Key words – Data Mining, Classification, Decision Rules, Decision Trees, Rule Induction

Max BRAMER[*]

A PUBLIC DOMAIN CLASSIFICATION WORKBENCH FOR DATA MINING

This paper describes the facilities available in *Inducer*, a public domain classification workbench aimed at users who wish to analyse 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 information generated in a form suitable for further processing by other packages. An experiment using the workbench is described.

1. INTRODUCTION

An increasing number of data mining packages have become available in recent years, both commercial and public domain. In discussing the commercial packages a recent paper [11] comments: "These packages, for the most part, essentially wrap a varying number of public domain … components or algorithms (sometimes re-implemented with rather small proprietary extensions) in an user-friendly graphical interface. Although such tools seemingly make DM technology more readily available to non-expert end-users, they tend either to provide only limited functionality … or to come with a non-negligible price-tag".

There are several public domain data mining packages available. Some such as MLC++ [1] and WEKA [13] include a wide range of algorithms but require a significant level of user sophistication, e.g. in typing complicated commands or linking modules from a program library. Other public domain packages concentrate on a single algorithm. Many of the packages provide few facilities to pre-process the user's data or to export information from the package for future use.

Although data mining is a wide-ranging field, one of its key task areas is *classification*. This paper describes *Inducer*, a public domain classification workbench for data mining aimed at users (in most cases non-computer scientists) who wish to analyse

[*] Department of Computer Science and Software Engineering, University of Portsmouth, Lion Terrace, Portsmouth PO1 3HE, United Kingdom. Max.Bramer@port.ac.uk

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 provides four basic algorithms for classification: tree induction, rule induction, nearest neighbour and naïve Bayes classification. Further information about these is given in Section 4. The algorithms may be used with a wide range of alternative parameter settings.

The package was designed to facilitate practical experimentation with a range of classification 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. It is expected that users studying a single dataset will run *Inducer* many times to gain a good understanding of their data. The package automatically keeps a log of basic information every time one of the classification algorithms is run.

No attempt has been made to incorporate a comprehensive collection of classification algorithms into *Inducer*. There are many new algorithms published each year, but few of them outperform the more established algorithms across a wide range of datasets. For practical purposes a new slightly improved algorithm is likely to be of considerably less value than a wide range of facilities for pre-processing the user's data, for exploring the effect of 'tuning' the basic algorithms in different ways and for exporting information for use outside the package.

The *Inducer* package includes a number of datasets, principally reformatted versions of datasets provided in the repository of machine learning datasets at the University of California at Irvine (UCI) [3]. The intention is that users will gain familiarity with the facilities available using these fairly well-behaved datasets before going on to analyse their own data.

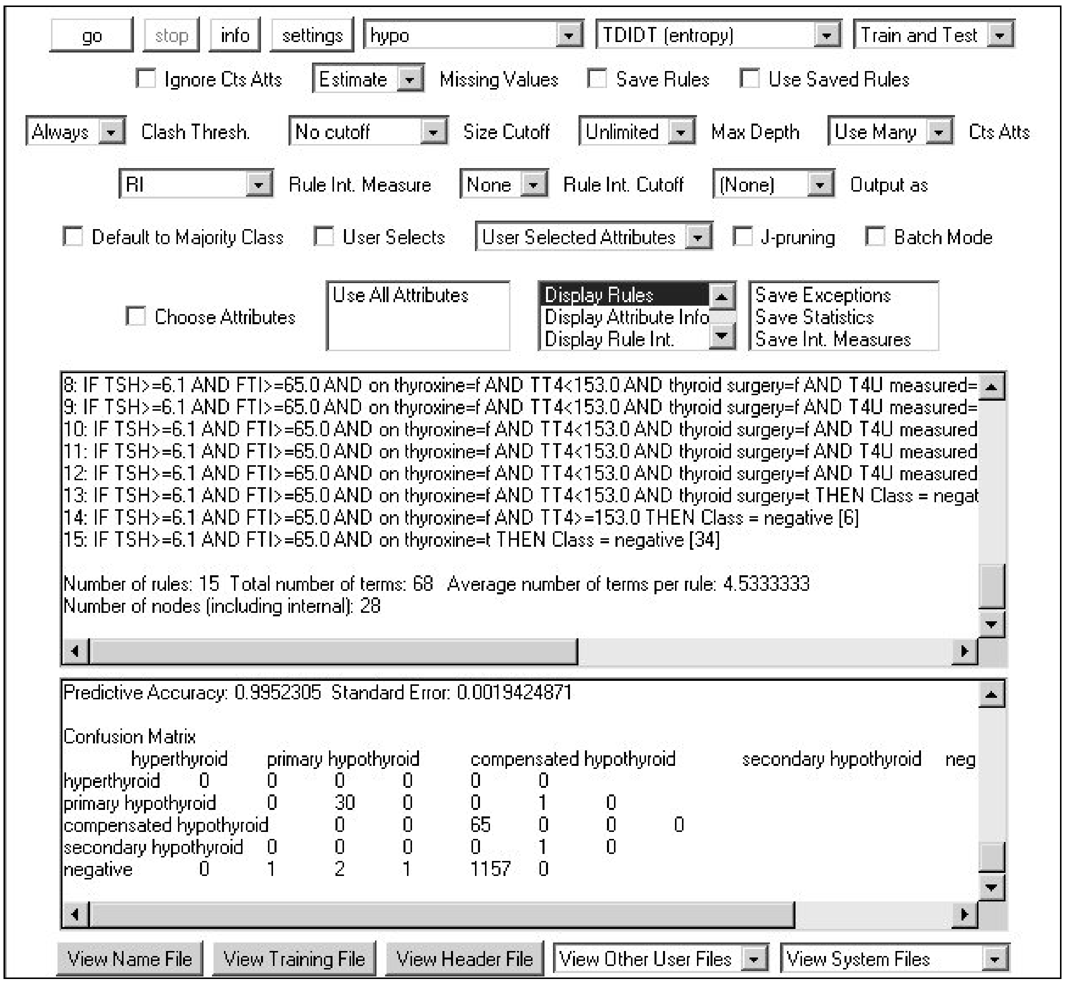
Some of the many facilities incorporated in *Inducer* are described in the following sections.

2. BASIC USE

Inducer is designed to be easy to use. It runs as a Java applet launched from a standard web browser, with extensive on-line documentation. The user can launch *Inducer* from any of three alternative web pages, the choice made determining the graphical interface displayed by the applet. There is a 'standard' rule/tree induction interface with few options, for use by newcomers, an 'advanced' rule/tree induction interface, for use by experienced users, and a nearest neighbour/naïve Bayes interface which omits all rule-specific facilities. All three interfaces have a standard look and feel, so familiarity gained from one will transfer across to using the other two.

Fig. 1 shows a screen image of *Inducer* running the hypothyroid dataset ('hypo') from the UCI Repository [3] and illustrates the 'advanced' interface of checkboxes and menus. Even for this interface, it requires only three mouse clicks to carry out a classification using the default settings: 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.

There are no limitations on the maximum size of the datasets 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 more than a few seconds for datasets of the size of most of those in the UCI Repository.



```
hypo    TDIDT (entropy)    Train and Test
☐ Ignore Cts Atts    Estimate  Missing Values    ☐ Save Rules    ☐ Use Saved Rules
Always  Clash Thresh.    No cutoff  Size Cutoff    Unlimited  Max Depth    Use Many  Cts Atts
RI  Rule Int. Measure    None  Rule Int. Cutoff    (None)  Output as
☐ Default to Majority Class    ☐ User Selects    User Selected Attributes    ☐ J-pruning    ☐ Batch Mode
☐ Choose Attributes    Use All Attributes    Display Rules        Save Exceptions
                                             Display Attribute Info  Save Statistics
                                             Display Rule Int.    Save Int. Measures
```

```
8: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4<153.0 AND thyroid surgery=f AND T4U measured=
9: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4<153.0 AND thyroid surgery=f AND T4U measured=
10: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4<153.0 AND thyroid surgery=f AND T4U measured
11: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4<153.0 AND thyroid surgery=f AND T4U measured
12: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4<153.0 AND thyroid surgery=f AND T4U measured
13: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4<153.0 AND thyroid surgery=t THEN Class = negat
14: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=f AND TT4>=153.0 THEN Class = negative [6]
15: IF TSH>=6.1 AND FTI>=65.0 AND on thyroxine=t THEN Class = negative [34]

Number of rules: 15  Total number of terms: 68   Average number of terms per rule: 4.5333333
Number of nodes (including internal): 28
```

```
Predictive Accuracy: 0.9952305  Standard Error: 0.0019424871

Confusion Matrix
          hyperthyroid     primary hypothyroid       compensated hypothyroid         secondary hypothyroid   neg
hyperthyroid      0        0        0        0        0        0
primary hypothyroid        0        30       0        0        1        0
compensated hypothyroid             0        0        65       0        0        0
secondary hypothyroid   0        0        0        0        1        0
negative          0        1        2        1        1157     0
```

Fig. 1. *Inducer* Screen Image

There are currently 24 datasets provided with the *Inducer* package in its default input directory. Alternative input directories can be specified if preferred. The format of datasets is essentially that specified in [12]. Each dataset comprises a name file, a training set and in many 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. An extension to the format given in [12] is that an attribute can also be specified as *ignore*. This is a helpful facility for datasets containing attributes such as *name, number of children* etc., which in most cases it would be meaningless to use for classification.

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.

The name file, training set and test set and all other files used by *Inducer* can be edited within the package itself using a standard text editor. Buttons for doing this are provided below the two text areas *Inducer* uses for displaying results.

For all four classification algorithms, estimates of the predictive accuracy of the classifier can be obtained by running *Inducer* in standard 'train and test' mode (i.e. using a training set and a separate test set) or using cross-validation or jack-knifing. In each case *Inducer* calculates and displays a *confusion matrix* with the entry for row i, column j giving the number of instances with a correct classification of i and a computed classification of j. In the case of a perfect classification all non-zero entries in the confusion matrix will occur on the leading diagonal. Other non-zero entries correspond to the number of incorrect classifications for each correct class/incorrect class combination.

Inducer displays the rules (if applicable) or other information about the classifier in one text area on the screen, with the confusion matrix in another. Information about the number of correct matches, incorrect matches and unclassified instances, the percentage of correct matches and other statistical information is also displayed. The contents of both text areas can easily be copied into word processor files for reports etc.

3. PRE-PROCESSING DATA

Real-world data can suffer from a variety of problems. Attributes are often included that are of little if any predictive value (e.g. the name or eye colour of a patient in a medical application). Attribute values are frequently ‘noisy’ or missing. Numerical values are often recorded with an unrealistically high level of (apparent) precision, e.g. 69.428176329. Large datasets may contain too many attributes and/or too many instances to process in a reasonable time. Some classifications may be so uncommon or meaningless that they will simply hinder the classification process.

Inducer provides a range of facilities for adjusting the input data before it is used for classification. Two of the options can be set using the package’s graphical interface. Some require the user to edit the ‘name’ file containing the specifications of all the attributes, which accompanies each data file. Others require directives to be entered in a *header file* associated with each dataset.

3.1 MISSING ATTRIBUTE VALUES

Two strategies are available in *Inducer* for dealing with missing attribute values: *discard* (delete any instance that has even one missing attribute value) and *estimate* (replace each missing value by an estimate of the true value). The former avoids the possibility of introducing errors, but runs the risk of discarding valuable information, especially if there are many instances with missing values. The latter strategy avoids discarding any data but runs the risk of introducing substantially incorrect estimates in some cases.

3.2 ACCURACY OF NUMERICAL MEASUREMENT

The precision of measurement of numerical attributes can be adjusted without altering the data itself in two ways. The standard specification of numerical attributes in the name file is *continuous*. *Inducer* also supports attribute specifications *continuous1*, *continuous2*, *continuous3*, *integer* and *integer10*, indicating that the numerical values of an attribute are to be rounded to 1, 2 or 3 decimal places, to an integer or to a multiple of 10, respectively, before they are used. The user can also specify (using a directive in the header file) that all attributes specified as *continuous* (but not *continuous1* etc.) in the name file are to be treated as, say, *continuous3*.

3.3 DEALING WITH IRRELEVANT ATTRIBUTES

There are several ways in which *Inducer* can be told to use only some of the available attributes for classification.

(a) Any individual attribute can be given a specification of *ignore* in the name file.

(b) There is a *choose attributes* option on the GUI. When it is selected, *Inducer* first reads in the information in the name file, then displays the names of all the (non-ignore) attributes and waits for the user to make a selection. Experiments with different combinations of attributes for the same dataset can be carried out easily and speedily using this facility and can give valuable insights into the user’s data.

(c) A final option, which is also available on the GUI, is to specify that all continuous attributes should be ignored. This is helpful when comparing the performance of the algorithms used in *Inducer* with other published algorithms, which frequently cannot handle continuous attributes.

3.4 ATTRIBUTE REDUCTION

Using a large number of attributes for classification can take a substantial amount of processing time, especially if they have continuous values. However it is often far from clear that the results would be less accurate if a much smaller number of attributes were available. It is hard to imagine many practical problems where the classification genuinely depends on the values of hundreds or thousands of attributes. *Inducer* includes a standard facility for attribute reduction by means of a header file directive such as *use best attributes 100*. *Inducer* does this by calculating for each attribute in turn the *information gain* [12] that would be obtained by prior knowledge of the value of the attribute and retains only those with the highest values.

A recent experiment (related to those reported in [2]) showed the practical value of this facility. A decision tree was generated from a training set of instances, each comprising the frequency of occurrence of 13,430 words in a set of web pages and their classification into one of two possible classes (using the Yahoo classification scheme). Using this decision tree to classify instances in an unseen test set gave 94% predictive accuracy. Reducing the number of attributes to just 50 before any tree generation took place resulted in a different decision tree which also had 94% predictive accuracy on the test data, but with a considerable reduction in the amount of processing required to generate it.
.

3.5 IGNORING MINOR OR IRRELEVANT CLASSIFICATIONS

Data not collected specifically with Data Mining in mind can sometimes include instances with inappropriate or unhelpful classifications. For example, a weather prediction dataset, with classifications *fair*, *sunny*, *rainy*, etc. might include a small proportion classified as *unrecorded*, corresponding to days when the recording equipment malfunctioned. To avoid these instances interfering with the classification process, the user could search through the data and remove the unrecorded instances, but a far easier way is to place a directive such as *ignore class unrecorded* in the header file.

3.6 USING ONLY PART OF A TRAINING SET

A further header file directive is available to specify that only a given number of instances should be read from the training file. This is very useful when the training set is large and also makes it straightforward to conduct experiments to establish the number of training instances needed to achieve a satisfactory level of predictive accuracy.

3.7 POSITIVE AND NEGATIVE EXAMPLES

A final pre-processing facility provided by *Inducer* is the ability to treat a dataset with multiple classifications as if there were only two. The instances are regarded as positive or negative examples of a specified concept (class), say *safe*. Effectively, all the other classes are combined into a single one, labelled in this case *non-safe*.

4. CLASSIFICATION ALGORITHMS AVAILABLE IN INDUCER

The four classification algorithms available in *Inducer* are:

(a) the widely used TDIDT (Top Down Induction of Decision Trees) algorithm which generates classification rules via the intermediate representation of a decision tree [12]

(b) the Prism rule induction algorithm which generates modular classification rules that do not fit into a tree structure [7]

(c) k-nearest neighbour matching, which uses a measure of the distance between each instance in the training set and an instance to be classified in the test set to find the k nearest instances and use their classifications to classify the test instance [10]

(d) naïve Bayes, which uses probability estimates to classify a test instance [10]

The first two methods generate rules from the training data. The third and fourth use the training data directly and do not form any explicit representation of the underlying classifier.

Many facilities are available to the user for controlling the rule generation process, when one of the interfaces for algorithms (a) and (b) is selected.

4.1 RULE GENERATION ALGORITHMS

Seven variants of the TDIDT tree generation algorithm [12] and five variants of the Prism rule generation algorithm [7] are provided in *Inducer*.

For TDIDT the criterion for selecting the attribute to use at each stage of the tree generation process can be: Information Gain (the default setting), Gain Ratio [12], the Gini Index, Evidential Power [9], 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 [4] are also available.

4.1.1 USING PRIOR KNOWLEDGE TO CONTROL RULE GENERATION

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. Users can also specify that they wish 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.

4.1.2 CUTOFFS DURING RULE GENERATION

To avoid rules being generated that are over-fitted to the data, either tree or rule generation can be *pre-pruned* using a 'size cutoff' (stop if the number of instances in the subset of 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 [5] is also available. This makes use of the J-measure, an information theoretic means of quantifying the information content of rules.

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.

4.1.3 DISCARDING RULES ON THE BASIS OF 'INTERESTINGNESS'

A topic of growing importance in recent years is that of rule interestingness [8], 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.

4.1.4 POST-PRUNING OF CLASSIFICATION TREES

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.

4.2 NEAREST NEIGHBOUR MATCHING

The user can experiment with the value of k, the number of nearest 'neighbours' and can specify whether continuous attributes are to be normalized to a standard range of values before use. In many cases it is desirable to discard some (perhaps many) of the less important attributes before using k-Nearest Neighbour classification, to avoid their having too great an influence on the classification. Methods (a) and (b) can be used in combination with the 'choose attributes' option described in Section 3.3 to identify and use the attributes that are most likely to be of value to the classification.

4.3 NAÏVE BAYES CLASSIFIER

This probabilistic method can only be used when all attributes are categorical. Again, using rule or tree induction can indicate which attributes are most likely to prove of most value when using this probabilistic method.

5. EXPORTING INFORMATION FROM INDUCER

Two important design requirements for any practical data mining package are that as far as possible the user is not restricted to just one mode of use and that the user is able to take information out of the package for subsequent processing elsewhere.

5.1 SAVING RULES

Having generated classification rules using one of the rule induction algorithms, it is standard practice to use them to predict the classification of a test set of instances that have already been classified, in order to estimate the accuracy of the classifier. However, the user may also wish to use the rules to classify either pre-classified data or genuinely unseen data, which may be in several separate datasets, possibly weeks or months later.

Inducer has a GUI option to save the rules as a file in a coded form, together with the values of the main system parameter settings, in a way suitable for processing by another program or manual editing by the user. The rule file can be used at any time in the future simply by selecting the 'Use Saved Rules' option for the same dataset name. The system will read in the rules from the appropriate saved rule file, automatically bypass the rule generation stage and use the rules to classify the data in the test file. Using the same saved rules to classify the instances in several different test sets can easily be achieved using the batch file facility described in Section 6.

5.2 EXPORTING RULES TO OTHER PACKAGES

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

```
public String classify (float sepalLength, float sepalWidth, float petalLength, float petalWidth) {
String classVal="";
if (petalLength<3.3) classVal="Iris-setosa";
else if (petalLength>=3.3 && petalLength<5.1 && petalWidth<1.6) classVal="Iris-versicolor";
else if (petalLength>=5.1 && petalWidth<1.6) classVal="Iris-virginica";
else if (sepalWidth<3.2 && petalLength>=3.3 && petalLength<4.9 && petalWidth>=1.6)
    classVal="Iris-virginica";
else if (sepalWidth>=3.2 && petalLength>=3.3 && petalLength<4.9 && petalWidth>=1.6)
    classVal="Iris-versicolor";
else if (petalLength>=4.9 && petalWidth>=1.6) classVal="Iris-virginica";
return classVal;
}
```

Fig.2. Rules Generated from the *iris* Dataset [3] 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 [6].

5.3 OTHER OUTPUT FILES

Three other output files can also be created giving information relating to the rule generation and execution process, for subsequent processing. All of them are all in comma delimited format, with export to standard spreadsheets, graphics packages etc. in mind.

(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 'interestingness' measures for each rule generated.

(d) The *log file*. This is automatically generated by the system and records the results of every run.

6. FACILITIES TO SUPPORT EXPERIMENTATION

Inducer provides two additional facilities to aid experimentation. The first is the log file facility mentioned in Section 5. Every time the 'Go' button is pressed, details of the main system parameter settings are appended to a log file, together with the number of rules and terms generated (if applicable), the number of instances correctly and incorrectly classified and other information. The log file is in comma delimited format, suitable for importing to a spreadsheet for further processing.

The second facility is the option to run *Inducer* in batch mode. When this option is checked a list of datasets to use is taken from the file *inducer.bat*. Pressing the *Go* button causes *Inducer* to run the currently selected classification algorithm with the current parameter settings on each of the datasets in turn.

The combination of these two facilities can enable a substantial series of experiments to be run in a simple and rapid fashion. As an example, an experiment was conducted to compare the sensitivity of the four classification algorithms to noise (i.e. incorrect attribute 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. Ten-fold cross-validation was used to estimate classification accuracy.

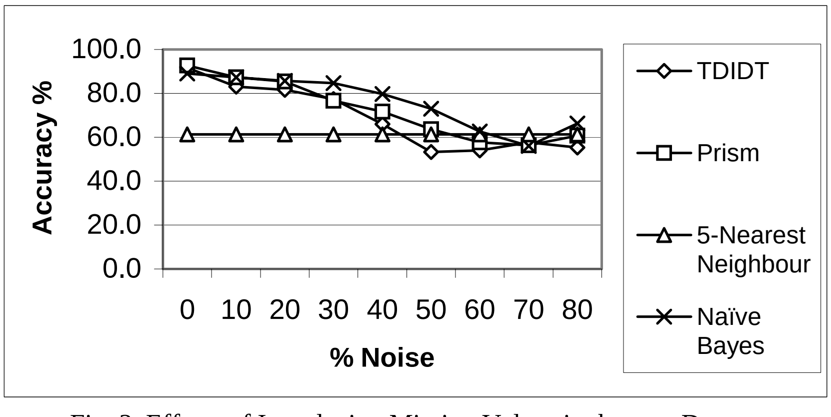


Fig. 3. Effects of Introducing Missing Values in the *vote* Dataset

Using *Datagen*, another of the packages in the *Inducer* suite, noise was systematically introduced into the training set in a random fashion with frequency from 10% up to 80%. Using the batch mode facility of *Inducer* the classification accuracy of the four algorithms was then computed for each of the nine datasets (the original noise-free one, plus the eight with noise added) in turn, as a single batch run. The results are summarised in Figure 3.

This example is presented not because of the significance of the results obtained but to illustrate the type of experiments that can be conducted easily using the batch mode facility in *Inducer*.

Once the datasets had been generated all that was necessary was to:

- Create a batch file giving the dataset name
- Start *Inducer* with the tree/rule interface, select the batch mode and 10-fold cross-validation options and press the *Go* button
- Start *Inducer* with the nearest neighbour/naïve Bayes interface, select the batch mode and 10-fold cross-validation options and press the *Go* button
- Import the log file into a spreadsheet as a comma delimited file, and position the results correctly in the spreadsheet for displaying as a chart.

The time taken to complete all four steps, in the case of this small dataset, was only a few minutes.

7. CONCLUSIONS

The *Inducer* workbench provides a powerful framework for in-depth experiments with alternative classification algorithms and related strategies. 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 classification experiments to be carried out by students 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] BENBRAHIM H. AND BRAMER M.A., *Impact on Performance of Hypertext Classification of Selective Rich HTML Capture*, In Artificial Intelligence Applications and Innovations (Proceedings of AIAI-2004, Toulouse, August 2004), Kluwer, 2004.

[3] BLAKE C.L. AND MERZ C.J., *UCI Repository of Machine Learning Databases*, University of California, Department of Information and Computer Science, 1998. [http://www.ics.uci.edu/~mlearn/MLRepository.html],

[4] BRAMER M.A., *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.

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

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

[7] CENDROWSKA J., *PRISM: An Algorithm for Inducing Modular Rules*, International Journal of Man-Machine Studies, vol. 27, pp. 349-370, 1987.

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

[9] McSHERRY D., *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.

[10] MITCHELL T., *Machine Learning*, McGraw-Hill, 1997.

[11] POVEL O. AND GIRAUD-CARRIER C., *SwissAnalyst: Data Mining Without the Entry Ticket*, In Artificial Intelligence Applications and Innovations (Proceedings of AIAI-2004, Toulouse, August 2004), Kluwer, 2004.

[12] QUINLAN J.R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[13] WITTEN I.H. AND FRANK E., *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 2000.