Inducer: a Rule Induction Workbench for Data Mining

Max Bramer
Faculty of Technology
University of Portsmouth
Portsmouth, UK
Email: Max.Bramer@port.ac.uk Fax: +44-2392-843030

Abstract

One of the key technologies of data mining is the automatic induction of classification rules from examples. A variety of methods have been proposed and many comparative studies of methods have been carried out. However the absence of a widely available common platform has made it difficult to carry out sufficiently in-depth studies to establish the superiority of one method over another.

This paper describes ***Inducer***, a rule induction workbench which aims to provide a common platform with a graphical user interface for analysing a variety of rule induction algorithms and related strategies with a range of datasets, without the need for any programming by the user. The principal features of ***Inducer*** are described with reference to two well-known rule induction algorithms. The paper concludes with an example of its use to compare these algorithms' performance when there are missing data values.

1 Introduction

The growing commercial importance of knowledge discovery and data mining techniques has stimulated new interest in the automatic induction of classification rules from examples, a field in which research can be traced back at least as far as the mid-1960s [1].

Many practical decision-making tasks can be formulated as classification problems, for example

- Customers who are likely to buy or not buy a particular product in a supermarket
- People who are at high, medium or low risk of acquiring a certain illness
- Student projects worthy of a distinction, merit, pass or fail grade
- Objects on a radar display which correspond to vehicles, people, buildings or trees.

In many fields, large collections of examples (often collected for other purposes) are readily available and automatically generating classification rules for such tasks has proved to be a realistic alternative to the standard Expert System approach of eliciting decision rules from experts. Reference [2] reports two large applications of 2,800 and 30,000+ rules, developed using automatic rule induction techniques in only one and 9 man-years, respectively, compared with the estimated 100 and 180 man-years needed to develop the celebrated 'conventional' Expert Systems Mycin and XCON.

Most work in this field to date has concentrated on generating classification rules in the intermediate form of a decision tree using variants of the TDIDT (Top-Down Induction of Decision Trees) algorithm [3]. Although the decision tree representation is widely used, it suffers from the fundamental problem that the corresponding classification rules can be excessively complex owing to the need for the rules invariably to 'link together' to form a tree structure. As rule induction techniques are applied to larger and larger commercial datasets this weakness is likely to become increasingly important.

One approach to overcoming this problem is the rule induction algorithm Prism [4,5], which generates classification rules directly, term by term from the training set without using the intermediate representation of a decision tree.

Algorithms for generating classification rules using other approaches (such as neural networks) have also been developed but will not be considered further in this paper.

Although there are many comparative studies of different algorithms reported in the research literature, the absence of a widely available common platform

for this work has inevitably made it difficult to carry out sufficiently in-depth studies to establish the relative strengths and weaknesses of different methods for particular types of application domain.

Two systems which seek to provide a common platform for the experimenter are MLC++ [6], a library of C++ classes for supervised machine learning which can be incorporated into a user's own programs, and WEKA [7], a library of machine learning algorithms written in Java which can be run from a Java command line.

The ***Inducer*** rule induction workbench is concerned specifically with the generation of classification rules and is aimed at investigators who prefer to use a graphical interface without the need for any programming or use of Unix commands. The package incorporates a wide range of user options and also permits the creation of a number of output files to facilitate further analysis.

The current version of ***Inducer*** includes implementations of TDIDT and N-Prism (a revised version of Prism, as described in [5]), as representatives of two of the most important classes of algorithm for automatic induction of classification rules. Further algorithms and additional user facilities will be added in future versions.

Following a brief introduction to the basic TDIDT and Prism algorithms, this paper goes on to describe the principal features of ***Inducer.*** A brief description of a series of experiments to compare TDIDT and N-Prism within the common framework of ***Inducer*** is also given.

2 Automatic Induction of Classification Rules

2.1 Example and Basic Terminology

The following example is taken from [3]. Table 1 records a golf player's decisions on whether or not to play each day based on that day's weather conditions.

Outlook	Temp (°F)	Humidity (%)	Windy	Class
sunny	75	70	true	play
sunny	80	90	true	don't play
sunny	85	85	false	don't play
sunny	72	95	false	don't play
sunny	69	70	false	play
overcast	72	90	true	play
overcast	83	78	false	play
overcast	64	65	true	play
overcast	81	75	false	play
rain	71	80	true	don't play
rain	65	70	true	don't play
rain	75	80	false	play
rain	68	80	false	play
rain	70	96	false	play

Table 1. The Golf Training Set

What combination of weather conditions determines whether the decision is to play or not to play?

The standard terminology is to call Table 1 a *training set* and each of its rows an *instance*. Each instance comprises the values of a number of *attributes* (variables) and the value of a corresponding *classification*. Attributes can be either *categorical* (such as outlook) or *continuous* such as humidity. However, continuous attributes need to be split into a discrete set of ranges (e.g. humidity <= 75 or > 75) before use.

One possible set of classification rules that can be derived from Table 1 is as follows:

1. IF outlook = sunny AND humidity <= 75
   THEN Class = play
2. IF outlook = sunny AND humidity > 75
   THEN Class = don't play
3. IF outlook = overcast THEN Class = play
4. IF outlook = rain AND windy = true
   THEN Class = don't play
5. IF outlook = rain AND windy = false
   THEN Class = play

2.2 Inducing Decision Trees: The TDIDT Algorithm

The TDIDT algorithm constructs a set of classification rules via the intermediate representation of a decision tree.

The algorithm begins by choosing an attribute and then divides the training set into subsets corresponding to each of its possible values. Each resulting branch of the tree then leads to either a leaf node, if all the instances in the corresponding subset have the same classification or otherwise a subtree constructed using the same algorithm recursively.

A possible decision tree corresponding to the above table is shown in Figure 1.

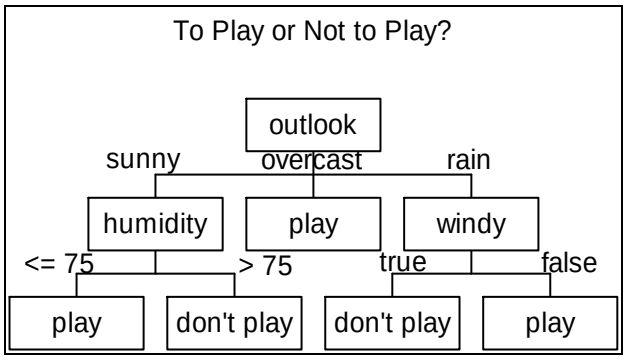


Figure 1. Decision Tree Corresponding to Table 1

This tree corresponds to the five classification rules given in Section 2.1, one rule per branch.

In general there are many possible decision trees and thus many different sets of rules corresponding to any given training set. The aim is to generate the set of classification rules that gives the best possible level of predictive accuracy when applied to a table of previously unseen instances, known as a *test set.*

The TDIDT algorithm can be summarised as follows.

IF all cases in the training set belong to the same class
THEN return the value of the class
ELSE
(a) select an attribute to split on *
(b) sort the instances in the training set into non-empty subsets, one for each attribute value
(c) return a tree with one branch for each subset, each branch having a descendant subtree or a class value produced by applying the algorithm recursively for each subset in turn.

* No attribute may be selected more than once in any branch.

Provided that no two instances with the same values of all the attributes belong to different classes, any method of selecting attributes at step (a) will suffice to produce a decision tree. However the predictive value of the corresponding set of classification rules on unseen instances in a test set may depend critically on how it is done. The most common attribute selection criterion is probably *Information Gain* [3], which uses the information theoretic measure *entropy* at each stage to split on the attribute which maximises the expected gain of information from applying the additional test. This is the approach adopted in the well-known system C4.5 [3].

2.3 Inducing Decision Rules: The Prism Algorithm

The basic Prism algorithm can be summarised as follows, assuming that there are n (>1) possible classes. Further details are given in [5].

For each class i from 1 to n inclusive:

(1) Calculate the probability that class = i for each attribute-value pair
(2) Select the attribute-value pair with the maximum probability and create a subset of the training set comprising all instances with the selected combination (for all classes)
(3) Repeat 1 and 2 for this subset until it contains only instances of class i. The induced rule is then the conjunction of all the attribute-value pairs selected in creating this subset
(4) Remove all instances covered by this rule from the training set

Repeat 1-4 until all instances of class i have been removed

3 The Inducer Rule Induction Workbench

3.1 Introduction to Inducer

The ***Inducer*** rule induction workbench [8] is one of a suite of packages developed to facilitate experiments with different techniques for generating classification rules. ***Inducer*** is implemented in Java (version 1.1) in the interests of portability and is available both as a standalone application and also as an applet.

The package was implemented partly for teaching purposes but principally to facilitate practical experimentation with a range of rule induction algorithms and associated strategies. It is written in a modular fashion to enable further algorithms and strategies to be added relatively easily in the future.

Figure 2 shows a screen image of Inducer running the hypothyroid dataset ('hypo') from the repository of machine learning datasets at the University of California at Irvine (UCI) [9].

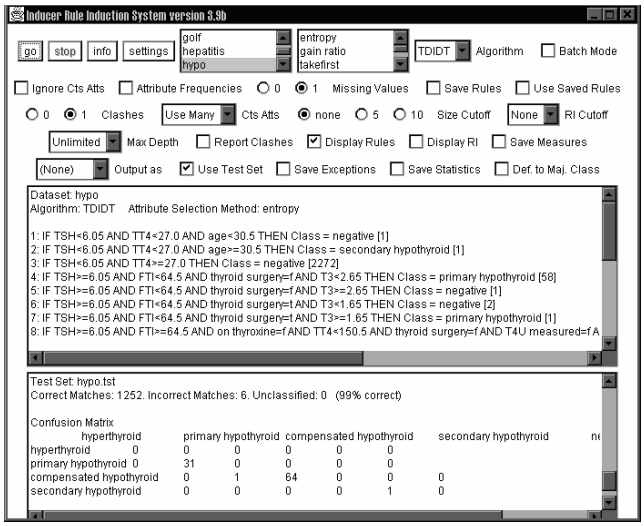


Figure 2. Inducer Screen Image

3.2 Basic Use

Inducer is designed to be easy to use. It has a graphical user interface which enables the expert user to select from a wide range of options, but for the inexperienced user it requires only two mouse clicks to obtain a set of classification rules: one to select a dataset from a menu in the top middle of the screen and another to press the *go* button in the top left-hand corner.

The default rule generation algorithm used is TDIDT, with entropy (or information gain) as the attribute selection criterion, as described in Section 2.2 above. Other parameters all have reasonable default values compatible with these.

There are limitations on the maximum size of training set and test set that can be processed, the maximum number of rules that can be generated etc. These are platform dependent and can be obtained by the user pressing the *info* (information) button, which displays the contents of file *inducer.inf*. Execution times are typically no more than a few seconds for datasets of the size of most of those in the UCI Repository.

Having generated a set of classification rules ***Inducer*** then runs the rules first against the original training set and then against a test set of previously unseen data (provided that such a test set exists and that the *Use Test Set* option has been selected).

For each training and test set examined ***Inducer*** calculates and displays a *confusion matrix* with one row and column per classification and one additional column corresponding to unclassified instances. If there are N possible classifications the matrix is thus of N rows by (N+1) columns.

The entry for row i, column j corresponds to the number of instances in the dataset with a correct classification of i and a computed classification of j. Entries in column N+1 occur when the induced rule set is unable to make a classification of a given instance.

In the case of a perfect classification all non-zero entries in the confusion matrix will occur on the leading diagonal of the main N x N matrix, with none in the rightmost ('unclassified') column. This will generally occur when the classification rules are run against the original training set which was used to generate them, but this is not invariably the case, e.g. if cutoffs (as described in Section 3.4.2) were applied during the rule generation process.

As well as the confusion matrix, ***Inducer*** displays the number of correct matches, incorrect matches and unclassified instances and the percentage of correct matches.

The options available for the experienced user are described in Sections 3.3 to 3.11 below. The current values of all system parameters, including those not displayed on the screen, can be obtained by the user by pressing the *settings* button at any time.

3.3 Data Input Parameters

3.3.1 Dataset Selection

There are currently 24 datasets available for selection from the default *input file directory*. These are mainly taken from the UCI Repository [9]. Alternative input file directories can be specified if preferred. The format of input files is essentially that specified in [3]. Each dataset comprises a name file, a training set and in most cases also a test set. The first line of the name file gives a list of all possible classifications. Subsequent lines give details of each of the attributes in turn, with the name of the attribute followed by either a list of its possible values, in the case of a categorical attribute, or the word *continuous*, denoting a numerical attribute, or *ignore*. The facility to specify an attribute as *ignore* is a valuable one, enabling the user easily to experiment with the effect of 'turning off' one or more attributes without having to change the data.

Training sets and test sets have the same format. Each

record corresponds to one instance and comprises the values of each of the attributes in the order in which they are listed in the name file, separated by commas as delimiters, followed by the corresponding classification as the last field in the record.

3.3.2 Ignore Continuous Attributes

The TDIDT algorithm as implemented in ***Inducer*** has a facility for local discretization of continuous attributes, i.e. dividing the values of an attribute X into two parts, $X<a$ and $X>=a$, say, at each stage of the tree generation process. However, many rule induction algorithms including N-Prism have no facilities for dealing (directly) with continuous attributes and it is sometimes helpful for the user to be able to 'turn off' such attributes, effectively treating them as if they were specified as *ignore* attributes in the name file.

3.3.3 Missing Value Strategy

For many practical applications the value of some attributes (or even in some cases the correct classification) may not be available for some or perhaps even all of the instances. An important practical requirement of a rule induction algorithm in many domains is the ability to make an accurate prediction of the classification of unseen test data even when there are missing values in the training set, the test set or both. Missing values in both training and test sets are conventionally identified by the symbol '?'.

Two missing value strategies are available in ***Inducer***.

(a) Ignore any instance in either the training or the test set that contains one or more missing values.
(b) Replace any missing values for a categorical attribute or for the classification by the most frequently occurring (non-missing) value for that attribute or classification and any missing values for a continuous attribute by the average of the (non-missing) values of that attribute. This is the default setting.

3.4 Rule Generation Parameters

3.4.1 Attribute Selection Method

There are five attribute selection criteria available when the TDIDT algorithm is selected, the last three being provided for teaching and comparative purposes only.

(1) *entropy* (or *information gain*) - the default
(2) *gain ratio* - a measure devised by Quinlan [3] aimed at overcoming the bias inherent in the use of information gain towards selecting attributes with a large number of values
(3) *takefirst* - work through the classes specified in the name file in order, from left to right
(4) *takelast* - as (3) but work from right to left
(5) *random* - select at random.

3.4.2 Cutoffs During Rule Generation

A problem that often arises during rule generation is the *over-fitting* of rules to data.

A rule such as

IF a = 1 AND b = 1 AND c = 3 AND d = 2 THEN Class=3

which is correct but corresponds to only a small number of instances in the training set may be of little value in predicting the classification for unseen data.

Generalising such a rule by stopping the rule generation process before the left-hand side is complete, e.g. *IF a = 1 AND b = 1 THEN Class=3* may be less accurate as far as the training set is concerned but of considerably more value when classifying data in an unseen test set.

Inducer has facilities for the user to specify two different types of cutoff during rule generation:

(a) a *depth cutoff* after either 1, 2 or 5 terms have been generated for a given rule (the default is 'unlimited')
(b) a *size cutoff* if the number of instances in the training set currently under consideration is below either 5 or 10 (the default is zero).

For both types of cutoff the result is a partly complete left-hand side of a rule, together with a subset of the training set containing instances with more than one classification, for which further subdivision has been inhibited. The rule is then either discarded or a single classification is taken depending on the *clash handling strategy* selected, as described in Section 3.4.4 below.

3.4.3 Discarding Rules on the Basis of 'Interestingness'

A topic of growing importance in recent years is that of rule 'interestingness' [10], the aim of which is to identify those rules in a generated ruleset which are likely to be of value in classifying unseen instances. There are many measures of rule interestingness

available, a basic one being Piatetsky-Shapiro's RI measure [11], which gives the difference between the actual number of instances matched by the rule in the training set and the number that would be expected by chance. ***Inducer*** allows the user to specify that a rule should be discarded (and not displayed) if the value of RI is less than zero, one or five. The default is 'never discard'.

3.4.4 Handling Clashes During Rule Generation

A *clash* occurs when a rule induction algorithm encounters a subset of the training set which has more than one classification but which cannot be further processed, either because there are no more attributes available or because of a cutoff. Such a subset is called a *clash set*.

Two strategies are currently implemented in ***Inducer*** for dealing with clashes encountered during rule generation:

(a) discard all instances in the clash set
(b) (default setting) treat the clash set as if all the instances in it had the same classification, i.e. the one that occurs most frequently. In the case of TDIDT a new rule is created. For N-Prism the rule is created only if the most frequently occurring classification is the one for which rules are currently being generated, otherwise the instances are discarded.

3.4.5 Dealing with Continuous Attributes

Inducer has a parameter that specifies how continuous attributes are to be handled during rule generation (TDIDT only):

(a) the attribute may be used once only in any given rule, as for categorical attributes
(b) (the default setting) the attribute may be used, i.e. subdivided into ranges, more than once in a rule if necessary, e.g.
IF a = yes AND x < 6.4 AND b=3 AND x < 3.2 THEN Class = 2

Inducer automatically combines multiple ranges of a continuous attribute; e.g. the above rule would be treated as
IF a = yes AND x < 3.2 AND b = 3 THEN Class = 2

3.5 Rule Execution Parameters

A 'default to majority class' facility is provided to force ***Inducer*** to classify all instances in the test set. Instances that would otherwise be unclassified are assigned to the most commonly occurring class. This is likely to be of practical value in domains where it is important to maximise the number of correct classifications and there is little or no penalty for incorrect ones. The default setting is false.

3.6 Display Parameters

Options are available (all defaulting to 'false') to display:

(a) the frequency of occurrence of each value of each categorical attribute, together with the minimum, maximum and average values of each continuous attribute
(b) the value of the RI measure for each rule
(c) details of clashes encountered during rule generation.

There is also an option to suppress the display of rules to the screen. This can speed up execution considerably. The total number of rules and terms and the average number of terms per rule are still displayed.

3.7 Saved Rule Parameters

Two parameters are available in connection with the saving of rules (the default for both is false):

(a) Save Rules - Save rules in a *saved rule file* in a coded form, together with the values of the main system parameter settings. Rule files are output to the *saved rules directory*, which is set by the ***Inducer*** initialisation file (see Section 3.10).
(b) Use Saved Rules - Read in rules from a saved rule file for the specified dataset. No processing of the training set takes place and the values of all rule generation parameters are ignored.

3.8 Log File

Every time the *go* button is pressed details of the main system parameter settings are recorded in a log file *inducer.log*, together with the number of rules and terms generated and the number of instances correctly classified, incorrectly classified or unclassified in the training set and (if applicable) the test set.

3.9 Other Output Files

The default for all four selections is false. All output files are saved to the *output files directory*, which is set by the ***Inducer*** initialisation file.

(a) A number of interestingness measures associated with each rule, including RI, are output to a file.
(b) Statistics giving the number of correctly classified, incorrectly classified and unclassified instances and the confusion matrix are output for the training set and (if selected) the test set for the chosen dataset.
(c) Detailed information about any misclassified instances can be output to an 'exceptions' file.
(d) Selecting the 'rule output to file' option enables the user to output the rules for possible use by another language or package. Three output formats are available:
    - a Java method
    - a set of Prolog clauses
    - (TDIDT only) a script file suitable for input to the SPSS flowcharting package *AllClear*.

3.10 Initialisation File

The initialisation file *inducer.ini* is invoked automatically when the application version of ***Inducer*** starts executing. It is used to specify the location of the input, output and rule file directories and the file extensions for the name, training and test sets, plus the saved rules file, the 'output rules' file and the rule interestingness measures, exceptions and statistics files.

For the applet version an equivalent effect to the initialisation file is achieved by embedding the information in the web page used to invoke the applet using the <PARAM> tag.

3.11 Batch Mode

An option is available (default setting false) to run ***Inducer*** in batch mode. When this is selected details of the input file, output file and saved rule file directories are taken from the file *inducer.bat*. These are followed by an unlimited number of triples specifying a name file, a training set and a test set. Pressing the *go* button causes ***Inducer*** to run on each of these triples of files in turn. All other parameter settings are taken from the graphical interface.

Running ***Inducer*** in batch mode enables a substantial series of experiments, say with a fixed name file and training set and a range of different test sets, to be run in a simple and rapid fashion, with the output automatically recorded in the log file.

4 Experiments in Rule Induction

The availability of the ***Inducer*** package makes it straightforward to conduct even very extensive comparisons of different rule induction algorithms and related strategies. A number of comparisons of TDIDT and N-Prism are reported in [5].

As a further example, an experiment was conducted to compare the sensitivity of the two algorithms to missing values. The dataset used for this experiment was the *Vote* dataset from the UCI Repository. The dataset has 16 attributes (all categorical), 2 classes (Democrat and Republican), with 300 instances in the training set and 135 in the test set.

Using ***Datagen***, another of the packages in the ***Inducer*** suite, missing values were systematically introduced into the training and test sets in a random fashion with frequency from 10% up to 70%. Using the batch mode facility of ***Inducer*** the classification accuracy of the two algorithms was then computed for missing value levels of 0%, 10%, 20%, 30%, 50% and 70% in each of the training and test sets, as a single batch run.

Both algorithms used the same strategy for dealing with missing values, with each missing value being replaced by the most frequently occurring value when generating or applying the classification rules.

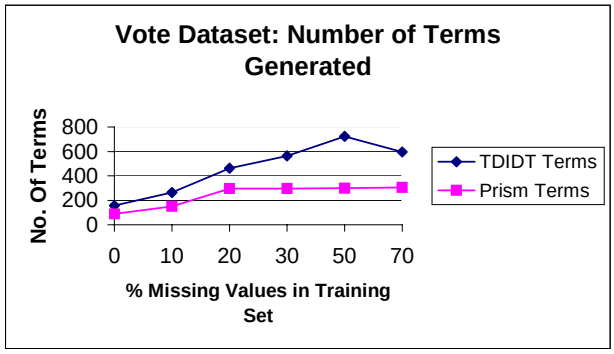
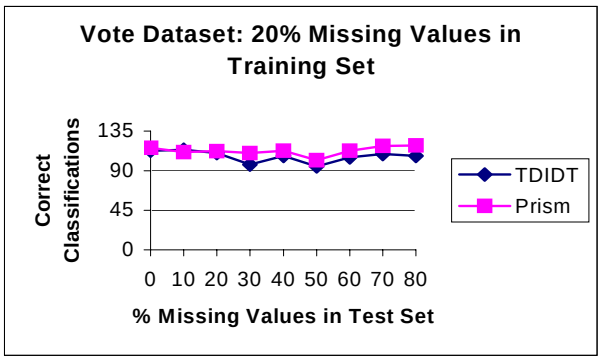


Figure 3. Number of Terms Generated for Varying Levels of Missing Values in the 'Vote' Training Set

The two algorithms produced virtually identical numbers of rules for each level of missing values in the training set. With 70% missing values there is some advantage to N-Prism (65 rules compared with 73). However, Figure 3 shows that N-Prism is considerably better than TDIDT when measured by the total number of terms generated and thus the average number of terms per rule. With no missing

values N-Prism generates a total of only 89 terms compared with 156 for TDIDT. Most strikingly, the total number of terms generated by N-Prism is not much more with 70% missing values (306 terms) than with 20% (296). By contrast TDIDT generates 462 terms with 20% missing values and this rises to 596 as the level of missing values increases to 70%.



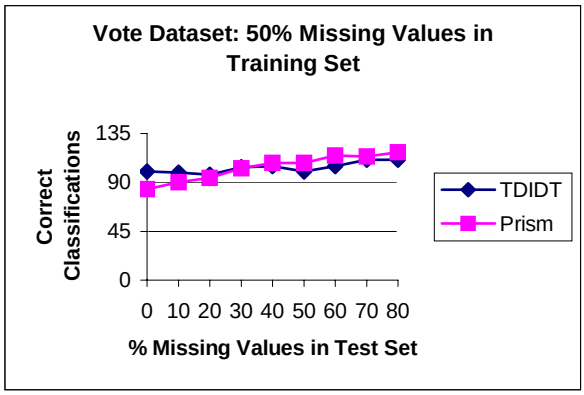


Figure 4. Effects of Introducing Missing Values in the 'Vote' Training and Test Sets

Figure 4 shows the comparative levels of classification accuracy of the two algorithms for missing value levels of 20% and 50% in the training set. Both algorithms perform well overall, even with high levels of missing values in both sets.

One way to extend these experiments would be to examine the effect of *pre-pruning* the rules, e.g. by means of a depth cutoff during the rule generation process, or of *post-pruning* them, say by discarding any rules with too low a value of the RI rule interestingness measure.

In general, a range of such experiments would need to be carried out to determine the most appropriate rule induction technique for a given application.

5 Conclusions

The ***Inducer*** rule induction workbench provides a powerful framework for in-depth experiments with alternative rule induction algorithms and related strategies. One such experiment has been reported briefly above. The package has been developed in a modular fashion to facilitate the addition of further algorithms and strategies as required.

References

1. Hunt, E.B., Marin J. and Stone, P.J. Experiments in Induction. Academic Press, 1966
2. Michie, D. Machine Executable Skills from "Silent" Brains. In: Research and Development in Expert Systems VII. Cambridge University Press, 1990
3. Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993
4. Cendrowska, J. PRISM: an Algorithm for Inducing Modular Rules. International Journal of Man-Machine Studies, 1987; 27: 349-370
5. Bramer, M.A. Automatic Induction of Classification Rules from Examples Using N-Prism. In: Research and Development in Intelligent Systems XVI. Springer-Verlag, pp. 99-121, 2000
6. MLC++ Machine Learning Library in C++. [A library of C++ classes, downloadable from http://www.sgi.com/Technology/mlc]
7. Witten, I.H. and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, 2000
8. Bramer, M.A. The Inducer User Guide and Reference Manual. Technical Report: University of Portsmouth, Faculty of Technology, 1999
9. Blake, C.L. and Merz, C.J. UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998
10. Freitas, A.A. On Rule Interestingness Measures. In: Research and Development in Expert Systems XV. Springer-Verlag, 1999, pp.147-158
11. Piatetsky-Shapiro, G. Discovery, Analysis and Presentation of Strong Rules. In: Piatetsky-Shapiro, G. and Frawley, W.J. (eds.), Knowledge Discovery in Databases. AAAI Press, 1991, pp. 229-248