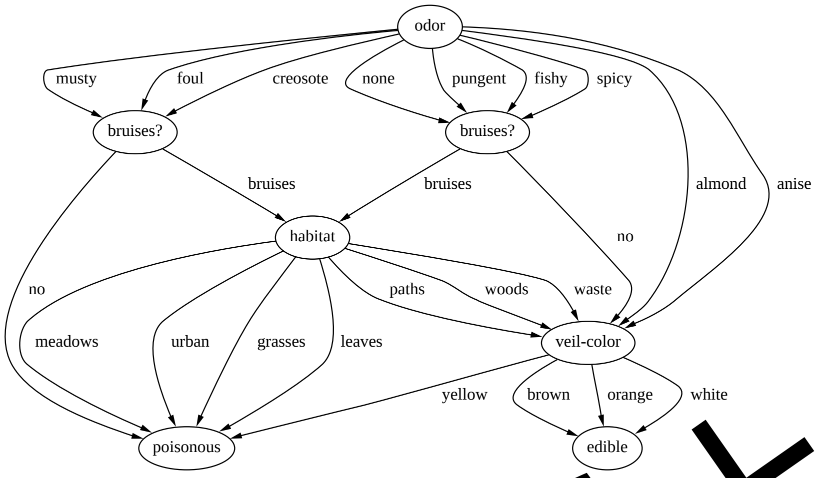
October 7, 1996



MLC++

Machine Learning library in C++

SGI $\mathcal{MLC}$++ Utilities 2.0

By: Ronny Kohavi
and
Dan Sommerfield

mlc@postofc.corp.sgi.com

The URL for $\mathcal{MLC}$++ is http://www.sgi.com/Technology/mlc

Contents

$\mathcal{MLC}$++ Utilities 2.0

Ron Kohavi and Dan Sommerfield
mlc@postofc.corp.sgi.com

October 7, 1996

> cuspy: /kuhs'pee/ [*WPI: from the DEC abbreviation CUSP, for 'Commonly Used System Program', i.e., a utility program used by many people*] *adj. 1. (of a program) Well-written. 2. Functionally excellent. A program that performs well and interfaces well to users is cuspy.*
> —*Jargon File 2.9.9*

This document describes some of the utility programs written using $\mathcal{MLC}$++.[1] Appendix A on page 36 describes the legal issues, installation procedure, and the mailing addresses for questions and bug reports. Appendix B describes some common error messages. Appendix C describes the additions and modifications that were done since the last major release. The reader is assumed to have general knowledge of machine learning and some experience with the Unix operating system.

> The examples in this document are usually displayed at LOGLEVEL 0 to save paper. We recommend that you use $\mathcal{MLC}$++ with the LOGLEVEL set to 1. To change the LOGLEVEL to 1 you can type
> `setenv LOGLEVEL 1`

1 Introduction

$\mathcal{MLC}$++ utilities take their options from environment variables or from the command-line. All examples are given assuming `csh` or `tcsh` is the shell. Text starting with the pound sign (#) is a comment. By default, required options that are not set will be prompted for.

Datasets are assumed to be in the $\mathcal{MLC}$++ format, which is very similar to the C4.5 (Quinlan 1993) format.[2]. Each dataset should include a `names` file describing how to parse the data, a `data` file containing the data, and an optional `test` file for estimating accuracy.

Example 1 (Running ID3)
Fisher's iris dataset (Fisher 1936) contains four attributes of iris plants: sepal length, sepal width, petal length, and petal width. The task is to categorize each instance into one of the three classes: Iris Setosa, Iris Versicolour, and Iris Virginica.

To run the ID3 induction algorithm (Quinlan 1986) on the iris dataset, consisting of iris.names, iris.data, and iris.test, one can type:

```
setenv DATAFILE iris              # The dataset stem
setenv INDUCER ID3                # pick ID3
setenv ID3_UNKNOWN_EDGES no       # Don't bother with unknown edges
setenv DISP_CONFUSION_MAT yes     # Show confusion matrix
setenv DISPLAY_STRUCT dotty       # Show the tree using dotty
Inducer
```

[1] $\mathcal{MLC}$++ is a Machine Learning library of C++ classes. General information about the library can be obtained through the world wide web at URL `http://www.sgi.com/Technology/mlc`

[2] Minor differences exist. $\mathcal{MLC}$++ does not support "ignore" in the names file. The "project" utility can be used instead of using ignore.

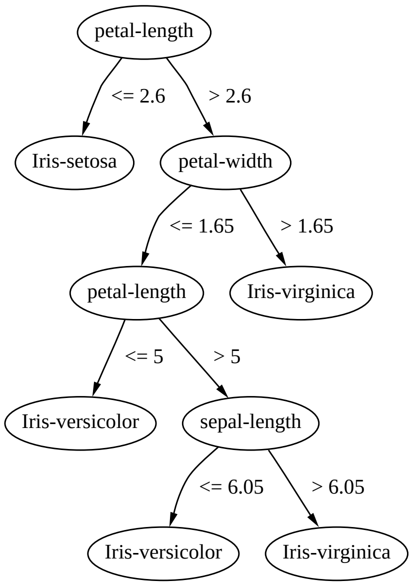


Figure 1: The file **iris.ps** depicting the decision tree induced by ID3 on the **iris** dataset.

```
dot -Tps -Gpage="8.5,11" -Gmargin="0,0" Inducer.dot > iris.ps
```

The output is:

```
Classifying (% done): 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%  done.
Number of training instances: 100
Number of test instances: 50.  Unseen: 50,  seen 0.
Number correct: 47.  Number incorrect: 3
Generalization accuracy: 94.00%.  Memorization accuracy: unknown
Accuracy: 94.00% +- 3.39% [83.78% - 97.94%]

Displaying confusion matrix...
 (a)  (b)  (c)    <-- classified as
---- ---- ----
  15    0    0    (a): class Iris-setosa
   0   15    2    (b): class Iris-versicolor
   0    1   17    (c): class Iris-virginica
```

If you have **dot** installed (see Appendix A on page 36), you can generate `iris.ps` file shown in Figure 1. If you have an X-terminal and **dotty**, $\mathcal{MLC}$++ will display the graph on the screen.

The generalization accuracy indicates the accuracy on unseen instances and the memorization accuracy indicates the accuracy on instances in the test set which were also in the training set. The accuracy is followed by ± and the theoretical standard deviation, and the range afterwards is the 95% confidence interval. See Section 3 for details.

Example 2 (Cross-Validation)
To cross-validate an inducer (induction algorithm) and a dataset, one can do:

```
setenv DATAFILE iris.all   # ".all" contains all the data
setenv INDUCER ID3
```

```
setenv ACC_ESTIMATOR cv
AccEst
```

The output is:

```
10 folds: 1 2 3 4 5 6 7 8 9 10
Method: cv
Trim: 0
Seed: 7258789
Folds: 10,  Times: 1
Accuracy: 94.00% +- 2.10% (80.00% - 100.00%)
```

cross-validating a different inducer can be done by simply changing the environment variable value.

```
setenv DATAFILE iris.all
setenv INDUCER IB         # A nearest-neighbor algorithm
setenv ACC_ESTIMATOR cv
AccEst
```

The output is:

```
10 folds: 1 2 3 4 5 6 7 8 9 10
Method: cv
Trim: 0
Seed: 7258789
Folds: 10,  Times: 1
Accuracy: 96.00% +- 1.47% (86.67% - 100.00%)
```

If you set the LOGLEVEL to 1, you will see all the available options. If you set the PROMPTLEVEL to "basic" or "all" (the default is "required-only"), $\mathcal{MLC}$++ will prompt you to fill in the options. Type '?' at any prompt for help.

2 Setting Options

$\mathcal{MLC}$++ utilities have the following command-line options:

```
<utility> {[-s] [-o <optionfile>] [-O <option>=<value>]}
```

The `-s` option suppresses environment options. The `-o` option allows reading options from a file containing `<option>=<value>` per line, and the `-O` option (uppercase) allows setting a specific option. The `-o` and `-O` flags may be repeated multiple times.

Each option used by a program has a unique name. By convention, option names appear in ALL CAPS and use underscores between words, although few common options do not use underscores (*e.g.*, LOGLEVEL). Options are set according to the following hierarchy:

$\mathcal{MLC}$++ default The default value set in the $\mathcal{MLC}$++ library. If the $\mathcal{MLC}$++ library does not provide a default value, the option must be supplied and is considered **required.**

Environment option An environment variable contains the default value. Users can set the environment variable with the same name as the option itself. For example, `setenv DATAFILE 'foo'` sets the DATAFILE option to the value "foo." An environment variable takes precedence over the library default value. The command-line flag "-s" will suppress looking at environment variables.

Command-line option As described in the beginning of this section.

User input Under certain settings of PROMPTLEVEL (described below), users can be prompted for option values. The default suggested to the user will be taken from the environment and the $\mathcal{MLC}$++ default described above. If the user types a value, the value will be taken as the option value. If the user types <**return**>, then the default value will be accepted. If the user types "?", then the system will provide some help info on the specific option.

An environment variable called `PROMPTLEVEL` determines when to prompt the user for an option. The variable has three possible values:

Required-only Prompt the user for required options only. Options that have a value set through an environment variable of through $\mathcal{MLC}$++ will not be prompted. This is the lowest level prompting mode.

> To get help and view the possible values for an enumerated option, type a question mark ('?') at the prompt or set the option to a question mark. For example,
> `setenv INDUCER '?'`
> will show you all the $\mathcal{MLC}$++ inducers available if you run the Inducer utility.

Basic Prompt for *basic options*, independent of whether they have a default value or not. Some options are defined as **nuisance** options in the $\mathcal{MLC}$++ library and will not be prompted by this mode. The purpose of this mode is to prompt for the most commonly used options.

If the option has an $\mathcal{MLC}$++ default, users can change it to be a nuisance option by setting the option value to an exclamation mark ('!'). A nuisance option can be changed to a non-nuisance option by setting its value to be a question mark ('?'). See Section 8 for more details.

All Prompt for all options, regardless of their setting.

When an option requires one of a given set of values, *i.e.*, an enumerated option, a prefix of the desired option value may be used, and comparison is case *insensitive*. If there are multiple values with the given prefix, the first one in the list will be chosen. For example, typing `setenv INDUCER naive` will match naive-bayes (prefix match) and `setenv INDUCER c4.5` will match C4.5 (first match, case insensitive).

To facilitate repeat runs of a program under the same options, options may be dumped to a file. The dump is in a format which can be sourced by `csh` or `tcsh`. The default is not to dump options. To set a dump file, set the environment variable `OPTION_DUMP` to the name of the new file. We recommend that you add the following to your .login

```
setenv OPTION_DUMP ~/.mlcoptions
```

To repeat a run that has been dumped, simply source it; for example,

```
source ~/.mlcoptions
```

Note that the dump file is generated as you input options in, so you can source it to fill-in the options you have already filled in even if you aborted the run.

2.1 Global Environment Variables

The following environment variables are applicable to any $\mathcal{MLC}$++ program. Note that they are not options and will not be prompted for, nor written to the dump file. These variables do not affect the inherent behavior of algorithms, only the display and the file name search paths.

Option name	Domain	Default	Explanation
PROMPTLEVEL	required, basic, all	required	Determines prompting of options (see above).

Option name	Domain	Default	Explanation
MLCPATH	pathnames	current directory	Any file that is opened with a relative name (*i.e.*, a name that does not begin with a slash (/)) will be searched for in the given colon-separates paths in the given order. To include the current directory in the search, put a period as one of the pathnames. For example, the value can be: **.:/u/mlc/db:/u/mlc/src/tests**.
LOGLEVEL	int ≥ 0	0	The amount of information to print. The higher the number, the more information is printed. Levels one and two are commonly used; higher levels help to understand the internals and to debug the algorithms.
OPTION_DUMP	filename	none	File to dump options (see Section 2).
DEBUGLEVEL	int ≥ 0	0	Internal debug level. Higher numbers execute more internal checks. This option is usually relevant only to developers. If you get an error, raising the level to 1 or 2 might give a better error message. Programs run slower by about 50% at level 1 and about 100% slower at level 2. Note: the library is currently compiled in FAST mode, which ignored DEBUGLEVEL.
LINE_WIDTH	int ≥ 1	79	The line width for $\mathcal{MLC}$++ output. Automatic wrapping will occur to break words before this width. Wrapped lines will begin with the WRAP_PREFIX string. The default line width of 79 is only set if the output stream is a tty (isatty()). If the output is redirected to a file, the default width is 32K.
WRAP_PREFIX	string	three spaces	The prefix to insert at the beginning of a wrapped line. Usually a few spaces.
SHOW_LOG_ORIGIN	yes, no	no	Show where in the source code the log messages are coming from. Log messages are messages that appear when the LOGLEVEL option is set to 1 or higher.
KEEPTEMP	yes/no	no	Keep temporary files that are generated. This is useful if you want to analyze the files used to interface external inducers.
TMPDIR	directory	/var/tmp	Where to generate temporary files. see the `tmpnam()` call for details.

2.2 Common Options

The following options are applicable in many $\mathcal{MLC}$++ utilities. A dash (—) denotes that the option must be set and has no default:

Option name	Domain	Default	Explanation
INDUCER	inducer name	—	The name of an inducer (see Section 4 on page 10).
DATAFILE	filename	—	The datafile to work on. This option should generally be the file stem (without any suffix), and implies .names for the names file, .data for the data file, and .test for the test file.
NAMESFILE	filename	datafile	The names file to help parse the datafile. It defaults to the stem of the datafile with a .names suffix.
TESTFILE	filename	datafile	Utilities that take an optional test file will use this option. It defaults to the filestem of the DATAFILE with a .test extension, except if the DATAFILE ends with ".all," in which case the default will be empty (no test file).
DUMPSTEM	filename	—	A file stem to write to. This option is used in utilities that generate files. See the `conv` utilities for example.
SEED	int	7258789	The default seed for the random generator.[3]
REMOVE_ UNKNOWN_ INST	yes, no	no	Removes unknown instances from the datafiles as they are read in.
CORRUPT_ UNKNOWN_ RATE	Real r $0 \leq r < 1$	0	The rate (probability) for an attribute to be corrupted to unknown.
DISPGRAPH	yes/no	no	Display the induced tree or graph using **dotty**.

3 Accuracy Estimation

Accuracy estimation refers to the process of approximating the future performance of a classifier induced by an inducer on a given dataset. We refer the reader to Kohavi (1995*b*) and Weiss & Kulikowski (1991) for an overview.

In many cases where $\mathcal{MLC}$++ presents an accuracy, it also presents the confidence of the result. The number after the $\pm$ indicates the standard deviation of the accuracy. If a single test set is available, the standard deviation is a theoretical computation that is reasonable for large test sets and for accuracies not too close to zero or one. If resamplings are used (cross-validation and bootstrap), then the standard deviation of the sample **mean** is given.[4] An accuracy range in square brackets is a 95% confidence bound that is computed by a more accurate formula (Kohavi 1995*b*, Devijver & Kittler 1982). An accuracy range in parentheses is a 95% percentile interval (Efron & Tibshirani 1993); the percentile bound is pessimistic in the sense that it includes a wider range due to the integral number of samples. Below 40 samples, it will give the lowest and highest estimates, so that one can see the variability of the estimates.

[3] The default seed is the phone number in the room where $\mathcal{MLC}$++ was developed.

[4] The standard deviation of the mean is not the standard deviation of the population. It shows how variable the mean is, which is smaller than the variability of the population itself by a factor of the number of samples. See any statistics book, such as Rice (1988) for details.

$\mathcal{MLC}$++ currently supports several methods of accuracy estimation:

Option name	Domain	Default	Explanation
ACC_ESTIMATOR	cv, strat-cv, bootstrap, hold-out, test-set	cv	Accuracy estimation method to use: cross-validation, stratified cv, bootstrap .632, hold-out, and testset. The testset "cheats" by looking at the test set and may be used to derive upper bounds on the performance.
ACC_TRIM	Real r $0 \leq r < .5$	0.0	Ratio of cross-validation folds to trim (the extreme values are trimmed). This might stabilizes the procedure because there may be unrepresentative folds. A reasonable value is 0.1. See Rice (1988, 331-333) for exact method of computing the mean and variance for trimmed data.

Holdout The dataset is split into two disjoint sets of instances. The inducer is trained on one set, the training set, and tested on the disjoint set, the test set. The accuracy on the test is the estimated accuracy.

Option name	Domain	Default	Explanation
HO_TIMES	int ≥ 1	1	The number of times to repeat holdout. If holdout is repeated more than once, it is sometimes referred to as random subsampling.
HO_NUMBER	int $ge0$	0	The number of instances to use in training the inducer, leaving the rest for the test set. If the value is zero, the HO_PERCENT option is used instead. If the number is negative, the absolute value specifies the number of instances to leave in the test set, with the rest used for training.
HO_PERCENT	Real r $0 \leq r \leq 1$	2/3	The percentage of the instances to use for the training set, leaving the rest for the test set.

Cross-validation In k-fold cross-validation, the dataset is randomly split into k mutually exclusive subsets (the folds) of approximately equal size. The inducer is trained and tested k times; each time tested on a fold and trained on the dataset minus the fold. The cross-validation estimate of accuracy is the average of the estimated accuracies from the k folds.

Option name	Domain	Default	Explanation
CV_FOLDS	int $\neq 0, 1$	10	The default number of folds to use in cross validations. A negative number k does leave-(-k)-out cross validation.

Option name	Domain	Default	Explanation
CV_TIMES	int ≥ 0	1	Number of times to repeat the cross-validation procedure. Higher numbers reduce the variance of the estimate. Zero (0) uses a heuristic that repeats cross-validation until the variance estimation goes below a threshold standard deviation (DES_STD_DEV) or a maximum number of times is reached (MAX_TIMES). Note that the estimate of the variance assumes independence of folds which is inaccurate when cross-validation is executed multiple times; this independence assumption becomes unrealistic as the number of executions increases. Given all that, it works pretty well in practice.

Stratified cross-validation Same as cross-validation, except that the folds are stratified so that they contain approximately the same proportions of labels as the original dataset.

Bootstrap The .632 Bootstrap (Efron & Tibshirani 1993) estimates the accuracy as follows. Given a dataset of size n, a **bootstrap sample** is created by sampling n instances uniformly from the data (with replacement). Since the dataset is sampled with replacement, the probability of any given instance not being chosen after n samples is $(1-1/n)^n \approx e^{-1} \approx 0.368$; the expected number of distinct instances from the original dataset appearing in the test set is thus 0.632. The $\epsilon 0$ accuracy estimate is derived by using the bootstrap sample for training and the rest of the instances for testing. Given a number b, the number of bootstrap samples, let $\epsilon 0_i$ be the accuracy estimate for bootstrap sample i. The .632 bootstrap estimate is defined as

$$\mathtt{acc}_{\text{boot}} = \frac{1}{b}\sum_{i=1}^{b}(0.632 \cdot \epsilon 0_i + .368 \cdot \mathtt{acc}_s)$$

where $\mathtt{acc}_s$ is the resubstitution error estimate on the full dataset (*i.e.*, the error on the training set).

Option name	Domain	Default	Explanation
BS_TIMES	int ≥ 1	10	The number of bootstrap samples (b).
BS_FRACTION	Real r $0 \leq r \leq 1$	0.632	The bootstrap fraction multiplying the unseen instances.

4 Inducers

As no one decision tree building method (or, for that matter, machine learning method) is the best for all datasets, we feel that a machine learning researcher/practitioner should experiment with as many methods as possible when attempting to solve a problem.
—S.K. Murthy, S. Kasif, S. Salzberg, README for OC1

$\mathcal{MLC}$++ supports many inducers (induction algorithms), but there is an important dichotomy. The first inducer type is called a *(regular) inducer* and it must be implemented in $\mathcal{MLC}$++ itself. The second type is called a *base inducer* and can either be implemented in $\mathcal{MLC}$++ or it can be an external inducer. A base inducer cannot categorize specific instances, only a set of instances. All external inducers, which are interfaced through $\mathcal{MLC}$++ (*e.g.*, C4.5, PEBLS, aha-IB, and OC1, T2) are base inducers. Base inducers are given the training set and test set and return the accuracy. Some $\mathcal{MLC}$++ algorithms are also base inducers or may behave like such under certain conditions. For example, if the FSS (feature subset selection) inducer option SHOW_REAL_ACC is not "never," then FSS behaves like a base inducer because it must have access to the test set to display the real accuracy as it progresses (this accuracy is not used in the induction process; it is only used for display purposes). Besides the technical details, some wrappers (*e.g.*, bagging) only support operations on regular inducers. Confusion matrices in the "Inducer" utility are an option provided only for regular inducers. We now describe the available inducers and their options.

4.1 Const

Const predicts a constant class—the majority class in the training set. The accuracy of the const inducer is commonly referred to as the *baseline* accuracy.

4.2 Naive Bayes

The Naive-Bayes inducer (Langley, Iba & Thompson 1992, Duda & Hart 1973, Good 1965) computes conditional probabilities of the classes given the instance and picks the class with the highest posterior. Attributes are assumed to be independent, an assumption that is unlikely to be true, but the algorithm is nonetheless very robust to violations of this assumption.

The probabilities for nominal (discrete) attributes are estimated by counts. The probability for zero counts is $1/2m$ for m instances. The probabilities for continuous attributes are estimated by assuming a normal distribution for each attribute and class. Unknown values in the test instance are skipped (equivalent to marginalizing over them).

Better results are commonly achieved by discretizing the continuous attributes. The disc-naive-bayes inducer provides this preprocessing step by chaining disc-filter-inducer to naive-bayes inducer (Dougherty, Kohavi & Sahami 1995). Further improvements can usually be achieved by running feature subset selection (Langley & Sage 1994, Kohavi & Sommerfield 1995) as shown below:

```
setenv INDUCER disc-filter
setenv DISCF_INDUCER FSS
setenv DISCF_FSS_INDUCER naive
setenv DISCF_FSS_CMPLX_PENALTY 0.001
setenv DISCF_FSS_CV_TIMES 0
setenv DISCF_FSS_ACC_ESTIMATOR cv
setenv DISCF_FSS_CV_FOLDS 5
setenv DISCF_FSS_DIRECTION backward
```

4.3 ID3, MC4

ID3 is a very basic decision tree algorithm with no pruning. MC4 includes pruning similar to C4.5 (Quinlan 1993). Except for unknown handling, which is different, MC4 should give you similar results to those of C4.5. Underneath, both are the same algorithm with different default parameter settings.

The MIN_SPLIT_WEIGHT is the minimum percent of training instances divided by the number of classes that are required to trickle down to at least two branches in a given node. The LBOUND_MIN_SPLIT and UBOUND_MIN_SPLIT bound this number from below and above. This provides a similar mechanism to C4.5's handling of splits. The determination of the bound is as follows. First, the minimum number of instances is computing using WEIGHT (this is computed as a floating point number), then if this number is higher than the UBOUND it becomes UBOUND. Finally, if this number if lower than LBOUND, it becomes LBOUND.

ID3_DEBUG adds information to each node, indicating the number of instances, entropy, and mutual information. Set DISPLAY_STRUCT to "dot" and view the resulting Inducer.dot file in dot/dotty after running the Inducer utility, or simply use the ID3 utility.

ID3_UNKNOWN_EDGES determines whether an edge is generated to handle unknown values. If this option is FALSE, you will get a nicer looking tree, but it will fail if there are instances with unknown values. Note that C4.5 has a better mechanism of handling unknowns.

ID3_SPLIT_BY determines the splitting criterion. Either regular mutual-information is used, or mutual-information normalized by the number of values is used.

4.4 Decision Tables

One of the simplest conceivable inducers. Stores a table of all instances, predicts according to the table. If an instance is not found, table-majority predicts the majority class of the table and table-no-majority returns "unknown" (always wrong against test-set).

When coupled with feature subset selection it provides a powerful inducer for discrete data (Kohavi 1995*a*). If discretization is done, it is also powerful for data with continuous attributes. For example, to run discretization and feature subset selection, one can define the following options:

```
setenv INDUCER disc-filter
setenv DISCF_INDUCER FSS
setenv DISCF_FSS_INDUCER table-majority
setenv DATAFILE cleve
```

and run the Inducer utility. Consider raising the LOGLEVEL to 2 to see the progress. You can use the project utility on the final node in order to study the selected attributes in isolation.

4.5 Instance Based Algorithms

IB is an instance-based inducer (Aha 1992, Wettschereck 1994). A good, robust algorithm, but still slow when there are many attributes.

NUM_NEIGHBORS determines the number of neighbors to use. In discrete domains, many neighbors will have the same distance, so NNKVALUE determines whether the number of neighbors will actually be neighbors of different distances, with tie-breaking instances counting as a single neighbor. NORMALIZATION determines whether the data should be normalized according to extreme values, according to the interquartile range (25% to 75%), or whether no normalization should take place. NEIGHBOR_VOTE determines whether the neighbors vote equally or with voting power inversely proportional to their distance. MANUAL_WEIGHTS allows setting the weights per attribute manually, i.e., by typing them for each attribute.

4.6 C4.5 Variants

A description of C4.5 is given in Quinlan (1993). The C4.5, C4.5-no-pruning, and C4.5-rules are interfaces to C4.5, which you need to install on your own. They requires "c4.5" and "c4.5rules" to be in the path, and use the file $MLCDIR/c45test.awk. C4.5 can be purchased with the C4.5 book by Ross Quinlan (ISBN: 1-55860-240). Patches can be retrieved by anonymous ftp to ftp.cs.su.oz.au, directory pub/ml, file patch.tar.Z.

You can modify the default behavior (options) for C4.5 by setting the C45_FLAGS (default is -u -f %s). The %s will be replaced by the file name stem as required by C4.5. C4.5-rules run C4.5 then C4.5rules and the appropriate options can be set using C45R_FLAGS1 for C4.5 and C45R_FLAGS2 for C4.5rules. Unless you use version 7 of C4.5rules, the return status is wrong, which is why we've added an "echo" dummy statement to C45R_FLAGS2.

`C45_STATS` allows you to generate statistics about the generated trees, including the number of nodes and the number of attributes. It is mostly useful if you want to see these statistics for 10-fold CV as opposed to a single run.

`MAX_C45_TRIES` determines the number of times to call C4.5 in case there is a problem running C4.5 or parsing its output. The default value of one suffices unless a cleaning job removes files from /tmp and may remove interface files. For example, in Kohavi (1995*b*) some runs made hundred of thousands of calls to C4.5 to compute a single number. It was crucial to make sure that even if the interface files were removed, a smooth recovery would occur by regenerating the files.

4.7 Holte's OneR

OneR is a simple classifier that makes a one-rule, *i.e.*, a rule based on the value of a single attribute (Holte 1993). MIN_INST is the minimum number of instances for a discretization interval. Holte recommends the value six for most datasets. OneR is currently implemented only as a base inducer.

OneR shows that it is easy to get reasonable accuracy on many tasks by simply looking at one attribute. Contrary to common claims and misinterpretations regarding Holte's results, the inducer is *significantly* inferior to C4.5. The average accuracy of OneR for the datasets tested by Holte is 5.7% lower than that of C4.5 (Holte 1993, page 67). If we look at the error rate, then C4.5 has an error of 14.07% and OneR therefore makes 40% more errors than C4.5.

4.8 T2

T2 is a two-level decision tree that minimizes the number of errors and discretizes continuous attributes (Auer, Holte & Maass 1995). It requires large amounts of memory if you have many clases.

4.9 HOODG/List-HOODG: Oblivious Decision Graphs

An inducer for building oblivious decision graphs bottom-up (Kohavi 1994*a*, Kohavi 1994*b*). Does not handle unknown values. HOODG suffers from irrelevant or weakly relevant features, which is why you should use feature subset selection. HOODG also requires discretized data, so disc-filter must be used. The following example shows an example run with the dotty output shown in Figure 2.

```
setenv DATAFILE monk1
setenv DRIBBLE false
setenv INDUCER disc-filter        # oodg can only work on discrete data
setenv DISCF_INDUCER fss          # feature subset selection
setenv DISCF_FSS_INDUCER hoodg
setenv DISCF_FSS_MAX_STALE 3      # how much to search before stopping
setenv DISCF_FSS_CV_TIMES 0       # heuristic for running cv multiple times
setenv DISCF_FSS_CV_FOLDS 5       # 5-fold CV
setenv DISCF_FSS_CMPLX_PENALTY 0.0001  # Small penalty for more features
setenv DISCF_FSS_SHOW_REAL_ACC never   # Otherwise it's a base inducer
setenv REMOVE_UNKNOWN_INST yes
setenv DISPLAY_STRUCT dotty            # Let's see final graph
setenv INDUCER_DOT oodg.dot            # Where to keep the dot output
Inducer
```

The output is:

```
Number of training instances: 124
Number of test instances: 432.  Unseen: 308,  seen 124.
Number correct: 432.  Number incorrect: 0
Generalization accuracy: 100.00%.  Memorization accuracy: 100.00%
Accuracy: 100.00% +- 0.00% [99.12% - 100.00%]

Note: this categorizer type does not support persistence
```

Invoking feature subset selection can be very slow. An alternative approach that is much faster is to use entropy to find a good set of attributes. The inducer "list-hoodg" encapsulates everything needed for the search. The option `AO_GROW_CONF_RATIO`, which is the proportion of misclassified instances at a given level, determines the stopping criteria. Thus a higher number means less attributes will be used. See Kohavi (1995*c*) for more information.

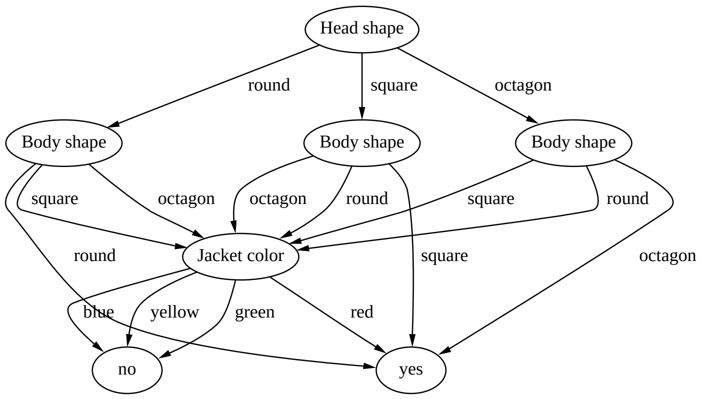


Figure 2: The Oblivious Decision Graph (OODG) for Monk1.

4.10 Aha Instance-based series (IBL)

Aha-ib is an external inducer that interfaces the IB1-4 series from 3/9/94 (Aha 1992). IB_CLASS should be set to one of the following values: ib1, ib2, ib3, or ib4. The seed and specific flags can be set in the options IBL_SEED and IBL_FLAGS. The executable "ibl" must be in the current path.

IBL is a research system and is not very robust. It does not check when its limits are exceeded and sometimes goes out of bounds on arrays, corrupting memory and usually core dumping. If it crashes, the most probable cause is that some constant in datastructures.h is too small. In the version distributed with $\mathcal{MLC}$++ we have increased the limits to 200 attributes and 10,000 instances.

There are some problems that we have discovered when trying to IBL on many datafiles:

1. If the files are too small (few instances), the test set accuracy is not reported (*e.g.*, soybean-small).
2. The program probably leaks memory. It required more than 150MB for the mushroom dataset.
3. It does not handle spaces in attributes values, which can cause problems in some files (this could be taken care of in the $\mathcal{MLC}$++ conversion code, but it is very rare so we do not handle it yet).

Contact David Aha (aha@aic.nrl.navy.mil) with questions, problems, and requests for the source code.

4.11 Perceptron and Winnow

Perceptron and Winnow are inducers that build linear discriminators. They are only capable of handling continuous attributes with no unknowns and two-class problems. For discrete data, you can use the "conv" utility to convert the input attributes to local encoding or binary encoding. The REMOVE_UNKNOWN_INST option can be used to remove instances with unknown values.

Perceptron uses the error correction rule (equation 5.19) in Hertz, Krogh & Palmer (1991). Winnow uses the algorithm described in Littlestone (1988).

All attributes are normalized to be in the range $[0 - 1]$ using extreme normalization (lowest values maps to 0, highest maps to 1). For different normalization types you can use cont-filter inducer as a preprocessor or run the "conv" utility. The reason for this normalization is that winnow overflows really fast when it raises numbers to powers.

4.12 OC1

OC1 is an external inducer that interfaces OC1 version 3 (Murthy, Kasif & Salzberg 1994). The source can be obtained from the Johns Hopkins University, Department of Computer Science. The latest version of the OC1 system can be directly obtained by anonymous FTP from blaze.cs.jhu.edu, in the directory pub/oc1.

Questions regarding OC1 should be directed to its authors, Sreerama Murthy, Steven Salzberg and Simon Kasif (E-mail: lastname@cs.jhu.edu URL: http://www.cs.jhu.edu/lastname).

Any commercial use of OC1 is strictly prohibited without the express written consent of the authors. If you use the OC1 software in the context of any of your publications, please reference Murthy et al. (1994).

The following options are Supported:

OC1_SEED to set the seed.

OC1_AXIS_PARALLEL_ONLY to force axis parallel splits.

OC1_CART_LINEAR_COMBINATION_MODE to force CART-like splits (Breiman, Friedman, Olshen & Stone 1984).

OC1_PRUNING_RATE to set the pruning rate.

The executable "mktree" must be in the current path.

4.13 PEBLS

PEBLS is an external inducer that interfaces the Parallel Exemplar-Based Learning System version 2.0 by Cost & Salzberg (1993). Please contact salzberg@cs.jhu.edu for more information. The sources can be retrieved at URL ftp://ftp.cs.jhu.edu/pub/pebls/.

Several changes were made to the original source code: The maximum number of instances was increased from 110 to 5,000 and the maximum number of characters in class names was increased to 20 (config.h). The program was changed to return status 0 upon exit if the run was successful. A severe bug was fixed in the way weighted voting is handled.

Supported options are: PEBLS_DISCRETIZATION_LEVELS, PEBLS_NEAREST_NEIGHBORS, and PEBLS_VOTING_SCHEME. The executable pebls must be in the current path.

4.14 CN2

CN2 is an external inducer that interfaces the CN2 program (Clark & Niblett 1989, Clark & Boswell 1991). Please see
`http://www.cs.utexas.edu/users/pclark/software.html` or contact `pclark@cs.utexas.edu` for more information.

4.15 Miscellaneous

Some inducers are still being developed or have esoteric uses. We briefly mention them.

Accuracy estimator is a wrapper inducer that runs a given inducer in ACC_INDUCER, estimates its accuracy using an accuracy estimation method, and returns that as the resulting accuracy. The `AccEst` utility provides a friendlier interface, but there are occasions where one wants to do two levels of accuracy estimation (*e.g.*, cross-validation accuracy on holdout sets), where this inducer is very useful (Kohavi 1995*b*).

NULL always predicts unknown and thus gets 0% accuracy. It is mostly used internally, but can be used with the Inducer utility and DISP_CONFUSION_MAT set to true in order to view the distribution of the labels. The "info" utility is probably a better way of getting basic statistics about a data file.

EODG is an inducer for building oblivious decision graphs top-down (Kohavi & Li 1995). Cannot handle unknown values.

LazyDT is a lazy decision tree algorithm, described in Friedman, Kohavi & Yun (1996).

Order-fss searches for an attribute ordering. Very researchy.

Disc-search is a wrapper discretizer that searches for the best number of intervals for each attribute. Very slow.

Weight-search is a wrapper discretizer that searches for the best weight for each attribute (from a uniform set of weights). Slow. Researchy. Not much improvement over feature subset selection.

CatDT Builds decision trees with categorizers you choose at the leaves. Researchy. Requires that inducers support copies, which very few do (*e.g.*, naive-bayes).

5 Wrapper Inducers

Wrapper inducers are not inducers in the ordinary sense, but are inducers that wrap around other inducers to modify their behavior and hopefully improve their performance or allow them to do operations that they could not otherwise perform (*e.g.*, discretize the data for them).

5.1 Discretization filter

Disc-filter is a wrapper inducer that takes the wrapped inducer in the DISCF_INDUCER option. Options for the wrapped inducer will be prefixed by the "DISCF_" prefix.

The most important option is the discretization type: entropy, 1r, bin, c4.5-disc, t2-disc. The entropy discretization seems to be the best discretization method from the allowed options for most practical datasets (Dougherty et al. 1995). Methods which require specifying the number of intervals need the option DISC_NUM_INTR, which determines the number of intervals. Possible options are: Algo-heuristic (algorithm dependent heuristic), Fixed-value (you specify the number), and MDL (based on Fayyad & Irani (1993)). The entropy method requires MIN_SPLIT, the minimum number of instances in an interval, and the algorithm heuristic defaults to MDL. The 1r method is Holte's method of discretization used in the OneR rule (Holte 1993). It requires MIN_INST, the minimum number of instances per bin (0 will be changed to 6, the default suggested by Holte). The bin method uses uniform binning (equal intervals) and the algorithm heuristic for the number of bins is to use twice the log (base 2) of the number of distinct values, a heuristic used in Splus (Spector 1994) and compared in Dougherty et al. (1995). The T2 algorithm (Maass 1994, Auer et al. 1995) heuristic is to form number of classes plus one bins.

5.2 Bagging

Bagging is a wrapper inducer that runs the wrapped inducer, specified in the BAG_INDUCER option, multiple times on subsets of the training set. During classification, the induced classifiers vote and the majority class is chosen (Breiman 1994, Wolpert 1992). The wrapped inducer must be a regular inducer (not a base inducer).

Bagging seems to work best on unstable inducers, that is, inducers that suffer from high variance because of small perturbations in the data (Geman, Bienenstock & Doursat 1992). Unstable inducers include decision trees (*e.g.*, ID3) and perceptrons; an example of a very stable inducer is nearest neighbor, which has a high bias in high-dimensional spaces, but very little variance. What you lose by bagging is the ability to understand the data: you have 20 experts that vote on the label and gains are achieved if they are all good but disagree between themselves many times!

The BAG_REPLICATIONS option determines the number of classifiers to create; the more, the "better" in the sense that the result will be more stable. BAG_PROPORTION determines the proportion of the training set that will be passed to each copy of the inducer. The higher the proportion, the larger the internal training set; however, if the data is not perturbed enough, the classifiers won't be different and bagging won't work well (Krogh & Vedelsby 1995). BAG_UNIF_WEIGHTS is a Boolean option that determines whether the votes are equal or estimated. If the votes are estimated, the portion of the training set that was not used for internal training is used as a test set, and the estimated accuracy is the weight associated with the induced categorizer. Due to high variance in the estimation, this option does not seem to work well in practice.

5.3 Feature Subset Selection

The feature subset selection is a wrapper inducer that selects a good subset of features for improved accuracy performance (Kohavi & Sommerfield 1995, Kohavi 1994*c*, John, Kohavi & Pfleger 1994).

All options in accuracy estimation (Section 3) can be used with the extra options listed below.

Option name	Domain	Default	Explanation
FSS_INDUCER	Inducer	—	Inducer to wrap around.
FSS_DOT_FILE	filename	FSS.dot	The name of the dot file that will contain the search space.
FSS_SEARCH_METHOD	hill-climbing, best-first	best	Search method in the space of feature subsets.
FSS_DIRECTION	forward, backward	forward	Start with the empty set of features (forward) or all features (backward).
FSS_EVAL_LIMIT	int ≥ 0	0	Maximum number of test-set evaluations to conduct. This is an one method of stopping the search (the other is MAX_STALE). The value zero (0) implies no limit.
FSS_SHOW_REAL_ACC	always, best-only, final-only, never	best	When to show the accuracy on the test set. Note that this accuracy is not used by the search mechanism to direct the search. If the accuracy is not "never," then this inducer behaves as a base inducer.
FSS_MAX_STALE	int > 0	5	A search is considered stale if this number of non-improving nodes were expanded. This determines one termination condition.
FSS_EPSILON	Real ≥ 0	0.001	Consider a node non-improving if estimated accuracy was better than the best by this number.
FSS_USE_COMPOUND	yes/no	yes	Generate nodes that combine the features of the best generated children (Kohavi & Sommerfield 1995).
FSS_CMPLX_PENALTY	Real	0.001	How much to penalize the estimate for each feature.

Example 3 (Feature Subset Selection)
To run the IB inducer on the monk1 dataset, one can do:

```
setenv LOGLEVEL 1
setenv INDUCER FSS
setenv FSS_INDUCER IB
setenv DATAFILE    monk1
setenv FSS_DOT_FILE IBFSS.dot
Inducer
```

The output is:

```
MLC++ Debug level is 0, log level is 1
```

```
OPTION PROMPTLEVEL = required-only
OPTION INDUCER = FSS
OPTION INDUCER_NAME = FSS
OPTION FSS_INDUCER = IB
OPTION FSS_INDUCER_NAME = IB
OPTION FSS_NUM_NEIGHBORS = 1
OPTION FSS_EDITING = false
OPTION FSS_NNKVALUE = num-distances
OPTION FSS_NORMALIZATION = extreme
OPTION FSS_NEIGHBOR_VOTE = inverse-distance
OPTION FSS_MANUAL_WEIGHTS = false
OPTION FSS_DOT_FILE = IBFSS.dot
OPTION FSS_SEARCH_METHOD = best-first
OPTION FSS_EVAL_LIMIT = 0
OPTION FSS_SHOW_REAL_ACC = best-only
OPTION FSS_MAX_STALE = 5
OPTION FSS_EPSILON = 0.001
OPTION FSS_USE_COMPOUND = true
OPTION FSS_CMPLX_PENALTY = 0
OPTION FSS_ACC_ESTIMATOR = cv
OPTION FSS_ACC_EST_SEED = 7258789
OPTION FSS_ACC_TRIM = 0
OPTION FSS_CV_FOLDS = 10
OPTION FSS_CV_TIMES = 1
OPTION FSS_CV_FRACT = 1
Method: cv
Trim: 0
Seed: 7258789
Folds: 10,  Times: 1
OPTION FSS_DIRECTION = forward
OPTION DATAFILE = monk1
OPTION NAMESFILE = monk1.names
OPTION REMOVE_UNKNOWN_INST = false
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading monk1.data.. done.
OPTION TESTFILE = monk1.test
Reading monk1.test..... done.

New best node (1 evals) #0[]: accuracy: 39.49% +- 2.45% (30.77% - 50.00%).
Test Set: 50.00% +- 2.41% [45.31% - 54.69%].  Bias: -10.51% cost: 10 complexity: 0
.......
New best node (8 evals) #5[4]: accuracy: 73.21% +- 2.92% (58.33% - 84.62%).
Test Set: 75.00% +- 2.09% [70.71% - 78.85%].  Bias: -1.79% cost: 10 complexity: 1
......
New best node (14 evals) #12[0, 1, 4]: accuracy: 99.17% +- 0.83% (91.67% - 100.00%).
Test Set: 100.00% +- 0.00% [99.12% - 100.00%].  Bias: -0.83% cost: 10 complexity: 3
...................
Final best node #12[0, 1, 4]: accuracy: 99.17% +- 0.83% (91.67% - 100.00%).
Test Set: 100.00% +- 0.00% [99.12% - 100.00%].  Bias: -0.83% cost: 10 complexity: 3
Expanded 8 nodes
Accuracy: 100.00% +- 0.00% [99.12% - 100.00%]
```

This example shows that one can improve the accuracy from 75% to (100%) by looking at only three features. In this case we know that these are the only three relevant features, but it is important to note that they were found automatically. Figure 3 shows the nodes visited and their information. The graph is automatically stored in the file FSS.dot. The edges show the difference in estimated accuracy between the two nodes. The information in each node of the graph is the following:

1. On the top line is the node number. This helps you see the order in which nodes were evaluated. Then

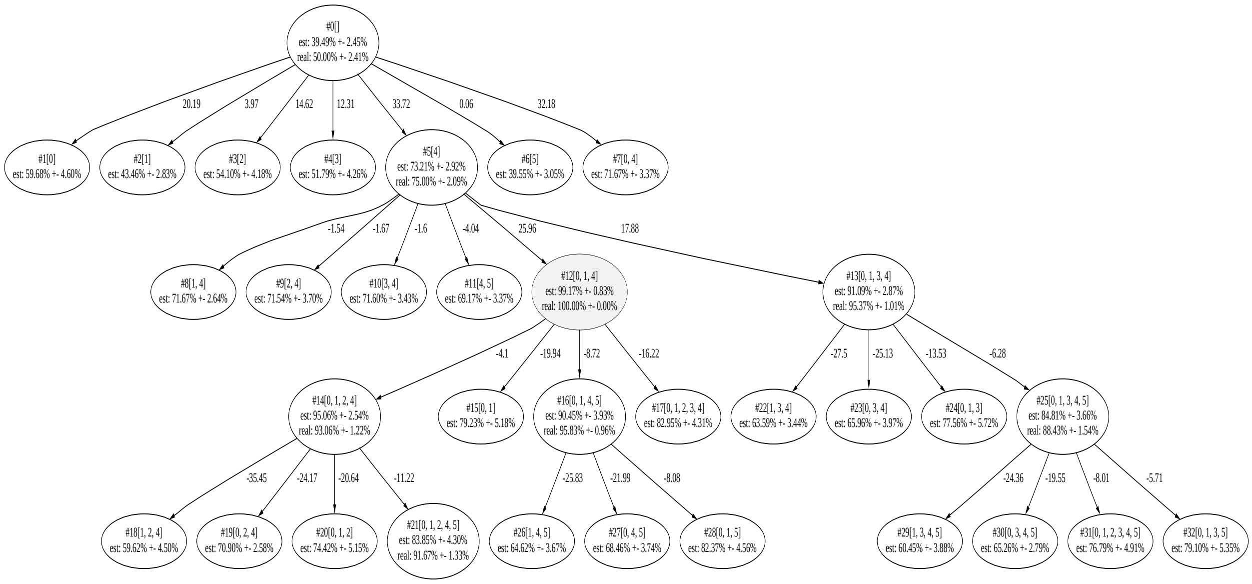


Figure 3: The search space for IB on the monk1 dataset

come the set of features used in brackets (starting from feature 0).

2. On the second line is the estimated accuracy, from whatever accuracy estimation used (*e.g.*, cross-validation, bootstrap, holdout) with the standard deviation of the mean.

3. The third line will appear only for nodes where the real accuracy was computed (accuracy on the test set). The evaluation depends on the setting of on FSS_SHOW_REAL_ACC. By default, only nodes that were "best" at some stage will have this number. Note that this accuracy is never used by the search algorithm.

5.4 C4.5 with Auto Parameters

C4.5-auto-parm is a wrapper algorithm that runs a search over the possible parameter settings for C4.5 and tries to pick the best one for the given dataset. You can decide which parameters to vary by setting the AP_VARY_X options, where X is either M,C,G, or S (see Quinlan (1993) for the meaning of these options). Almost all options applicable to the FSS search (Section 5.3) are applicable here with the AP_ prefix instead of FSS_ prefix. The search space explored is dumped into the file AP.dot and can be viewed using dot or dotty.

The algorithm was reported in Kohavi & John (1995), although some changes since then mean results won't exactly match. Specifically, we do not add and subtract 5 from the array of possibilities because the reviewers considered this a bad hack. Also note that AP_CV_TIMES was set to 0 in our experiment, which takes more time.

6 Utilities

Utilities provided by $\mathcal{MLC}$++ are simple programs (usually less than 100 lines of code) that interface the library to perform common functions.

6.1 Inducers

The **Inducer** utility runs the given inducer on the given datafile and reports the following statistics:

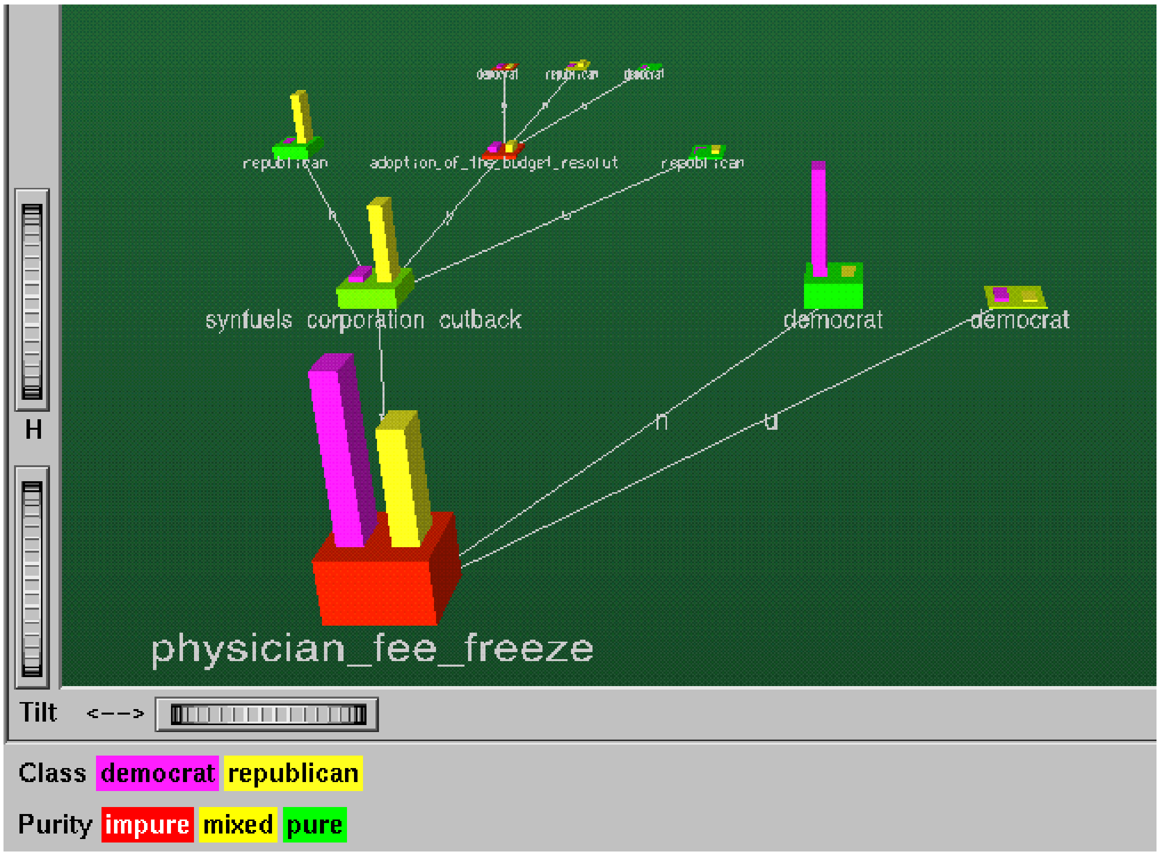


Figure 4: A snapshot of the MineSet tree visualizer fly-through.

Instance Counts The number of training instances, test instances, the number of unseen test instances, and the number of instances seen.

Classification counts The number of correct and incorrect classifications.

Generalization accuracy The accuracy on the unseen instances.

Memorization accuracy The accuracy on the seen instances. A big discrepancy between the generalization and memorization accuracy usually indicates overfitting.

Accuracy The overall accuracy on the test set.

Option name	Domain	Default	Explanation
DISP_CONFUSION_MAT	yes, no	no	Display a table of classifications versus correct classes.
DISP_MISCLASS	yes, no	no	Display the misclassified instances.

Option name	Domain	Default	Explanation
DISPLAY_STRUCT	none, ASCII, dot, dotty, treeviz	none	Display the inducer classifier. ASCII outputs the classifier to the screen. Dot will output an Inducer.dot file for inducers that support dot output (only decision tree and decision graph classifiers). treeviz will output files compatible with Silicon Graphics' MineSet tree visualizer and call it. The tree visualizer allows "flying" through the tree. Figure 4 shows a snapshot for the vote dataset.
INDUCER_DOT	filename	Inducer. dot	Only appears if DISPLAY_STRUCT is dot.
DIST_DISP	yes, no	no	Display distribution at nodes. This option will be available only when relevant.
CAT_NAME	filename		The persistent categorizer name which contains the categorizer. A ".cat" suffix will be appended. Use with the Categorize utility. Few inducers now support persistence.

6.2 Accuracy Estimation

The **AccEst** utility gives estimated accuracy on future instances by different accuracy estimation methods. See Section 3 for details.

6.3 Info

The **info** utility provides basic statistical information about a dataset. It reports the number of instances in the ".data" file, ".test" file, and ".all" file (the ".all" is optional and should contain the ".train" and ".test" for used in AccEst).

It reports the class probabilities, the number of attributes, and their type (continuous or nominal). If the option SHOW_ATTR_INFO is yes, then the number of values for each attribute is given. This may help pinpoint inappropriate declarations of attributes or even continuous attributes which simply have very few values.

Converting attributes with only two values to nominal is generally suggested to gain speedup. For example, the running time for C4.5 (excluding $\mathcal{MLC}$++ overhead) on the StatLog DNA dataset (Taylor, Michie & Spiegalhalter 1994) is 14 seconds on an SGI Indy if the attributes are declared continuous and 4.7 seconds if they are declared nominal. Minor accuracy differences may result due to slightly different ways of handling such attributes.

Example 4 (The "info" utility)
To get information about the attributes in the datafile "labor-neg" one can type:

```
setenv DATAFILE labor-neg
setenv SHOW_ATTR_INFO yes
info
```

The output is:

```
Data + Test == All
Number of instances in labor-neg.all = 57
   Duplicate or conflicting instances : 0
```

```
Number of instances in labor-neg.data = 40
   Duplicate or conflicting instances : 0
Number of instances in labor-neg.test = 17
   Duplicate or conflicting instances : 0
Class probabilities for labor-neg.all file
Probability for the label 'good' : 64.91%
Probability for the label 'bad' : 35.09%
Majority accuracy: 64.91% on value good
Number of attributes = 16 (continuous : 8 nominal : 8)
Information about .all file :
   3 distinct values for attribute #0 (duration) continuous
  17 distinct values for attribute #1 (wage increase first year) continuous
  15 distinct values for attribute #2 (wage increase second year) continuous
   9 distinct values for attribute #3 (wage increase third year) continuous
   4 distinct values for attribute #4 (cost of living adjustment) nominal
   8 distinct values for attribute #5 (working hours) continuous
   4 distinct values for attribute #6 (pension) nominal
   7 distinct values for attribute #7 (standby pay) continuous
  10 distinct values for attribute #8 (shift differential) continuous
   3 distinct values for attribute #9 (education allowance) nominal
   6 distinct values for attribute #10 (statutory holidays) continuous
   4 distinct values for attribute #11 (vacation) nominal
   3 distinct values for attribute #12 (longterm disability assistance) nominal
   4 distinct values for attribute #13 (contribution to dental plan) nominal
   3 distinct values for attribute #14 (bereavement assistance) nominal
   4 distinct values for attribute #15 (contribution to health plan) nominal
```

6.4 Bias-Variance Decomposition

The **biasVar** provides a bias-variance decomposition as described in Kohavi & Wolpert (1996).

Option name	Domain	Default	Explanation
NUM_TEST_INST	integer	0	Choose the number of test instances or select zero for a test fraction.
TEST_FRACT	real	0.6667	Fraction of instances to reserver as test set.
INTERNAL_TRAIN_FRACT	real	0.5	Internal split of training set.
TRAIN_TIMES	integer	10	Number of replicates to form.
REPEAT_TIMES	integer	1	Number of times to repeat replicate sets.

6.5 Categorize

The **Categorize** utility provides a way to categorize new unlabelled records using a persistent categorizer.

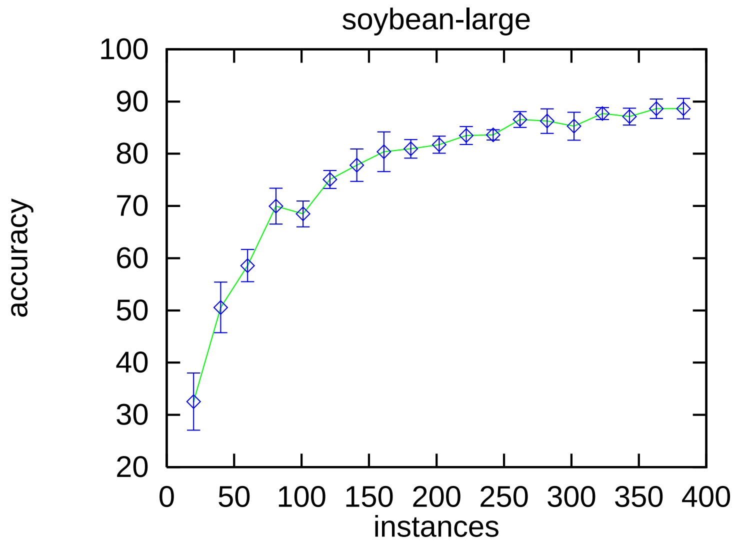


Figure 5: The learning curve for C4.5 on soybean-large

Option name	Domain	Default	Explanation

Option name	Domain	Default	Explanation
CAT_NAME	filename		The persistent categorizer name which contains the categorizer. A ".cat" suffix will be appended. Generated by the Inducer utility.
DUMPSTEM	filename		The filename to contain the predicted labels, one per line.

6.6 LearnCurve

The **LearnCurve** utility generates a learning curve for a given induction algorithm and a dataset. Given a dataset, the x-axis represents the number of training instances and the y-axis represents the accuracy when trained on the given number of instances and tested on the unseen instances.

Option name	Domain	Default	Explanation
NUM_INTERVALS	int > 0	10	Number of intervals on the x-axis. Samples points are approximately equally spaced.
NUM_REPEATS	int > 0	5	Number of times to run the induction algorithm for each sample point. The higher the number, the smaller the standard deviation of the mean.

Option name	Domain	Default	Explanation
MIN_TEST_SIZE	int ≥ 0	0	Minimum number of dataset instances to save for testing. If 0, then this number is set to be the number of instances divided by the number of intervals.
SEED	int	7258789	Seed for creating samples.
DUMPSTEM	filename	none	Stem to dump training sets created. The test set is always the full file. This allows comparing results with other induction algorithms outside $\mathcal{MLC}$++.
LC_OUTPUT_TYPE	none, mathematica, gnuplot	none	Output data suitable for mathematica or gnuplot. For mathematica, the file LearnCurve.m can be used with the generated output.

Example 5 (Learning Curve)
To generate a learning curve for the performance of C4.5 on the soybean-large dataset, one can do:

```
setenv INDUCER  C4.5
setenv DATAFILE soybean-large.all    # This contains the full dataset
setenv NUM_INTERVALS 20              # number of intervals on X-axis
setenv NUM_REPEATS   10              # number of runs at each point
setenv MIN_TEST_SIZE 300             # leave at least 300 for testing
setenv DUMPSTEM                      # no dump stem
setenv LC_OUTPUT_TYPE gnuplot
LearnCurve
gnuplot soybean-large.gnuplot
```

The output is:

```
Inducer: c4.5.  Intervals: 20, Repeats: 10, Min test size: 300.  Seed: 7258789
DATAFILE: soybean-large.all (size=683)
Size,  Acc, std-dev of mean
    20, 32.55% +- 2.79%
    40, 50.58% +- 2.48%
    60, 58.59% +- 1.58%
    81, 69.97% +- 1.75%
   101, 68.49% +- 1.26%
   121, 75.09% +- 0.88%
   141, 77.80% +- 1.58%
   161, 80.38% +- 1.94%
   181, 80.96% +- 0.91%
   202, 81.73% +- 0.83%
   222, 83.49% +- 0.87%
   242, 83.63% +- 0.50%
   262, 86.56% +- 0.76%
   282, 86.26% +- 1.19%
   302, 85.28% +- 1.37%
   323, 87.69% +- 0.58%
   343, 87.12% +- 0.82%
   363, 88.63% +- 0.95%
   383, 88.63% +- 1.00%
Gnuplot output in soybean-large.gnuplot
```

Figure 5 on the preceding page shows the gnuplot graph generated by LearnCurve.

6.7 C45Tree

The **C45Tree** utility will run C4.5 and generate files based on the DATAFILE with -unpruned.dot and -pruned.dot suffixes. These can be viewed using dot or dotty. If the option DISPGRAPH is yes, the graphs will pop up using dotty.

Limitations: DIST_DISP should not be set to yes, and subset splits (-s in C4.5) are not implemented.

6.8 Project

The **project** utility allows projecting a dataset into a space containing only a subset of the attributes (the rest of the attributes are discarded). This set of attribute numbers to project on is queried when the utility is executed. The names file, data file, and test file are all converted to the projected space.

Example 6 (Feature Subset Selection) Feature subset selection on Table-majority inducer found that out of 180 bits used in the DNA splice-junction dataset used in the StatLog project (Taylor et al. 1994), a small subset of 11 bits were most useful for Table-majority (Kohavi 1995*a*). To generate this subset, one can project on the features numbered 81, 83, 84, 89, 92, 93, 94, 95, 96, 101, 104. The performance of Table-majority is 94.6% (on the independent test set), versus 92.7% for C4.5 when given all the attributes.

6.9 Discretization

The **discretize** utility provides discretization ability. See Section 5.1 for a description of available options.

Note that discretization must be done using the training set only.

This utility indeed only looks at the training set to form the intervals, and these intervals are used to discretize both the training set and the test set. It is a mistake to discretize all the data and then to run cross-validation, because the discretization intervals will then be chosen based on the internal folds that serve as test sets. The $\mathcal{MLC}$++ disc-filter will do the right thing if used within cross-validation, *i.e.*, for each of the cross-validation folds, different intervals will be formed as if these were training and test sets.

6.10 Conversions

The **conv** utility provides simple conversions of the data for algorithms that do not deal well with categorical attributes or that require a slightly different input format. Two encodings for nominal attributes are provided:

Local encoding Each value of a categorical attribute is made into an indicator attribute. For a given value in the data file, the appropriate indicator attribute is set to one, and all other indicator attributes that share the representation are set to zero. An unknown value causes all indicator attributes to be zero.

Binary encoding A categorical variable with k possible values is assigned into $\lceil log_2(k+1)\rceil$ bits. Value i is mapped into the binary representation of $i+1$, and the binary zero is allocated for unknown values.

Option name	Domain	Default	Explanation
CONVERSION	local, binary, none, aha	local	See above. The "Aha" format converts to David Aha's IBL programs format (Aha 1992).
ATTR_DELIM	space, comma, period, semicolon, colon	comma	the attribute delimiter.

Option name	Domain	Default	Explanation
LAST_ATTR_DELIM	ditto	comma	The last delimiter before the label.
END_OF_LINE_DELIM	ditto	period	End of line marker.
NORMALIZATION	none, normalDist	none	NormalDist normalizes the continuous attributes to have mean 0, variance 1. If there is only one element, the variance is assumed to be 1. A variance of zero is modified to be 0.01.

6.11 General Logic Diagrams

General Logic Diagrams (GLDs) are graphical projections of multi-dimensional discrete spaces onto two dimensions. They are similar to Karnaugh maps, but are generalized to non Boolean inputs and outputs. A GLD provides a way of displaying up to about ten dimensions in a graphical representation that can be understood by humans. GLDs were described in Michalski (1978) and later used in Wnek, Sarma, Wahab & Michalski (1990). They were used in Thrun *et al.* (1991) and in Wnek & Michalski (1994) to compare algorithms. GLDs have a long history and have been rediscovered many times. They are sometimes called *Dimensional Stacking* (LeBlank, Ward & Wittels 1990).

GLDs will only work with inducers, not base inducers.

Each possible instance in the space defines exactly one box in the GLD. The GLD utility has the following display options (GLD_SET):

Test Show the test set instances with their classes.

Overlay Show the test set instances. The display shows correct or incorrect prediction by the categorizer.

Predicted Train Show the full predicted space and overlay the training set classes. You must have a color/grey-scale display.

Predicted Test Show the full predicted space and overlay the test set instances, showing the mistakes as X's. You must have a color/grey-scale display.

The output of the GLD utility can either be an X window popup or a file that can be read using Xfig. The main advantage of the Xfig output is that the attribute names and other comments may be added and then inserted into a document. There are currently only eight colors and they begin to cycle if there are more classes.

Option name	Domain	Default	Explanation
GLD_MANAGER	Motif,Xfig	Motif	Display the output to an X window or generate Xfig output in GLD.fig.
GLD_SET	test, overlay, predicted-Train, predicted-Test	predicted-Test	What to show in the GLD (see above).

Option name	Domain	Default	Explanation
GLD_MANUAL_ ORDER	yes, no	no	Restrict the inducer and the display to a given set of attributes with manually specified ordering on the axes.
GLD_FONT	font	9x15	The font to use if the manager is Motif. The smaller the font, the more of the attribute value will fit in the designated boxes.
GLD_HORIZ- ONTAL_MARGIN	int	300	How much space (pixels) to leave on the left side of the diagram for the labels.
GLD_VERT- ICAL_MARGIN	int	100	How much space (pixels) to leave on the bottom of the diagram for the labels.
GLD_HORIZ- ONTAL_PIXELS	int	400	Horizontal size of diagram in pixels, excluding margins.
GLD_VERT- ICAL_PIXELS	int	400	Vertical size of diagram in pixels, excluding margins.
GLD_COLOR_ DISPLAY	yes,no	yes	Different shapes have different colors. Most Postscript printers do a good job of grey-scales with colors. For PredictedTrain/Test combinations, you must have a color display.
GLD_COLOR_FILL	yes,no	yes	Use shapes or fill in the box with a color. Only relevant if COLOR_DISPLAY is yes.
GLD_OUTFILE	filename	GLD.fig	The file name to output if the display manager is Xfig.
GLD_MAX_ LINE_WIDTH	int	4	The thickness of lines in the GLD is determined by the position of the line. Line 2^i has width i, unless i is greater than this parameter.
GLD_ALT_ COLOR_SCHEME	yes,no	no	An alternate coloring scheme, which sometimes works better.

Example 7 (General Logic Diagrams)
We now show the four different sets displayable in the GLD. The four GLDs are shown in Figures 6 and 7.

```
setenv INDUCER ID3
setenv DATAFILE monk1
setenv GLD_MANAGER xfig
setenv GLD_OUTFILE monk1-GLD-test.fig
setenv GLD_SET test
GLD
setenv GLD_OUTFILE monk1-GLD-overlay.fig
setenv GLD_SET overlay
GLD
setenv GLD_OUTFILE monk1-GLD-predTrain.fig
setenv GLD_SET predictedTrain
GLD
setenv GLD_OUTFILE monk1-GLD-predTest.fig
setenv GLD_SET predictedTest
GLD
```

The output is:

```
Generating GLD for monk1.data
Classifying (% done): 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%  done.
Generating GLD for monk1.data
Classifying (% done): 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%  done.
Generating GLD for monk1.data
Classifying (% done): 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%  done.
Generating GLD for monk1.data
Classifying (% done): 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%  done.
```

7 Useful Scripts and Tricks

7.1 Comparing two Inducers

Here is an example of a shell script that compares ID3 with a bagged ID3. Because ID3 is very unstable, large improvements can be seen for some files. For this script, monk1 improves from 81.25% to 90.51%; crx improves from 74.00% to 79.00%; and waveform improves from 70.21% to 78.87%.

```
#!/bin/tcsh
#Script to compare bagging of ID3

# first time, log 1; otherwise, log 0
setenv LOGLEVEL 1
foreach i (monk1 crx waveform-21)
   echo ------------------- $i ------------------------
   setenv DATAFILE $i
   setenv INDUCER id3
   Inducer
   setenv INDUCER bagging
   setenv BAG_INDUCER id3
   Inducer
   setenv LOGLEVEL 0
end
```

7.2 Conversions

Here is an example of a shell script that compares a perceptron with a bagged perceptron. The file datasets.txt should contain a list of file names to compare. The datafile is first converted to local encoding using the conv utility, then an inducer is run and then bagging is done.

```
#!/bin/tcsh
#Script to compare bagging of perceptron.

set ind = perceptron
setenv MAX_EPOCHS 25
setenv BAG_REPLICATIONS 20
setenv REMOVE_UNKNOWN_INST yes
setenv LOGLEVEL 0

foreach i (`cat datasets.txt`)
```

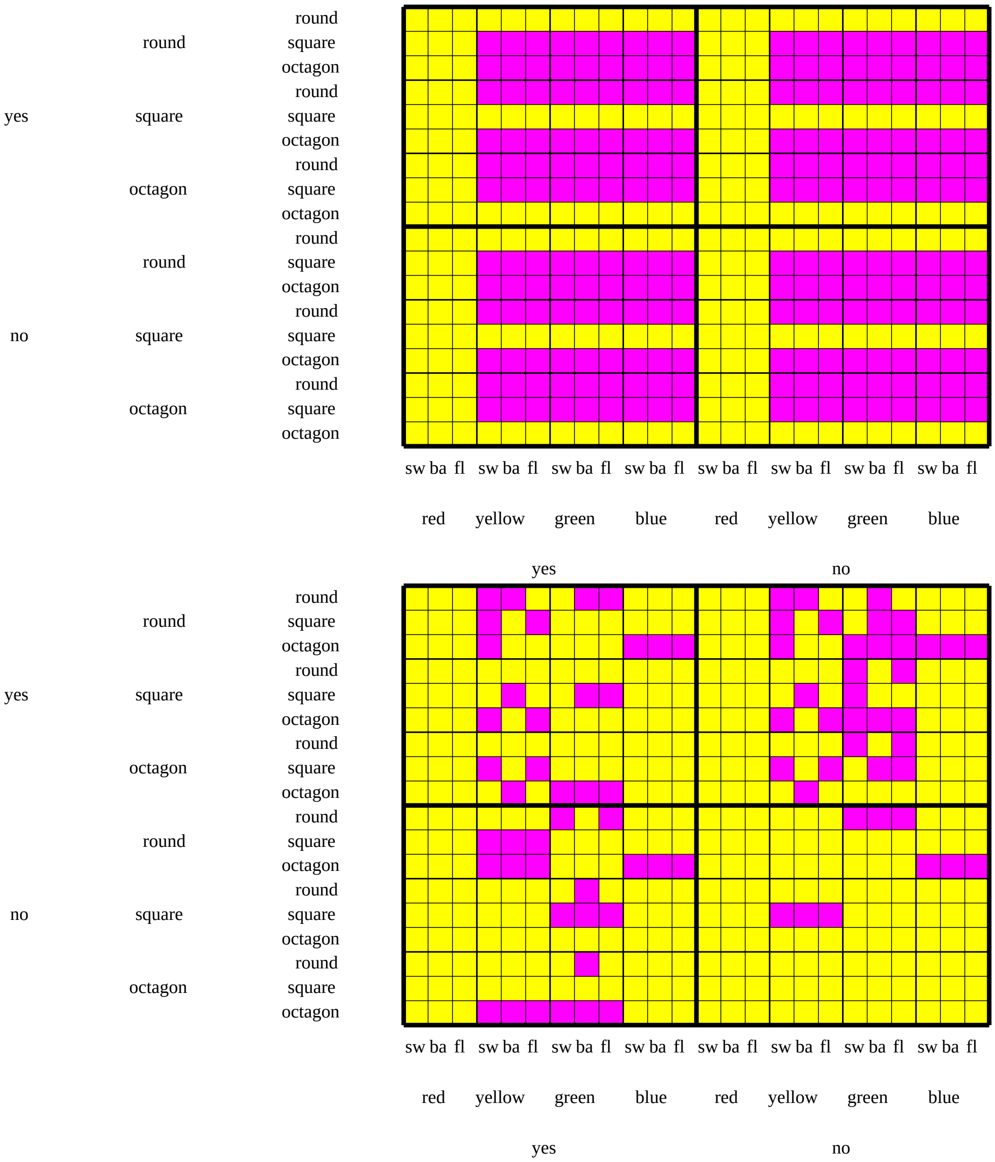
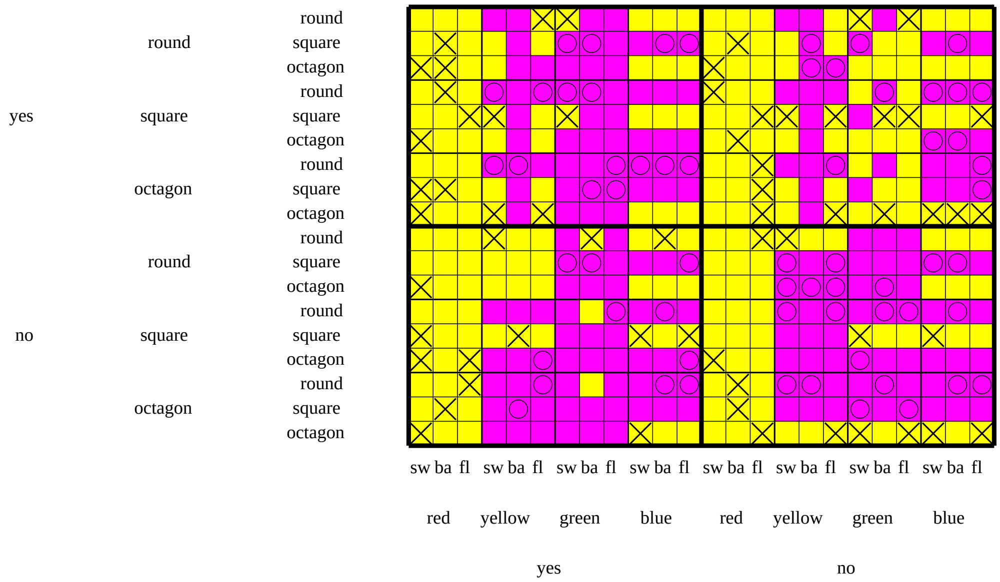


Figure 6: GLD for ID3/monk1. Set=test on the top and Set=overlay on the bottom.



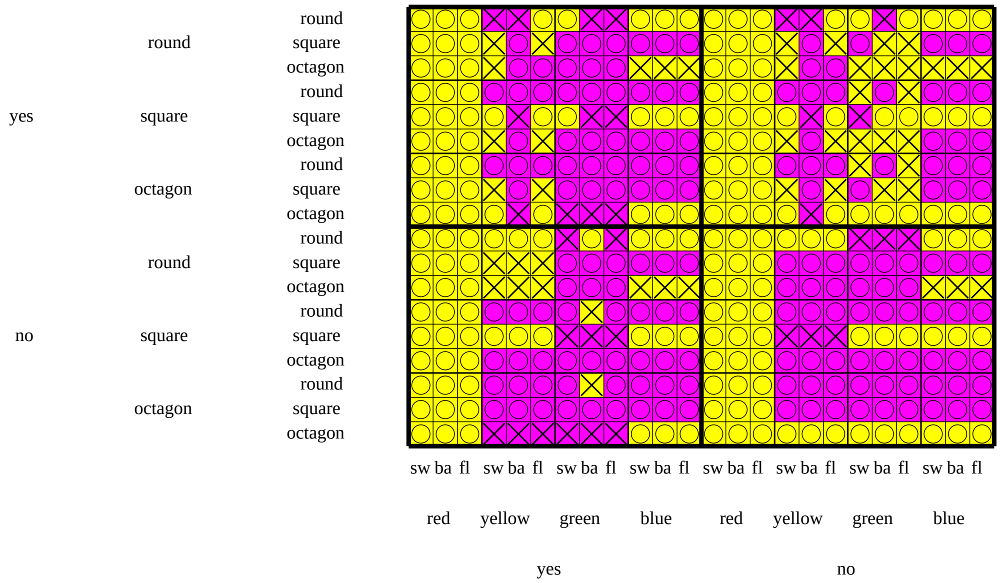


Figure 7: GLD for ID3/monk1. Set=predictedTrain on the top and Set=predictedTest on the bottom.

```
    set file = `basename $i \.all`
    echo ------------------- $file -----------------------
    setenv DATAFILE $file
    setenv DUMPSTEM /tmp/a
    conv
    setenv DATAFILE /tmp/a
    setenv INDUCER $ind
    echo -n "$ind        "
    Inducer | grep Accuracy
    setenv INDUCER bag
    setenv BAG_INDUCER $ind
    echo -n "Bagged $ind "
    Inducer | grep Accuracy
    setenv LOGLEVEL 0
end
```

7.3 Converting Files for External Inducers

If you want to use one of the external inducers directly (*e.g.*, PEBLS, OC1, Aha's IB) then you need to convert the data files to their format. One simple trick is to define a two line script in your directory that contains a single exit statement that returns a bad status. For example, to get a file in PEBLS format you can do:

```
echo "exit 1" > pebls
chmod a+x pebls
setenv INDUCER pebls
Inducer
```

The inducer will abort with an error message indicating the command line used to call pebls. The line will contain the name of the temporary file names used, which you can then rename.

Note: for this trick to work properly, your path must contain the current directory before the $MLCDIR directory. Do "which pebls" to verify that it is indeed the above script that will get executed. Don't forget to remove the "pebls" file when you're done.

8 Mastering Options

This section describes in detail the special option values that can be used.

8.1 Special Values

Options can take on special values as follows:

? Force prompting of the given option, regardless of the prompt level. This is useful if you want to set the value of an option, without being prompted for other options. Specifically, this will cause prompting of nuisance options. Note that because most shells have special meanings for a question mark, the value must be quoted:

```
setenv OPTION '?'
```

'!' Force the option to the $\mathcal{MLC}$++ default value and avoid prompting for it unless the PROMPTLEVEL is set to "all". If there is no default value, the program will abort.

value! The value preceding the exclamation mark will be treated as a default, and the status of the option will be changed to nuisance, avoiding further prompting in the basic prompt level. Use this feature to avoid prompting for an option you do not want to change when doing multiple runs. For example,

```
setenv CV_FOLDS '5!'
```

8.2 String Options

Options that require a string value are treated specially because the empty string is sometimes *allowed* as a choice. Here are the differences:

1. `setenv OPTION` (no value set) will set the option to the empty string. For other options, this is treated as if the environment variable is not set at all.
2. `setenv OPTION ' '` does the same as above.
3. Typing `' '` at the prompt sets the option to an empty string.
4. If the option has no default and the user hits return, the option will be set to the empty string (*i.e.*, the empty string is the default for string options which appear to have no default).

8.3 The Option Dump File

When a program is running, option values that are required for the particular execution are output into the dump file the same way they were input. There are several rules governing dump file behavior:

1. If an option was set via `setenv`, then a similar setenv string will appear in the dump-file, including options using '!'.
2. If an option was prompted, then the user has some control over what will appear in the dump-file:
   (a) Typing a value will place that value into the dump file. If the option was a nuisance option, then the value will be followed by '!'.
   (b) Typing a value following by '!' will accept the value and place '!' after the value in the dump file.
   (c) Typing `<return>` to accept the $\mathcal{MLC}$++ default will place an `unsetenv OPTION_NAME` in the dump file, which will cause similar behavior if the dump file is sourced.
   (d) Typing '!' at the prompt will accept the default but place a '!' in the dump file, causing the option not to be prompted if the dump file is sourced.

Acknowledgments

Wray Buntine, George John, Pat Langley, Ofer Matan, Karl Pfleger, and Scott Roy contributed to the design of $\mathcal{MLC}$++. Nils Nilsson and Yoav Shoham supported this project. Many students at Stanford have worked on $\mathcal{MLC}$++, including: Robert Allen, Eric Bauer, Brian Frasca, James Dougherty, Steven Ihde, Ronny Kohavi, Alex Kozlov, Clay Kunz, Jamie Chia-Hsin Li, Richard Long, David Manley, Svetlozar Nestorov, Mehran Sahami, Dan Sommerfield, and Yeogirl Yun. $\mathcal{MLC}$++ was partly funded by Silicon Graphics Inc, ONR grants N00014-94-1-0448, N00014-95-1-0669, and NSF grant IRI-9116399.

References

Aha, D. W. (1992), 'Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms', *International Journal of Man-Machine Studies* **36**(1), 267–287.

Auer, P., Holte, R. & Maass, W. (1995), Theory and applications of agnostic PAC-learning with small decision trees, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann Publishers, Inc.

Breiman, L. (1994), Bagging predictors, Technical Report Statistics Department, University of California at Berkeley.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984), *Classification and Regression Trees*, Wadsworth International Group.

Clark, P. & Boswell, R. (1991), Rule induction with CN2: Some recent improvements, *in* Y. Kodratoff, ed., 'Proceedings of the fifth European conference (EWSL-91)', Springer Verlag, pp. 151–163. *http://www.cs.utexas.edu/users/pclark/papers/newcn.ps

Clark, P. & Niblett, T. (1989), 'The CN2 induction algorithm', *Machine Learning* **3**(4), 261–283.

Cost, S. & Salzberg, S. (1993), 'A weighted nearest neighbor algorithm for learning with symbolic features', *Machine Learning* **10**(1), 57–78.

Devijver, P. A. & Kittler, J. (1982), *Pattern Recognition: A Statistical Approach*, Prentice-Hall International.

Dougherty, J., Kohavi, R. & Sahami, M. (1995), Supervised and unsupervised discretization of continuous features, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann, pp. 194–202.

Duda, R. & Hart, P. (1973), *Pattern Classification and Scene Analysis*, Wiley.

Efron, B. & Tibshirani, R. (1993), *An Introduction to the Bootstrap*, Chapman & Hall.

Fayyad, U. M. & Irani, K. B. (1993), Multi-interval discretization of continuous-valued attributes for classification learning, *in* 'Proceedings of the 13th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Inc., pp. 1022–1027.

Fisher, R. A. (1936), 'The use of multiple measurements in taxonomic problems', *Annals of Eugenics* **7**(1), 179–188.

Friedman, J., Kohavi, R. & Yun, Y. (1996), Lazy decision trees, *in* 'Proceedings of the Thirteenth National Conference on Artificial Intelligence', AAAI Press and the MIT Press, pp. 717–724.

Geman, S., Bienenstock, E. & Doursat, R. (1992), 'Neural networks and the bias/variance dilemma', *Neural Computation* **4**, 1–48.

Good, I. J. (1965), *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*, M.I.T. Press.

Hertz, J., Krogh, A. & Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation*, Addison Wesley.

Holte, R. C. (1993), 'Very simple classification rules perform well on most commonly used datasets', *Machine Learning* **11**, 63–90.

John, G., Kohavi, R. & Pfleger, K. (1994), Irrelevant features and the subset selection problem, *in* 'Machine Learning: Proceedings of the Eleventh International Conference', Morgan Kaufmann, pp. 121–129.

Kohavi, R. (1994*a*), Bottom-up induction of oblivious, read-once decision graphs, *in* F. Bergadano & L. D. Raedt, eds, 'Proceedings of the European Conference on Machine Learning', pp. 154–169.

Kohavi, R. (1994*b*), Bottom-up induction of oblivious, read-once decision graphs : strengths and limitations, *in* 'Twelfth National Conference on Artificial Intelligence', pp. 613–618.

Kohavi, R. (1994*c*), Feature subset selection as search with probabilistic estimates, *in* 'AAAI Fall Symposium on Relevance', pp. 122–126.

Kohavi, R. (1995*a*), The power of decision tables, *in* N. Lavrac & S. Wrobel, eds, 'Proceedings of the European Conference on Machine Learning', Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, Heidelberg, New York, pp. 174–189.

Kohavi, R. (1995*b*), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Inc., pp. 1137–1143.

Kohavi, R. (1995*c*), Wrappers for Performance Enhancement and Oblivious Decision Graphs, PhD thesis, Stanford University, Computer Science department. STAN-CS-TR-95-1560, ftp://starry.stanford.edu/pub/ronnyk/teza.ps.

Kohavi, R. (1996), Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid, *in* 'Proceedings of the Second International Conference on Knowledge Discovery and Data Mining', p. to appear.

Kohavi, R. & John, G. (1995), Automatic parameter selection by minimizing estimated error, *in* A. Prieditis & S. Russell, eds, 'Machine Learning: Proceedings of the Twelfth International Conference', Morgan Kaufmann Publishers, Inc., pp. 304–312.

Kohavi, R., John, G., Long, R., Manley, D. & Pfleger, K. (1994), MLC++: A machine learning library in C++, *in* 'Tools with Artificial Intelligence', IEEE Computer Society Press, pp. 740–743. `http://www.sgi.com/Technology/mlc`.

Kohavi, R. & Li, C.-H. (1995), Oblivious decision trees, graphs, and top-down pruning, *in* C. S. Mellish, ed., 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Inc., pp. 1071–1077.

Kohavi, R. & Sommerfield, D. (1995), Feature subset selection using the wrapper model: Overfitting and dynamic search space topology, *in* 'The First International Conference on Knowledge Discovery and Data Mining', pp. 192–197.

Kohavi, R., Sommerfield, D. & Dougherty, J. (1996), Data mining using MLC++: A machine learning library in C++, *in* 'Tools with Artificial Intelligence', IEEE Computer Society Press, p. To Appear. `http://www.sgi.com/Technology/mlc`.

Kohavi, R. & Wolpert, D. H. (1996), Bias plus variance decomposition for zero-one loss functions, *in* L. Saitta, ed., 'Machine Learning: Proceedings of the Thirteenth International Conference', Morgan Kaufmann Publishers, Inc. Available at http://robotics.stanford.edu/users/ronnyk.

Koutsofios, E. & North, S. C. (1994), Drawing graphs with dot. Available by anonymous ftp from `research.att.com:dist/drawdag/dotdoc.ps.Z`.

Krogh, A. & Vedelsby, J. (1995), Neural network ensembles, cross validation, and active learning, *in* 'Advances in Neural Information Processing Systems', Vol. 7, MIT Press.

Langley, P., Iba, W. & Thompson, K. (1992), An analysis of bayesian classifiers, *in* 'Proceedings of the tenth national conference on artificial intelligence', AAAI Press and MIT Press, pp. 223–228.

Langley, P. & Sage, S. (1994), Induction of selective bayesian classifiers, *in* 'Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence', Morgan Kaufmann Publishers, Inc., Seattle, WA, pp. 399–406.

LeBlank, J., Ward, M. & Wittels, N. (1990), Exploring n-dimensional databases, *in* 'Proceedings of Visualization', pp. 230–237.

Littlestone, N. (1988), 'Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm', *Machine Learning* **2**, 285–318.

Maass, W. (1994), Efficient agnostic PAC-learning with simple hypotheses, *in* 'Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory', pp. 67–75.

Michalski, R. S. (1978), A planar geometric model for representing multidimensional discrete spaces and multiple-valued logic functions, Technical Report UIUCDCS-R-78-897, University of Illinois at Urbaba-Champaign.

Murthy, S. K., Kasif, S. & Salzberg, S. (1994), 'A system for the induction of oblique decision trees', *Journal of Artificial Intelligence Research* **2**, 1–33.

Quinlan, J. R. (1986), 'Induction of decision trees', *Machine Learning* **1**, 81–106. Reprinted in Shavlik and Dietterich (eds.) *Readings in Machine Learning*.

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, Inc., Los Altos, California.

Rice, J. A. (1988), *Mathematical Statistics and Data Analysis*, Wadsworth & Brooks/Cole.

Spector, P. (1994), *An Introduction to S and S-PLUS*, Duxbury Press.

Taylor, C., Michie, D. & Spiegalhalter, D. (1994), *Machine Learning, Neural and Statistical Classification*, Paramount Publishing International.

Thrun *et al.* (1991), The Monk's problems: A performance comparison of different learning algorithms, Technical Report CMU-CS-91-197, Carnegie Mellon University.

Weiss, S. M. & Kulikowski, C. A. (1991), *Computer Systems that Learn*, Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Wettschereck, D. (1994), A Study of Distance-Based Machine Learning Algorithms, PhD thesis, Oregon State University.

Wnek, J. & Michalski, R. S. (1994), 'Hypothesis-driven constructive induction in AQ17-HCI : A method and experiments', *Machine Learning* **14**(2), 139–168.

Wnek, J., Sarma, J., Wahab, A. A. & Michalski, R. S. (1990), Comparing learning paradigms via diagrammatic visualization, *in* 'Methodologies for Intelligent Systems, 5. Proceedings of the Fifth International Symposium', pp. 428–437. Also technical report MLI90-2, University of Illinois at Urbaba-Champaign.

Wolpert, D. H. (1992), 'Stacked generalization', *Neural Networks* **5**, 241–259.

A Installation, Registration, Questions

A.1 Legal Issues

SGI $\mathcal{MLC}$++ is provided "as is" without warranty of any kind, either expressed or implied. No one who has been involved in the creation, production, or delivery of this software and documentation shall be liable for any direct, incidental, or consequential damages resulting from use of the software or documentation, regardless of the theory of liability.

The entire risk as to quality, performance, or results due to use of the software is assumed by the user, and the software is provided without obligation of any kind to assist in its use, modification, or enhancement.

SGI $\mathcal{MLC}$++ is research domain. The specific terms detailing its use can be found in

`http://www.sgi.com/Technology/mlc/terms.html`

External inducers, for which we provide interfaces, have other restrictions; please contact the authors of those tools for specific permissions.

A.2 $\mathcal{MLC}$++ Installation

$\mathcal{MLC}$++ utilities are available through the world wide web at URL

`http://www.sgi.com/Technology/mlc`

Since version 1.3, the object code for the utilities is provided for Silicon Graphics hardware running IRIX 5.3 or 6.2.

Databases from UC Irvine that have been converted to our format (compatible with C4.5) are in

`ftp://starry.stanford.edu/pub/ronnyk/mlc/db`

Please look at the README file that comes with the distribution.

A.3 Questions, Problems, Bug Reports

Please fill in the registration form at URL **http://www.sgi.com/Technology/mlc/mail.html** The purpose of the registration form is to let us know who is working on $\mathcal{MLC}$++, and to allow you to optionally join a mailing list for discussions of problems.

Please look at the "known bugs" item in the $\mathcal{MLC}$++ home page for descriptions and workarounds of known bugs (http://www.sgi.com/Technology/mlc).

Questions, help requests, and bug reports should be addressed to `mlc@postofc.corp.sgi.com`. For bug reports, please execute the utilities with LOGLEVEL set to 1 or higher, and with DEBUGLEVEL 2.

Comments on this document, suggestions on how to improve the utilities are encouraged.

A.4 Dot and Dotty

$\mathcal{MLC}$++ interfaces **dot** and **dotty** from AT&T (Koutsofios & North 1994). These programs allow you to display graphs on the screen and to generate postscript for printing and inserting into documents.

A web server is available to download graph viewing tools. You can obtain precompiled binaries and source for dot, dotty, and some associated programs. The URL is

`http://www.research.att.com/sw/tools/reuse/`

Click on source or binary. If you accept the no-cost non-commercial license, then you'll be presented with a form to fill in, and then finally hotlinks from which to download the archives.

Please direct dot/dotty related questions directly to Stephen North (north@research.att.com).

A.5 Referencing $\mathcal{MLC}$++

A paper on $\mathcal{MLC}$++ will be published in the Tools with Artificial Intelligence conference (Kohavi, Sommerfield & Dougherty 1996). Please use it as the reference to $\mathcal{MLC}$++ if you are using some of the utilities provided by $\mathcal{MLC}$++ and would like to acknowledge this fact. LaTeX users can add the following to their bib file:

```
@inproceedings{mlc-new-intro,
   author = {Ron Kohavi and Dan Sommerfield and James Dougherty},
   title = {Data Mining Using {MLC++}: A Machine Learning Library in {C++}},
   booktitle={Tools with Artificial Intelligence},
   year = 1996,
   pages = {To Appear},
   note= {\texttt{http://www.sgi.com/Technology/mlc}},
   publisher={IEEE Computer Society Press}}
```

The LaTeX macros to display $\mathcal{MLC}$++ is:

```
\newcommand{\mlc}{\ensuremath{\mathcal{MLC\hspace{-.05em}\raisebox{.4ex}{\tiny\bf ++}}}}
```

B Common Error Messages

Shown below are common error messages and their explanations.

If there is a problem in parsing an input file, just SETENV LOGLEVEL 10 and you will see how each attribute is being parsed. This usually suffices to solve the problem.

1. ```
   6119:Inducer: rld: Fatal Error: cannot successfully map soname
   'libMWrapper.so' under any of the filenames
   /usr/lib/libMWrapper.so:/lib/libMWrapper.so
   ```

   $\mathcal{MLC}$++ uses dynamically shared objects. The runtime loader must know where these are through the environment variable `LD_LIBRARY_PATH`. Add

   setenv LD_LIBRARY_PATH "/usr/lib:/lib:$MLCDIR"

   After you set `MLCDIR` to the $\mathcal{MLC}$++ directory. Another alternative is to move all the files ending with `.so` into `/usr/lib`.

2. `Error - LinearDiscriminant::LinearDiscriminant: Number of categories 7 != 2.`

   The inducer you are running attempts to generate a LinearDiscriminant for more than two categories. Remember that perceptron and winnow are limited to two-class problems.

3. ```
   sh: c4.5: not found
   C4.5 program returned bad status.  Line executed was
   c4.5 -u -f /tmp/aaaa0063_.MLC  | awk -f $MLCDIR/c45test.awk...
   ```

   The csh or tcsh variable `path` does not include the c4.5 executable.

4. ```
   sh: awk: not found
   C4.5 program returned bad status.  Line executed was
      c4.5 -u -f /tmp/aaaa00636.MLC  | awk -f $MLCDIR/c45test.awk...
   ```

   The environment variable `$MLCDIR` is not defined to be the path where $\mathcal{MLC}$++ is installed. Specifically, c45 results are parsed using the c45test.awk script.
   ```

5. `Error - FileNames::test_file: No TESTFILE specified.`

   A TESTFILE must be specified if the DATAFILE ends with the ".all" suffix. If you are following the $\mathcal{MLC}$++ naming conventions, then you are probably doing something wrong. A ".all" suffix indicates that all your data is in the given file, so no TESTFILE should be available. You may be running the Inducer utility instead of the AccEst utility.

6. `Error - mlcIO::file_exists: File '.names' does not exist in colon separated paths '.:/u/mlc/db:'.`

   An empty DATAFILE was given.

C Differences from Previous Versions and Known Bugs

C.1 Differences from 1.3.2

1. The main change is the policy regarding the status of $\mathcal{MLC}$++. SGI $\mathcal{MLC}$++, which is $\mathcal{MLC}$++ 2.0 and above is not public domain anymore, but research domain. This means that it can be used for research purposes but cannot be used in any commercial product without prior agreement from Silicon Graphics. For more details, see the $\mathcal{MLC}$++ home page.

2. The preferred reference to $\mathcal{MLC}$++ changed from Kohavi, John, Long, Manley & Pfleger (1994) to Kohavi et al. (1996).

3. The distribution is compiled in FAST mode, which is about 30% faster than non-fast mode (non-fast mode was the mode previously distributed).

4. The utilities distribution is given using dynamically shared objects that save space. The compressed tar file is now about half the size of the last version.

5. Persistent categorizers are now supported. Persistent decision trees and Naive-Bayes are implemented. This allows a categorizer to be saved and later read in. Assimilation code allows instances with more attributes than required by the categorizer to be assimilated and categorized.

6. Decision trees were improved as follows:

   (a) Decision trees now provide pruning in a way similar to C4.5. Branch replacement is not being done and the fudge factors (C4.5 adds 0.1 in certain places are not in the code). The `MC4` inducer defaults to a setting very similar to C4.5's setting.
   (b) A new option, adjust thresholds, allow splits in decision trees to be adjusted to actual data elements as in C4.5. The option is implemented much more efficiently than in C4.5.
   (c) Quinlan's MDL penalty for continuous attributes (C4.5 rel 8) is now available.
   (d) Gain ratio is supported as a splitting criterion. This is implemented exactly as the C4.5 version (with all the hacks), so that except for unknown handling and tie breakers, the unpruned trees are the same.
   (e) More statistics are provided about the number of attributes and depth of tree.
   (f) Improved output for MineSet™ Tree Visualizer.

7. Naive-Bayes changes:

   (a) Naive-Bayes uses a value of 0 as a default for `NO_MATCHES_FACTOR`, which is the value used when there are no records matching a given attribute value and label value. This was made to make the probability distribution consistent and (surprisingly?) results sometimes improve. The previous default was 0.5 over the number of instances.

(b) Naive-Bayes now supports Laplace corrections.

(c) Naive-Bayes now outputs MineSet™ Evidence Visualizer format files.

8. The biasVar utility has been added for the bias-variance decomposition based on Kohavi & Wolpert (1996).

9. NBTree described in Kohavi (1996) can be setup with the following parameters:

```
setenv INDUCER catdt
setenv CATDT_LEAF_INDUCER naive
setenv LBOUND_MIN_SPLIT 30
setenv CATDT_CV_FOLDS 5
setenv CATDT_CV_TIMES 1
setenv CATDT_LEAF_NB_NO_MATCHES_FACTOR = 0.5
setenv IMPROVE_RATIO 0.05
```

10. A "dribble" support was added to show progress on big files. Dribble is done for decision trees and discretization.

11. Unlabelled instance lists are partially supported. The syntax is to say "nolabel" in the names file.

12. Options in OODG allow you to build an oblivious decision tree to determine the cumulative purity of chosen attributes.

13. Control-c and kill signals are caught and handled. A cleanup of temporary files is done.

14. Governors remove attributes with too many values and avoid runs with too many label values. These limits are artificial and can be increased by changing the appropriate options in the message. With this change, dynamic projections of instance lists are allowed during reading (mainly an efficiency issue).

15. Misc changes:

    (a) The URL for dotty has change with the breakup of AT&T.

16. Major source code changes:

    (a) At the source level Bag, BagCounter, List, and CounterList have been unified into one smart class, making a lot of the programming easier.

    (b) The GNU library is not being used any more.

    (c) Code is available to upload data into a database (Oracle, Sybase, and Informix variations are available) through database loaders (not a straight load, but the sql files are written out).

C.2 Differences from 1.3

1. Decision tree induction can now output to the tree visualizer from SGI's MineSet.

2. Utilities now support command-line flags (options) as described in Section 2.

3. Names files now support "discrete" as a description for nominal attributes. As data is read, the list of values is dynamically added. Values in the test set that were not seen are considered unknown.

4. The C45Tree utility now supports the DIST_DISP option for displaying the distribution at every node.

5. The "info" utility now gives the majority accuracy.

6. Temporary files are now erased on interrupts. The environment variable KEEPTEMP can be set to "yes" to override this behavior.

7. Heuristics for determining the number of bins for discretization were added.

8. A new discretization option (t2-disc) was added. T2 is now given with the distribution (for discretization).

9. A new discretization option (c4.5-disc) was added.

10. CN2 algorithm is now given with the distribution. We thank Rick Kufrin from NCSA for modifying the original CN2 so that it easily compiles on SGI.

11. Discretization intervals were changed to conform with C4.5. The left branch is now $\leq$ instead of $<$.

12. Weights in nearest neighbor can be set "manually," *i.e.*, through user input.

13. The C45test.awk script is no longer used nor needed.

C.3 Known Bugs

1. Trimming doesn't work properly in accuracy estimation.

2. If FSS_ACC_ESTIMATOR is set to "test" in feature subset selection then you'll get:
   `Error - AccData::check_real:  real accuracy is undefined.`

   The workaround is to do `setenv FSS_SHOW_REAL_ACC always`