A Data Structure to Represent Association Rules based Classifiers

Paulo J Azevedo*
Departamento de Informática
Universidade do Minho
Braga, Portugal
pja@di.uminho.pt

ABSTRACT

We tackle the problem of representing association rules for a prediction purpose. We approach this problem by introducing a novel data structure for representing association rules (now seen as classification/regression rules). Unseen cases are fitted into a graph like structure that avoids any type of sorting procedure. The graph indexes the items present in the rules so that only rules with the antecedent covered by the new case are visited.

A detailed description of the data structures to store the association rules is given along with the most important steps of the algorithm. Benchmarking and discussion on the main features is also presented.

1. INTRODUCTION

Many association rule based classifiers had been proposed in the literature. The basic idea is to generate rules with a single item in the consequent and to select rules with the defined target attribute occurring at the consequent. These rules are known as CAR (Classification Association Rules) rules. The prediction procedure works by selecting rules whose antecedent covers a new instance (case) to be classified. Then, an order is imposed on these rules according to a measure, typically rule strength (e.g. confidence, lift). The best rule is chosen to fire and the new case prediction is the consequent of this rule. This procedure is known as BestRule prediction. Examples of this approach can be found in [6; 5; 4], However, not many proposals have considered the problem of efficiently representing association rules as prediction models i.e with the aim of performing prediction for new cases. In this paper, we tackle the program of efficiently store rules as prediction models. The aim is to store rules in such a way that:

1. Fast indexing of the rules that cover a specific unseen case is provided.
2. Rules are efficiently stored by eliminating redundancy between items on the antecedent.
3. Easily gather and order the set of triggered consequents for an unseen case.

*Supported by the POSI/SRI/39630/2001/Class Project (Fundação Ciência e Tecnologia), FEDER e Programa de Financiamento Plurianual de Unidades de I & D.

The first requirement has to do with the expected ability to identify the rules that contribute to the unseen case. The efficiency of the prediction procedure is highly dependent on this capability. The second statement aims for a very compact method to store the rules. Items that appear in two rules should not appear twice to represent the rules. Finally, the last requires that, for each new case, the set of consequents fired to build the prediction be computationally efficient obtained. In this paper we will suggest a data structure that covers the three requirements.

The rest of the paper is organized as follows: We proceed by describing the hash graph structure. Then, a description of the prediction algorithm is presented. The implementation of the CAREN system [8] is described. Finally, benchmarking and related work are discussed.

2. CAR RULES

As proposed by several authors [6; 5; 4], association rules can be used as classification rules. The idea is to impose constrains into the frequent mining algorithm so that it derives rules with a single item consequent belonging to the target (class) attribute.

Association rules are rules of the form

$$a_1 \ \& \ a_2 \ \& \ ... \ \& \ a_n \rightarrow c$$

Such rules describe association (or simply co-occurrence) between atomic elements present in data. These elements can be items bought in supermarket or genes present in a certain chromosome, or simply a pair of attribute/value items from a relational database. Rules are made out of itemsets observed in the dataset, i.e. combination of items. For instance, itemset $a_1a_2a_3...a_nc$ gave rise to the former rule. Quality of rules is measured through statistical measures like support and confidence. Support describes incidence of the itemset in the database and confidence measure the predictability strength of the rule. Support is calculated by itemset counting among the transactions contained in the dataset. Although in general they have multiple items in the consequent, here we focus on rules with a singleton right-hand-side. This emphasizes the classification purpose, where the consequent is expected to contain the class attribute.

The prediction procedure follows closely the one proposed in [5]. These authors suggest that confidence should be used to obtain rule's ordering. Ties are solved by consulting support and rule's length. We follow the same policy but generalize to any rule's measure (not just confidence). Such measures

asses the predictive ability of the rules. In this setting, measures like *confidence*, *conviction* and *lift* can be used. We will refer to these, generally, as interest measures. The ordering procedure is summarized as follows:
Given R_1 and R_2 we say that R_1 precedes R_2

$R_1 \succ R_2$ if
$int(R_1) > int(R_2)$ or
$int(R_1) == int(R_2) \wedge sup(R1) > sup(R2)$ or
$int(R_1) == int(R_2) \wedge sup(R1) == sup(R2)$
$\wedge \#ant(R1) < \#ant(R2)$.

where *int* is the used interest measure and $\#ant$ is the length of the antecedent.
The approach described in [5] and [6] contain a rules selection procedure that reduces to a coverage algorithm. First, it orders the rules following the rank descending order described before (relation $\succ$). Then, it selects a rule from the top of the rank that covers and correctly classifies an instance from the training set. The procedure stops when all the instances are covered by a selected rule. This process is executed in a post-processing step i.e after the derivation of rules occurs. Our approach specifically does not apply any sort of coverage algorithm to select rules. Instead, we make use of the *improvement* measure [2], the χ^2 test between antecedent and consequent and the traditional minconf, minsup constraints to select rules. Furthermore, rules selection occurs during rules derivation rather than as a post-processing step. We believe that applying a coverage algorithm entails information loss and consequently prediction power degradation. Although no evidences for this claim will be shown, experiments performed in the past suggest that there is a large opportunity for the degration to occur.

A question arises when using rules as prediction models, which is how to efficiently select rules that fire for a given case. We call this problem *the antecedent cover problem.* To solve it most authors propose dataset indexing techniques rather than rules indexing data structures. We proceed by proposing a data structure to attain the antecedent cover problem. This data structure indexes and efficiently stores rules without antecedent redundancy.

3. DATA STRUCTURE

We introduce a novel data structure to represent association rules for a classification task. The purpose of this data structure is to optimize storing space and computational time related to the prediction task. In the sequel, we will replace the term "prediction" by "classification". However, one could be performing a regression (numeric prediction) task using the same prediction algorithm. Here, classification refers to the task of collecting the consequents of the rules whose antecedents are covered by the new case, i.e., rules that fire for the new case.

The structure (which in the sequel will be referred as *HashGraph*) is item oriented. It grows out of an array of frequent items. The order in this array is the same as the items order imposed by the frequent patterns mining algorithm (we used support ascending order). Rules are represented through a *trie* like structure, a kind of discrimination tree. Tries are well known data structures suited to index strings, enabling to collapse operations of insertion and retrievial. Each item in the array contains an associated trie to represent rules where the same item is the first element at the antecedent. Rule's antecedents also follow the order imposed in the array.
We will follow figure 1 to describe the hashgraph data structure. A rule is represented by a set of items, corresponding to the antecedent, plus a last item representing the consequent (in our case it would correspond to a class value). The leaf node in a path of the trie (class node) represents the last item of the antecedent. It also contains the consequent item and information about the rule's metrics. In figure 1 it is pictorially represented by hexagon nodes. Each position in the array of items contains a Boolean field. This is used to signal that an unseen case (to be classified) contains the item. We refer to an item having the Boolean field with the *true* value as *light on* item. The procedure to verify whether a new case covers the antecedent of a rule reduces to check if the items in the antecedent are on. Notice that this simple mechanism eliminates the need to reorder the items of a new case (according to the order in items array), each time it is used for classification.
In figure 1 an hashgraph is pictorially described. Notice the class nodes (hexagons) where the last item of the antecedent, the class label and rule's metrics are represented. Antecedent nodes (rounded boxes) store a single item from the antecedent. The "look back" arrows represent the checking for items in *light on* state.

In terms of compactness, the hashgraph structure depends on the items ordering. The ability of item sharing between rules is proportional to the support of that item. That is, high support items yield higher sharing capability.

4. ALGORITHM - PREDICTION TASK

The algorithm to collects the rules that fire given a new case is described as follows:

Input: Unseen case(set of items)
Switch to *light on* (in the binary array) the items present in the new case;
foreach *item in the array that is on* **do**
  Follow each path of the trie that contains items *on* ;
  if *consequent node reached* **then**
    collect it;
  end
end
Switch to *light off* the items present in the new case;
Output: the set of consequents
Algorithm 1: Algorithm Collect_Firing_Rules

As we can see, the prediction algorithm is based on the simple idea of following *light on* items and process the associated rules. No case (instance) reorder is required.

5. IMPLEMENTATION

We implemented the described algorithm in the Caren system [8]. Caren contains a specific module (*carenclass*) to generate association rules with the purpose of classification. Besides the basic parameters, this module requires the user to specify the consequent (attribute or item) to

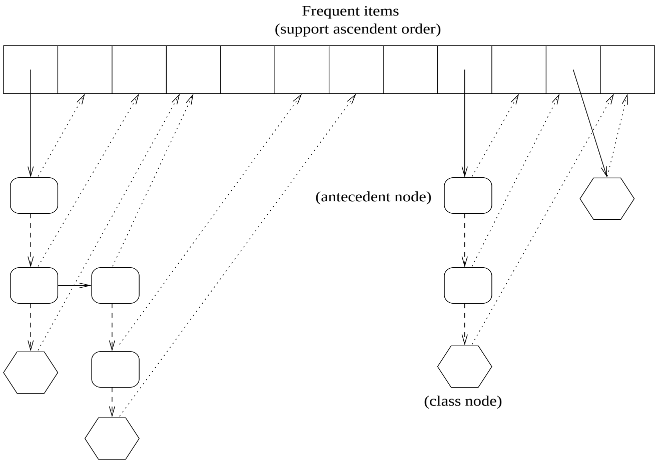

Figure 1: HashGraph for storing association rules as a prediction model

generate rules for. *Carenclass* implements a bitwise depth-first frequent patterns mining algorithm. It resembles the ECLAT algorithm proposed in [9], since it is a depth first algorithm that also makes use of a vertical representation of the database. It also has similarities to the OPUS_AR procedure [11] since it aims to construct the rules rather than first deriving the itemsets. We use bitmaps to represent items coverage. This is the formalism used to obtain a vertical representation of the database. Along the execution of the algorithm, bitmaps for the itemsets are obtained by performing bitwise operation between the bitmaps of the composed items. The complete procedure is summarized in algorithm 2.

In this algorithm, the operator # refers to bitcounting for determining the support of itemsets and rules. It is given a database DB, a set of consequent items $CONS$ and threshold values for *minsup* and *minint*. The latter is a constraint for a rule's predictability strength. The algorithm returns a set of rules that satisfies the constraint for consequent, support and strength.

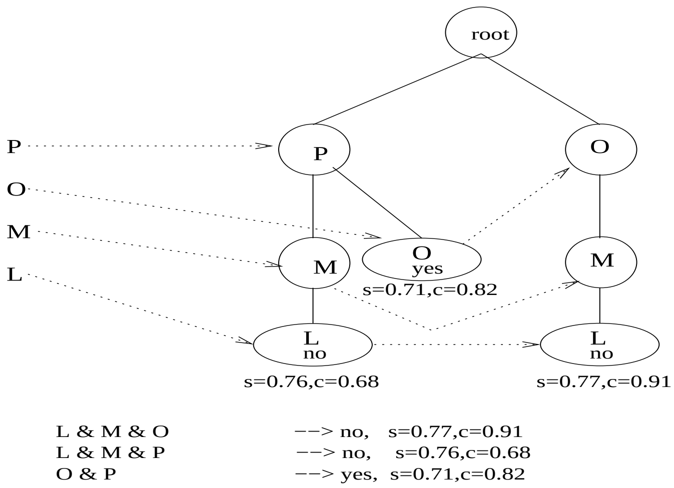

Figure 2: Comparison with CR-trees.

	T1	T2	T3	T4
minsup	0.05	0.03	0.03	0.02
minconf	0.4	0.4	0.3	0.5
num cases	125 811	671 599	671 599	1 463 927
num rules	2 295	4 437	4 444	2 799 617
time	0'26"	0'56"	0'55"	1h51'53"

Table 1: Benchmark on models built out of **t40.i10.d100** dataset

	M1	M2	M3
minsup	0.1	0.07	0.05
minconf	0.5	0.6	0.4
num rules	279 046	675 038	1 843 608
time	4'32"	9'01"	23'53"

Table 2: Benchmark on models built out of **mushroom** dataset (num of cases = 8124)

	C1	C2	C3	C4
minsup	0.5	0.57	0.6	0.65
minconf	0.65	0.6	0.6	0.65
num rules	1317840	86920	16304	52
time	14h02'02"	43'23"	7'39"	9"

Table 3: **connect-4** dataset results (num of cases = 67557)

Input: minsup,minint,DB,CONS
Rules := $\emptyset$;
First *DB* scan (count items)
  $A := \{\forall x \in Items(DB), x \notin CONS : \ count(x) \geq minsup\}$;
  $C := \{\forall x \in CONS : count(x) \geq minsup\}$;
  Reorder the items in A according to support ascendant order;
Second *DB* Scan
  Define a bitmap for each frequent item in A and C;
  Define a flat matrix ($sup2[\,]$) for 2-itemset counting;
/*(derive cover for each frequent item) */
foreach *transaction* $t \in DB$ **do**
    Set correspondent bit in each bitmap of a frequent item (in A and C) present in t;
    Count 2-itemsets occurring in t;
end
/*(Expansion phase) */
foreach *frequent item* $i \in A$ **do**
    $Rules := Rules \cup \{\forall c \in C,\ sup2[i][c] \geq minsup,\ int(i \rightarrow c) \geq minint \ : \ i \rightarrow c\}$;
    foreach *frequent item* $i' \in A \ : \ i' \succ i$ *($\succ$ refers to item ordering)* **do**
        if $sup2[i][i'] \geq minsup$ **then**
            $a := \{i, i'\}$;
            $bitmap(a) := bitmap(i) \oplus bitmap(i')$;
            $sup(a) := \#(bitmap(a))$;
            $Rules := Rules \cup \mathtt{Expansion}(\mathtt{a}, \mathtt{i}', \mathtt{A}, \mathtt{C})$;
        end
    end
end
Output: *Rules*

Algorithm 2: Algorithm Depth First Expansion

Input: (itemset,lastitem,A,C)
$R := \emptyset$;
foreach $i \in A \ : \ i \succ lastitem$ *($\succ$ refers to item ordering)* **do**
    if $\forall\ a \in itemset \quad sup2[a][i] \geq minsup$ **then**
        $new := itemset \cup \{i\}$;
        $bitmap(new) := bitmap(itemset) \oplus bitmap(i)$;
        $sup(new) := \#(new)$;
        if $sup(new) \geq minsup$ **then**
            foreach $c \in C$ **do**
                if $\#(bitmap(new) \oplus bitmap(c)) \geq minsup$ **then**
                    if $int(new \rightarrow c) \geq minint$ **then**
                        $R := R \cup \{new \rightarrow c\} \cup \mathtt{Expansion}(\mathtt{new}, \mathtt{i}, \mathtt{A}, \mathtt{C})$;
                    end
                end
            end
        end
    end
end
return R

Function `Expansion`(*itemset,lastitem*)

Typically, association rules generation is a post-processing task. That is, it is executed after the frequent pattern mining algorithm determines which itemsets are valid. For efficiency purposes, it is desirable to push the rules generation task into the frequent pattern mining phase. This way, as soon as an frequent itemset is counted and checked valid (for instance, that it contains the required consequent item), rule generation for that itemset is triggered. However, depth-first approaches to itemset counting face a problem. It may happen that subsets of the itemset in question are not yet determined. Thus, we might have a rule ready to be derived but that does not have the antecedent support counted. Algorithms like Apriori [1] do not suffer from this problem. Being a breath-first approach imply that all subsets are determined for a counted itemset. Carenclass has a simple and elegant approach to this problem. Since it knows in advance what items it will generate rules for (they will occur in the consequent) it re-allocates the items defined as consequents in a separate list (variable C in algorithm 2). This ensures two things: first, consequent items are the last to "join" the itemset obtained from depth-first expansion; secondly, when about to generate a rule, the support of the antecedant itemset (without the consequent item) is already known. The same approach is used to apply χ^2 filtering to itemset counting.

In terms of memory requirements, *carenclass* consumption is at most:

$$memory = (2 \,\mathtt{x}\, \#frequent\_items) \,\mathtt{x}\, BitMap\_size$$

$BitMap\_size$ is the size of the memory word to represent an item coverage. The memory word size in bits must be at least the number of database transactions. $\#frequent\_items$ is the number of frequent items. The *times two* represents the bitmap for each frequent item and the bitmap for the branch of depth-first expansion, which is at most the size of the maximal itemset minus one i.e. $\#frequent\_items - 1$. An extra bitmap is needed to represent the actual itemset. The carenclass module constructs a prediction model out of the derived rules. It organizes the rules into an hashgraph. The Caren system performs classifications through the *Predict* module. *Predict* [8] makes use of an hashgraph structure to represent association rules and classify test data.

6. BENCHMARKING

We projected several testes to demonstrate the suitability of the proposed data structure. Some datasets considered computationally heavy were selected for benchmarking. One, *t40.i10.d100*, was generated using IBM synthetic datasets tool. The very dense *connect-4* and *mushroom* datasets from the Irvine collection [12] were also used. The sets of rules derived from these datasets tend to be very large, which entails prediction models with exagerated number of rules. A typical association model for prediction contains hundreds of rules, not millions. However, for benchmarking purpose we chose very large sets of rules to verify linearity along the number of rules.
The tests, shown in table 2 and 3, actually show linearity along the number of rules. The test described in table 1 shows linearity along the number of rules and number of cases. For *connect-4*, the same accuracy was obtained for all 4 models. All datasets were used both as training set, to derive association rules, and as test set. In basket data like dataset *t40.i10.d100*, each case is used in a all-but-one fashion. Thus, a transaction with n items (size) yields n different cases, where the hidden item plays the role of the class value to be predicted.
Test were performed using a 1.3MHz Pentium III machine with 2 Gigabyte of main memory.

7. RELATED WORK

Most proposals for implementing algorithms to build models for classification seem to focus their overall performance in the efficient retrieval of training cases. The training dataset is indexed and typically loaded into main memory e.g. [3]. For instance, [7] proposes a data structure to index data according to the values that each example takes for each attribute. This enables a fast evaluation of a new case on a set of decision rules since, to evaluate one instance, it does not require processing all the examples.

Our work concentrate on organizing rules to favor a faster evaluation. In this context, the CMAR [5] approach to represent association rules is the most similar to ours. This work represents rules using a FP-tree like structure, called CR-tree. Figure 2 presents the CR-tree representation of the last three rules described in figure 3. The CMAR approach requires transaction reordering. To classify a new transaction, items present in the case (transaction) must be ordered according to the imposed item order supplied by the CR-tree. In an HashGraph, no items reordering is required. The only computational burden required to classify a new case is to signal, in the binary array, the items present in the transaction.

Figures 3 and 2 suggest that an HashGraph collapses more information than a CR-tree. For the same set of rules (the ones in figure 2), a smaller number of nodes is required by an hashgraph than a CR-tree to represent them (three last rules in figure 3). Thus, apart from being a faster data structure in supplying prediction for new instances, Hashgraphs provides the most compact formalism to represent the same set of rules.

8. CONCLUSIONS

Caren was developed with the aim of producing a tool capable of generating classification rules, although it can also be used for undefined item consequent association rules generation. By classification rules, we meant rules that can be used with a classification purpose where a prediction model makes use of them in the form of a decision list.

We proposed a novel data structure ($HashGraph$) for storing association rules as classification rules. It covers the three requirements that is expected when performing prediction using rules.
We also propose a simple solution to push into the itemset mining process the association rules derivation procedure. As shown before, it is not trivial to implement this feature into a depth-first frequent pattern mining algorithm.

One should bear in mind this fact and that Caren and *Predict* are Java-based, when comparing performance with implementations like [5]. Despite this issue, benchmarking suggests that hash graph is an interesting formalism to represent rules for prediction.

Acknowledgments

Thanks to Alípio Jorge for all the suggestions and testing. This work was developed during the execution of the CLASS project (POSI/SRI/39630/2001/Class Project - Fundação Ciência e Tecnologia)

9. REFERENCES

[1] Agrawal R., Srikant R.
Fast Algorithms for Mining Association Rules
in Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile, Sept. 1994

[2] Bayardo, R.J., Agrawal, R., Gunopulos, D.,
Constraint-Based Rule Mining in Large, Dense Databases
in Data Mining and Knowledge Discovery, Volume 4, Issue 2 - 3, Pages 217 - 240, (2000)

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

[4] Jovanoski V., Lavrac N.
Classification Rule Learning with Apriori-C
in Proceedings of the 10th Portuguese Conference on Artificial Intelligence, EPIA 2001, LNCS 2258, Porto, Portugal.

[5] Li W., Han J., Pei J.
CMAR: Accurate and Efficient Classification based on MultiClass-Association Rules
in Proceedings of the IEEE International Conference on Data Mining, 2001.

[6] Liu B., Hsu W., Ma Y.
Integrating Classification and Association Rule Mining
in Proceedings of the ACM SIGKDD International COnference on Knowledge Discovery and Data Mining (KDD98), New York 1998.

[7] Girldez R., Aguilar-Ruiz J., Riquelme J.
An efficient data structure for decision rules discovery
in Proceedings of the ACM Symposium of Applied Computing (SAC2003), Florida 2003.

[8] Azevedo P., Jorge A.
The CLASS Project
http://www.niaad.liacc.up.pt/~amjorge/Projectos/Class/

[9] Zaki M.J.
Scalable algorithms for association mining
IEEE Transactions on Knowledge and Data Engineering, 12(3):372-390, May-June 2000.

[10] Merz J., Murphy P.,
UCI repository of Machine Learning Database
http://www.cs.uci.edu/ ~mlearn

[11] Webb G.
Efficient search for association rules
in the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00), Boston, MA 2000.

[12] C. J. Merz and P. Murphy.,
Uci repository of machine learning database
in http://www.cs.uci.edu/~mlearn, 1996.

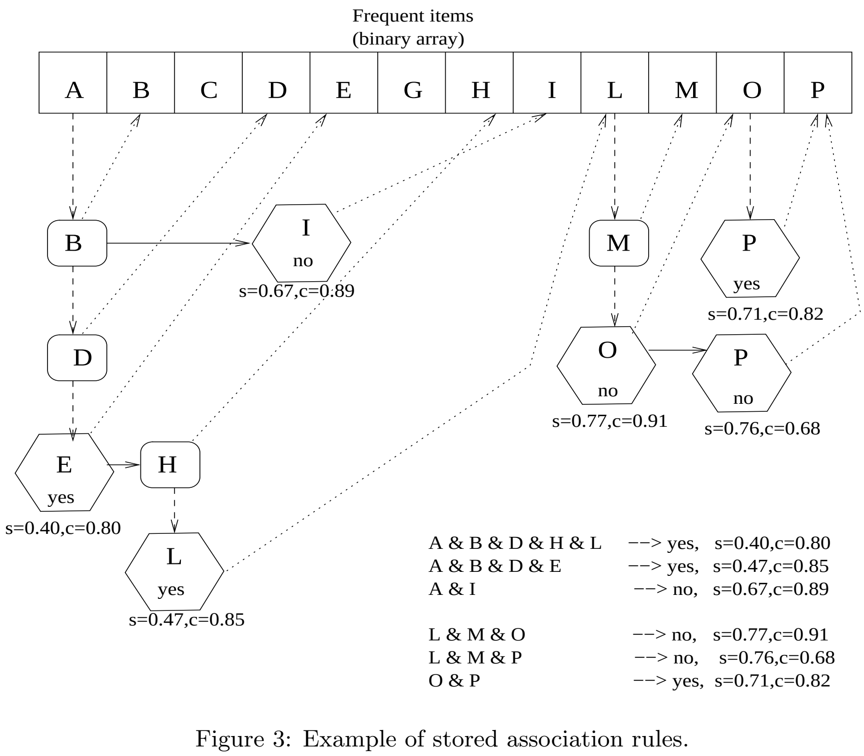

Figure 3: Example of stored association rules.