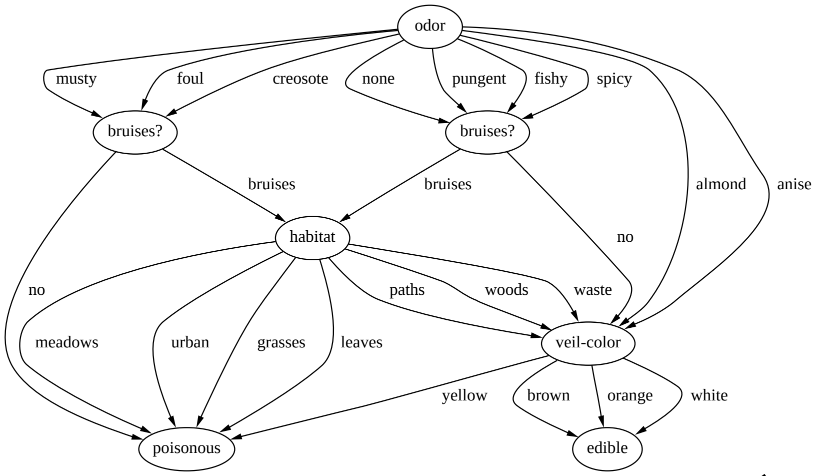
July 1, 1995



MLC++

Machine Learning library in C++

$\mathcal{MLC}$++ Utilities 1.2

By: Ronny Kohavi
and
Dan Sommerfield

mlc@CS.Stanford.EDU

The URL for $\mathcal{MLC}$++ is http://robotics.stanford.edu/users/ronnyk/mlc.html

Contents

$\mathcal{MLC}$++ Utilities V1.2

Ron Kohavi and Dan Sommerfield
Computer Science Department
Stanford University
Stanford, CA. 94305
`mlc@CS.Stanford.EDU`

July 1, 1995

`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 28 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 space. We recommend that you use $\mathcal{MLC}$++ with the LOGLEVEL set to 1, *i.e.*, type
>
> `setenv LOGLEVEL 1`
>
> and add it to your .login file. This will show you the possible options that can be set and display some extra information.

1 Introduction

$\mathcal{MLC}$++ utilities currently take their options from environment variables. 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 features of iris plants: sepal length, sepal width, petal length, and petal width. The task is to classify 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 in directory `/u/mlc/db/` consisting of iris.names, iris.data, and iris.test, one can type:

[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://robotics.stanford.edu:/users/ronnyk/mlc.html`

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

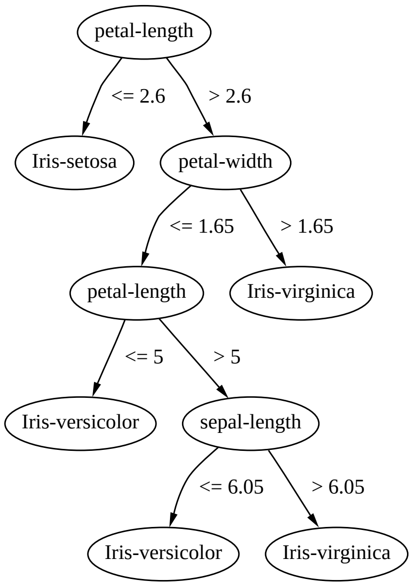


Figure 1: The file **iris.ps** depicting the decision tree induced by ID3 on the **iris** dataset.

```
setenv MLCPATH /u/mlc/db          # Where to search for the files
setenv DATAFILE iris              # The dataset stem
setenv DISP_CONFUSION_MAT yes     # Show confusion matrix
setenv DISPGRAPH yes              # Show the resulting graph (requires dotty)
ID3                               # ID3.dot is always produced
dot -Tps -Gpage="8.5,11" -Gmargin="0,0" ID3.dot > iris.ps # Generate postscript
                                                          #   from ID3.dot
```

The output is:

```
Tree has 9 nodes, and 5 leaves.
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 28), 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 $\pm$ 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:

```
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:

```
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

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.

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).
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**.

Option name	Domain	Default	Explanation
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.
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.

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.

Option name	Domain	Default	Explanation
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. 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.

$\mathcal{MLC}$++ 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.

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

Option name	Domain	Default	Explanation
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.

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

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

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 classify specific instances, only a set of instances. All external inducers, which are interfaced through $\mathcal{MLC}$++ (*e.g.*, C4.5, PEBLS, IB, and OC1) 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. Features 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 features. 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).

4.3 Perceptron and Winnow

Perceptron and Winnow are inducers that build linear discriminators. They are only capable of handling continuous features with no unknowns and two-class problems. For discrete data, you can use the "conv" utility to convert the input features 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 Herz, Krogh & Palmer (1991). Winnow uses the algorithm described in Littlestone (1988).

4.4 ID3

ID3 is a very basic decision tree algorithm with no pruning and a simple unknown-handling mechanism (separate value). If you are seriously interested in decision trees, use C4.5. This is not meant to be a serious decision tree induction algorithm, at least if used without feature subset selection, which indirectly provides the pruning (Kohavi & Sommerfield 1995).

The MIN_SPLIT_WEIGHT is the minimum percent of training instances 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.

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.

4.5 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 features. 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 features in isolation.

4.6 Instance Based Algorithms

IB is an instance-based inducer (Aha 1992, Wettschereck 1994). A good, robust algorithm, but still slow. A new version with kd-trees is now being implemented.

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.

4.7 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.

4.8 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 feature. 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.

4.9 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. Try the following for some interesting graphs (look at Inducer.dot output using dotty):

```
setenv INDUCER disc-filter
setenv DISCF_INDUCER fss
setenv DISCF_FSS_INDUCER hoodg
setenv DISCF_FSS_MAX_STALE 3
setenv DISCF_FSS_CV_TIMES 0
setenv DISCF_FSS_CV_FOLDS 5
setenv DISCF_FSS_CMPLX_PENALTY 0.0001
setenv DISCF_FSS_SHOW_REAL_ACC never
setenv REMOVE_UNKNOWN_INST yes
setenv DISPLAY_STRUCT dot
Inducer
```

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 features 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 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.

4.12 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/.

Two changes were made to the original source code: The maximum number of instances was increased from 110 to 5,000 in config.h, and the program was changed to return status 0 upon exit if the run was successful.

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

4.13 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. Unpublished.

Order-fss searches for an attribute ordering. Current research for OODGs.

Disc-search is a wrapper discretizer that searches for the best number of intervals for each feature. Too slow, but will probably be rewritten soon with a more efficient algorithm.

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, or bin. The entropy discretization seems to be the best discretization method from the allowed options for most practical datasets (Dougherty et al. 1995). It requires the MIN_SPLIT, the minimum number of instances in an interval, and INITIAL_VAL, the number of intervals to create (the default of 0 invokes the heuristic described in Fayyad & Irani (1993)). 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). INITIAL_VAL determines the number of bins (the default of 0 invokes a heuristic that discretizes into 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).

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 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.

Option name	Domain	Default	Explanation
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_EPSILON_INSERT	$[-1, 1]$	0.0	Boost (negative value) or penalty (positive value) to add to estimate when computing compound node for an inserted feature.
FSS_EPSILON_DELETE	$[-1, 1]$	0.0	Boost (negative value) or penalty (positive value) to add to estimate when computing compound node for a deleted feature.
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 iris dataset, one can do:

```
setenv LOGLEVEL 1
setenv INDUCER FSS
setenv FSS_INDUCER IB
setenv DATAFILE    iris
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_DOT_FILE = FSS.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_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
```

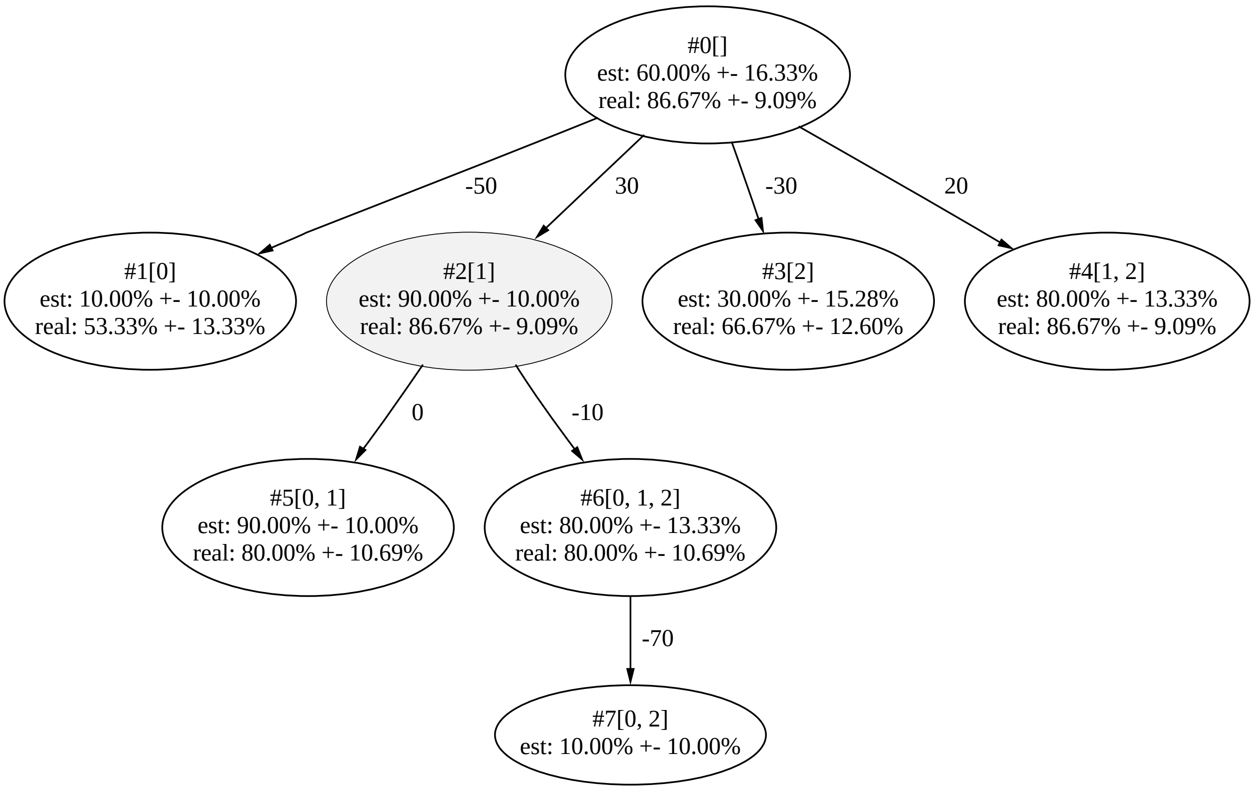


Figure 2: The search space for IB on the iris dataset

```
OPTION FSS_DIRECTION = forward
OPTION DATAFILE = iris
OPTION NAMESFILE = iris.names
OPTION REMOVE_UNKNOWN_INST = false
OPTION CORRUPT_UNKNOWN_RATE = 0
Reading iris.data. done.
OPTION TESTFILE = iris.test
Reading iris.test done.
new best node (1 evals) #0[]: accuracy: 23.00% +- 3.67% (10.00% - 50.00%).
Test Set: 30.00% +- 6.55% [19.10% - 43.75%].  Bias: -7.00% cost: 10 complexity: 0
new best node (7 evals) #5[2, 3]: accuracy: 97.00% +- 1.53% (90.00% - 100.00%).
Test Set: 98.00% +- 2.00% [89.50% - 99.65%].  Bias: -1.00% cost: 10 complexity: 0
final best node #5[2, 3]: accuracy: 97.00% +- 1.53% (90.00% - 100.00%).
Test Set: 98.00% +- 2.00% [89.50% - 99.65%].  Bias: -1.00% cost: 10 complexity: 0
expanded 7 nodes
Accuracy: 98.00% +- 2.00% [89.50% - 99.65%]
```

This example shows that one can improve the accuracy to (98%) by looking at only two features. Figure 2 shows the nodes visited and their information. The graph is automatically stored in the file FSS.dot.

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 Inducer

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

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.
DISPLAY_STRUCT	none, ASCII, dot	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).
DIST_DISP	yes, no	no	Display distribution at nodes. This option will be available only when relevant.

6.2 ID3

The **ID3** utility is similar to Inducer, except that ID3 is always executed and the DISPGRAPH option will pop the decision tree on your screen if dotty is installed. The default for ID3_UNKNOWN_EDGES is false because this utility is usually used to look at tree, which are more comprehensible without the unknown edges.

6.3 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.

6.4 AccEst

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

6.5 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 features or even continuous features which simply have very few values.

Converting features 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 22 seconds on our Sparc10 if the features are declared continuous and 6 seconds if they are declared nominal. Minor accuracy differences may result due to slightly different ways of handling such features.

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 .all = 57 (duplicate or conflicting instances : 0)
Number of instances in labor-neg.data = 40 (duplicate or conflicting instances : 0)
Number of instances in .test = 17 (duplicate or conflicting instances : 0)
.all file - Class probability for the label 'good' : 64.91%
.all file - Class probability for the label 'bad' : 35.09%
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
```

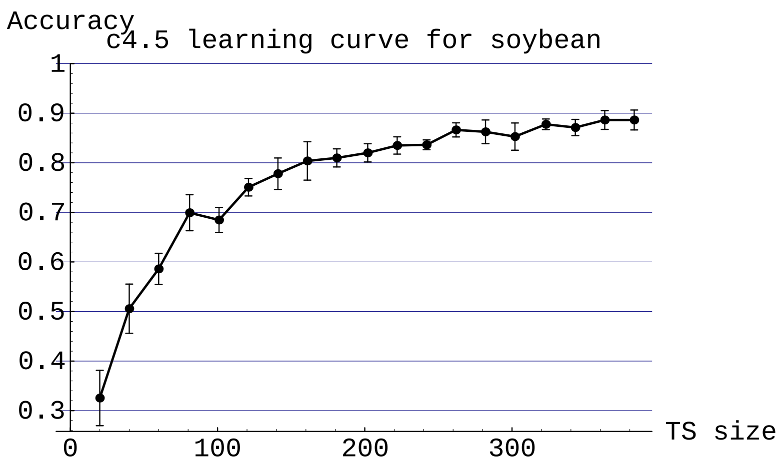


Figure 3: The learning curve for C4.5 on soybean-large

```
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
```

6.6 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.
NUM_TEST_SIZE	int > 0	0	Minimum number of dataset instances to save for testing. If 0, then this is the number of intervals in the 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 $\mathcal{MLC}$++.

Option name	Domain	Default	Explanation
MATH_DISPLAY	yes/no	no	Output data line that can be copied into Mathematica. LearnCurve.m can then be executed in Mathematica to generate the appropriate graph and statistics.

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
LearnCurve
```

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, 50.57% +- 2.48%
    60, 58.60% +- 1.57%
    81, 69.97% +- 1.74%
   101, 68.48% +- 1.26%
   121, 75.09% +- 0.88%
   141, 77.80% +- 1.58%
   161, 80.37% +- 1.93%
   181, 80.98% +- 0.91%
   202, 81.73% +- 0.83%
   222, 83.49% +- 0.87%
   242, 83.62% +- 0.50%
   262, 86.56% +- 0.76%
```

```
282, 86.24% +- 1.20%
302, 85.28% +- 1.37%
323, 87.68% +- 0.57%
343, 87.11% +- 0.82%
363, 88.63% +- 0.95%
383, 88.63% +- 1.00%
```

Figure 3 on page 20 shows the Mathematica graph generated by running `LearnCurve.m` on the generated data.

6.7 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.

6.8 Discretization

The utility "discretize" 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 more 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.

6.9 Conversions

The utility "conv" 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 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).

Option name	Domain	Default	Explanation
ATTR_DELIM	space, comma, period, semicolon, colon	comma	the attribute delimiter.
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.

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`)
   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 $\mathcal{MLC}$++. Many students have worked and continue to work on $\mathcal{MLC}$++, including: Robert Allen, Brian Frasca, James Dougherty, Steven Ihde, Ronny Kohavi, Jamie Chia-Hsin Li, Richard Long, David Manley, Svetlozar Nestorov, Mehran Sahami, Dan Sommerfield, and Yeogirl Yun. $\mathcal{MLC}$++ was partly funded by ONR grant N00014-94-1-0448 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.

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.

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* "Machine Learning: Proceedings of the Twelfth International Conference". Available by anonymous ftp from `starry.Stanford.EDU:pub/ronnyk/disc.ps`.

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, pp. 1022–1027.

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

Geman, S. & Bienenstock, E. (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.

Herz, 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. Available by anonymous ftp from:
`starry.Stanford.EDU:pub/ronnyk/ml94.ps`.

Kohavi, R. (1994*a*), Bottom-up induction of oblivious, read-once decision graphs, *in* "Proceedings of the European Conference on Machine Learning". Available by anonymous ftp from
`starry.Stanford.EDU:pub/ronnyk/euroML94.ps`.

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. Available by anonymous ftp from
`Starry.Stanford.EDU:pub/ronnyk/aaai94.ps`.

Kohavi, R. (1994*c*), Feature subset selection as search with probabilistic estimates, *in* "AAAI Fall Symposium on Relevance", pp. 122–126. Available by anonymous ftp from:
`starry.Stanford.EDU:pub/ronnyk/aaaiSymposium94.ps`.

Kohavi, R. (1995*a*), The power of decision tables, *in* N. Lavrac & S. Wrobel, eds, "Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning, 1995)", Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, Heidelberg, New York, pp. 174 – 189. Available by anonymous ftp from `starry.Stanford.EDU:pub/ronnyk/tables.ps`.

Kohavi, R. (1995*b*), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* "Proceedings of the 14th International Joint Conference on Artificial Intelligence". Available by anonymous ftp from
`starry.Stanford.EDU:pub/ronnyk/accEst.ps`.

Kohavi, R. & John, G. (1995), Automatic parameter selection by minimizing estimated error, *in* "Machine Learning: Proceedings of the Twelfth International Conference".

Kohavi, R. & Li, C.-H. (1995), Oblivious decision trees, graphs, and top-down pruning, *in* "Proceedings of the 14th International Joint Conference on Artificial Intelligence". Available by anonymous ftp from `starry.Stanford.EDU:pub/ronnyk/eodg.ps`.

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". Available by anonymous ftp from `starry.Stanford.EDU:pub/ronnyk/fssWrapper.ps`.

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. Available by anonymous ftp from: `starry.Stanford.EDU:pub/ronnyk/mlc/toolsmlc.ps`.

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

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.

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

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, 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.

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

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

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

A Installation, Registration, Questions

A.1 Legal Issues

$\mathcal{MLC}$++ 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.

$\mathcal{MLC}$++ is public domain and can be freely used (including for commercial purposes). The sources are freely available from our anonymous ftp site. Some 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 by anonymous ftp to **starry.stanford.edu** in **pub/ronnyk/mlc/util.** Please look at the README file there. Databases from UC Irvine that have been converted to our format (compatible with C4.5) are in **pub/ronnyk/mlc/db.** General information about the library (including the installation kit) can be obtained through the world wide web at URL

http://robotics.stanford.edu:/users/ronnyk/mlc.html

$\mathcal{MLC}$++ utilities are currently provided in executable form only for Sun platforms. They should work on SunOS 4.1.3 (1.1.3) and on Solaris 2.3 (5.3).

A.3 Questions, Problems, Bug Reports

Please fill in the mosaic form at URL **ftp://starry.stanford.edu/pub/ronnyk/mlc/mlc-register.html** or send the mlc-register.txt file available with the installation kit to **mlc@cs.stanford.edu**. 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://robotics.stanford.edu/users/ronnyk/mlc.html).

Questions or help requests related to the utilities should be addressed to `mlcpp-help@CS.Stanford.EDU`. Bug reports should be sent to `mlcpp-bugs@CS.Stanford.EDU`. 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, and requests for additional utilities should be sent to **mlc@CS.Stanford.EDU**.

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/orgs/ssr/book/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 $\mathcal{MLC}$++ was published in the Tools with Artificial Intelligence conference (Kohavi, John, Long, Manley & Pfleger 1994). 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-intro,
   author = {Ron Kohavi and George John and Richard Long and David Manley
             and Karl Pfleger},
   title = {{MLC++}: A Machine Learning Library in {C++}},
   booktitle={Tools with Artificial Intelligence},
   year = 1994,
   pages = {740--743},
   publisher={IEEE Computer Society Press},
   note = {Available by anonymous ftp from: \\
          {\tt starry.Stanford.EDU:pub/ronnyk/mlc/toolsmlc.ps}}}
```

B Common Error Messages

Shown below are common error messages and their explanations.

1. `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.

2. ```
   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.

3. ```
   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.

4. `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.

5. ```
   Error - mlcIO::file_exists: File '.names' does not exist in colon separated
      paths '.:/u/mlc/db:'.
   ```

   An empty DATAFILE was given.
   ```

C Differences from Version 1.1

The changes to the library (since version 1.1) include:

1. The code executes faster and the data files requires less memory.
2. Discretization is now supported. Both a utility and a discretization filter inducer are provided.

   The new discretization code (updated from 1.1.1) stores the actual intervals as discrete values, prefixing the period by a backslash. This makes the datafile easier to interpret and allows the dot displays to look reasonable (with ranges on the edges).
3. Feature subset selection (search in general) supports compound operators (Kohavi & Sommerfield 1995).
4. C4.5 with automatic parameter tuning was added (Kohavi & John 1995).
5. Bagging was added (Breiman 1994, Wolpert 1992).
6. An entropy-based decision graphs inducer was added (Kohavi & Li 1995).
7. Perceptron inducer added.
8. Winnow inducer added.
9. New external interfaces are provided (base inducers):
   (a) C4.5rules is now interfaced too (requires C4.5).
   (b) New interface to PEBLS (Cost & Salzberg 1993).
   (c) New interface to OC1 (Murthy et al. 1994).
   (d) New interface to ib1-4 (Aha 1992).
10. The "Inducer" utility can display confusion matrices, misclassified instances, and the structure of the induced classifier.
11. The "info" utility displays basic information about a dataset.
12. C45Tree utility to display trees from C4.5.
13. Minor changes:
   (a) Minor changes in option names. Please set the LOGLEVEL to 1 to make sure your options are correctly set.
   (b) Automatic wrapping is not done if output is redirected to a file. This sometimes allows for easier automatic parsing of the output.
   (c) Accuracy estimator now defaults to cv, not strat-cv.
   (d) The conv utility now declares the attributes as continuous (used to be nominal). This is because the main reason for using it is for Perceptron/Winnow or Neural net packages.
   (e) Very minor bug fix in the 1R discretizer's handling of unknowns.
   (f) TEMP_FILE_NAME option is not used any more. We call the tmpnam() function and add a ".MLC" suffix.