ART: A Hybrid Classification Model

Fernando Berzal (fberzal@decsai.ugr.es) *
Dept. Computer Science and AI, University of Granada (Spain)

Juan-Carlos Cubero (jc.cubero@decsai.ugr.es) *
Dept. Computer Science and AI, University of Granada (Spain).

Daniel Sánchez (daniel@decsai.ugr.es)
Dept. Computer Science and AI, University of Granada (Spain)

José María Serrano (jmserrano@decsai.ugr.es)
Dept. Computer Science and AI, University of Granada (Spain)

July 23, 2002

Abstract. This paper presents a new family of decision list induction algorithms based on ideas borrowed from the association rule mining context. ART, which stands for 'Association Rule Tree', builds decision lists which can be viewed as degenerate, polythetic decision trees. Our method is a generalized "Separate and Conquer" algorithm suitable for Data Mining applications because it makes use use of efficient and scalable association rule mining techniques.

Keywords: supervised learning, classification, decision lists, decision trees, association rules, Data Mining

1. Introduction

Classification is a major problem in Artificial Intelligence. The aim of any classification algorithm is to build a classification model given some examples of the classes we are trying to model; i.e. given the training set. The abstract model we obtain can then be used to classify new examples or simply to achieve a better understanding of the available data.

In this paper, we present a new algorithm for building classifiers which can yield excellent results while being suitable to cope with huge data sets usually found in Data Mining problems. In order to ensure good scalability properties, we make use of association rules to efficiently build a decision list, which can be viewed as a degenerate, polythetic decision tree, hence its name ART (Association Rule Tree). As any symbolic classification model, its main strength is that it provides understanding of and insight into the data.

* Contact information: Juan-Carlos Cubero and Fernando Berzal, Departamento de Ciencias de la Computación e Inteligencia Artificial, ETS Ingeniería Informática, Universidad de Granada, Granada 18071 SPAIN.

Our method, as a decision list learner, is a generalized "Separate and Conquer" algorithm (Pagallo and Haussler, 1995), in contrast to the standard "Divide and Conquer" algorithms used to build decision trees. However, our induction process is faster than general decision list and rule inducers which need to discover rules one at a time. As decision tree learners, ART is able to build classification models in an efficient and scalable way. Moreover, our classifiers tend to be smaller than the decision trees generated by standard TDIDT classifiers.

In the following section we introduce the learning techniques our method is based on. Afterwards, we present a complete description of our classification model, whose intuitive parameters can be automatically adjusted. Some experimental results are also reported in this paper. Finally, we present our conclusions and pointers to future work.

2. Background

2.1. Rule learners and decision lists

There are many algorithms which induce sets of **if - then** rules directly from data. Some algorithms perform a beam search through the space of hypotheses such as INDUCE or AQ (Sestito and Dillon, 1994), as well as CN2 (Clark and Nibblett, 1989) (Clark and Boswell, 1991) and CWS (Domingos, 1996). Genetic algorithms, such as REGAL (Giordana and Neri, 1996), also fall in this category.

Decision lists can be considered to be an extended **if - then - else if - ... - else -** rule. In fact, *k-DL* (decision lists with conjunctive clauses of size at most k) are polynomially learnable (Rivest, 1987). Although rule learning systems are computationally expensive, more efficient algorithms exist for decision lists, e.g. IREP (Fŕnkranz and Widmer, 1994), RIPPERk (Cohen, 1995), and PNrule (Joshi et al., 2001). These algorithms are incremental, in the sense that they add one rule at a time to the classification model they build (usually through a "separate and conquer algorithm"). Some proposals, however, try to discover the whole set of rules in a single search, e.g. BruteDL (Segal and Etzioni, 1994) performs a depth-bound search of rules up to a certain length.

An alternative approach to obtain a set of **if-then** rules consists of building a decision tree using a "divide and conquer" algorithm. Rules can be easily extracted from decision trees, and there are methods, such as C4.5rules (Quinlan, 1993), which convert decision trees into decision lists simplifying the set of rules which is derived from a decision tree.

2.2. Decision trees

Decision trees, also known as classification or identification trees, probably constitute the most popular and commonly-used classification model; e.g. see (Quinlan, 1986b) and (Gehrke et al., 1999b).

The knowledge obtained during the learning process is represented using a tree where each internal node holds a question about one particular attribute (with one offspring per possible answer) and each leaf is labeled with one of the possible classes.

A decision tree may be used to classify a given example beginning at the root and following the path given by the answers to the questions at the internal nodes until a leaf is reached.

Decision trees are built recursively following a top-down approach (from general concepts to particular examples). That is the reason why the acronym TDIDT, which stands for Top-Down Induction on Decision Trees, is used to refer to this kind of algorithms.

The TDIDT algorithm family includes classical algorithms such as CLS (Concept Learning System), ID3 (Quinlan, 1986a), C4.5 (Quinlan, 1993), and CART (Classification And Regression Trees) (Breiman et al., 1984), as well as more recent ones such as SLIQ (Mehta et al., 1996), SPRINT (Shafer et al., 1996), QUEST (Loh and Shih, 1997), PUBLIC (Rastogi and Shim, 1998), RainForest (Gehrke and Ramakrishnan, 1998) and BOAT (Gehrke et al., 1999a). Other decision tree classifiers include BCT (Chan, 1989), which builds polythetic decision trees combining ideas from ID3 and CN2, and TX2steps (Elder, 1995), which performs a lookahead in decision tree construction.

2.3. Association rules

Association rule mining has been traditionally applied to databases of sales transactions (referred to as basket data). In this kind of database, a transaction T is a set of items, henceforth itemset, with a unique identifier and some additional information (e.g. date and customer). A transaction contains a set of items I if I is contained in T. Such a set of items is called *k-itemset*, where k is the number of items in the set.

An association rule is an implication $X \Rightarrow Y$ where X and Y are itemsets with empty intersection (i.e. with no items in common). The intuitive meaning of such a rule is that the transactions (or tuples) which contain X also tend to contain Y.

The confidence of an association rule $X \Rightarrow Y$ is the proportion of transactions containing X which also contain Y. The support of the rule is the fraction of transactions in the database which contain both X and Y.

Given a database, association rule mining algorithms try to extract all the association rules with support and confidence above user-specified thresholds, *MinSupp* and *MinConf* respectively.

Many algorithms have been proposed to solve this problem, from the original proposals, such as AIS (Agrawal et al., 1993), SETM (Houtsma and Swami, 1993), or, in particular, Apriori (Agrawal and Skirant, 1994), to more advanced algorithms such as DHP (Park et al., 1995), DIC (Brin et al., 1997), CARMA (Hidber, 1999), FP-Growth (Han et al., 2000), and TBAR (Berzal et al., 2001). See (Hipp et al., 2000) for a recent survey of the problem and (Han and Plank, 1996) for a somewhat older comparison of some selected algorithms.

2.4. Related work

Some fundamental differences exist between classification and association rule discovery (Freitas, 2000). Association rules do not involve prediction, nor do they provide any mechanism to avoid underfitting and overfitting apart from the crude *MinSupport* user-specified threshold. An inductive bias is also needed to solve classification problems, i.e. a basis for favoring one hypothesis over another (e.g. Occam's razor). This bias, like any other bias, must be domain-dependent.

Association rules have however been used to solve classification problems directly. In (Ali at al., 1997), association rules are used to build partial classification models in domains where conventional classifiers would be ineffective. For example, traditional decision trees are problematic when many values are missing and also when the class distribution is very skewed.

In (Wang et al., 2000), a tree of rules is built from an arbitrary set of association rules without using an ad-hoc minimum support threshold. The authors have observed that predictivity often depends on high confidence, and rules of high support tend to have low confidence, so *MinSupport* pruning is not suitable for their classification purposes.

CBA is an algorithm for building complete classification models using association rules which was proposed in (Liu et al., 1998). In CBA [Classification Based on Associations], all "class association rules" are extracted from the available training dataset (i.e. all the association rules containing the class attribute in their consequent), and the most adequate rules are selected to build an "associative classification model", which uses a default class to make it complete. This classifier builder uses a brute-force exhaustive global search, and yields excellent results when compared to C4.5. In (Liu et al., 2000c), CBA performance was "improved" allowing multiple minimum support thresholds for the

different problem classes and recurring to traditional TDIDT classifiers when no accurate rules are found.

A similar strategy to that of CBA is used to classify text documents into topic hierarchies in (Wang et al., 1999). All the generalized association rules with the class attribute in their consequent are extracted, these rules are ranked, and some of them are selected to build a classifier which takes context into account, since class proximity is important when classifying documents into topics.

Hybrid approaches have also been suggested in the literature. LB (Meretakis and Wüthrich, 1999), which stands for "Large Bayes", is an extended Naïve Bayes classifier which uses an Apriori-like frequent pattern mining algorithm to discover frequent itemsets with their class support. This class support is an estimate of the probability of the pattern occurring with a certain class. The proposed algorithm achieves good results. However, it lacks the understandability of symbolic models (such as decision trees).

Emerging patterns are itemsets whose support increases significantly from one dataset to another (Dong and Li, 1999). They have been used to build classifiers following the LB philosophy. For example, CAEP (Dong et al., 1999) finds all the emerging patterns meeting some support and growth rate thresholds for each class. It then computes an aggregated differentiating score to determine the most suitable class for a given instance. This computation allows the algorithm to perform well when the class populations are unbalanced, although it gives no further insight into the data.

Liu, Hu and Hsu (Liu et al., 2000a) propose the use of a hierarchical representation consisting of general rules and exceptions in order to replace the usual flat representation model where too many association rules hamper the understanding of the underlying data. The same approach is followed in (Liu et al., 2000b) in order to obtain a good summary of the knowledge contained in an arbitrary set of rules. In some way, ART takes a further step in that direction, as we will see in the following sections.

3. ART Classification model

Instead of discovering rules one at a time, as most decision list learners do, ART discovers multiple rules simultaneously. ART builds partial classification models using sets of association rules. The instances in the input dataset which are not covered by the selected association rules are then grouped together in 'else' branches to be further processed following the same algorithm.

3.1. Building the classifier

ART constructs a decision list using a greedy algorithm where each decision is not revoked once it has been taken. Since it employs an association rule mining algorithm to efficiently build partial classification models, it requires the typical user-specified thresholds used in association rule mining, *MinSupp* (minimum support for the frequent itemsets) and *MinConf* (minimum association rule confidence), although the latter can be omitted, as we shall see. On the other hand, the minimum support threshold can be established as a percentage over the current dataset size and it implicitly restricts the tree branching factor. Therefore, primary or candidate keys in the input dataset will never be selected by ART.

3.1.1. *ART Overview*

The special kind of decision list ART obtains can be considered as a degenerate, polythetic decision tree. Unlike most traditional TDIDT algorithms, ART is able to use several attributes simultaneously to branch the decision tree. This fact has proved to improve the performance of decision tree learning in terms of both *higher prediction accuracy and lower theory complexity* (Zheng, 2000).

In our context, *a good rule* is an accurate rule, i.e. an association rule with a high enough confidence value so that it can be helpful in order to build an accurate classifier. As we will see in Section 3.1.3, the *best set of good rules* is the most promising set of rules according to some preference criterion. In other words, it is the set of rules which best seems to serve our purpose of building a predictive model (from a heuristic point of view, as is evident). ART will choose such a set to branch the decision tree.

In its quest for candidate hypothesis, ART begins by looking for simple association rules such as $\{A_i.a_i\} \Rightarrow \{C.c_j\}$, where A is an attribute, a is its value, C is the class attribute and c is one of the problem classes. The attribute A with *the best set of good rules* $\{A_i.a_i\} \Rightarrow \{C.c_j\}$ is then selected to branch the tree. A leaf node is generated for *each good rule* and all the data instances not covered by any of these rules are grouped in an 'else' branch to be further processed in the same way.

When no suitable association rules are found for any single attribute-value pair $A.a$, ART looks for more complex association rules. It begins with rules of the form $\{A_1.a_1\ A_2.a_2\} \Rightarrow \{C.c_j\}$. If no good candidate rules are found, ART then searches for rules with three attribute-value pairs in their antecedents, such as $\{A_1.a_1\ A_2.a_2\ A_3.a_3\} \Rightarrow \{C.c_j\}$, and so on. An upper bound on the size of the left-hand side (LHS) of the association rules is then advisable to stop the search in the rare case

when no suitable rules are found. This parameter is designed as *MaxSize* and its maximum possible value equals the number of predictive attributes in the training dataset (recall the First Normal Form). In fact, this parameter can be set to that value by default, unless the user explicitly changes it.

ART therefore begins by using simple hypotheses to classify the training data and makes more complex hypothesis only when no simple hypotheses are found to work well. ART thus incorporates an explicit preference for simpler models using Occam's Razor as its inductive bias. Despite recent criticisms about using Occam's razor in the KDD field (Domingos, 1998) (Domingos, 1999), we believe Occam's Razor is still appropiate for classification tasks. It should be noted that an inductive bias is necessary for any classification task anyway, since, given a set of observed facts, the number of hypotheses that imply these facts is potentially infinite (Freitas, 2000).

A pseudo-code description of the ART algorithm is sketched in Figure 1.

3.1.2. *Rule mining: Candidate rules*

The first step in the ART algorithm is to discover potentially predictive rules in the training dataset. These rules will be used to grow the decision tree which ART builds. An algorithm is needed to look for underlying rules in the input dataset.

The simplest possible rule mining algorithm is to discover all the association rules which include the class attribute as their consequent and satisfy the user-specified *MinSupp* and *MinConf* thresholds.

Since dense datasets are common in the relational databases typically used for classification problems (i.e. datasets with a relatively small number of missing values), an algorithm like TBAR is the best choice (Berzal et al., 2001) because it takes advantage of the First Normal Form to reduce the size of the candidate set in the association rule mining process (i.e. the number of potentially relevant itemsets).

The *MinSupp* parameter can be a fixed number of tuples, or a percentage of the size of the current training dataset. In the first case, an absolute minimum support threshold is established beforehand and is fixed during all the stages of the decision tree learning. Using a relative minimum support threshold, the actual support is adapted to the size of the remaining dataset. For example, if *MinSupp* is set at 0.1 and we begin with 1000 tuples, the absolute minimum support will be 100 tuples at the root of the tree while it will be lower in later stages of the algorithm. If there are N tuples left in the training dataset, 0.1*N will be used as the minimum support threshold in order to obtain all the frequent itemsets in the remaining dataset, which might not be frequent

```
function ART (data, MaxSize, MinSupp, MinConf): classifier;
// data:      Training dataset
// MaxSize:   Maximum LHS itemset size
//            (default value = number of predictive attributes)
// MinSupp:   Minimum support threshold
//            (default value = 0.05 = 5% )
// MinConf:   Minimum confidence threshold
//            (automatic selection by default)

k = 1;          // LHS itemset size
tree = null;    // Resulting tree

while ( (tree is null) and (k ≤ MaxSize) )

    // Rule mining
    Find all the good rules from input data with
    k items in the LHS and the class attribute in the RHS
        e.g. {A1.a1 .. Ak.ak} ⇒ {C.cj}

    if there are candidate rules to branch the tree

      // Rule selection
      Select the best set of rules with the same set of attributes
      {A1..Ak} in the LHS according to the preference criterion.

      // Tree branching
      tree = Tree resulting from using the selected rules, where
            - Each association rule {A1.a1 .. Ak.ak} ⇒ {C.cj}
              is used to build a new leaf (labeled cj).
            - All training examples not covered by the selected
              association rules are grouped into an 'else' branch
              which is built calling the algorithm recursively:
              data = uncovered data // Transaction trimming
              tree.else = ART (data, MaxSize, MinSupp, MinConf);

    else
      k = k + 1;

if tree is null // no decision tree has been built
  tree = leaf node labeled with the most frequent class;

return tree;
```

Figure 1. ART Algorithm Outline

in the complete dataset. When setting *MinSupp* as a percentage of the size of the remaining dataset, this parameter adjusts itself to the size of the current dataset, so it does not need to be tuned by the end-user. A value between 0.05 and 0.1 seems reasonable for building decision trees which are neither too leafy nor too bare. See Section 3.4 for a more formal discussion of the *MinSupp* threshold value effect on the ART decision tree properties.

Let us now examine how to work with the *MinConf* threshold. This parameter is directly related to the confidence of each of the association rules considered and thus, it should be a near-to-1 value (0.9, for instance). But we have found that *MinConf* should not be finely tuned by the expert user. As we will show, it is better to let ART adjust it automatically.

In any case, if the expert user so requires, *MinConf* can be manually tuned. A typical example of this situation can be found in the MUSHROOM dataset from the UCI Machine Learning Repository, where a single classification mistake could have disastrous consequences. In this particular case, a false positive is not allowable because a poisonous mushroom would be labeled as edible. Obviously, that is not advisable, since a classification mistake could provoke health problems, even death. A stringent 100% minimum confidence threshold is required for the MUSHROOM dataset. In cases such as these, a user-established *MinConf* threshold must be used to ensure that the classification model obtained by ART has the desirable properties (no false positives in the previous example).

The MUSHROOM dataset is an extreme example, however. Setting a minimum confidence threshold beforehand can be counterproductive if this threshold is not realistic, since no classification model would be obtained if *MinConf* were too high for the actual dataset. In such a case, no association rules would be discovered and no suitable decision tree could be built. Moreover, a higher *MinConf* threshold does not imply greater classifier accuracy, and, from a practical point of view, our algorithm training time could be significantly increased (see Section 4).

Consequently, we need to introduce an automatic *MinConf* threshold selection method into ART. The idea underlying any such algorithm is that once the most accurate association rule has been found (i.e. the one with the best confidence value), only similarly accurate rules are used to build the tree. One heuristics we have found to work quite well is to consider only those rules with confidence above *MaxConf* - Δ in each step, where *MaxConf* stands for the maximum confidence among the discovered association rules. This heuristics uses a parameter Δ to establish a "tolerance" interval for the confidence of the rules. We select

the best possible rule and only allow slightly worse rules to be included in each set of good rules. Using this rather restrictive approach, the algorithm ensures that no bad rules will be considered in order to build new leaves. In the worst case, a good rule might not be considered at the current level of the decision tree. In any case, it will be considered later when processing the remaining data, where it will hold with even more support (since the dataset is trimmed every time that new leaves are added to the decision tree).

Our experiments have demonstrated that the automatic confidence threshold selection obtains nearly optimal solutions (when, e.g., Δ is made equal to *MinSupp*). In our opinion, a user-established minimum confidence threshold should only be used when our problem domain requires it and it therefore becomes strictly necessary (as in the MUSHROOM example).

3.1.3. *Rule selection: Preference criterion*

Once we have obtained some suitable rules for classifying input data, i.e. a partial classification model has been built, some of them must be selected to build the current level in the decision tree (which, once built, is a complete classification model).

Our algorithm checks for the existence of at least one set of suitable attributes $\{A_1\ A_2\ ..\ A_k\}$ for branching the decision tree. The selected set of attributes must lead to predictive and accurate rules of the form $\{A_1.a_1\ A_2.a_2\ ..\ A_k.a_k\} \Rightarrow \{C.c_j\}$, which will be used to classify input data as belonging to class c_j.

ART tries to make good use of the information obtained from the rule mining process. All the rules obtained at each stage of the algorithm involve k-itemsets in their left-hand side. Each k-itemset corresponds to a set of k different attributes (recall again the First Normal Form). The rules corresponding to each set of k different attributes are grouped together and heuristics are employed to choose the best candidate among these sets in order to branch the tree.

ART follows a simple heuristics which consists in choosing the set of attributes which correctly classifies more instances in the training dataset (using the previously obtained rules), provided that the corresponding rules verify the *MinConf* constraint. Such a heuristics maximizes the number of classified examples and, thus, minimizes the number of unclassified examples at each level of the tree: ART tends to reduce the height of the decision tree following Occam's Economy Principle.

3.1.4. *'Else' branches*

Once a set of rules with the same set of attributes in their antecedents has been selected during the rule selection phase, we proceed now to branch the tree using the discovered information.

- Each of the selected rules leads to a new leaf in the tree. Each leaf is labeled with the class value which appears in the consequent of the corresponding rule.
- All the training examples which are not covered by the selected rules are grouped together into an 'else' branch to be processed in later stages of our algorithm.

ART differs from traditional TDIDT algorithms because it uses 'else' branches to group all the tuples which have not been covered by the selected rules at each level of the decision tree. In other words, the remaining training examples are treated as a whole. ART does not therefore need to build a new branch for each possible value of the selected attributes. ART focuses its attention only on those values which yield interesting results, reducing the decision tree branching factor. This fact, in turn, lets ART choose a set of attributes to branch the decision tree, instead of the single attribute or binary test most TDIDT algorithms use.

The use of 'else' branches in decision lists can make classification models harder to understand by humans when compared to standard decision trees, since the meaning of a given rule depends on its position in the list (Segal and Etzioni, 1994). However, it also helps to build better classifiers by grouping small datasets which would otherwise correspond to different subtrees in TDIDT algorithms. Moreover, these datasets would eventually become too small to be able to build accurate classifiers. Better prediction accuracy is then achieved by joining all these small datasets, since the effect of noise in input data is diminished and the overfitting problem is avoided.

In some sense, ART embeds the philosophy of the algorithms proposed by Liu et al. (Liu et al., 2000a) (Liu et al., 2000b), who try to organize and summarize the set of all the discovered rules intuitively by using general rules, summaries and exceptions. The idea is to replace a set of potentially too many rules by a more concise representation of the discovered knowledge, which is also easier to work with. The ART approach achieves this goal while creating its classification model, while Liu et al. attain it with hindsight.

It should be noted that ART also differs from conventional decision list learners because it discovers multiple rules at a time, so that the rules learnt at a given level in the tree are independent and mutually

exclusive, whereas rules are strictly ordered in standard decision lists. This fact reduces the decision list depth and partially mitigates the understandability problem mentioned above, which is inherent to the decision list model.

Once we have described our proposed classification model, we present one application where we have obtained interesting results in order to illustrate ART main features.

3.2. An example application: The SPLICE dataset

Our aim is to determine the type of a DNA splice junction (exon/intron, intron/exon, or neither) given a primate splice-junction gene sequence, which consists of 60 nucleotides. The following problem description is a verbatim transcript from the UCI Machine Learning Repository documentation:

> Splice junctions are points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as EI sites), and recognizing intron/exon boundaries (IE sites). (In the biological community, IE borders are referred to as "acceptors" while EI borders are referred to as "donors".)

For this particular problem, ART algorithm builds a classifier which achieves excellent classification accuracy and whose size is still manageable. TDIDT algorithms, such as C4.5, obtain slightly better classification accuracy (above 90% using cross-validation) but they also require much larger decision trees which are harder for humans to understand. The C4.5 decision tree, for example, would require between three and four times the space occupied by our tree even after pessimistic pruning.

Moreover, ART exploits the symmetries around the junction (which occurs between P30 and P31) to attain a better understanding of the underlying patterns. Observe, for example, the P29-P31, P28-P32 and P25-P35 pairs used to branch the decision tree. Typical TDIDT algorithms cannot use such associations and thus their applicability to problems where correlations play an important role is restricted. Although they might obtain better classification accuracy, ART provides simpler and much more understandable classification models.

```
P30 = A : TYPE = N (473|62)
P30 = C : TYPE = N (441|24)
P30 = T : TYPE = N (447|57)
else
  P28 = A and P32 = T : TYPE = EI (235|33)
  P28 = G and P32 = T : TYPE = EI (130|20)
  P28 = C and P32 = A : TYPE = IE (160|31)
  P28 = C and P32 = C : TYPE = IE (167|35)
  P28 = C and P32 = G : TYPE = IE (179|36)
  else
    P28 = A : TYPE = N (106|14)
    P28 = G : TYPE = N (94|4)
    else
      P29 = C and P31 = G : TYPE = EI (40|5)
      P29 = A and P31 = A : TYPE = IE (86|4)
      P29 = A and P31 = C : TYPE = IE (61|4)
      P29 = A and P31 = T : TYPE = IE (39|1)
      else
        P25 = A and P35 = G : TYPE = EI (54|5)
        P25 = G and P35 = G : TYPE = EI (63|7)
        else
          P23 = G and P35 = G : TYPE = EI (40|8)
          P23 = T and P35 = C : TYPE = IE (37|7)
          else
            P21 = G and P34 = A : TYPE = EI (41|5)
            else
              P28 = T and P29 = A : TYPE = IE (66|8)
              else
                P31 = G and P33 = A : TYPE = EI (62|9)
                else
                  P28 = T : TYPE = N (49|6)
                  else
                    P24 = C and P29 = A : TYPE = IE (39|8)
                    else
                      TYPE = IE (66|39)
```

Figure 2. ART classifier for the SPLICE dataset

```
function classify (art, instance): class;
// Input:    art = ART classifier
//           instance = Unlabeled instance
// Output:   class = Assigned class

Match the value(s) of the instance attribute(s)
      with the value(s) used to branch the tree

if the value(s) correspond(s) to
     any of the branches leading to a leaf
  return the corresponding leaf class
else if there exists an else branch // follow it
  return classify(art.else,instance);
else
  return default class;
```

Figure 3. Unlabeled data classification

3.3. Using the classifier

The classifier obtained by ART can be used to classify unlabeled instances as any other algorithm of the TDIDT family. Beginning at the root of the tree, a given example follows different branches of the tree depending on its attribute values (as if the tree nodes were railroad switches). Any example will reach a tree leaf eventually, and it will be labeled with the most common class in that leaf. The ART classification process is described in Figure 3.

When building the classifier, null values can be automatically sent to the 'else' branch, since they are not covered by the discovered association rules. However, when classifying data with unknown values for some attributes, another alternative could be followed, such as the one used by C4.5, for instance.

Decision trees are also popular because IF-THEN rules can be derived from them trivially. When trying to obtain a set of rules from the decision tree, ART differs from traditional TDIDT algorithms since it employs 'else' branches. This kind of branch causes negations to appear in the rule antecedents when deriving production rules from the decision tree.

In addition, a simple rule-based procedure to evaluate an ART classifier exists, since ART classifiers can be viewed as decision lists. These decision lists may end up in a default class value (produced when no association rules are found to branch the decision tree further). Even worse, when the association rules used to build the tree cover all the

examples in the input dataset, a given instance may lead to nowhere in the decision tree. In this unlikely but possible case, that instance would be labeled with the most common class in the current subtree (i.e. in the training data which led to that subtree).

3.4. ART Classifier properties

In this section, we will discuss further several key properties of our method for building classifiers:

Search strategy: ART performs an exhaustive global search of potentially interesting rules in the training dataset but it makes that search local to the remaining dataset in the current level of the tree. In this way, the efficiency which characterizes the heuristic greedy search used in typical TDIDT algorithms is combined with the power of the exhaustive search performed by the association rule mining algorithm. In fact, trying to classify as many examples as possible at each level of the decision tree helps ART to take better local decisions when building the classification model.

In some respects, ART follows Michalski's STAR methodology (Sestito and Dillon, 1994, chapter 3), as CN2 does, since it generates rules iteratively until a complete classification model has been built. It should be noted however that ART, as decision list inducers, removes both positive and negative examples covered by the generated rules from the current dataset, while other STAR algorithms must keep all negative examples when trying to build new hypotheses, which leads to more complex rules. This removal of covered instances is referred to as 'transaction trimming' in Data Mining (Park et al., 1997) and it helps to reduce the number of I/O operations needed to build the classifier.

It should also be noted that ART searches for sets of rules in parallel, while other algorithms try to find one rule at a time. Moreover, ART performs this search without incurring in a severe performance penalty (as BruteDL does, for instance).

Although a greedy algorithm was chosen to build the ART classifier, other approaches would also be feasible. For example, a beam search could be performed to find better classifiers. However, efficiency is a must in Data Mining problems and greedy algorithms are the best choice in this case.

Robustness (outliers & primary keys): The use of concepts taken from association rule mining helps to build more robust classifiers, minimizing the effects of noise in the input data.

The minimum support threshold makes isolated outliers harmless since they are not taken into account when deciding how to build the classifier: association rules handle noisy data seamlessly. Moreover, the support threshold also removes the problems faced by other TDIDT algorithms when some attributes are nearly keys (e.g. ID3 would always choose these kinds of attributes because they minimize the entropy in the resulting subtrees) without the need for more artificial mechanisms (such as C4.5 gain ratio criterion).

ART therefore provides a uniform treatment of outliers (which do not contribute to association rules significantly), and primary and candidate keys (whose values do not have enough support to generate association rules on their own). Both outliers and primary keys are thorny issues for traditional TDIDT algorithms.

Tree complexity: Both the height and the width of the decision tree are restricted by the minimum support threshold.

Given an absolute minimum support threshold *MinSupp* as a normalized value between 0 and 1, the ART decision tree will have at most *1/MinSupp* levels, because at each level *n*MinSupp* instances will be trimmed at least, where n is the number of instances in the training dataset.

When an absolute minimum support threshold is used, no more than *MaxSize*(1/MinSupp)* scans over the input data will be needed to build the complete classifier, since all the association rules can be obtained with, at the most, *MaxSize* scans over the training data using simple algorithms such as Apriori. In other words, ART is *O(n)* on the size of the training dataset. This fact is essential for successful data mining.

The tree branching factor is also determined by the minimum support threshold. *1/MinSupp* is an upper bound on the tree branching factor regardless of whether the minimum support threshold *MinSupp* is absolute or relative, because no more than *1/MinSupp* rules can be selected at each level of the tree (an extreme case would occur when all training instances are classified at the same level of the tree using the maximum possible number of rules, all of them with the minimum allowable support).

Table I. Datasets used in our experiments

Dataset	*Examples*	*Attributes*	*Classes*
AUDIOLOGY	226	70	24
CAR	1728	7	4
CHESS	3196	36	2
HAYES-ROTH	160	5	3
LENSES	24	6	3
LUNG CANCER	32	57	3
MUSHROOM	8124	23	2
NURSERY	12960	9	5
SOYBEAN	683	36	19
SPLICE	3175	61	3
TICTACTOE	958	10	2
TITANIC	2201	4	2
VOTE	435	17	2

4. Empirical results

We have implemented ART in Java 2 using Sun JDK 1.3. Our program accesses data stored in relational databases through JDBC (which stands for 'Java Database Connectivity', the standard call-level interface of the Java programming language). Our implementation of ART makes use of TBAR (Berzal et al., 2001), an Apriori-like algorithm for finding frequent itemsets (Agrawal and Skirant, 1994). We also used AspectJ, an aspect-oriented extension of the Java programming language (Kiczales et al., 2001), in order to monitor I/O operations in our experiments.

All the results reported in this section were obtained using 10-CV (ten-folded cross-validation). The tests were carried out on a Pentium III 1100MHz PC running MS Windows NT 4.0 WorkStation with 128MB of RAM. The back-end database used in our experiments was InterBase 6 although any other database would work (in fact, we also tested ART against Oracle and IBM DB2 servers).

Table I shows the datasets we used in our experiments, which were downloaded from the UCI Machine Learning Repository.