

# Finding representative association rules from large rule collections

Warren L. Davis IV\*

Peter Schwarz†

Evimaria Terzi‡

## Abstract

One of the most well-studied problems in data mining is computing association rules from large transactional databases. Often, the rule collections extracted from existing data-mining methods can be far too large to be carefully examined and understood by the data analysts.

In this paper, we address exactly this issue of overwhelmingly large rule collections by introducing and studying the following problem: Given a large collection  $\mathcal{R}$  of association rules we want to pick a subset of them  $S \subseteq \mathcal{R}$  that best represents the original collection  $\mathcal{R}$  as well as the dataset from which  $\mathcal{R}$  was extracted. We first quantify the notion of the goodness of a ruleset using two very simple and intuitive definitions. Based on these definitions we then formally define and study the corresponding optimization problems of picking the best ruleset  $S \subseteq \mathcal{R}$ . We propose algorithms for solving these problems and present experiments to show that our algorithms work well for real datasets and lead to large reduction in the size of the original rule collection.

## 1 Introduction

The problem of discovering association rules from transaction data was introduced by Agrawal, Imielinski and Swami [1]. The same basic idea of finding associations (or causal relationships) between attributes has motivated lots of research on how to find “interesting” association rules and how to compute them efficiently. Inevitably, different measures of interestingness lead to different algorithmic and other technical challenges. For an indicative, though not complete set of references on those topics see [2, 10, 7, 17] and references therein. The main characteristic of this line of research is that it focuses on the discovery of *all* association rules that satisfy a minimum interestingness criterion. Although the extracted rules may find hidden associations from the data, it is often the case that the collection of association rules output by such algorithms is too large. Thus, it is impractical for someone to go through all

these rules and draw conclusions from them.

In this paper we consider the problem of representing a set of association rules extracted using any rule-mining algorithm, by a smaller, yet sufficient, subset of them. We want the rules that we select in this subset to be *representative* of the input collection. The premise of our work is that small sets of representative rules can give a better understanding of the global structure of the dataset without significant sacrifice of information.

We measure the goodness of a set of rules by how well they can predict certain right-hand sides. As an example consider the database shown in Table 1 and rules  $r_1 : \{\text{acetaminophen}\} \rightarrow \{\text{liver damage}\}$ ,  $r_2 : \{\text{acetaminophen, alcohol}\} \rightarrow \{\text{liver damage}\}$ ,  $r_3 : \{\text{alcohol, flu shot}\} \rightarrow \{\text{liver damage}\}$  and  $r_4 : \{\text{alcohol}\} \rightarrow \{\text{acetaminophen}\}$ . Assume that we are interested in all conditions that lead to  $\{\text{liver damage}\}$ . That is, we fix the consequent of interest to  $\{\text{liver damage}\}$ . Among all the rules that have  $\{\text{liver damage}\}$  as their consequent (rules  $r_1, r_2$  and  $r_3$ ) we want to pick the ones that are good predictors for this specific consequent. More specifically, given an arbitrary transaction from the database, and hiding the knowledge of whether  $\{\text{liver damage}\}$  appears in the transaction, we want to use the set of rules we select to predict whether the transaction has that consequent. The goal is to make as many correct predictions as possible.

Of course, it all depends on how the information from a set of rules is aggregated in order to make the prediction. For example,  $r_1$  above suggests that  $\{\text{acetaminophen}\}$  is enough indication to predict  $\{\text{liver damage}\}$ , while  $r_2$  says that  $\{\text{acetaminophen}\}$  by itself is not enough if it is not accompanied by  $\{\text{alcohol}\}$ . Therefore, we need an aggregation function that combines information from sets of rules and predicts  $\{\text{liver damage}\}$  for a specific transaction. In the rest of the paper we show how all these notions can be formally defined and how they lead to some interesting combinatorial problems.

**1.1 The problem** At a high-level we address the following problem: Given a set of transactions  $T$  and a set of association rules  $\mathcal{R}$  mined from  $T$ , pick  $S \subseteq \mathcal{R}$  so that  $S$  is a good representation of both  $\mathcal{R}$  and  $T$ .

In our setting, we split this problem into subprob-

\*IBM Almaden Research Center, San Jose, CA, USA. Email: wdavis@us.ibm.com

†IBM Almaden Research Center, San Jose, CA, USA Email: schwarz@us.ibm.com

‡IBM Almaden Research Center, San Jose, CA, USA. Email: eterzi@us.ibm.com

lems defined by specific rule consequents: if there are  $\ell$  distinct consequents appearing in the rules in  $\mathcal{R}$ , then we partition  $\mathcal{R}$  into  $\ell$  groups  $\mathcal{R} = \{\mathcal{R}_{y_1}, \dots, \mathcal{R}_{y_\ell}\}$ ; all rules in  $\mathcal{R}_{y_i}$  share consequent  $Y_i$ . For each  $\mathcal{R}_{y_i}$  we solve the problem of finding a set  $S_{y_i} \subseteq \mathcal{R}_{y_i}$  of good representative rules. Then  $S = \bigcup_{i=1}^{\ell} S_{y_i}$ . The selection of rules in each  $S_{y_i}$  is made based on how well rules in  $S_{y_i}$  can predict the existence (or not) of consequent  $Y_i$  in the original set of transactions, had this information been hidden in the prediction process.

The way the original set of rules  $\mathcal{R}$  is extracted is indifferent to the methods presented in this paper. In practice, it can be extracted using any of the data-mining algorithms that appear in the literature or simply domain knowledge.

**1.2 Roadmap** The rest of the paper is organized as follows: in Section 2 we give the necessary notation, and in Section 3 we provide the formal problem definitions and give some complexity results. Algorithms are described in Section 4 and experiments in Section 5. We compare our work to previous work in Section 6 and conclude the paper in Section 7.

## 2 Preliminaries

Assume a transaction database  $T = \{t_1, \dots, t_n\}$  consisting of  $n$  transactions, and a collection  $\mathcal{X} = \{X_1, \dots, X_N\}$  of different items that every transaction may contain. For transaction  $t \in T$  and  $X \subseteq \mathcal{X}$  we use  $t[X] = 1$  to denote that all items in  $X$  appear in transaction  $t$ , otherwise we use  $t[X] = 0$ . We use  $T_X$  to denote the subset of transactions in  $T$  for which  $t[X] = 1$ .

An association rule  $r : X \rightarrow Y$  consists of a *left-hand side* (LHS)  $X \subseteq \mathcal{X}$  and a *right-hand side* (RHS)  $Y \subseteq \mathcal{X}$ , such that  $X \cap Y = \emptyset$ . In practice, association rules are also associated with a score, e.g., confidence of a rule. In our discussion, and mostly for ease of exposition, we ignore these scores.

We use  $\mathcal{R}$  the set of all association rules extracted from transactions  $T$ . We assume that there are  $\ell$  distinct RHSs appearing in the rules in  $\mathcal{R}$  and that  $\mathcal{R}$  is partitioned into sets  $\{\mathcal{R}_{y_1}, \dots, \mathcal{R}_{y_\ell}\}$ ; all rules in  $\mathcal{R}_{y_i}$  share the same RHS  $Y_i$ . We focus the rest of our discussion on a particular RHS  $Y$  and show how to pick  $S_y \subseteq \mathcal{R}_y$ .

We say that rule  $r : X \rightarrow Y$  *applies* to transaction  $t \in T$  if  $t[X] = 1$  and we denote it by  $\alpha(r, t) = 1$ . Otherwise, if  $t[X] = 0$  we have that  $\alpha(r, t) = 0$ .

The following definitions of true positive and false positive transactions with respect to a single rule are necessary.

Table 1: Example of a transaction database of patients' histories .

$t_1$ :	{acetaminophen, alcohol}
$t_2$ :	{acetaminophen, alcohol, liver damage}
$t_3$ :	{acetaminophen, alcohol, flu shot, liver damage}
$t_4$ :	{acetaminophen}
$t_5$ :	{alcohol}

**DEFINITION 2.1.** A transaction  $t \in T$  is a true positive (TP) with respect to rule  $r : X \rightarrow Y$  if the rule predicts that  $Y$  is going to be observed in  $t$ ,  $\alpha(r, t) = 1$ , and  $t[Y] = 1$ .

**DEFINITION 2.2.** A transaction  $t \in T$  is a false positive (FP) with respect to rule  $r : X \rightarrow Y$  if the rule predicts that  $Y$  is going to be observed in  $t$ ,  $\alpha(r, t) = 1$ , and  $t[Y] = 0$ .

For rule  $r : X \rightarrow Y$  we use  $TP(r)$  to denote the set of tuples in  $T$  to which rule  $r$  applies ( $\alpha(t, r) = 1$ ) and in which  $Y$  is actually observed. That is,

$$TP(r) = \{t | t \in T \wedge t[Y] = 1 \wedge \alpha(r, t) = 1\}.$$

Similarly, the set of tuples in  $T$  that are false positives for rule  $r$ , are defined by the set:

$$FP(r) = \{t | t \in T \wedge t[Y] = 0 \wedge \alpha(r, t) = 1\}.$$

**EXAMPLE 2.1.** Consider the transaction database shown in Table 1 that represent patients' histories.

Rule  $r : \{\text{acetaminophen, alcohol}\} \rightarrow \{\text{liver damage}\}$  predicts that transactions (patients)  $t_1, t_2$ , and  $t_3$  will all have  $\{\text{liver damage}\}$ . Note that in the case of transaction  $t_1$  this is a wrong prediction (FP). For transactions  $t_2$  and  $t_4$  this is the correct prediction (TP).

**2.1 Rule-aggregation functions:** In the case of a single rule  $r : X \rightarrow Y$  and a transaction  $t \in T$ , predicting whether  $Y$  appears in  $t$  can be done according to the following simple rule: "RHS  $Y$  is predicted for transaction  $t$  if  $\alpha(r, t) = 1$ ". The prediction task gets more complicated when there is more than one rule involved in the decision making.

**EXAMPLE 2.2.** Consider again the database shown in Table 1. Further assume two rules  $r_1 : \{\text{acetaminophen}\} \rightarrow \{\text{liver damage}\}$  and  $r_2 : \{\text{acetaminophen, alcohol}\} \rightarrow \{\text{liver damage}\}$ . In this case,  $r_1$  by itself predicts  $\{\text{liver damage}\}$  for patients that correspond to transactions  $t_1, \dots, t_4$ . Rule  $r_2$  when considered alone says that  $\{\text{acetaminophen}\}$  needs to

be combined with  $\{\text{alcohol}\}$  in order to predict  $\{\text{liver damage}\}$  for a transaction. Therefore, it suggests that  $\{\text{liver damage}\}$  will be predicted only for transactions  $t_1, \dots, t_3$ , where both  $\{\text{acetaminophen}\}$  and  $\{\text{alcohol}\}$  are observed. The question is how should the information from rules  $r_1$  and  $r_2$  be combined in order to collectively decide which patients will be predicted to have  $\{\text{liver damage}\}$ .

Denote by  $\mathcal{A}$  the aggregation function that takes all rules in  $S_y$  into account and decides whether to predict  $Y$  for a specific tuple  $t \in T$ . Let the outcome of this aggregation function be  $\mathcal{A}(S_y, t) \in \{0, 1\}$ :  $\mathcal{A}(S_y, t) = 1$  when rules  $S_y$  when aggregated using  $\mathcal{A}$  predict  $Y$  for  $t$ . Otherwise,  $\mathcal{A}(S_y, t) = 0$ .

We now describe two instantiations of  $\mathcal{A}$ , that we call  $\mathcal{A}_1$  and  $\mathcal{A}_{avg}$ . We focus on those because we believe they are simple, natural and intuitive.

**The 1-RULE aggregation function  $\mathcal{A}_1$ :** Given a set of rules  $S_y \subseteq \mathcal{R}_y$  and transaction  $t \in T$ ,  $\mathcal{A}_1(S_y, t) = 1$  if there exists at least one rule  $r \in S_y$  such that  $\alpha(r, t) = 1$ . Otherwise,  $\mathcal{A}_1(S_y, t) = 0$ .

**EXAMPLE 2.3.** Consider again the database in Table 1 and rule set  $S = \{r_1, r_2\}$  where  $r_1$  and  $r_2$  are defined as before to be  $r_1 : \{\text{acetaminophen}\} \rightarrow \{\text{liver damage}\}$  and  $r_2 : \{\text{acetaminophen}, \text{alcohol}\} \rightarrow \{\text{liver damage}\}$ . In this case, for  $1 \leq i \leq 4$   $\mathcal{A}_1(S, t_i) = 1$ . That is, all patients  $t_i$  with  $1 \leq i \leq 4$  are predicted to have  $\{\text{liver damage}\}$ . Note, that only for transactions  $t_2$  and  $t_3$  this decision leads to a true positive. The rest of the transactions are false positives.

The definitions of true positives and false positives of a single rule (Definitions 2.1 and 2.2) can be easily generalized to sets of rules with the same RHS. When aggregation function  $\mathcal{A}_1$  takes into account rules  $S_y$ , then we use  $\text{TP}_1(S_y)$  for the set of tuples  $t \in T$  for which  $t[Y] = 1$  and  $\mathcal{A}_1(S_y, t) = 1$ . That is,

$$\text{TP}_1(S_y) = \{t | t \in T \wedge t[Y] = 1 \wedge \mathcal{A}_1(S_y, t) = 1\}.$$

An alternative representation of the set  $\text{TP}_1(S_y)$  is the following

$$\text{TP}_1(S_y) = \bigcup_{r \in S_y} \text{TP}(r).$$

False positives are handled in a symmetric way. That is, for a set of rules  $S_y$  we use  $\text{FP}_1(S_y)$  to specify the set of tuples  $t \in T$  for which  $t[Y] = 0$  and  $\mathcal{A}_1(S_y, t) = 1$ . That is,

$$\text{FP}_1(S_y) = \{t | t \in T \wedge t[Y] = 0 \wedge \mathcal{A}_1(S_y, t) = 1\}.$$

As before, the set  $\text{FP}_1(S_y)$  can be alternatively represented by

$$\text{FP}_1(S_y) = \bigcup_{r \in S_y} \text{FP}(r).$$

So far we have limited our discussion in rulesets  $S_y$  in which all rules have the same RHS  $Y$ . For an arbitrary ruleset  $S$  where  $\ell$  distinct RHSs appear we have that

$$\text{TP}_1(S) = \biguplus_{i=1}^{\ell} \text{TP}_1(S_{y_i}),$$

and

$$\text{FP}_1(S) = \biguplus_{i=1}^{\ell} \text{FP}_1(S_{y_i}).$$

$S_{y_i}$  refers to all the rules in  $S$  that share the same  $i$ -th RHS;  $\biguplus_{i=1}^{\ell} \text{TP}_1(S_{y_i})$  (or  $\biguplus_{i=1}^{\ell} \text{FP}_1(S_{y_i})$ ) denotes the union of transactions in  $\text{TP}_1(S_{y_i})$  (or  $\text{FP}_1(S_{y_i})$ ), when duplicates are not ignored.

**The AVERAGECHOICE aggregation function  $\mathcal{A}_{avg}$ :** Given a set of rules  $S_y \subseteq \mathcal{R}_y$  and transaction  $t \in T$ , let

$$(S_y, t)^+ = \{r | r \in S_y \wedge \alpha(r, t) = 1\}$$

be the set of rules in  $S_y$  that apply to transaction  $t$ , and

$$(S_y, t)^- = \{r | r \in S_y \wedge \alpha(r, t) = 0\}$$

the set of rules in  $S_y$  that do not apply to transaction  $t$ . Obviously,  $(S_y, t)^+ \cup (S_y, t)^- = S_y$  and  $(S_y, t)^+ \cap (S_y, t)^- = \emptyset$ . For a given set of rules  $S_y$  transaction  $t \in T$  has  $\mathcal{A}_{avg}(S_y, t) = 1$  with probability

$$p_{avg}(S_y, t) = \frac{|(S_y, t)^+|}{|S_y|},$$

and not to have  $Y$  with probability  $1 - p_{avg}(S_y, t)$ .

**EXAMPLE 2.4.** Consider again the database in Table 1 and rule set  $S = \{r_1, r_2\}$  where  $r_1$  and  $r_2$  are defined as in Example 2.3. The probability of predicting  $\{\text{liver damage}\}$  for transactions  $t_i$  with  $1 \leq i \leq 3$  is:  $p_{avg}(S, t_i) = 1$ . For transaction  $t_4$ ,  $p_{avg}(S, t_4) = 1/2$ , and for transaction  $t_5$ ,  $p_{avg}(S, t_5) = 0$ .

A transaction  $t \in T$  for which  $t[Y] = 1$  has  $\mathcal{A}_{avg}(S_y, t) = 1$  with probability

$$(2.1) \quad p_{avg}^{TP}(S_y, t) = \frac{\sum_{r \in S_y} (\mathbf{I}_{\alpha(r,t)=1} \cdot \mathbf{I}_{t[Y]=1})}{|S_y|},$$

where  $\mathbf{I}_{condition}$  is an indicator variable that takes value 1 if “condition” is true and value “0” otherwise. Similarly, a transaction  $t \in T$  for which  $t[Y] = 0$  is predicted as 0 by  $\mathcal{A}_{avg}$  with probability

$$(2.2) \quad p_{avg}^{FP}(S_y, t) = \frac{\sum_{r \in S_y} (\mathbf{I}_{\alpha(r,t)=1} \cdot \mathbf{I}_{t[Y]=0})}{|S_y|}.$$

Although for  $\mathcal{A}_1$  the sets  $TP_1(S_y)$  and  $FP_1(S_y)$  were easily defined, the definitions of the corresponding tuple sets for  $\mathcal{A}_{avg}$  are a bit more tricky. This is because  $\mathcal{A}_{avg}$  makes probabilistic, rather than deterministic, decisions. As a result, we cannot define sets  $TP_{avg}$  and  $FP_{avg}$  deterministically. However, we can compute the expected cardinality of these sets as follows.

$$(2.3) \quad |TP_{avg}(S_y)| = \sum_{t \in T} p_{avg}^{TP}(S_y, t),$$

and

$$(2.4) \quad |FP_{avg}(S_y)| = \sum_{t \in T} p_{avg}^{FP}(S_y, t),$$

where the probabilities  $p_{avg}^{TP}(S_y, t)$  and  $p_{avg}^{FP}(S_y, t)$  are computed as in Equations (2.1) and (2.2) respectively.

As before, the generalization of Equations (2.3) and (2.4) for arbitrary collections of rules  $S$  that have  $\ell$  distinct RHSs is simply

$$|TP_{avg}(S)| = \sum_{i=1}^{\ell} |TP_{avg}(S_{y_i})|,$$

and

$$|FP_{avg}(S)| = \sum_{i=1}^{\ell} |FP_{avg}(S_{y_i})|.$$

### 3 Problem definitions

We measure the goodness of a set of rules as a function of the correct predictions they make; the more correct prediction the better the set of rules. In this section we translate this abstract objective into a concrete optimization function.

Given a dataset  $T$  and a collection of rules  $R_y$ , with RHS  $Y$ , and aggregation function  $\mathcal{A}$ , our goal is to pick set  $S_y \subseteq \mathcal{R}_y$  such that the number of true positives predictions on tuples in  $T$  is maximized and the number of false positive predictions is minimized. Maximization of true positives and minimization of false positives are two different objective functions. Therefore, we could formalize this requirement in a bi-objective optimization problem. In that case though, the optimal solution would be hard to define; there could be solutions that are good for one objective but not so good for the other, or vice versa. In order to deal with this, we would have to settle with finding all *pareto-optimal* solutions, rather than a single solution. For a discussion on bi-objective optimization problems and pareto-optimal solutions see [8]. Reporting all pareto-optimal solutions has its advantages and disadvantages; the main disadvantage being that there can be exponential number of them.

Two alternatives to bi-objective optimization exist:

- (a) To incorporate both the objectives into a single objective function e.g., by taking their (weighted) sum.
  - (b) To bound the one objective and optimize the other.
- In this paper we pick alternative (b) because we believe that gives a more structured output.

We define two RULE SELECTION problems one for aggregation function  $\mathcal{A}_1$  and the other for aggregation function  $\mathcal{A}_{avg}$ .

**PROBLEM 3.1.** [ $\mathcal{A}_1$ -(RULE SELECTION)] *Consider transactions  $T$ , a collection of rules  $\mathcal{R}_y$  sharing the same right-hand side  $Y$ , and an integer  $\mathbf{B}$ . Find the set of rules  $S_y \subseteq \mathcal{R}_y$  such that  $|TP_1(S_y)|$  is maximized and  $|FP_1(S_y)| \leq \mathbf{B}$ .*

We have the following result.

**PROPOSITION 3.1.** *The  $\mathcal{A}_1$ -RULE SELECTION problem is NP-hard.*

*Proof.* We reduce the RED-BLUE SET COVER problem to the  $\mathcal{A}_1$ -RULE SELECTION problem. In the RED-BLUE SET COVER problem we are given disjoint sets  $R$  and  $B$  of red and blue elements respectively, and a collection  $S = \{S_1, \dots, S_n\} \subseteq 2^{R \cup B}$ . The goal is to find a collection  $C \subseteq S$  that covers all blue elements, i.e.,  $B \subseteq \cup C$ , while minimizing the number of covered red elements.

Given an instance of the RED-BLUE SET COVER, we create an instance of the  $\mathcal{A}_1$ -RULE SELECTION as follows: For each element in  $B$  and each element in  $R$  we create a transaction. Let the transactions corresponding to elements in  $B$  be  $\{t_1, \dots, t_{|B|}\}$  and the transactions corresponding to elements in  $R$  be

$\{t_{|B|+1}, \dots, t_{|B|+|R|}\}$ . Then for every transaction  $t_{i_1}$  with  $1 \leq i_1 \leq |B|$  we have that  $t_{i_1}[Y] = 1$  and for every transaction  $t_{i_2}$  with  $|B| + 1 \leq i_2 \leq |B| + |R|$  we have that  $t_{i_2}[Y] = 0$ . For every set  $S_j \in S$  we create a rule  $r_j \in \mathcal{R}_y$ . For a rule  $r_j \in \mathcal{R}_y$  and transaction  $t_{i_1}$  with  $1 \leq i_1 \leq |B|$  we set  $\alpha(t_{i_1}, r_j) = 1$  if  $S_j$  contains the blue element that corresponds to transaction  $t_{i_1}$ . In this case  $t_{i_1}$  is a true positive for rule  $r_j$ . For a rule  $r_j \in \mathcal{R}_y$  and transaction  $t_{i_2}$  with  $|B| + 1 \leq i_2 \leq |B| + |R|$  we set  $\alpha(t_{i_2}, r_j) = 1$  if  $S_j$  contains the red element that corresponds to transaction  $t_{i_2}$ . That is, transaction  $t_{i_2}$  is a false positive for rule  $r_j$ . Otherwise,  $\alpha(t_{i_2}, r_j) = 0$  and transaction  $t_{i_2}$  is correctly decided not to have  $Y$ .

It is easy to see that every solution  $S_y \subseteq \mathcal{R}_y$  to the  $\mathcal{A}_1$ -RULE SELECTION problem with benefit  $|\text{TP}_1(S_y)|$  and cost  $|\text{FP}_1(S_y)| \leq \mathbf{B}$  translates a solution  $C$  to the RED-BLUE SET COVER problem such that if  $r_i \in S_y$  then  $S_i \in C$  that covers  $|\text{TP}_1(S_y)|$  blue elements and  $|\text{FP}_1(S_y)|$  red elements.

The RED-BLUE SET COVER problem was first presented in [6] where it was also proven that (a) unless  $\text{NP} \subseteq \text{DTIME}(n^{\text{poly} \log(n)})$ , there exist no polynomial-time approximation algorithms to approximate the RED-BLUE SET COVER to within a factor of  $2^{(4 \log n)^{1-\epsilon}}$  for any  $\epsilon > 0$ , and (b) there are no polynomial-time approximation algorithms to approximate RED-BLUE SET COVER to within  $2^{\log^{1-(\log \log \beta)^{-c}} \beta}$  for any constant  $c < 1/2$  unless  $\text{P} = \text{NP}$ . The best upper bound for the RED-BLUE SET COVER is due to [14], who has recently presented a randomized  $2\sqrt{n \log \beta}$ -approximation algorithm for it. In both cases above  $\beta$  corresponds to the cardinality of the set of blue elements  $\beta = |B|$ .

The  $\mathcal{A}_1$ -RULE SELECTION problem is not parameter free; parameter  $\mathbf{B}$  needs to be given as part of the input. An intuitive way of setting this parameter is by observing that the quantity  $|\text{FP}_1(S_y)|$  is in fact the number of *false positives per rule*. When  $S_y = \mathcal{R}_y$  we get the number of false positives per rule achieved by the input set of rules  $\mathcal{R}_y$ . We can thus set  $\mathbf{B}$  to be a factor  $\lambda \in \mathbf{R}^+$  of this quantity. That is,

$$(3.5) \quad \mathbf{B} = \lambda |\text{FP}_1(\mathcal{R}_y)|.$$

For example,  $\lambda = 1$  requires that the average false positives per rule in the output is the same as in the original ruleset.

**PROBLEM 3.2.** [ $\mathcal{A}_{\text{avg}}$ -(RULE SELECTION)] *Given a collection of rules  $\mathcal{R}_y$  sharing the same right-hand side  $Y$  and rational number  $\mathbf{B}$ , find a set of rules  $S_y \subseteq \mathcal{R}_y$  such that  $|\text{TP}_{\text{avg}}(S_y)|$  is maximized and  $|\text{FP}_{\text{avg}}(S_y)| \leq \mathbf{B}$ .*

As before, the above problem definition is not parameter free, and depends on the setting of  $\mathbf{B}$ . This

will be handled in the same way as for the  $\mathcal{A}_1$ -RULE SELECTION problem.

In order to be symmetric to the  $\mathcal{A}_1$ -(RULE SELECTION) we note here that the  $\mathcal{A}_{\text{avg}}$ -(RULE SELECTION) can be solved optimally in polynomial time. Although we describe the algorithm for solving it in the next section we summarize this fact in the following proposition.

**PROPOSITION 3.2.** *The  $\mathcal{A}_{\text{avg}}$ -(RULE SELECTION) problem can be solved optimally in polynomial time.*

Problems 3.1 and 3.2 have the same underlying goal. Their only difference is in the rule aggregation function used to predict the existence of a specific RHS in transactions in  $T$ . We refer to the generic RULE SELECTION problem, where either of the two aggregation functions are used, as the  $\mathcal{A}$ -RULE SELECTION problem.

#### 4 Algorithms for RULE SELECTION problems

In this section we present heuristic and optimal algorithms for the  $\mathcal{A}_1$  and  $\mathcal{A}_{\text{avg}}$ -RULE SELECTION problems. Note that these algorithms work for sets of rules that share a specific RHS. That is, given a rule collection  $\mathcal{R}_y$ , where all rules in  $\mathcal{R}_y$  share the same RHS  $Y$ , they pick  $S_y \subseteq \mathcal{R}_y$  to optimize the objective functions of Problems 3.1 or 3.2 respectively.

---

**Algorithm 1** The global rule-selection routine GRSR.

---

**Input:** A collection of rules  $\mathcal{R}$  with  $\ell$  distinct RHSs.

**Output:** A subset of rules  $S \subseteq \mathcal{R}$   
 $S = \emptyset$

- 1: Partition  $\mathcal{R} = \{\mathcal{R}_{y_1}, \mathcal{R}_{y_2}, \dots, \mathcal{R}_{y_\ell}\}$
  - 2: All rules in  $\mathcal{R}_{y_i}$  share RHS  $Y_i$ .
  - 3: **for**  $i = 1$  to  $\ell$  **do**
  - 4:      $S_i = \text{solve } \mathcal{A}\text{-RULE SELECTION}$
  - 5:      $S = S \cup S_i$
  - 6: **return**  $S$
- 

In practice though, the input rule collections  $\mathcal{R}$  may contain rules with many different RHSs. Our approach for dealing with such rule collections is shown in Algorithm 1; the set of rules in  $\mathcal{R}$  is partitioned into groups  $\{\mathcal{R}_{y_1}, \mathcal{R}_{y_2}, \dots, \mathcal{R}_{y_\ell}\}$ . All rules in  $\mathcal{R}_{y_i}$  share the same RHS  $Y_i$ . Then the algorithms for the  $\mathcal{A}$ -RULE SELECTION problem operate on each ruleset  $\mathcal{R}_{y_i}$  separately. The union of their outputs is the output of the global rule-selection routine.

#### 4.1 Algorithms $\mathcal{A}_1$ -(RULE SELECTION) problem

In Section 3, Proposition 3.1, we showed that the  $\mathcal{A}_1$ -RULE SELECTION problem is NP-hard, with little hope for good polynomial-time algorithms with bounded approximation factors. Here, we give a simple, intu-

itive and efficient greedy-based heuristic algorithm that works well in practice.

---

**Algorithm 2** The Greedy\_1 algorithm.

---

**Input:** A set of rules  $\mathcal{R}_y$  with right-hand side  $Y$  and transaction database  $T$  and integer  $\mathbf{B}$ .  
**Output:** A subset of rules  $S_y \subseteq \mathcal{R}_y$

- 1:  $C = \mathcal{R}_y, S_y = \emptyset$
- 2: **while**  $|\text{FP}_1(S_y)| \leq \mathbf{B}$  **do**
- 3:   pick  $r \in C$  such that
- 4:    $r = \arg \max_{r' \in C} \text{Marginal\_Gain}(S_y, r')$
- 5:   **if**  $\text{TP}_1(S_y \cup \{r\}) > \text{TP}_1(S_y)$  **then**
- 6:      $C = C \setminus \{r\}$
- 7:      $S_y = S_y \cup \{r\}$
- 8:   **else**
- 9:     return  $S_y$
- 10: return  $S_y$

---

The pseudocode for the Greedy\_1 algorithm is given in Algorithm 2. It first starts with all rules in the  $\mathcal{R}_y$  as candidates for being selected, i.e.,  $C = \mathcal{R}_y$ . Also, the output set of selected rules is originally set to the empty set,  $S_y = \emptyset$ . As long as the total cost of the selected rules does not exceed the pre-specified threshold,  $|\text{FP}_1(S_y)|\mathbf{B}$ , the rule  $r$  with the maximal marginal gain is picked to be added to set  $S_y$ . If no such rule exists, the algorithm terminates and returns the rules that have been picked in set  $S_y$ .

The selection of the rule  $r$  to be added to set  $S_y$  is done in the Marginal\_Gain function that is simply defined to be

$$(4.6) \quad \begin{aligned} \text{Marginal\_Gain}(S_y, r) &= \\ &= \frac{|\text{TP}_1(S_y \cup \{r\}) \setminus \text{TP}_1(S_y)|}{|\text{FP}_1(S_y \cup \{r\}) \setminus \text{FP}_1(S_y)|}. \end{aligned}$$

**Running time:** For a database with  $n$  transactions and ruleset  $\mathcal{R}_y$  with  $m$  rules, the running time of the Greedy\_1 algorithm is  $\mathcal{O}(Imn)$ :  $I$  refers to the number of iterations of the **while** loop. In the worst case, the while loop repeats as many times as the number of rules in  $\mathcal{R}_y$ . However, in practice much less iterations are needed. In each iteration the marginal gain of every remaining rule in set  $C$  needs to be updated. In the worst case, every iteration requires time  $\mathcal{O}(mn)$ .

#### 4.2 Algorithms $\mathcal{A}_{avg}$ -(RULE SELECTION) problem

In Section 3 in Proposition 3.2 we stated, without a proof, that the  $\mathcal{A}_{avg}$ -RULE SELECTION problem can be solved optimally in polynomial time. We start this section by describing the optimal polynomial-time algorithm for solving it.

The key observation for our solution is that if the quantities  $|\text{TP}_{avg}(S_y)|$  and  $|\text{FP}_{avg}(S_y)|$  given in Equations (2.3) and (2.4) respectively, were lacking their denominators,  $|S_y|$ , then, Problem 3.2 would have been identical to the KNAPSACK problem [19].

Our polynomial-time algorithm for  $\mathcal{A}_{avg}$ -(RULE SELECTION) is based on exactly this observation. The pseudocode of the algorithm that we call DP\_Avg is given in Algorithm 3. The core idea of the DP\_Avg algorithm is that it fixes the cardinality of the ruleset to be selected from the whole collection of rules  $\mathcal{R}_y$  and it finds the optimal solution for each such cardinality by calling the DP routine. The DP algorithm is a dynamic-programming recursion that we shall describe shortly. Among all solutions with different cardinalities, the one that maximizes the objective function is picked.

---

**Algorithm 3** The DP\_Avg algorithm.

---

**Input:** A set of rules  $\mathcal{R}_y$  with right-hand side  $Y$  and transaction database  $T$  and rational number  $\mathbf{B}$ .  
**Output:** A subset of rules  $S_y \subseteq \mathcal{R}_y, S_y = \emptyset$

- 1: **for**  $k = 1, \dots, |\mathcal{R}_y|$  **do**
- 2:    $S'_y = \text{DP}(\mathcal{R}_y, [k \times \mathbf{B}], k)$
- 3:   **if**  $|\text{TP}_{avg}(S'_y)| > |\text{TP}_{avg}(S_y)|$  and  $|\text{FP}_{avg}(S'_y)| < \mathbf{B}$  **then**
- 4:      $S_y = S'_y$
- 5: return  $S_y$

---

The most important line of the DP\_Avg algorithm is the call of the DP routine. The goal of this routine is to find  $S_y \subseteq \mathcal{R}_y$  of cardinality  $k$  such that quantity

$$\begin{aligned} \overline{\text{TP}}_{\#}(S_y) &= |S_y| \cdot |\text{TP}_{avg}(S_y)| \\ &= \sum_{r \in S_y} \sum_{t \in T} (\mathbf{I}_{\alpha(r,t)=1} \cdot \mathbf{I}_{t[Y]=1}), \end{aligned}$$

is maximized and quantity

$$\begin{aligned} \overline{\text{FP}}_{\#}(S_y) &= |S_y| \cdot |\text{FP}_{avg}(S_y)| \\ &= \sum_{r \in S_y} \sum_{t \in T} (\mathbf{I}_{\alpha(r,t)=1} \cdot \mathbf{I}_{t[Y]=0}), \end{aligned}$$

is below fixed bound  $\mathbf{B}_k = [k \times \mathbf{B}]$ . Maximizing  $\overline{\text{TP}}_{\#}(S_y)$  under the constraint that  $\overline{\text{FP}}_{\#}(S_y) \leq \mathbf{B}_k$  bears similarity with the classical KNAPSACK problem. There are only two differences; first, the classical KNAPSACK problem does not impose any constraint on the cardinality of the solution, while in our case we need to output set  $S_y \subseteq \mathcal{R}$  of fixed cardinality  $k$ . Second, in our instance the bound  $\mathbf{B}_k$  on the cost of the solution  $\overline{\text{FP}}_{\#}(S_y)$  is bounded by the size of the input, that is,

$\mathbf{B}_k = \mathcal{O}(mn)$ . The latter observation, makes the running time of the DP algorithm polynomial to the size of the input.

We now give the dynamic-programming recursion that is the core of the DP routine. For expressing it, we consider an arbitrary ordering of the rules in ruleset  $\mathcal{R}_y$ , that is,  $R_y = \{r_1, \dots, r_m\}$ . Denote now by  $\overline{\text{TP}}_{\#}[i, c, k]$  the maximum value of  $\overline{\text{TP}}_{\#}$  that can be achieved using a subset of size  $k$  out of the first  $i$  rules  $(r_1, \dots, r_i)$  of the original ruleset  $\mathcal{R}_y$ , when the cost  $\overline{\text{FP}}_{\#}$  is at most  $b \leq \mathbf{B}_k$ . Then the following recursion allows us to compute all the values  $\overline{\text{TP}}_{\#}[i, b, k]$

$$(4.7) \overline{\text{TP}}_{\#}[i, b, k] = \max \left\{ \overline{\text{TP}}_{\#}[i-1, b, k], \overline{\text{TP}}_{\#}[i-1, b - |\text{FP}(r_i)|, k-1] + |\text{TP}(r_i)| \right\}.$$

**Running time:** For a database with  $n$  transactions and ruleset  $\mathcal{R}_y$  with  $m$  rules, the running time of the DP routine is  $\mathcal{O}(\mathbf{B}_k |\mathcal{R}_y| k)$ ; recall that  $\mathbf{B}_k = \mathcal{O}(nm)$ ,  $|\mathcal{R}_y| = \mathcal{O}(m)$  and  $k = \mathcal{O}(m)$  we have that the worst-case running time of the DP routine is  $\mathcal{O}(m^3 n)$ . The DP\_Avg algorithm makes  $m$  calls to the DP routine, and therefore a naive implementation would require time  $\mathcal{O}(m^4 n)$ . However, observe that the entries of the  $\overline{\text{TP}}_{\#}$  table that have been computed for  $k = k_1$  need not be recomputed for  $k = k_2$ , with  $k_2 > k_1$ . Although this observation considerably improves the running time of the DP\_Avg algorithm in practice, it leaves its worst-case complexity intact.

Using standard arguments, like, for example, the ones used in [19] Chapter 8. It is easy to show that the DP\_Avg algorithm gives the optimal solution to the  $\mathcal{A}_{avg}$ -(RULE SELECTION) problem.

Though provably optimal, the DP\_Avg algorithm is computationally too expensive for reasonably large datasets. For that reason we give an alternative *greedy* version of the DP\_Avg algorithm, which we call the **Greedy\_Avg** algorithm. This algorithm is conceptually identical to the DP\_Avg algorithm shown in Algorithm 3 except that instead of invoking the DP routine for every  $k$  we invoke a greedy-selection algorithm. The pseudocode in Algorithm 4 describes the **Greedy\_Avg** algorithm. The algorithm initially computes the gain  $\mathbf{Gain}(r)$ , for every rule  $r$  in the input rule collection  $\mathcal{R}_y$ . The  $\mathbf{Gain}(r)$  of a rule  $r$  is simply defined as the ratio of the expected number of TP to the expected number of FP introduced by rule  $r$  alone. Note that for a single rule the expected number of true positives (false positives) coincides with the cardinality of sets  $\text{TP}(r)$  ( $\text{FP}(r)$ ). That is,

---

**Algorithm 4** The **Greedy\_Avg** algorithm.

---

**Input:** A set of rules  $\mathcal{R}_y$  with right-hand side  $Y$  and transaction database  $T$  and rational  $\mathbf{B}$ .

- 1: **for**  $r \in \mathcal{R}_y$  **do**
- 2:     Compute  $\mathbf{Gain}(r)$
- 3: Sort elements in  $\mathcal{R}_y$  in decreasing order of their  $\mathbf{Gain}$ 's  $\mathbf{Gain}(r_1) \geq \dots \geq \mathbf{Gain}(r_m)$
- 4:  $\text{maxBenefit} = 0$ ,  $\text{maxIndex} = -1$ ,  $\text{benefit} = 0$ ,  $\text{weight} = 0$
- 5: **for**  $i = 1 \dots m$  **do**
- 6:      $\text{benefit} = \text{benefit} + \overline{\text{TP}}_{\#}(r_i)$
- 7:      $\text{weight} = \text{weight} + \overline{\text{FP}}_{\#}(r_i)$
- 8:     **if**  $\frac{\text{benefit}}{i} > \text{maxBenefit}$  and  $\frac{\text{weight}}{i} \leq \mathbf{B}$  **then**
- 9:          $\text{maxBenefit} = \frac{\text{benefit}}{i}$
- 10:          $\text{maxIndex} = i$
- 11:  $S_y = \bigcup_{i=1, \dots, \text{maxIndex}} r_i$
- 12: **return**  $S_y$

---

$$\mathbf{Gain}(r) = \frac{|\text{TP}_{avg}(r)|}{|\text{FP}_{avg}(r)|} = \frac{|\text{TP}(r)|}{|\text{FP}(r)|}.$$

The rules are then sorted in decreasing order of their gain values. The **Greedy\_Avg** algorithm then goes through the sorted rules and each rule  $r_i$ , that is ranked  $i$ , is added to the solution ruleset  $S_y$  as long as its addition improves the objective function (benefit) and keeps the average weight below the input threshold  $\mathbf{B}$ .

**Running time:** The running time of the **Greedy\_Avg** algorithm is dominated by the sorting of the rules in  $\mathcal{R}_y$  with respect to their  $\mathbf{Gain}$  values. Therefore, if computing the  $\mathbf{Gain}$  of every rule takes time  $\mathcal{O}(T_G)$ , then the running time of the **Greedy\_Avg** algorithm is  $\mathcal{O}(|\mathcal{R}_y| \log |\mathcal{R}_y|) \mathcal{O}(m \log m + m T_G)$ . In our case,  $T_G = \mathcal{O}(n)$ , and thus **Greedy\_Avg** needs time  $\mathcal{O}(m \log m + mn)$ .

## 5 Experiments

The goal of the experimental evaluation is to illustrate that our algorithms pick a small number of representative association rules  $S$  from a large input rule collection  $\mathcal{R}$ . At the same time, we show that  $|\text{TP}_x(S)| \geq |\text{TP}_x(\mathcal{R})|$  and  $|\text{FP}_x(S)| \leq |\text{FP}_x(\mathcal{R})|$ , where  $x \in \{1, avg\}$ .

**Datasets:** We experiment with seven different datasets described in Table 2. All these datasets are available from the UCI Machine Learning Repository ([3])

For the majority of our experiments we generate the input rule collection  $\mathcal{R}$  using the standard **Apriori** algorithm. Recall that **Apriori** has two parameters, the *minimum support* (*min\_sup*) and the *minimum*

Dataset	# of items	# of transactions
LENSES	12	24
HAYESROTH	18	132
BALANCE	23	625
CAR	25	1728
LYMPH	63	148
CONGRESS	50	435
MUSHROOM	119	8124

Table 2: Number of items and tuples of the seven datasets used in our experimental evaluation.

*confidence (min\_conf)*. For a rule  $r : X \rightarrow Y$  the support of the rule  $sup(r)$  is the number of transactions  $t \in T$  for which  $t[X \cup Y] = 1$ . The confidence of the rule  $conf(r)$  is the ratio of the transactions for which  $t[X \cup Y] = 1$  divided by the transactions for which  $t[X] = 1$ .

Table 3 shows the number of rules extracted from the different datasets using **Apriori** algorithm. For the smaller datasets (LENSES, HAYESROTH, BALANCE and CAR) we set both the support and confidence thresholds to zero. For the larger datasets (LYMPH, CONGRESS and MUSHROOM) we set the minimum support threshold to the smallest integer that is greater or equal to  $0.2 \times |T|$ , where  $|T|$  is the number of transactions in the dataset. For these datasets the confidence threshold is set to 0.2. The selection threshold is somewhat arbitrary; the trends of our results do not change for different threshold values. A value of 0.2 is reasonable since it does not cause the **Apriori** algorithm to explore the whole itemset lattice.

Dataset	<i>min_sup</i>	<i>min_conf</i>	# of rules in $\mathcal{R}$
LENSES	1	0	2682
HAYESROTH	1	0	8858
BALANCE	1	0	46708
CAR	1	0	862906
LYMPH	30	.2	191460
CONGRESS	87	.2	1110512
MUSHROOM	1625	.2	19191656

Table 3: Minimum support and confidence thresholds used for **Apriori** algorithm; number of rules in the mined collection  $\mathcal{R}$

**Performance measures:** We measure the qualitative performance of the different algorithms using three evaluation measures: the *R-ratio* ( $rR$ ), the *TP-ratio* ( $rTP$ ), and the *FP-ratio* ( $rFP$ ).

Given rule-selection algorithm SA and a collection of mined rules  $\mathcal{R}$  used as input to SA, we define the

R-ratio to be

$$rR(SA, \mathcal{R}) = \frac{|SA(\mathcal{R})|}{|\mathcal{R}|}.$$

SA can be any rule-selection algorithm, like for example **Greedy\_1**, **DP\_Avg**, or **Greedy\_Avg**. We use  $SA(\mathcal{R})$  to denote the set of rules picked by algorithm SA on input  $\mathcal{R}$ . The R-ratio takes values in  $[0, 1]$ . The smaller the value of R-ratio, the smaller the set of representatives picked by the SA algorithm and thus the better the algorithm in reducing the size of the original ruleset.

For  $\mathcal{A}_1$  or  $\mathcal{A}_{avg}$  aggregation functions, the TP-ratio for algorithm SA and input rule-collection  $\mathcal{R}$  is defined to be

$$rTP_1(SA, \mathcal{R}) = \frac{|TP_1(SA(\mathcal{R}))|}{|TP_1(\mathcal{R})|},$$

and

$$rTP_{avg}(SA, \mathcal{R}) = \frac{|TP_{avg}(SA(\mathcal{R}))|}{|TP_{avg}(\mathcal{R})|}$$

respectively. The TP-ratio takes values in  $[0, \infty)$ . The larger the value of the TP-ratio, the better the performance of the algorithm. The definitions of  $rFP_1$  and  $rFP_{avg}$  are analogous to those of  $rTP_1$  and  $rTP_{avg}$ ; the only difference is that FP is used instead of TP. We omit the actual definitions due to space constraints. Note however, that the FP-ratios take values in  $[0, \infty)$ . The smaller the value of the FP-ratio, the better the performance of an SA algorithm.

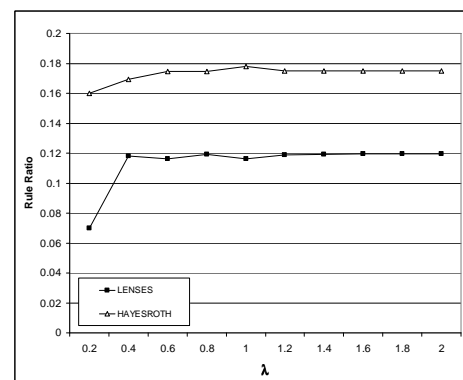


Figure 1: Reduction of the number of rules achieved by **Greedy\_1** for the HAYESROTH and LENSES datasets; y-axis: Rules ratio  $rR$ ; x-axis: varying values of  $\lambda$ .

**Experiment I: Impact of threshold  $\lambda$  on the performance of **Greedy\_1** algorithm:** Figure 1 shows the

rules ratio for the `Greedy_1` algorithm as a function of the value of  $\lambda$  (evaluated by Equation (3.5)). At  $\lambda=.2$ , the false positive constraint is extremely limiting, allowing few rules to be selected. However, for  $\lambda \geq .4$  `Greedy_1` is able to select more rules. It is interesting to note that the rules ratios for  $\lambda \in [0.4, 2]$  are approximately the same for both datasets.

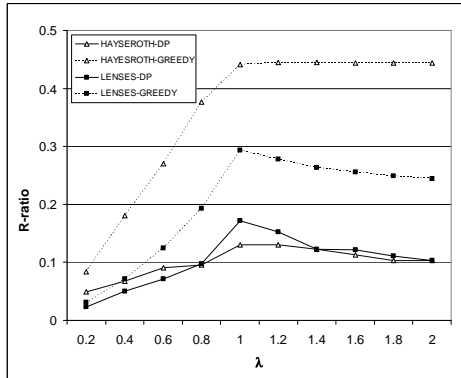


Figure 2: Reduction of the number of rules achieved by  $\mathcal{A}_{avg}$ -RULE SELECTION algorithms `DP_Avg` and `Greedy_Avg` for the HAYESROTH and LENSES datasets; y-axis: Rules ratio  $rR$ ; x-axis: varying values of  $\lambda$ .

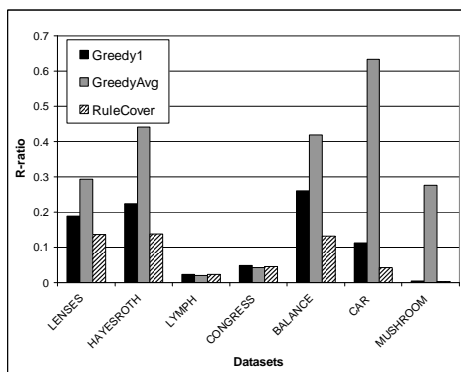


Figure 3: Values of  $rR(SA, \mathcal{R})$  for  $SA \in \{\text{Greedy}_1, \text{Greedy\_Avg}, \text{RuleCover}\}$  and ruleset  $\mathcal{R}$  produced by Apriori.

**Experiment II: Comparison of DP\_Avg and Greedy\_Avg:** The goal of this experiment is to explore the difference in the results of `DP_Avg` and `Greedy_Avg` algorithms. Recall that both algorithms solve the  $\mathcal{A}_{avg}$ -RULE SELECTION problem; `DP_Avg` solves it optimally, while `Greedy_Avg` is a suboptimal, but much more efficient, algorithm for the same problem.

Figure 2 shows the rules ratio for the `DP_Avg` and the `Greedy_Avg` algorithms as a function of the value

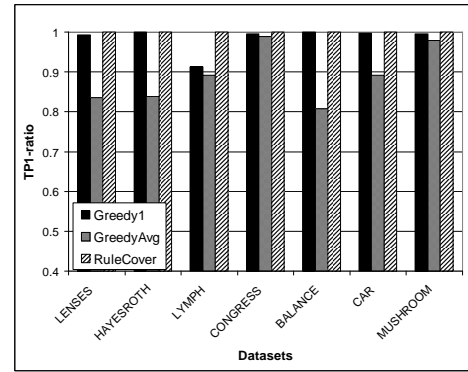


Figure 4: Values of  $rTP_1(SA, \mathcal{R})$  for  $SA \in \{\text{Greedy}_1, \text{Greedy\_Avg}, \text{RuleCover}\}$  and ruleset  $\mathcal{R}$  produced by Apriori.

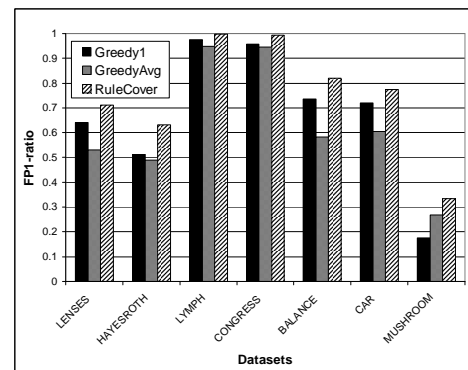


Figure 5: Values of  $rFP_1(SA, \mathcal{R})$  for  $SA \in \{\text{Greedy}_1, \text{Greedy\_Avg}, \text{RuleCover}\}$  and ruleset  $\mathcal{R}$  produced by Apriori.

of  $\lambda$  (evaluated by Equation 3.5). As  $\lambda$  increases from 0.2 to 1, the number of rules for both algorithms and in both datasets increases. As in Experiment I, The false positive constraint is initially limiting to the rules that can be selected. Relaxing that constraint allows the algorithms to maximize  $|TP_{avg}|$  by selecting sets of rules that have a higher ratio of false positives per rule.

For  $\lambda > 1$  the rule-selection is allowing more false positives per rule than the original set of rules. In this case, the two algorithms become more risky by selecting rules with considerable number of true positives but very large number of false positives. Such selections lead to a slight decrease in the number of representative rules picked by `DP_Avg` and `Greedy_Avg` as shown in Figure 2.

It should be noted that, overall, the reduction of rules for `Greedy_Avg` is not nearly as great as for `DP`. However, we will restrict ourselves to the usage of `Greedy_Avg` algorithm for the rest of the experiments since the memory requirements of the `DP_Avg` algorithm

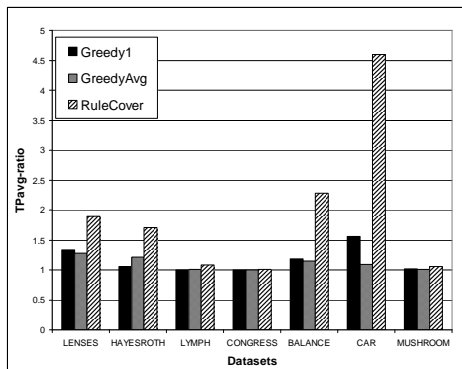


Figure 6: Values of  $rTP_{avg}(SA, \mathcal{R})$  for  $SA \in \{\text{Greedy}_1, \text{Greedy\_Avg}, \text{RuleCover}\}$  and ruleset  $\mathcal{R}$  produced by Apriori.

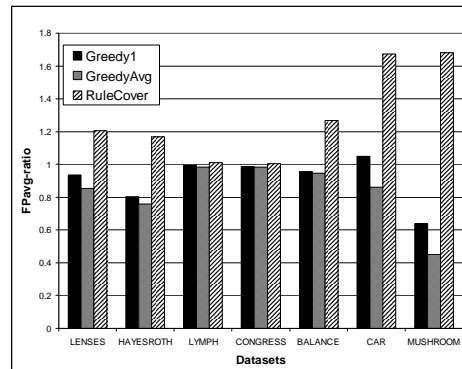


Figure 7: Values of  $rFP_{avg}(SA, \mathcal{R})$  for  $SA \in \{\text{Greedy}_1, \text{Greedy\_Avg}, \text{RuleCover}\}$  and ruleset  $\mathcal{R}$  produced by Apriori.

make it impractical for large datasets.

**Experiment III: Evaluation of rule-selection algorithms across different datasets:** In experiment III we perform a more thorough comparison of different rule-selection algorithms for all datasets. Performance is evaluated using the R, TP and FP ratios. In our comparisons we include three rule-selection algorithms: **Greedy<sub>1</sub>**, **Greedy<sub>Avg</sub>** and **RuleCover**. The first two algorithms have been described in Algorithms 2 and 4. The **RuleCover** algorithm is the one presented in [18]. We include it in our evaluation because it is very close in spirit to our algorithms. For each distinct RHS  $Y$ , the **RuleCover** algorithm greedily forms the set  $S_y$ . In each greedy step, the current set  $S_y$  is extended by rule  $r$  that maximizes the difference  $|\text{TP}(S_y \cup \{r\})| - |\text{TP}(S_y)|$ . Note that **RuleCover** is very similar to the **Greedy<sub>1</sub>** algorithm. Their only difference is that the number of transactions in  $FP_1$  is not taken into account by **RuleCover**.

For **Greedy<sub>1</sub>** and **Greedy<sub>Avg</sub>**, we use set  $\lambda = 1$ . That is we require that the algorithms produce no more false positive predictions per rule than the original ruleset. This seems to be a reasonable and intuitive requirement for a benchmark bound.

Figure 3 shows the  $rR(SA, \mathcal{R})$  for  $SA \in \{\text{Greedy}_1, \text{Greedy\_Avg}, \text{RuleCover}\}$ , and for different datasets. The original rulesets  $\mathcal{R}$  have been mined using the **Apriori** algorithm and the configurations we gave in Table 3. In all datasets and for all SA algorithms, the size of the set of representative rules is much smaller than the size of the input rule collection. In the worst case, the R-ratio is at most 0.63, while in most cases it is much lower, reducing to as low as 0.004 in the MUSHROOM dataset for **Greedy<sub>1</sub>**. On average, **Greedy<sub>Avg</sub>** does not achieve as small R-ratios as the other two al-

gorithms. It is not clear whether this observation is an artifact of the suboptimality of **Greedy<sub>Avg</sub>** algorithm or its objective function. However, based on the results of Experiment II, one would guess that the former reason is responsible for the results. On the positive side, one should point out that the R-ratio of **Greedy<sub>Avg</sub>**, though worse than the other algorithms, still achieves great reduction of the original set of rules. The **RuleCover** algorithm usually achieves lower R-ratios than **Greedy<sub>1</sub>** and **Greedy<sub>Avg</sub>**, although sometimes only marginally.

Of course, small R-ratios are good, but in order for the output rule collection to be useful they should be accompanied by large TP-ratios and small FP-ratios. Figure 4 shows the values of  $rTP_1(SA, \mathcal{R})$  for the same set of SA algorithms as before. As expected, the  $rTP_1$  takes value 1 for the for **RuleCover** algorithm; this is by definition in the case of **RuleCover** algorithm. The  $rTP_1$  values of **Greedy<sub>1</sub>** and **Greedy<sub>Avg</sub>** are also high (more than 0.8) even though the algorithm is not explicitly optimizing  $TP_1$  objective.

The corresponding  $rFP_1(SA, \mathcal{R})$  values for the same algorithms are shown in Figure 5. The trend here is that **RuleCover** exhibits the highest values for  $rFP_1$ ; this is expected given that the rule-selection process in this algorithm does not take into account the false positives.

Figures 6 and 7 show the  $rTP_{avg}$  and  $rFP_{avg}$  for SA algorithms **Greedy<sub>1</sub>**, **Greedy<sub>Avg</sub>** and **RuleCover**. **RuleCover** again exhibits noticeably good  $rTP_{avg}$  ratios for some datasets, like for example LENSES or CAR. For the majority of the datasets though, all three algorithms have similar  $rTP_{avg}$  ratios and output rulesets with  $TP_{avg}$  and expected  $TP_{avg}$  cardinality around that of the input rule collection. When it comes to  $rFP_{avg}$ , **Greedy<sub>1</sub>** and **Greedy<sub>Avg</sub>** perform the best. In this case **RuleCover**, as expected, introduces lots of false-positive predictions. For example, in the MUSHROOM

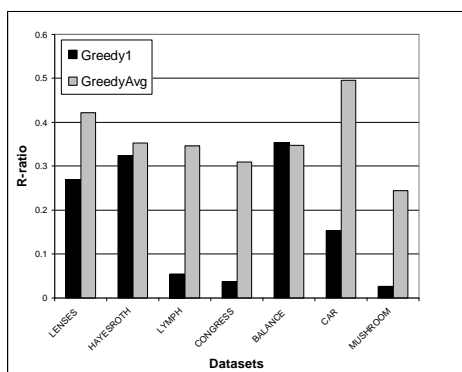


Figure 8: R-ratio for **Greedy\_1** and **Greedy\_Avg** when input ruleset  $\mathcal{R}$  was produced by NDI.

dataset it introduces 2.63 times more false positives than **Greedy\_1** and 3.73 times more than **Greedy\_Avg**.

**Experiment IV: NDI rules:** The goal of this experiment is to study the behavior of the rule-selection algorithms when the input rule collection  $\mathcal{R}$  is not mined using **Apriori** algorithm. For that purpose we use the NDI mining algorithm. NDI uses the frequent-itemset mining algorithm of [5] to generate all frequent non-derivable itemsets. It then reports the rules that can be constructed from these itemsets and which are above the *min\_conf* threshold. Although the rule collection  $\mathcal{R}$  mined by the NDI algorithm is much more condensed than the collections output by the **Apriori** algorithm our results indicate that the **Greedy\_1** and **Greedy\_Avg** algorithms manage to find even smaller representative sets of rules. Figure 8 shows the R-ratio of **Greedy\_1** and **Greedy\_Avg** algorithms when the NDI mining algorithm is used for producing the original rule collection  $\mathcal{R}$ . The smallest R-ratio (0.026) is achieved by **Greedy\_1** on the MUSHROOM dataset. The TP-ratio and FP-ratio of **Greedy\_1** and **Greedy\_Avg** exhibit the same behavior as in the case of rules mined by **Apriori** and thus the corresponding plots are omitted.

**Summary of the experiments:** The takeaway message is that our algorithms produce collections of representative association rules that are significantly smaller than the mined rule collections. There are cases where the size of the set of representatives is .4% of the size of input rule set. At the same time, we illustrate that these small sets of representative rules have high true-positive and low false positive rates when compared both with the input rule collection as well as the outputs of other algorithms proposed in the literature (e.g., **RuleCover**). A practical example of the utility of this reduction can be seen in an application of our methods to the MUH-

SROOM dataset. The original set of rules consists of 8615 rules which correspond to the singular consequent of a mushroom being poisonous. The **Greedy\_1** algorithm reduces this to 8 rules (< .1%). These 8 rules cover over 98% of the true positives covered by the original set of rules, and only 7% of the false positives.

## 6 Related work

Association rules have been a central and recurring theme in data-mining research. Therefore, any attempt to summarize this work would be far from complete. A large body of the published work has focused on devising efficient algorithms that, given an input dataset, extract all association rules that have support and confidence above specified thresholds. Although interesting, this work is not directly related to ours. The rules output by these methods are an input to our problem, and our approaches are indifferent to how this input was generated.

More related to ours is the work on *reduced rule mining*. In this case, the goal is to reduce the set of the mined association rules by considering an additional criterion that allows certain sets of rules not to be included in the output. See for example [10, 15, 16, 20], for some indicative work along these lines. The main difference between this work and ours is that in the former approaches the decision whether to disregard a rule is done *during* the mining process. Rules are disregarded on-the-fly based on some statistics extracted from the data and the rules that have already been mined. Contrary to this, our approach considers the whole set of mined rules and decides which ones to pick as representatives. Therefore, the output of the former approaches can also be seen as candidate inputs to our algorithms.

Identifying a core set of essential rules from a collection of mined rules has also been studied before. For example in [18] the authors approach the problem as a set cover problem: each mined rule represents the set of tuples that contain all the items in both the antecedent and the consequent of the rule. Then, the problem is to pick the minimum set of rules such that all tuples are covered at least once. The combinatorial nature of this formulation has some similarities with ours. However, the key difference between the two approaches is that, for each distinct RHS, not only do we want to correctly predict its appearance in tuples that have this RHS (as [18] does), but we also want to make correct (negative) predictions for the tuples that do not have the RHS. As a result the combinatorial problems that arise from our formulations are very different from the ones described in [18]. Similar to [18] is the formulation proposed in [4]. The latter work

also considers all rules independently of their RHSs and the false predictions are not taken into account in their formulation either.

At a high-level, related to ours is also the work on building classifiers using association rules. For example, [9, 11, 12] and references therein, represent this line of research. The key difference between these approaches and ours is that they consider rules competing towards classifying transactions to different classes. Classifiers' objective is to assign the right RHS (class) for every transaction. In our case, rules with different RHSs do not compete with each other, and the same transaction can be predicted to have two different RHSs. The prediction mechanism in our case is just a means for evaluating the goodness of a collection of rules rather than the goal itself. Finally, the combinatorial problems that arise in our setting are totally different from those in the previous work on classification using association rules.

## 7 Conclusions

We presented a framework for selecting a subset of representative rules from large collections of association rules produced by data-mining algorithms. We believe that such small subsets of representatives are easier to be handled by data-analysts. We defined two versions of the RULE SELECTION problem and presented simple combinatorial algorithms for solving them. Via a broad set of experiments with a wide selection of datasets, we showed that our algorithms work well in practice and pick small number of representatives which adequately describe both the original data and the input rule collection.

In our framework we defined the goodness of a set of representative rules by using how well they predict certain RHSs. In the future we would like to explore different notions of goodness and study the problems that arise from such definitions.

## References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328, 1996.

[3] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[4] T. Brijs, K. Vanhoof, and G. Wets. Reducing redundancy in characteristic rule discovery by using integer programming techniques. *Intell. Data Anal.*, 4(3,4):229–240, 2000.

[5] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. pages 74–85. Springer, 2002.

[6] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *SODA*, pages 345–353, 2000.

[7] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, J. D. Ullman, C. Yang, R. Motwani, and R. Motwani. Finding interesting associations without support pruning. In *IEEE Transactions on Knowledge and Data Engineering*, pages 489–499, 2000.

[8] I. Diakonikolas and M. Yannakakis. Succinct approximate convex pareto curves. In *SODA*, pages 74–83, 2008.

[9] G. Dong, X. Zhang, L. Wong, and J. Li. Caep: Classification by aggregating emerging patterns. In *Discovery Science (DS)*, pages 30–42, London, UK, 1999. Springer-Verlag.

[10] B. Goethals, J. Muhonen, and H. Toivonen. Mining non-derivable association rules. In *SDM*, 2005.

[11] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.

[12] B. Liu, Y. Ma, and C. K. Wong. Improving an association rule based classifier. In *PKDD*, pages 504–509, London, UK, 2000. Springer-Verlag.

[13] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *KDD*, pages 189–194. AAAI Press, 1996.

[14] D. Peleg. Approximation algorithms for the label-cover<sub>max</sub> and red-blue set cover problems. *J. Discrete Algorithms*, 5(1):55–64, 2007.

[15] J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[16] J. Quinlan. Determinate literals in inductive logic programming. In *IJCAI*, pages 746–750, San Mateo, CA, 1991. Morgan-Kaufman.

[17] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Min. Knowl. Discov.*, 2(1):39–68, 1998.

[18] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Htnen, and H. Mannila. Pruning and grouping discovered association rules. In *ECML-95 Workshop on Statistics, Machine Learning, and Knowledge Discovery in Databases*, pages 47–52, Heraklion, Crete, Greece, April 1995.

[19] V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[20] X. Yin and J. Han. Cpar: Classification based on predictive association rules. In D. Barbará and C. Kamath, editors, *SDM*. SIAM, 2003.