanged by simply pressing carriage return, or can be altered by providing new information in the same manner as before.

9.3 Conducting experiments

Up to this point, the method suggested for estimating the reliability of a classification model is to divide the data into a training and test set, build the model using only the training set, and examine its performance on the unseen test cases. This is quite satisfactory when there is plenty of data, but in the more common circumstance of having less data than we would like, two problems arise. First, in order to get a reasonably accurate fix on error rate, the test set must be large, so the training set is impoverished. Secondly, when the total amount of data is moderate (several hundred cases, say), different divisions of the data into training and test sets can produce surprisingly large variations in error rates on unseen cases.

A more robust estimate of accuracy on unseen cases can be obtained by *cross-validation*. In this procedure, the available data is divided into N blocks so as to make each block's number of cases and class distribution

as uniform as possible. N different classification models are then built, in each of which one block is omitted from the training data, and the resulting model is tested on the cases in that omitted block. In this way, each case appears in exactly one test set. Provided that N is not too small—10 is a common number—the average error rate over the N unseen test sets is a good predictor of the error rate of a model built from all the data.[8]

Cross-validation may seem complex, but two small programs and the shell script that invokes them have been included to facilitate trials. The command to execute the script is

xval.sh *filestem* N [*options*] [*+identifier*]

where

- *filestem* is the name of the task, as above. The script expects to find the corresponding names and data files and, if there is also a test file, the cases in the data and test files are merged before being divided into blocks.
- N is the number of blocks to be used, and so the number of train/test runs to be made. N should never exceed the number of cases available.
- *options*, if they appear, are any options that are to be applied to C4.5 and/or C4.5RULES, in any order.
- the optional *+identifier* is used to label and recognize the output from this particular cross-validation run. There should not be any spaces between the "+" and the identifier. Since this suffix will be attached to every file name generated, it should be short—two or three characters is recommended.

The script generates successive data and test files, builds a decision tree model and a production rule model from the training cases, and uses the models to classify the unseen test cases. The average figures for the two model types are written to the files *filestem*.tres[*+identifier*] and *filestem*.rres[*+identifier*]. Files with extension to*i*[*+identifier*] and ro*i*[*+identifier*] contain the output from C4.5 and C4.5RULES for the ith of the N runs.

8. This gives a slight overestimate of the error rate, since each of the N models is constructed from a subset of the data.

For example, suppose that we wished to carry out a ten-way cross-validation of mytask, grouping tests for the construction of trees and setting a redundancy factor of 2.5 when rules are produced. The command

xval.sh mytask 10 -s -r 2.5 +me

would invoke the cross-validation script as above. This would divide any data into ten blocks, numbered 0 through 9, and would produce the following output files:

- Output from individual runs from C4.5 in files mytask.to0+me to mytask.to9+me and a summary in mytask.tres+me.
- Similar output from the ten C4.5RULES runs in files mytask.ro0+me to mytask.ro9+me and a summary in mytask.rres+me.

The summary files are quite terse, giving the lines from each individual run that describe errors on the training and test sets, and similar lines that show the average errors on training and test. For instance, when a ten-way cross-validation was carried out on the labor-neg data from Chapter 1, the summary file for C4.5RULES (Figure 9–1) indicates that, on the first block, there were three errors on the training data and one on the test data, none on each for the second block, and so on. Averaged over the ten blocks, the rules had a 3.7% error rate on the training data and a 13.7% error rate on unseen test cases. This last figure would be the cross-validation estimate of the error rate of a ruleset built from all the data.

9.4 Using options: A credit approval example

We close this chapter with a small example that illustrates the use of some of the options above. The domain for this case study concerns approval of credit facilities using a dataset provided by a bank. The 690 cases are split 44% to 56% between two classes; the 15 attributes include 6 with numeric values and 9 discrete-valued attributes, the latter having from 2 to 14 possible values. This dataset is distributed with the system under the name crx, but the names of classes, attributes, and attribute values have been disguised by replacing them with symbols to protect the bank's interest in the data. (Such bowdlerization does not affect the learning task in any way but makes the resulting classification models rather uninformative.)

```
Tested 51, errors 3 (5.9%) <<
Tested 6, errors 1 (16.7%) <<
Tested 51, errors 0 (0.0%) <<
Tested 6, errors 0 (0.0%) <<
Tested 51, errors 2 (3.9%) <<
Tested 6, errors 2 (33.3%) <<
Tested 51, errors 1 (2.0%) <<
Tested 6, errors 0 (0.0%) <<
Tested 51, errors 3 (5.9%) <<
Tested 6, errors 3 (50.0%) <<
Tested 51, errors 1 (2.0%) <<
Tested 6, errors 0 (0.0%) <<
Tested 51, errors 3 (5.9%) <<
Tested 6, errors 1 (16.7%) <<
Tested 52, errors 1 (1.9%) <<
Tested 5, errors 1 (20.0%) <<
Tested 52, errors 2 (3.8%) <<
Tested 5, errors 0 (0.0%) <<
Tested 52, errors 3 (5.8%) <<
Tested 5, errors 0 (0.0%) <<

train:  Tested 51.3, errors 1.9 (3.7%) <<
test:   Tested 5.7, errors 0.8 (13.7%) <<
```

Figure 9–1. Summary file for rule models, labor-neg data

As a sighting shot, a ten-way cross-validation using all default options is carried out by the command

```
xval.sh crx 10
```

The key information extracted from the summary files crx.tres and crx.rres is given in the first line of Table 9–1. For this cross-validation, the average simplified decision tree of 58.8 nodes has an error rate of 17.5% on unseen cases whereas the pessimistic error rate predicted from the training data is 12.4%. The unseen error rate for production rules, 17.4%, is quite close to that for the trees.

Inspection of the simplified trees in files crx.to0 to crx.to9 reveals that the multivalued discrete attributes appear infrequently. This could be because they contain little information relevant to classification, or alternatively because they are not being treated equitably by the default gain ratio selection criterion. To explore this further, the cross-validation

Table 9–1. Results with selected options

Options	Decision Trees			Rules
	Size	Unseen Error Rate	Predicted Error Rate	Unseen Error Rate
default	58.8	17.5%	12.4%	17.4%
-s	69.5	17.4%	11.6%	15.2%
-m15	15.9	14.5%	14.4%	14.3%
-c10	39.1	15.8%	15.6%	16.1%
-m15 -c10	10.7	14.5%	16.6%	14.2%
-t10	67.1	17.1%	11.8%	16.8%
-t10 -m15	15.9	15.1%	14.1%	14.5%

is rerun with the -s option that enables the formation of value groups. As Table 9–1 shows, this appears to have little effect on the decision trees (in fact, their average size increases slightly), but results in a lower error rate for the rule-based classifiers. The lack of significant improvement for trees suggests that value grouping is not going to be helpful for this task.

The feature that stands out in the first default run is the disparity between the observed and estimated error rates for unseen cases. This sort of situation can arise when the attributes allow an almost "pure" partition of the training cases into single-class subsets, but where much of the structure induced by this partition has little predictive power. (Since a single continuous attribute can give rise to numerous possible divisions, the phenomenon often occurs when there are many independent continuous attributes.) There are two ways to address the problem: increasing the -m value so as to prevent overly fine-grained divisions of the training set, or reducing the -c option with the effect that more structure is pruned away. When the same cross-validation is repeated with -m15 and with -c10, Table 9–1 shows that the former is more effective here. The error rates on unseen cases obtained with -m15 are much smaller for both trees and rules; moreover, the predicted error rate for trees is almost exactly correct, indicating an appropriate level of pruning. If we

try to gild the lily by using both options together, Table 9–1 shows that the error rate is still low but the simplified trees are smaller.

So far, we have reduced the error rate from 17.5% to 14.5% for pruned trees and from 17.4% to 14.2% for rules, an improvement of roughly 20%. Next, we investigate whether windowing would help in this domain. The `-t10` option causes ten trees to be produced, one of which is selected as the best tree, but all of which are used when a ruleset is constructed. The windowing option slows down the cross-validation by a factor of 20 or so and has little apparent benefit, either on its own or when invoked with the `-m15` option.

At this stage it seems sensible to call a halt to further trials and to recommend using the options `-m15` and `-c10`. The values 15 and 10 were more or less plucked from the air, so we could perhaps see whether other values near these give noticeably better results; in my experience, such fine-tuning is not terribly productive because the system is relatively insensitive to small changes in parameter values. In an actual application we would now return to the dataset and generate a single tree or production rule classifier from all the available cases using these options.