A Rules-to-Trees Conversion in the Inductive Database System VINLEN

Tomasz Szydło[1], Bartłomiej Śnieżyński[1], and Ryszard S. Michalski[2,3]

[1] Institute of Computer Science, AGH University of Science and Technology, Kraków, Poland
[2] Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA
[3] Institute of Computer Science, Polish Academy of Science
e-mail: tomsz@student.agh.edu.pl, sniezyn@agh.edu.pl, michalski@gmu.edu

Abstract. Decision trees and rules are completing methods of knowledge representation. Both have advantages in some applications. Algorithms that convert trees to rules are common. In the paper an algorithm that converts rules to decision tree and its implementation in inductive database VINLEN is presented.

Keywords: machine learning, decision tree learning, decision making, inductive databases.

1 Introduction

Decision trees and decision rules are very common knowledge representation used in machine learning and data mining. Algorithms for learning decision trees are simple to implement, and relatively fast. Algorithms for learning decision rules are more complex, but resulting rules may be easier to interpret.

To exploit advantages of both knowledge representations, learned decision trees are often converted into rules. Opposite conversion is also possible. The AQDT-2 method proposed by Imam and Michalski is able to convert rules into a decision tree that is optimized according to a given criterion of decision tree optimality [1].

A major advantage of such a conversion is that it allows an adaptation of the decision tree to changing conditions. To explain this advantage, note that decision trees are built under certain assumptions. If these assumptions are not satisfied, e.g., the cost of measuring of some attributes changes, it is not possible to measure some attribute, new data become available, or the probability distribution of classes changes, the learned decision tree needs to be modified. Once a tree is built, however, such a modification is difficult or even impossible without learning the tree from scratch again.

The AQDT-2 program takes a set of rules learned from examples, and generates a decision tree tailored to the given decision task. Generating a tree from rules is faster than generating a decision tree from examples.

The AQDT-2 method was implemented by Imam as an independent program. Nowadays, we can observe a trend of building integrated machine learning environments (see [5,7,6]). Such an approach not only makes performing machine learning experiments easier, but also provides an opportunity to apply different methods by users who are not specialists in this domain.

This paper describes an implementation of the AQDT-2 method as a module of the inductive database system VINLEN that aims at integrating conventional databases with a range of inductive inference capabilities [2]. The following sections describe the AQDT-2 method, its implementation as a VINLEN module, and some preliminary results from testing the module as a VINLEN operator.

2 AQDT-2 as a Knowledge Generation Operator

As it was mentioned above, AQDT-2 is implemented as a part of inductive database VINLEN [2]. In this system database and knowledge base form knowledge system (KS), standard database operators are completed by knowledge generation operators (KGOs) that operate on elements of KS.

There are several KGOs implemented in the system so far. The most important is rule induction operator that is an implementation of AQ algorithm. It transforms database table into rulefamily (set of attributional rules expressed using Attributional Calculus [4]). AQDT-2 is another operator. It takes a rulefamily as an input and produces a decision tree that can be stored in a knowledge base. Classic decision tree induction algorithm – C4.5 is also implemented to make comparison with AQDT-2 in the future.

Generated decision tree is presented in a result window, where user has possibility to expand or collapse nodes of the tree, what is useful for watching some branches of the tree in different level of abstraction. There is also possibility to print and copy the tree to the clipboard.

3 AQDT-2 Algorithm

In this section the AQDT-2 algorithm is described. It builds a decision tree from a set of attributional rules instead of examples. Algorithm is presented below:

Input: A family of attributional rulesets.
Output: A decision tree.
Step 1. Evaluate each attribute occurring in the ruleset using the LEF attribute ranking measure (see below). Select the highest ranked attribute. Suppose it is attribute A.
Step 2. Create a node of the tree, and assign to it the attribute A. Create as many branches from the node, as there are legal values of the attribute A, and assign these values to the branches.

Step 3. For each branch, associate a group of rules from the ruleset context which contain a condition satisfied by the value assigned to this branch. If there are rules in the ruleset context that do not contain attribute A, add these rules to all rule groups associated with the branches.

Step 4. If all the rules in a ruleset context for some branch belong to the same class, create a leaf node and assign to it that class. If all branches of the tree have class assigned, stop. Otherwise, repeat steps 1 to 4 for each branch that has no class assigned.

The difference between AQDT-2 and methods of learning decision trees from examples is that it chooses attributes that are assigned to the nodes using criteria based on the properties of the input attributional rules. At each step, algorithm chooses the attribute from available set by determining the attribute utility in the given set of attributional rules. The attribute (test) utility is based on five elementary criteria: cost, disjointness, importance, value distribution, and dominance [1].

The above criteria can be combined into one general test ranking measure using the "lexicographic evaluation functional with tolerances" – LEF [3]. LEF combines two or more elementary criteria by evaluating them one by one (in the order defined by LEF) on the given set of tests. A test passes to the next criterion only if it scores on the previous criterion within the range defined by the tolerance. In AQDT-2 method the following LEF is used:

$$< C, \tau_1, D, \tau_2, I, \tau_3, V, \tau_4, Do, \tau_5 > \tag{1}$$

where C, D, I, V, Do represent cost, disjointness, importance, value distribution, and dominance; τ_1, τ_2, τ_3, τ_4 and τ_5 are tolerance thresholds.

The decision tree can be generated in *Compact Mode* or *Normal Mode*. In *Normal Mode* standard decision trees are generated: each branch has one specific attribute and value assigned. In *Compact Mode* a decision tree may contain nodes representing conditions expressed in attributional calculus (e.g. internal disjunction in the form $x = v_1$ or v_2) that are generated using selectors appearing in rules.

4 Example

In this section a simple example of AQDT-2 method application is presented. A robot domain with the following nine nominal attributes is used: *head*, *body*, *smile*, *holding*, *height*, *antenna*, *jacket tie*, and *group*. The last one is a target attribute with three possible values: *bad*, *do_not_know*, *good*. Rules generated by AQ21 KGO are presented below:

```
A robot is unfriendly, if
    - its head is square or triangular,
      and its body is square or round;      (r1)
A robot is unclassified, if
```

```
    - its head is pentagonal;              (r2)
A robot is friendly, if
    - its body is round, or                (r3)
    - its head is square or triangular,
      and its body is triangular;          (r4)
```

From these rules two trees are generated using AQDT-2 method in compact mode and standard mode. First one has six nodes, four of them are leafs. It looks as follows:

```
IF robot's head is pentagonal THEN it is unclassified
IF robot's head is round THEN it is friendly
IF robot's head is square or triangular THEN
     IF robot's body is round or square THEN it is unfriendly
     IF robot's body is triangular THEN it is friendly
```

We show how it is constructed. At the beginning, attribute *head* with the highest rank is chosen (using LEF method mentioned above) and a node with this attribute assigned is created. Compact mode causes that whole selectors are used to assign values to branches. *head* appears in three selectors, therefore three new nodes are created with branches with the following values assigned: `pentagon` (r2), `round` (r3), and `square or triangle` (r1, r4). Rule labels in parenthesis show which rules are copied to the new nodes. Two of these nodes are recognized as leafs. Third one is processed recursively because rules copied there belong to different classes. Attribute *body* is chosen (from rules r1 and r4). There are two values in selectors containing this attribute, hence two nodes are created with values `round or square` (r1), and `triangle` (r4) assigned to branches. These nodes are leafs, therefore tree construction is finished.

In the normal mode there are always as many branches as there are values defined in an attribute domain. Therefore tree is bigger. It has eleven nodes and eight leaves. It is presented below:

```
IF robot's head is pentagonal THEN it is unclassified
IF robot's head is round THEN it is friendly
IF robot's head is square THEN
     IF robot's body is round THEN it is unfriendly
     IF robot's body is square THEN it is unfriendly
     IF robot's body is triangular THEN it is friendly
IF robot's head is triangle THEN
     IF robot's body is round THEN it is unfriendly
     IF robot's body is square THEN it is unfriendly
     IF robot's body is triangular THEN it is friendly
```

5 Conclusion and Further Research

The goals of this project are to develop an efficient and versatile program for transforming attributional rules learned to optimized decision trees and to integrate it with the VINLEN inductive database system. The generated tree is stored in the VINLEN's knowledge system, and used as an input to other operators.

In the near future, we would like to add new features to the module, such as the ability for edit manually the decision tree and to test it on a database of examples. We would also like to develop an operator that produces code in a given programming language that implements the generated decision tree.

Interesting results of comparison of AQDT-2 and C4.5 programs can be found in [1]. Imam and Michalski tested both programs on several datasets. The following properties of the generated decision trees were measured: the number of nodes, accuracy, and execution time. Decision trees obtained using AQDT-2 tended to be simpler and had higher predictive accuracy. In further research, we plan to conduct more experiments that test the decision rules to decision tree conversion on a number of problem domains.

6 Acknowledgments

The authors thank Janusz Wojtusiak for help with using AQ21 and tuning it to produce better rules.

References

1. I. Imam and R. S. Michalski. An empirical comparison between learning decision trees from examples and from decision rules. In *Proc. of the Ninth International Symposium on Methodologies for Intelligent Systems (ISMIS-96)*, Zakopane, Poland, 1996.
2. K. A. Kaufman and R. S. Michalski. The development of the inductive database system VINLEN: A review of current research. In M. Kłopotek et al., editor, *Intelligent Information Processing and Web Mining*, Advances in Soft Computing, pages 393–398, Zakopane, Poland, 2003. Springer.
3. R. S. Michalski. AQVAL/1-computer implementation of a variable-valued logic system VL1 and examples of its application to pattern recognition. In *Proceeding of the First International Joint Conference on Pattern Recognition*, Washington, D.C., USA, 1973.
4. R. S. Michalski. *Attributional Calculus: A Logic and Representation Language for Natural Induction.* Reports of the Machine Learning and Inference Laboratory, MLI 04-2. George Mason University, 2004.
5. I. Mierswa, R. Klinkberg, S. Fischer, and O. Ritthoff. A Flexible Platform for Knowledge Discovery Experiments: YALE – Yet Another Learning Environment. In *LLWA 03 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivitat*, 2003.
6. M. Staudt, J. U. Kietz, and U. Reimer. A data mining support environment and its application on insurance data. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.* AAAI Press, 1998.
7. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.