RandT: a DataEngine plug-in for Fuzzy Rules and Fuzzy Decision Trees Generation

Juan Ramón Velasco, Juan Pedro Somolinos
DAEDALUS - Data, Decisions and Language, S.A.
Pirineos 27, 1º Madrid (Spain)
Phone: +34-91-311-3386, Fax: +34-91-450-4058
E-mail: {jvelasco, jsomolinos} @daedalus.es
http://www.daedalus.es

ABSTRACT: RandT system is a DataEngine plug-in for building classifiers inductively from data collections by discovering and analysing patterns of behaviour in such data collections. In order to do this, RandT system uses the C4.5 algorithm, which builds both a decision tree and a rule base from the data. The application also allows the integration of the obtained models from the data in other programs as an application development library in an easy and powerful way.

KEYWORDS: RandT, C4.5, decision tree, rule base, symbolic knowledge, data mining, classifier, generalisation, fuzzy C4.5, DataEngine, application development library.

INTRODUCTION AND GOALS OF RandT SYSTEM

DataEngine is a powerful data-mining tool that includes several techniques and algorithms for data analysis as neural networks, Kohonen networks, fuzzy clustering, etc. However, it has not a native decision tree classifier, which is a very important algorithm for data classification. To solve this problem, there exists a DataEngine plug-in, Decision Xpert, but it is only available in German language.

The RandT system uses the DataEngine plug-in facilities (MIT GmbH, 1999b) to provide to the DataEngine user a way for building decision trees and rule bases from data collections, and at the same time, to allow them the use of these obtained models in other applications. This second goal is achieved by the definition of an easy interface, from which users can invoke the generated model (either a decision tree or a rule base) and perform classification tasks in their own application with this model.

There is a third goal for this tool: to provide an integrated mechanism to provide fuzzy classifiers and fuzzy rules to DataEngine. In this way, decision tree algorithms are *fuzzyfied*, to deal with imprecision and uncertainty.

DECISION TREES

A decision tree is a structure made by nodes and leafs. Nodes are the place where decisions are taken, i.e., some test are carried out on an attribute, with one branch and a subtree for every possible value of this attribute. Leafs represents the decision class of the tree (see figure 1). To classify an example, the system starts at the root node of the tree (in the figure, on the top) and it moves through the decision nodes that correspond with the example until a leaf is reached. The class represented in the leaf is the decision for this case.

There are several algorithms that build these structures from data collections. The oldest versions of these algorithms only were able to deal with discrete attributes (properties with a limited number of different values and with no order relationship between them). RandT system is based on ID3 and C4.5 algorithms and it can manage both discrete and numerical attributes.

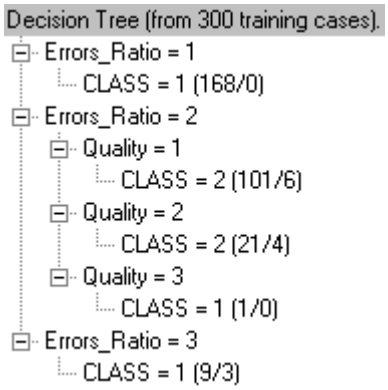

Figure 1: An example of ID3 decision tree.

THE ID3 ALGORITHM

ID3 is the most extended algorithm for generating decision trees. It was proposed by J. Ross Quinlan (1979, 1986). In a general view, ID3 algorithm is able to generate a decision tree from data collections made by discrete attributes and a class for every example. The aim of the model is to classify new examples (that is, examples not contained in the data collection used to generate the decision tree).

The construction of the tree is made by seeking for the attribute that provides the largest amount of information in each node of the tree. The original ID3 algorithm uses a criterion called the *gain criterion*; this criterion selects the attribute that maximises the information gain. The information of a message depends on its probability and can be measured in terms of *entropy*, that is, a number (bits) that explains the disorder of a set of cases. Selecting the attributes of each node in this way minimises the deep of the final decision tree. Said in an easy way: the idea is to select the attribute that splits the set of data in subsets as well classified as possible.

The entropy of a set of examples *S* is given by the expression:

$$Entropy(S) = -\sum_{j=1}^{k} \frac{freq(Cj,S)}{|S|} \times \log 2 \left(\frac{freq(Cj,S)}{|S|} \right)$$

Where *freq (Cj, S)* measures the number of values of the set *S* that belong to the class *Cj*, $|S|$ is the total cases of the set *S* and *k* represents the number of classes. After the entropy of a set of data is calculated, for every attribute, the average entropy of the derived subsets is obtained. The attribute that minimises the average entropy is selected.

THE C4.5 ALGORITHM

The C4.5 algorithm is the natural evolution of ID3, and was developed by Quinlan (1993) in order to improve the capabilities of its predecessor. The most relevant new features are:

- The ID3 gain criterion has a serious deficiency, because it has a strong bias in favour of attributes with too many outcomes (for instance, any identity number on the data, which is unique for every instance, is a perfect attribute for classifying with no error, but there is no possibility of learning on it). To solve this problem, C4.5 integrates the gain ratio criterion, based on the ID3 criterion, but introducing a kind of normalisation in which the apparent gain to be assigned to attributes with many outcomes is adjusted.
- C4.5 is able to manage continuous attributes, which ID3 uses in a wrong way. In this case, decision nodes are obtained by testing every possible value contained in the input data and selecting the best threshold. A decision node with a test on a continuous attribute splits the data into two sets according to the select threshold.
- C4.5 allows missing or unknown values in the data collection used for generating the models and in the non-seen cases.

- C4.5 introduces the concept of pruning of trees. This is a way to avoid the overfitting of a classification model, that is, a tree that infers more structure than is justified by the training cases. By pruning, the structure of a tree is reduced in order to fit better the new cases to classify. This fact has two benefits: it improves the accuracy of the model to classify new non-seen cases and at the same time reduces the complexity of the tree, allowing a better understanding of the problem.

An ID3 decision tree generated by RandT with numerical attributes is shown in the figure 2 (left).

As an added value from ID3, C4.5 can also generate a rule base derived from a decision tree. This rule base has a number of rules, each of them looks like "IF antecedent1 AND antecedent2 AND... AND antecendentN THEN class = X". A case is covered by a rule if the case satisfies all the conditions or antecedents of that rule.

An example of a rule base is shown in the figure 2 (right)

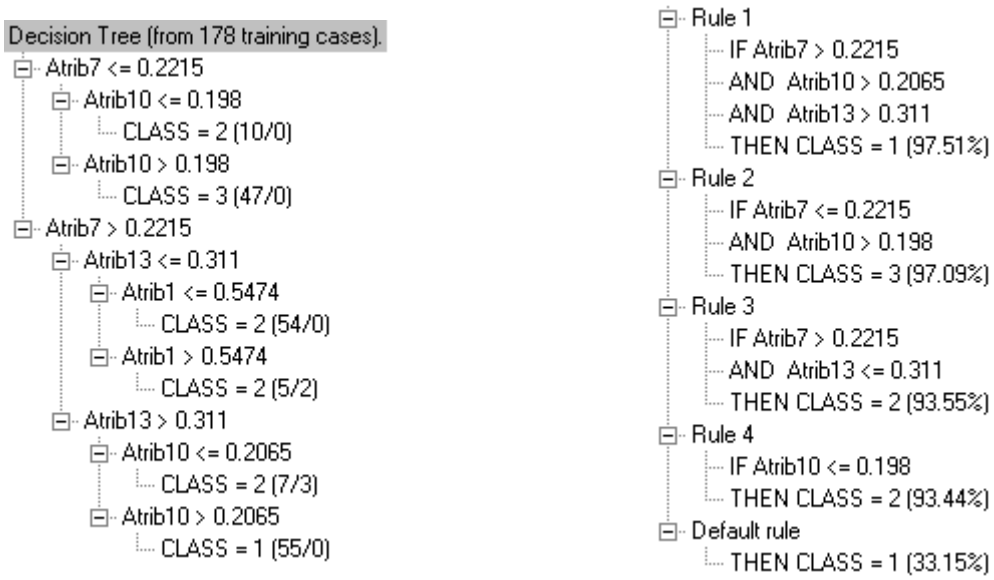

Figure 2: An example of a decision tree with numerical attributes (left) and an example of a rule base (right).

To create the derived-tree rule base, C4.5 begins by creating a rule for every leaf from the tree and deleting the superfluous conditions of rules. This kind of generalisation avoids the concepts to be fragmented in several subtrees of the decision tree. The final rule base is obtained by grouping rules by its predicted class and ordering them to minimise false positive errors (that is, cases covered by a rule whose predicted class is wrong), and choosing a default rule, which is the one that is applied when any other one can't cover a case.

Rule bases usually provides a more accurate model of the problem than decision trees, and they express the concepts of the problems in an easier and more understandable way. The structure of the tree, if complex, has serious understanding problems. Each node of the tree has a specific context established by the outcomes of the antecedent nodes, that is, a test on an attribute only makes sense if it is read with the outcomes of all the antecedent tests. Simple trees don't seem to be a problem, but it is when the tree structure grows that the understanding problem arises. The advantage of a rule is that its antecedents have not to be evaluated in a fixed order, and a rule is able to show together all the conditions that a case has to satisfy in order to decide the class it belongs to. Nevertheless, the problem for right interpretation is not completely solved: C4.5 interprets the rule base in order, so the second rule only will be tested when the first one fails, and so on.

THE RandT SYSTEM

The first version of RandT provides a way for building decision trees and rule bases from data collections, and at the same time, to allow the use of these obtained models in other applications. As it was said in the introduction, this second goal is achieved by the definition of an easy interface, from which users can invoke the generated model (either a decision tree or a rule base) and perform classification tasks in their own application with this model.

The fuzzy version of these algorithms will be available on RandT 2.0. This version will give an integrated mechanism to provide fuzzy classifiers and fuzzy rules for DataEngine.

MAIN FEATURES OF RandT 1.0

These are the basic features implemented in the first version of the RandT plug-in.

- RandT is able to generate a decision tree from an input data made up of examples of a problem. For each case, this input data collection must specify the values for the different attributes of the problem and the real class the case belongs to. In order to achieve a good decision tree, RandT allows the user to manage a lot of parameters that have influence in the generation process:
  - The user can decide if the final decision tree must be pruned, and if so, a pruning confidence level can be use to indicate the level of pruning. The pruning of the tree is done by collapsing in a leaf all the subtrees that has an estimated error rate higher than a leaf. This value depends on the pruning confidence level: the lower the confidence level, the heavier the tree pruning is, with a most pronounced effect in small data sets. The user can introduce a value between 0 and 99 per cent.
  - A kind of *pre-pruning* can be forced by using the minimum cases in a decision node parameter. RandT will not continue dividing a node that has fewer cases than this given value. Thus, this parameter should be used to prevent a "specialisation" of the tree structure. A higher value may be good for data collections where there is a lot of noise.
  - RandT is able to check if grouping discrete attribute values make the final tree more accurate. This feature has interest if the distinct values of a discrete attribute have any relationship.
  - Both of the best-attribute selection criterions can be used: the *gain* criterion (from ID3) or the *gain ratio* one (C4.5).
  - RandT also implements a windowing algorithm, which allows the user to deal with big data sets. The windowing algorithm starts with a small part of the total data, and applies the decision tree algorithm on it. The decision tree is tested over a different test data set. If the obtained error is higher than a given value, the non-classified cases are added to the training data set. The algorithm continues in an iterative way. RandT allows the user to define:
    - The initial size of the window, that is, the number of training cases used in the first iteration of the algorithm to generate each tree.
    - The maximum number of training cases that can be added to the generation subset of the tree in each step.
    - The number of generated trees in this way (starting with different windows).
  - RandT can ignore some attributes of the problem in order to construct the classifiers. This option is useful if any of the discrete attributes has too many values or if the user simply doesn't wish an attribute to be tested.
  - If user checks the proper option on the configuration window, RandT will display the final decision tree. The dialog window that shows the tree also displays more information about the classifier (number of classes, error rate, size of the training set and detailed information of each node). The user can also print the decision tree in a readable form.
- RandT is able to build a rule base from the final unpruned decision tree as well. These are the available options for the rule base generation process:
  - The confidence level. RandT uses this value in order to estimate the real error rate of each rule from the error rate of such rule with the training cases. Its meaning is the same as the pruning confidence level: higher values of this parameter cause better estimated error rates than lower ones.
  - Maximum error rate of each rule. This parameter causes RandT to discard all the rules that have an error rate in the training cases up to the given value. This is useful to reduce the number of rules in the final rule base if the user is looking for a simple rule base.
  - Theory weight. This parameter represents the influence of the number of training cases covered by a set of rules in order to choose the best set of rules that covers each class. RandT uses a special algorithm to do this task. Best class ruleset is chosen by minimising the total cost of encoding the ruleset. This total encoding is calculated using a weighted sum:

    $$Total\ encoding = exception\ encoding + theory\ weight \times theory\ encoding$$

    Where the *exception encoding* is the cost (in bits) of encode all the non-covered training cases and the theory encoding represents the costs of encode all the rules in the class ruleset. So, the theory weight value can be

used to bias the choice of the set of rules that cover each class: the higher the theory weight the fewer number of rules obtained.

- And, of course, the system can classify new data collections by using a saved classifier (either a decision tree or a rule base).

RandT WORK MODES

RandT has two basic ways of work, which can be selected by configuring the RandT function block:

- *Model generation.* In this function mode, RandT constructs both a decision tree and a rule base from the input data collection; the models are saved on disk for later use. RandT output is the confusion matrix of both the decision tree and the rule base, as shown in the figure 3 (the two confusion matrixes can easily be sorted out by *Split* and *Select Columns* DataEngine functions).

ConfusionMatrix.dat

	CLASS 1 (tree) []	CLASS 2 (tree) []	CLASS 3 (tree) []	CLASS 1 (rule base) []	CLASS 2 (rule base) []	CLASS 3 (rule base) []
1	5,000	1,000	0,000	4,000	2,000	0,000
2	1,000	5,000	0,000	0,000	6,000	0,000
3	0,000	0,000	6,000	0,000	0,000	6,000
*						

Figure 3: An example of confusion matrixes generated by RandT (problem with three classes).

Confusion matrix shows how a model classifies each case (i.e. which class decides the model), depending on the real class the case belongs to. The (i,j) cell contains the number of cases belonging to the i class and being classified as class j by the model. Obviously, the matrix diagonal contains the correct classified cases (where i=j). RandT also performs tests of the real accuracy of a model if the user provides a test data collection as second input.

- *Classification mode.* RandT classifies the input data using a saved classifier. The output matrix is the same as the input adding one more column, which contains the predicted class for each case.

INTEGRATION OF CLASSIFICATION MODELS IN OTHER APPLICATIONS

RandT system has also been developed to allow integration of the obtained models in not DataEngine programs. Thus, user develops a classifier by using the RandT plug-in into the DataEngine environment, and includes it into an application for making classification task there. This integration requires only few lines of C/C++ code.

Classifiers obtained by RandT in DataEngine are saved in a binary format. RandT allows the user to recover the classifiers from a C/C++ code, eliminating the need of recompiling the program if the model changes, in the same way than DataEngine ADL (MIT GmbH 1999a). The user system only needs the file containing the classifier; RandT library reads and interprets this file when the model is invoked from the program. Besides, DataEngine is not required to be installed on the platform in which your application is going to be used.

RandT include all functions needed for loading, initialising and executing a classifier, and the interface given to the programmer is very easy and intuitive.

FUZZY CLASSIFIERS

RandT 2.0 will be focused on the implementation of a fuzzy C4.5 algorithm (Rives, 1990 and García and Garín, 1995). Frequently, the attributes of a problem don't have a discrete value, that is, the cases don't belong to a given category in a crisp way but they usually have certain degree of membership to each category of the attribute. This fact also applies to the class an example belongs to.

Therefore, the implementation of a fuzzy C4.5 algorithm (a generalisation of the original C4.5 algorithm) can be very useful to cover these situations. This implies the inclusion of a new kind of attribute into the RandT system, a fuzzy

attribute. The improvement of the algorithm for building the classifiers must include methods for managing and evaluating this type of attributes: calculation of gain and gain ratio criterion with fuzzy attributes, pruning trees, generalising rules, etc.

In this case, the entropy for selecting the attribute that is going to be used for splitting the data is calculated taking into account two important aspects:

- The membership of the examples to the classes
- The membership of the examples to the nodes: While in the classic ID3/C4.5 algorithm, the data splitting is completely crisp (data are divided in the different branches), in this case any example may belong to different branches with a given membership value.

$$FuzzyEntropy(S) = -\sum_{i=1}^{k} \frac{\sum_{j=1}^{Ne} \mu_i(Ej) \cdot \sigma_X(Ej)}{\sum_{j=1}^{Ne} \sigma_X(Ej)} \times \log_2 \left(\frac{\sum_{j=1}^{Ne} \mu_i(Ej) \cdot \sigma_X(Ej)}{\sum_{j=1}^{Ne} \sigma_X(Ej)} \right)$$

In the formula, $\mu_i(E_j)$ is the membership value of the example E_j to the class i. $\sigma_X(E_j)$ is the membership value of the example E_j to the node X.

The algorithm needs several extra values, as the minimum membership value for an example to belong a node.

REFERENCES

García, J.A. and M.L. Garín, 1995, “Variante del Algoritmo ID3 para Atributos con Valores Borrosos.”, In: VI Conferencia de la Asociación Espanola para la Inteligencia Artificial. CAEPIA'95 Actas, 217-224. Alicante, Spain. (in spanish)

MIT GmbH, 1999a, “DataEngine ADL 3.0 – User’s Guide & Reference Manual”, Aachen, Germany.

MIT GmbH, 1999b, “Programmer’s Guide to User-Defined Function Blocks“, Aachen, Germany.

Quinlan, J. Ross, 1979, "Discovering Rules from Large Collections of Examples, in Expert Systems in the Microelectronic Age", Michie, D. (Ed.), Edimburgo University Press, Edimburgo.

Quinlan, J. Ross, 1986, "Induction of Decision Trees (ID3 algorithm), Machine Learning, Morgan Kauffman Publishers, California.

Quinlan, J. Ross, 1993, "C4.5: Programs for machine learning", Morgan Kauffman Publishers, California.

Rives, J., 1990, “FID3: Fuzzy Induction Decision Tree”, in Ayyub, B.M. (ed.), Uncertainty Modelling and Analysis. Proceedings of ISUMA’90, IEEE, New York.