The Inducer Rule Induction Workbench

Max Bramer

***Abstract*—This paper describes the facilities available in *Inducer*, a public domain rule induction workbench aimed at users who may not be computer scientists, who wish to analyze their own datasets using a range of data mining strategies or to conduct experiments with a given technique or combination of techniques across a range of datasets. *Inducer* has a graphical user interface which is designed to be easy to use by beginners, but also includes a range of advanced features for experienced users, including facilities to export the rules generated in several formats and other information in a form suitable for further processing by other packages. An experiment using the workbench is described.**

***Index Terms*—Data Mining, Decision Rules, Decision Trees, Rule Induction**

I. Introduction

The automatic induction of classification rules from examples is one of the key technologies of data mining. However, many of the available software tools are aimed at specialist academic researchers and require a high level of user sophistication, e.g. in manipulating a complex graphical interface, typing complicated commands or linking modules from a program library. In some cases the user is presented with a 'closed world', i.e. the classification rules and any statistical or other information generated cannot easily be exported from the package.

Two such packages are MLC++ [1], a library of C++ classes for supervised learning which can be incorporated into a user's own programs, and WEKA [2], a library of algorithms written in Java which can be run from a Java command line.

This paper describes *Inducer*, a public domain rule induction workbench aimed at users who may not be computer scientists, who wish to analyze their own datasets using a range of alternative data mining strategies or to conduct experiments with a technique or combination of techniques across a range of datasets. *Inducer* has a simple graphical user interface of checkboxes and menus (basic use of the package relying on default settings requires just three mouse clicks) and also includes a range of advanced features for more experienced users. The package is supplied with a number of standard datasets. It is written in Java in the interests of portability but no knowledge of that language is needed by the user. *Inducer* runs as an applet launched from a standard web browser. There is extensive on-line documentation.

The package was designed 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.

Inducer is intended for use with small to medium-size datasets. It can handle datasets with an unlimited number of instances but is not designed for processing very large datasets. The expectation is that users studying a single dataset will run *Inducer* many times to gain a good understanding of their data. The availability of a wide range of facilities for exploring data is considered more important than attempting to incorporate the latest 'state of the art' algorithms.

Inducer incorporates several variants of each of two families of rule generation algorithms: the widely used TDIDT (Top Down Induction of Decision Trees) algorithm which generates classification rules via the intermediate representation of a decision tree [3] and the Prism rule induction algorithm which generates modular classification rules that do not fit into a tree structure [4], [5]. Estimates of the predictive accuracy of the rules generated can be obtained by running the package in standard 'train and test' mode (i.e. using a training set and a separate test set) or using cross-validation or jack-knifing. A batch mode facility is available for conducting experiments using a fixed combination of techniques (selected via the GUI) across a number of datasets, as a single *Inducer* run. The rules generated are displayed in one text area on the screen, with confusion matrices and information about predictive accuracy etc. in another. The contents of both these areas can easily be copied into word processor files for reports etc.

Some of the many facilities incorporated in *Inducer* are described in the following sections.

II. Basic Use

Fig. 1 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) [6].

Inducer is designed to be easy to use. For the inexperienced user it requires only three mouse clicks to obtain a set of classification rules: two 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. For the expert user there is a wide range of options and facilities available.

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

There are no 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., except the limitations of the memory available. Execution times are typically no

Max Bramer is Professor of Information Technology in the Department of Computer Science and Software Engineering, University of Portsmouth UK (tel: +44-2392-846380, e-mail: Max.Bramer@port.ac.uk).

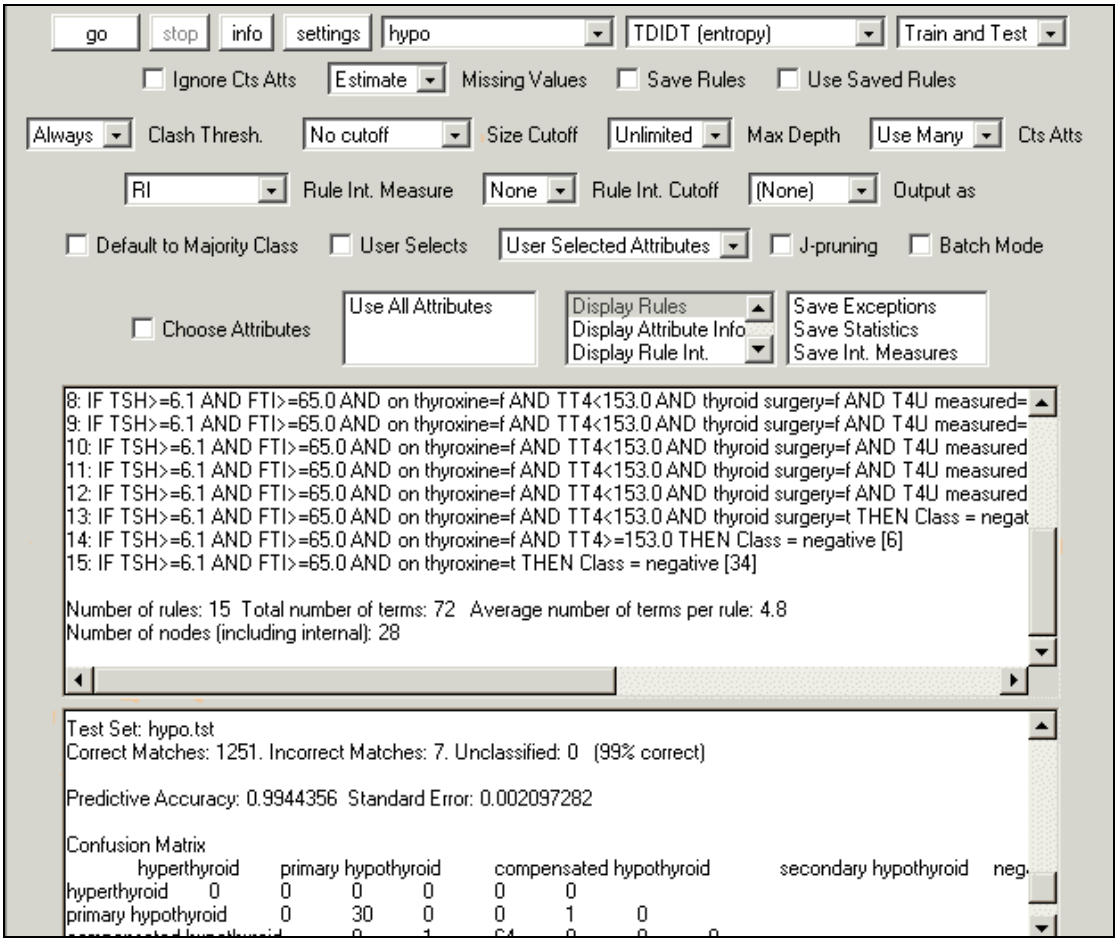

Fig. 1 *Inducer* Screen Image

more than a few seconds for datasets of the size of most of those in the UCI Repository.

There are currently 24 datasets available for selection from the default *input file directory*. These are mainly taken from the UCI Repository [6]. 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.

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 *Train and Test* 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.

Entries that are not on the leading diagonal correspond to the number of incorrect classifications for each correct class/incorrect class combination.

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

III. Rule Generation and Execution

Many facilities are available to the user for controlling the rule generation process.

A. Rule Generation Algorithms

Seven variants of the TDIDT tree generation algorithm [3] and five variants of the Prism rule generation algorithm [4], [5] are provided in *Inducer*.

For TDIDT the criterion for selecting the attribute to use at each stage of the tree generation process can be: Entropy

(or Information Gain), probably the most widely used attribute selection criterion for tree generation, 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, the well-known Gini Index, Evidential Power [7], as well as three other simpler methods.

As well as the standard version of Prism, there are four other versions available. The rules may be generated by processing the instances for each class in turn, working either from the largest class to the smallest or vice versa. The TC and TCS rule generation strategies described in [8] are also available.

B. Using Prior Knowledge

In cases where some important rules are known in advance, the user can give them to *Inducer* in a separate 'seed rules' file, the decision tree or rules only being generated for the instances the seed rules do not cover.

Another unusual feature of *Inducer* is that the user can specify that he or she wishes to choose the attribute to be used at each stage of the decision tree or rule generation process. Processing pauses until the user selects an attribute to use from a menu or selects 'automatic' indicating that from then on the system should make its own selections. Specifying the first few choices in this way is designed to allow users to take advantage of the knowledge that some attributes are more important to the classification than others.

C. 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.

Either tree or rule generation can be *pre-pruned* using a 'size cutoff' (stop if the number of instances in the training set currently under consideration is below a specified value) or a 'depth cutoff' (stop once a specified number of terms have been generated for a given branch or rule).

The author's J-pruning technique ([9], [8]) is also available. This makes use of the J-measure, an information theoretic means of quantifying the information content of rules [10].

If a clash arises during rule generation (e.g. because a pruning criterion has been met) a 'clash threshold' technique is used: if more than a user-specified percentage of the instances under consideration belong to the most frequent class they are all treated as belonging to that class, otherwise they are all discarded.

D. Discarding Rules on the Basis of 'Interestingness'

A topic of growing importance in recent years is that of rule interestingness [11], the aim of which is to identify those rules in a generated rule set that are likely to be of most value in classifying unseen instances. Seven measures of rule interestingness are calculated by *Inducer* for each rule generated and the user can choose to discard all rules with the value of a specified measure below a threshold level.

E. Post-pruning

A facility for the post-pruning of the decision trees produced by the TDIDT algorithm, using 'expected error pruning' is also provided. The decision tree is generated and branches are then progressively removed from the bottom up provided that the expected error at the leaf nodes is not increased at any stage.

F. Rule Execution Parameters

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

IV. Exporting Rules and Other Information

A. Exporting Rules

An important requirement of any practical data mining package is that it must be possible to export the rules generated by the package for use in the user's own programs, without the need for extensive retyping.

Using the 'Output as' option on the GUI, the rules generated by *Inducer* can be exported to a text file in four different formats, including as a Java method (see Fig.2) and as a set of Prolog clauses.

```
public String classify (String age,String specRx,
    String astig,String tears) {
String classification="";
if (tears.equals("1")) classification="3";
else if (astig.equals("1") && tears.equals("2"))
    classification="2";
else if (age.equals("1") && astig.equals("2") &&
    tears.equals("2")) classification="1";
else if (age.equals("2") && astig.equals("2") &&
    tears.equals("2")) classification="3";
else if (age.equals("3") && astig.equals("2") &&
    tears.equals("2")) classification="1";
return classification;
}
```

Fig.2 Rules Generated by the *lens24* Dataset from [6] as a Java Method

The other options are to output the rules in propositional form, i.e. as they are displayed on the screen, or in a scripting language suitable for input to the author's 'Knowledge Web' expert system delivery environment [12].

The rule set can also be saved as a rule file in a coded form, together with the values of the main system parameter settings, in a way that is suitable for processing by another program.

Rule files can also be imported by *Inducer* (perhaps after modification by the user or by another program) for use in analyzing further data. Selecting 'Use Saved Rules' overrides all the current rule generation parameter settings. No processing of the training set takes place.

B. Other Output Files

Output files can also be created giving information relating to the rule generation and execution process, for subsequent processing. There are three main output files, all optional.

(a) The *exceptions file*, which contains information about each instance misclassified by the generated rules: its reference number, the number of the rule that 'fired', i.e. was used to generate a classification for the instance, the predicted class, i.e. the incorrect classification generated and the correct classification.

(b) The *statistics file*, which contains a great deal of information about each run of *Inducer*. For both the training set and the test set (if there is one) it contains the confusion matrix generated, plus for each instance a numerical reference number, the number of the rule that was used to generate a classification for the instance, the predicted class, i.e. the classification generated, the correct classification, plus for each rule, information about its classification performance on the instances.

(c) The *rule interestingness file*, which contains the values of seven measures for each rule generated.

These files are all in CSV (comma delimited) format, with export to standard spreadsheets, graphics packages etc. in mind.

In addition to the above the system automatically generates a log file recording the results of every run. This can be very useful when conducting experiments, as described below.

V. Modifying the Input Data

Inducer provides a range of facilities to enable the user to adjust the data or to use only part of it in a given run.

The simplest way of doing this is by editing the name, data or test files using a text editor. Buttons for doing this are provided below the two text areas *Inducer* uses for displaying results. Changing the specification of an attribute in the name file from 'categorical' or 'continuous' to 'ignore' is a simple way of specifying that that attribute should not be used. However, for non-trivial changes it is usually better to be able to change the data without editing the files themselves.

A. Selecting Attributes

When the 'Choose Attributes' option is selected, pressing the 'Go' button causes the system to wait until the user chooses the attributes to be used in the current run. The word 'Go' changes to 'Continue'. Selecting a number of attributes and pressing 'Continue' restarts the run.

B. Missing Values

For many practical applications the value of some attributes 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*: ignore any instance in either the training or the test set that contains one or more missing values or replace any missing values for a categorical attribute by the most frequently occurring (non-missing) value for that attribute and any missing values for a continuous attribute by the average of the (non-missing) values of that attribute. This is the default setting.

C. Discretizing Continuous Attributes

The TDIDT and Prism algorithms as implemented in *Inducer* both have 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 rule generation process. However, many other rule induction algorithms have no facilities for dealing (directly) with continuous attributes and for purposes of comparison 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.

D. Header File

The user can modify the input data in several major ways by associating a header file with a dataset. There are facilities available:

(a) To specify the character used to separate data values in the training and test files.

(b) To specify the character used to denote a missing value in the training and test files.

(c) To specify that all attributes specified in the name file for the current dataset as 'continuous' be treated as if they were specified as a different type, including 'continuous1' (rounded to one decimal place before use) and 'integer' (rounded to the nearest integer before use). This enables the user to experiment with adjusting the precision of numerical data before any run.

(d) To specify that only a given number of instances in the training file for the current dataset should be read.

(e) To specify that all instances with a given classification in the training set and the test set (if there is one) be ignored.

(f) To specify that all instances with a given classification be treated as 'positive' instances of a concept and all instances with other classifications be treated as belonging to a single class, which is the negative of that concept.

VI. Facilities to Support Experimentation

Inducer provides two additional facilities to aid experimentation. The first is the log file facility mentioned previously. Every time the 'Go' button is pressed details of the main system parameter settings are recorded in a 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.

The second facility is the option to run *Inducer* in batch mode. When this option is checked details of the directories used by the *Inducer* system 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, perhaps with a series of different datasets, or possibly 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. Fig.3 is a typical example of a log file.

```
[file stems]
c:\inducer_data\datasets\
c:\inducer_data\outfiles\
c:\inducer_data\rulefiles\
[data files]
chess.nam,chess.dat,chess.tst
contact_lenses.nam,contact_lenses.dat,contact_lenses.tst
crx.nam,crx.dat,crx.tst
degrees.nam,degrees.dat,degrees.tst
vote.nam,vote.dat,vote.tst
```

Fig.3. An *Inducer* Batch File

VII. 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 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.

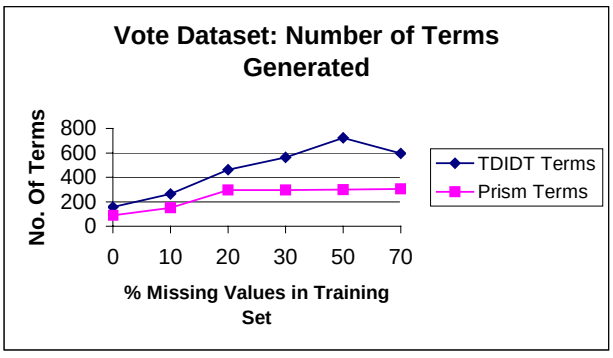

Fig. 4. Number of Terms Generated for Varying Levels of Missing Values in the 'Vote' Training Set

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 Prism (65 rules compared with 73). However, Fig. 4 shows that 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 Prism generates a total of only 89 terms compared with 156 for TDIDT. Most strikingly, the total number of terms generated by 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%.

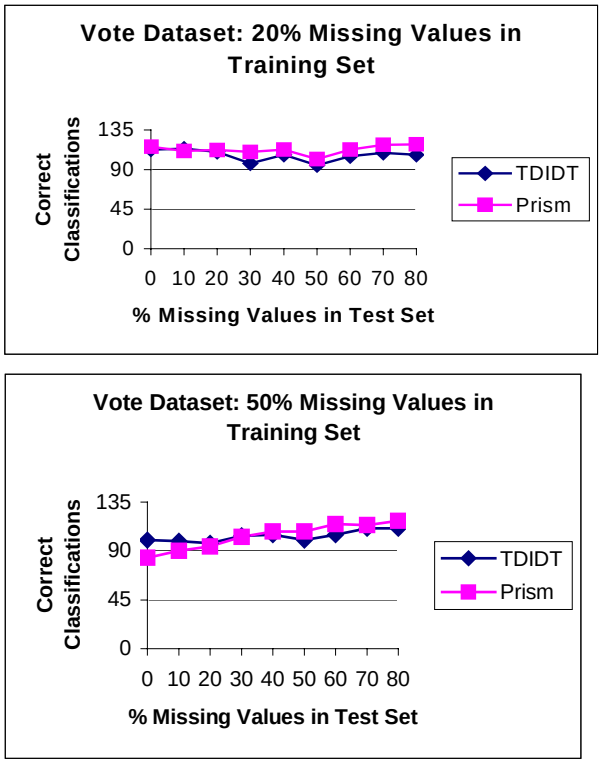

Fig. 5. Effects of Introducing Missing Values in the 'Vote' Training and Test Sets

Fig. 5 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.

VIII. Conclusions

The *Inducer* 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.

Although not part of its original purpose *Inducer* has also been used as a teaching tool and has proved valuable for this, enabling quite elaborate rule induction experiments to be carried out without any need for programming.

REFERENCES

[1] MLC++ Machine Learning Library in C++. [A library of C++ classes, downloadable from http://www.sgi.com/Technology/mlc].

[2] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, 2000.

[3] J. R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[4] J. Cendrowska, PRISM: an Algorithm for Inducing Modular Rules. International Journal of Man-Machine Studies, vol. 27, pp. 349-370, 1987.

[5] M. A. Bramer, 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] C. L. Blake and C. J. Merz, 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.

[7] D. McSherry, Explanation of Attribute Relevance in Decision Tree Induction, in *Research and Development in Intelligent Systems XVIII*, M. A. Bramer, F. Coenen and A. Preece, Eds. Springer, 2002.

[8] M. A. Bramer, An Information-Theoretic Approach to the Pre-pruning of Classification Rules, in *Intelligent Information Processing*, M. Musen, B. Neumann and R. Studer, Eds. Kluwer, 2002.

[9] M. A. Bramer, Using J-Pruning to Reduce Overfitting in Classification Trees. *Knowledge Based Systems*, vol. 15, no. 5-6, pp. 301-308, 2002.

[10] P. Smyth and R. M. Goodman, *Rule Induction Using Information Theory*, in *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. J. Frawley, Eds. AAAI Press, pp. 159-176, 1991.

[11] A. A. Freitas, On Rule Interestingness Measures, in *Research and Development in Expert Systems XV*. Springer-Verlag, pp.147-158, 1999.

[12] M. A. Bramer, Knowledge Web: A Public Domain Expert System Delivery Environment. *IEEE International Conference on Systems, Man And Cybernetics*, 2003.