




Finding Probabilistic Rule Lists using the Minimum Description Length Principle

John O. R. Aoga¹(✉) , Tias Guns^{2,3}, Siegfried Nijssen¹, and Pierre Schaus¹

¹ ICTEAM, UCLouvain, Ottignies-Louvain-la-Neuve, Belgium
{john.aoga,siegfried.nijssen,pierre.schaus}@uclouvain.be

² VUB, Brussels, Belgium

³ KU Leuven, Leuven, Belgium

tias.guns@vub.ac.be, tias.guns@cs.kuleuven.be

Abstract. An important task in data mining is that of rule discovery in supervised data. Well-known examples include rule-based classification and subgroup discovery. Motivated by the need to succinctly describe an entire labeled dataset, rather than accurately classify the label, we propose an MDL-based supervised rule discovery task. The task concerns the discovery of a small rule list where each rule captures the probability of the Boolean target attribute being true. Our approach is built on a novel combination of two main building blocks: (i) the use of the Minimum Description Length (MDL) principle to characterize good-and-small sets of probabilistic rules, (ii) the use of branch-and-bound with a best-first search strategy to find better-than-greedy and optimal solutions for the proposed task. We experimentally show the effectiveness of our approach, by providing a comparison with other supervised rule learning algorithms on real-life datasets.

1 Introduction

Rule learning in supervised data is a well-established problem in data mining and machine learning. Compared to many other methods, a clear benefit of rule-based methods is that the rule format is more easy to interpret, and hence is useful in knowledge discovery. Well-known examples of rule learning are

Rule-based classification, in which the aim is to find a set of rules that predicts the class of examples well;

Subgroup discovery, in which the aim is to find a set of rules that describes subgroups of examples in the data; in these subgroups, the distribution of the target attribute is different from the overall population.

The main difference between subgroup discovery and rule-based classification is that rule-based classification aims to find a set of rules that can be applied on

J.O.R. Aoga—This author is supported by the FRIA-FNRS (Fonds pour la Formation à la Recherche dans l'Industrie et dans l'Agriculture, Belgium).

Table 1. Probabilistic rule lists example

(a) From door opening data			(b) From mushroom dataset		
	Rule list	Probability		Rule list	Probability
IF	WEDNESDAY <i>and</i> MORNING	0.879	IF	Gill-spacing is closed <i>and</i> No odor	0.95
ELSE IF	HOLIDAY <i>and</i> THURSDAY	0.011	ELSE IF	Gill-spacing is closed <i>and</i> Stalk-shape is tapering	0.0
ELSE IF	THURSDAY <i>and</i> AFTERNOON	0.987	ELSE IF	Stalk-color-above-ring is white <i>and</i> Gill-size is broad	1.0
ELSE IF	SUNDAY	0.001	ELSE IF	Gill-spacing is closed	0.0
ELSE	(Default rule)	0.101	ELSE	(Default rule)	0.56

any example to obtain a prediction for that example. Subgroup discovery aims to characterize subgroups of examples, but not necessarily all examples.

Similar to rule-based classification, in this work we are also interested in finding a set of rules that describe a target attribute fully and in an interpretable manner. However, we make a specific assumption that is not common in rule-based classification: we assume that the class attribute has a skewed distribution, and that exact prediction is certainly not possible. The following example illustrates a problem that has these characteristics.

Example 1. Assume that we characterize every minute in a year in terms of the following attributes: the part of the day the minute belongs to (morning, afternoon), the day the minute belongs to (Sunday, Monday, ...), the month the minute belongs to (January, ...) and the minute of the day (1, 2, ..., 24×60); furthermore, over a year we use a sensor to monitor when an individual opens a specific door in his house. Can we use rules to characterize when this individual opens her door?

In this example, the event of “opening a door” is expected to be a rare event; if we use a classification algorithm on the above dataset, we will notice that the class attribute is very unbalanced. Most classification algorithms will either prefer to always predict the default label (the door is closed), or will construct many very specific rules to cover the small number of examples that are the exception. The reason for this is that many rule-based classifiers find *lists* of rules; a rule that makes an error in its prediction, cannot be corrected by a later rule. Hence, most classification rule learning algorithms favor rules with lower recall but high precision.

In this paper, we propose a new algorithm for finding rule lists, designed to work well in this specific setting. It identifies simple probabilistic rule lists, such as in Table 1. Hence, the rule mining setting studied in this work can be characterized by these properties:

- it learns rules with probabilities in the head; these probabilities represent the class distribution for the examples covered by the rule, and should not be understood as class prediction;
- the list of rules is intended to characterize the class distribution over the entire data, in contrast to subgroup discovery;
- it favors smaller rule lists to ease interpretation.

Finding lists of rules that satisfy these requirements is not a straightforward task. To address these challenges, this paper proposes the following contributions.

1. We propose a new optimization criterion based on the *Minimum Description Length* (MDL) principle; this criterion aims to find rule lists that are small, yet characterize the target distribution well.
2. We propose a new search algorithm based on branch-and-bound search; this search algorithm aims to find the global optimum for the proposed optimization criterion under given constraints.

The approach that we take in this work is a *pattern set mining* approach. We first use itemset mining algorithms to find a candidate set of itemsets. From this set, we select a subset that describes the target attribute well. From the pattern set mining perspective, we propose a new supervised optimization criterion for selecting a set of *free* patterns, and a new search algorithm for finding a set of patterns that optimizes the criterion.

In the remainder of the paper, we first present related work in Sect. 2. In Sect. 3 we present the problem of finding probabilistic rule lists. Then, we describe our Minimum Description Length (MDL)-based approach in terms of the formalization and algorithms for solving it. Finally, we show experiments in Sect. 5 before concluding.

2 Related Work

This work builds on a number of areas in the literature.

Rule-based classification. There is a large literature on rule-based classification; a good overview of these algorithms, including classic algorithms such as CN2 and RIPPER, can be found in a textbook by Fürnkranz et al. [4]. Two types of rule-based classifiers can be distinguished: classifiers based on rule sets and on rule lists. In set-based classifiers, all rules that match an example are used to obtain a prediction for that example. In list-based classifiers, the first matching rule is used; we build on this class of methods.

Covering algorithms are the most popular type of rule learning algorithm. These algorithms iteratively search for a rule to add to a rule set or list. Most often, in each iteration a greedy search algorithm is used, which constructs a rule by iteratively adding the condition that improves the quality of the rule the most.

The main challenge faced by pure covering algorithms is that later rules cannot correct errors made by earlier rules in a rule list. Such algorithms hence need to favor precision over recall to obtain accurate classifiers. As a result rule lists may become unnecessarily long. One way to solve this is using *pruning*: the rule set is reduced in a post-processing step.

Pattern-based classification. Compared to traditional rule learning algorithms, pattern-based classifiers use pattern mining algorithms, such as frequent

itemset mining algorithms, to identify candidate rules [12]. These frequent itemsets are post-processed to construct rule sets or rule lists. Most of these post-processing approaches use heuristic search algorithms, although the use of exact search has also been studied [6].

Pattern set mining. From a pattern mining perspective, selecting a small set of patterns from a larger set of patterns can be seen as a pattern set mining problem [12]. In contrast to unsupervised methods, supervised methods aim to find a balance between pattern sets that are non-redundant and that are accurate. One popular approach for evaluating the quality of a pattern set is based on the Minimum Description Length principle, as pioneered in the unsupervised setting by the KRIMP algorithm [10]. Exact methods for pattern set mining were studied by Guns et al. [6], among others, but these studies did not consider scoring functions based on MDL or did not exploit freeness, as we do.

Subgroup discovery. Strongly related to both pattern mining and rule-based classification is subgroup discovery. Subgroup discovery differs from classification in that it does not aim to build a predictive model; rather, subgroup discovery algorithms are intended to return small and interpretable sets of *local* patterns; subgroups are not necessarily ordered in a specific manner. For this reason, traditional subgroup discovery algorithms were modifications of covering based rule-learning algorithms to explicitly allow for overlap between patterns [7].

Bayesian rule lists. Most related to this work is recent work by Yang et al. [11] on probabilistic rule lists. This work also finds ordered lists of probabilistic rules. Contrary to our work, however, the aim of the work of Yang et al. is to identify accurate classifiers, and not to identify as small and interpretable representations of the class distribution as possible. Furthermore, Yang et al. use a sampling based algorithm to identify good sets of patterns. We propose an alternative, exact algorithm in this work.

3 The Probabilistic Rule List Mining Problem

This work is motivated by the creation of a probabilistic rule list that summarizes labeled data well. In order to be easily interpretable, the rule list and the individual rules should be concise.

We assume the data is described by a set of discrete attributes. These attributes can be represented as a set of Boolean properties using a one-hot encoding. These properties are referred to as items in the following, in line with the itemset mining literature.

More formally, let $\mathcal{I} = \{1, \dots, m\}$ represent a set of m possible items and let $\mathcal{F} \subseteq 2^{\mathcal{I}}$ be a set of itemsets built on those items. A probabilistic rule list (PRL) built on \mathcal{F} is a sequence of rules of the form $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (I^{(2)}, p^{(2)}), \dots, (I^{(k)}, p^{(k)}) \rangle$ with p^i being a probability and $I^i \in \mathcal{F}, \forall i = 1, \dots, k-1$ and $I^k = \emptyset$. This latter is the default rule. The sequence of itemsets in the rule list can be expressed as membership to the regular language: $\langle I^1, \dots, I^k \rangle \in \mathcal{L}(\mathcal{F}^* \cdot \emptyset)$ with

\mathcal{F}^* the Kleene operator on \mathcal{F} . Table 1 shows two example rule lists (generated from different data).

The rule list has a sequential interpretation, in that the set of data instances that match the first rule I^1 are assumed to have a positive label with probability p^1 . The other data instances, those that do not match I^1 , but do match I^2 have a probability of p^2 to be positive, etc. The final empty set $I^k = \emptyset$ hence captures all instances not matched by the other rules.

We now formalize the problem of creating the probabilistic rule list based on \mathcal{F} and a dataset \mathcal{D} .

Definition 1. *As input we receive a set of itemsets \mathcal{F} that can be used to compose the rule list, and a database \mathcal{D} of instances, with for each a Boolean target attribute: $\mathcal{D} = \{(t, I_t, a_t) \mid t \in \mathcal{T}, I_t \subseteq \mathcal{I}, a_t \in \{+, -\}\}$, where the set \mathcal{T} contains the instance or transaction identifiers $\mathcal{T} = \{1, \dots, n\}$. The database can be split into a positive \mathcal{D}^+ and negative \mathcal{D}^- database, based on the target attribute value (+ or -).*

The problem of finding a probabilistic rule list is formalized as: $\operatorname{argmin}_{\mathcal{R}} \operatorname{score}(\mathcal{R}, \mathcal{F}, \mathcal{D})$ where $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (I^{(2)}, p^{(2)}), \dots, (I^{(k)}, p^{(k)}) \rangle$ is a probabilistic rule list such that $\langle I^{(1)}, \dots, I^{(k)} \rangle \in \mathcal{L}(\mathcal{F}^* \cdot \emptyset)$ and score is an optimization criterion. Various optimization criteria can be defined, including criteria inspired by classification rule learning, subgroup discovery and pattern set mining. Our aim in this work is to develop an optimization criterion that explicitly favors smaller rule lists that describe the entire target distribution well. For this purpose, we will use the Minimum Description Length principle, discussed in the next section.

4 Discovering Probabilistic Rule Lists

4.1 Coverage and Probability of a Rule List

To evaluate the quality of a rule set on a given dataset, we will use a number of concepts taken from the itemset mining literature [1].

Definition 2 (Coverage and support of an itemset). *The set of transactions in a database \mathcal{D} containing an itemset I is called the cover: $\varphi(\mathcal{D}, I) = \{(t, I_t, a_t) \in \mathcal{D} \mid I \subseteq I_t\}$. The size of the cover is called the support $\psi(\mathcal{D}, I) = |\varphi(\mathcal{D}, I)|$.*

Example 2. An example itemset database is given in Fig. 1a. $I = \{A, C\}$ is an example itemset; $\varphi(\mathcal{D}, I)$ contains transaction identifiers $\{1, 2, 5\}$, so $\psi(\mathcal{D}, I) = 3$. The set of frequent itemsets with support at least 4 is $\{\emptyset, \{A\}, \{B\}, \{C\}, \{E\}, \{B, E\}\}$ (Fig. 1b).

In the remainder of this paper, for the sake of simplicity we denote $\varphi(\mathcal{D}, I)$ as $\varphi(I)$ when no ambiguity regarding \mathcal{D} is possible. Similarly, we will use $\varphi^+(I)$ to

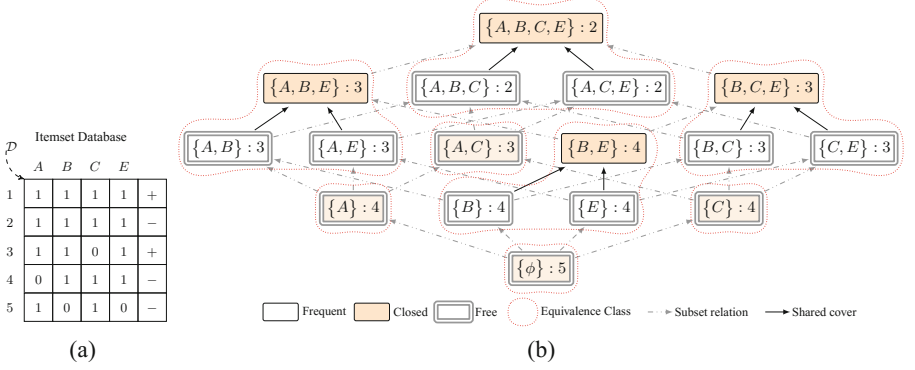


Fig. 1. (a) Itemset Database with positive/negative classes; (b) Powerset lattice of \mathcal{D} with equivalence classes.

denote $\varphi(\mathcal{D}^+, I)$ where $\mathcal{D}^+ = \{(t, I_t, a_t) \in \mathcal{D} \mid a_t = +\}$ and likewise for $\varphi^-(I)$ with $a_t = -$.

We are interested in finding a list of rules. Each itemset in the list has a cover that is defined as follows.

Definition 3 (Coverage of an itemset in a sequence). Assume the sequence of itemsets $\langle I^{(1)}, \dots, I^{(k)} \rangle$, the coverage of an itemset $I^{(j)}$ over \mathcal{D} is its cover in the database of transactions not covered by the previous itemsets $I^{(1)}, I^{(2)}, \dots, I^{(j-1)}$:

$$\Phi(\mathcal{D}, \langle I^{(1)}, \dots, I^{(k)} \rangle, j) = \varphi(\mathcal{D} \setminus (\varphi(I^{(1)}) \cup \varphi(I^{(2)}) \cup \dots \cup \varphi(I^{(j-1)})), I^{(j)}) \quad (1)$$

with $\Phi(\mathcal{D}, \langle I^{(1)}, \dots, I^{(k)} \rangle, 1) = \varphi(\mathcal{D}, I^{(1)})$.

Note that in a rule list \mathcal{R} , the last itemset is always $I^{(k)} = \emptyset$, which is the *default rule* or *final else-case*. This *empty set* inherently covers all instances not covered by any of the $k - 1$ previous rules since $\varphi(\mathcal{D}, \emptyset) = \{(t, I_t, a_t) \in \mathcal{D} \mid \emptyset \subseteq I_t\} = \mathcal{D}$ for any \mathcal{D} .

Given a rule list $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (I^{(2)}, p^{(2)}), \dots, (I^{(k)}, p^{(k)}) \rangle$ we will denote by $\Phi(\mathcal{D}, \mathcal{R}, j)$ the cover of the j th itemset in the rule list's sequence of itemsets. If no ambiguity is possible we simply write Φ_j . Similarly $\Phi_j^+ = \Phi(\mathcal{D}^+, \mathcal{R}, j)$ and $\Phi_j^- = \Phi(\mathcal{D}^-, \mathcal{R}, j)$.

When creating a rule list \mathcal{R} from a dataset \mathcal{D} given \mathcal{F} , we define the probability $p^{(j)}$ of a rule $I^{(j)}$ as $p^{(j)} = P(a_t = + \mid (t, I_t, a_t) \in \Phi(\mathcal{D}, \mathcal{R}, j)) = \frac{|\Phi_j^+|}{|\Phi_j^+| + |\Phi_j^-|}$.

Example 3. Assume the running example database (Fig. 1a) and a rule list with corresponding sequence of itemsets $\langle \{A, B, C\}, \{C\}, \emptyset \rangle$. The coverage of $I^{(2)} = \{C\}$ over \mathcal{D} is $\Phi_2 = \{4, 5\}$, instead of $\{1, 2, 4, 5\}$, as the transactions 1 and 2 were already covered by $I^{(1)} = \{A, B, C\}$. Its probability is hence $p^{(2)} =$

$\frac{|\Phi_2^+|}{|\Phi_2^+|+|\Phi_2^-|} = \frac{0}{0+2} = 0$, which indicates that no positive transaction was observed with the condition of the rule, after observing the previous rules.

At this stage, an open question is how to evaluate the quality of a probabilistic rule list \mathcal{R} . In this work, we propose to evaluate how well the rule list allows to *compress* the values for the class attribute observed in a training dataset. For this, we will use the Minimum Description Length (MDL) principle.

4.2 Minimum Description Length Encoding of Rule Lists

The Minimum Description Length (MDL) principle [5, 8] is a general method for *inductive inference*, based on the idea that ‘*the more we can compress the data, the more there are regularities in it and the more we learn from it*’ [5]. MDL allows making a trade-off between the complexity of rules and their ability to capture the distribution of the class attribute. To do this, we use a two-part code that minimizes the *number of bits* needed to encode the data with a model, as well as the number of bits to encode the model itself. As stated earlier, the focus in this work is on a code that favors simplicity.

Let $\mathcal{M} = M_1, M_2, \dots$ be a list of model candidates. In two-part MDL, the best model $M \in \mathcal{M}$ to capture information in a given database \mathcal{D} is the one which minimizes the code length $L(M) = L_{model}(M) + L_{data}(\mathcal{D}|M)$, where $L_{model}(M)$ is the length, in bits, of the description of the model itself and $L_{data}(\mathcal{D}|M)$ the length of the data, in bits, when it is encoded with this model.

In our case, models correspond to rule lists of the form $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (I^{(2)}, p^{(2)}), \dots, (I^{(k)}, p^{(k)}) \rangle$ with $I^{(j)} \in \mathcal{F}, \forall j \in 1, \dots, k-1, I^{(k)} = \emptyset$ and $p^{(j)} = \frac{|\Phi_j^+|}{|\Phi_j^+|+|\Phi_j^-|}$. We thus need to define an encoding with $L_{model}(\mathcal{R})$ an encoding of the rule list, and $L_{data}(\cdot|\mathcal{R})$ such that $L_{data}(\mathcal{D}|\mathcal{R})$ can be interpreted as the *coding length* of the distribution of $+/-$'s in \mathcal{D} when it is encoded with \mathcal{R} . The best rule list is then the one that minimizes the total length $L(\mathcal{R})$:

$$\mathcal{R}^* = \operatorname{argmin}_{\mathcal{R} \in \mathcal{L}(\mathcal{F}^* \cdot \emptyset)} L_{data}(\mathcal{D}|\mathcal{R}) + L_{model}(\mathcal{R}), \quad (2)$$

where we identified \mathcal{R} by its sequence of itemsets to ease notation; each itemset has a probability $p^{(j)}$ as defined earlier.

We now first discuss how we encode \mathcal{R} when $k \leq 2$ (i.e. $\mathcal{R} = \langle (\emptyset, p^{(1)}) \rangle$ or $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (\emptyset, p^{(2)}) \rangle$) and then generalize to the case $k > 2$.

Case $k = 2$: To understand the computation of the coding length of \mathcal{R} , we first show how we can encode a target attribute if we have an itemset I and then a *default rule*. Given a rule $(I^{(1)}, p^{(1)})$, we assume that the positive and negative labels in $\varphi(\mathcal{D}, I^{(1)})$ follow a *Bernoulli distribution*, with a probability $p^{(1)}$ for the class label. The probability density of the labels according to I is hence (omitting \mathcal{D} from the notation):

$$P_r(a_t = + | \varphi(I)) = (p^{(1)})^{|\varphi^+(I)|} (1 - p^{(1)})^{|\varphi^-(I)|}. \quad (3)$$

Theorem 1 (Local Coding length of data). *Using Shannons Noiseless Channel Coding Theorem [3] the number of bits needed to encode the class labels of \mathcal{D} using I is at least the logarithm¹ of the probability density of the class labels in \mathcal{D} given I : $L_{local\ data}(\mathcal{D}|I) = -\log P_r(a_t = + | \varphi(\mathcal{D}, I))$. Using (3) we can hence encode each positive label at a cost of*

$$L_{local\ data}(\mathcal{D}|I) = \mathcal{Q}(\varphi^+(I), \varphi^-(I)) + \mathcal{Q}(\varphi^-(I), \varphi^+(I)), \quad (4)$$

with $\mathcal{Q}(a, b) = -a \log \frac{a}{a+b}$.

We will use this bound, which can be approximated closely using arithmetic coding, as the coding length for the class labels. Based on the above theorem and assuming a rule list is $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (\emptyset, p^{(2)}) \rangle$, the coding length of Φ is

$$L_{data}(\mathcal{D}|\mathcal{R}) = L_{local\ data}(\mathcal{D}|I^{(1)}) + L_{local\ data}(\mathcal{D} \setminus \varphi(I^{(1)})|\emptyset) \quad (5)$$

Example 4. Assume the rule list is $\mathcal{R} = \langle (\{A, B, C\}, 0.50), (\emptyset, 0.33) \rangle$ and that our database \mathcal{D} (Fig. 1a) is duplicated 256 times. $L_{local\ data}(\mathcal{D}|\{A, B, C\}) = -256 \log 0.5 - 256 \log(1 - 0.5) = 512\text{bits}$ and $L_{local\ data}(\mathcal{D} \setminus \varphi(I^{(1)})|\emptyset) = -256 \log 0.33 - 256 \log(1 - 0.33) = 705\text{bits}$; then $L_{data}(\mathcal{D}|\mathcal{R}) = 1217\text{bits}$.

When we encode the class label using this model, we do not only need to encode the data, but also the model itself.

Definition 4 (Length of the model). *Assume a rule list $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (\emptyset, p^{(2)}) \rangle$, we represent $(I^{(1)}, p^{(1)})$ as a string “ $m_1 I_1^{(1)} \dots I_{m_1}^{(1)} n_1^+$ ” where, $m_1 = |I^{(1)}|$ is the number of items in $I^{(1)}$, followed by the identifiers of each item in $I^{(1)}$ and finally the number of positive labels in \mathcal{D} : $n_1^+ = |\varphi^+(I^{(1)})|$. The length, in bits, to encode this string is:*

$$L_{local\ model}(I^{(1)}) = \underbrace{\log m}_{|I^{(1)}|} + \underbrace{|I^{(1)}| \log m}_{I_1^{(1)} \dots I_{|I^{(1)}|}^{(1)}} + \underbrace{\log n_1^+}_{n_1^+} \quad (6)$$

where $\log m$ bits are required to represent m_1 , as $m_1 \leq m = |I|$, and also $\log m$ bits for each item identifier plus $\log n$ bits to encode n_1^+ . Coding n_1^- is unnecessary as it can be retrieved from the data using the itemset: $n_1^- = |\varphi(\mathcal{D}, I^{(1)})| - n_1^+$. From there, assuming that the itemset database \mathcal{D} and the set of items \mathcal{I} are known, one can easily retrieve the coverage of $I^{(1)}$ and then compute the probability $p^{(1)}$ using the number of positive labels n_1^+ . The coding length of the model \mathcal{R} is $L_{model}(\mathcal{R}) = L_{local\ model}(I^{(1)}) + L_{local\ model}(\emptyset)$.

Example 5. We continue on Example 4. To encode the model, the string “3 A B C 256” is encoded: $L_{local\ model}(\{A, B, C\}) = \log 4 + 3 \log 4 + \log 1280 = 19\text{bits}$ similarly $L_{local\ model}(\emptyset) = \log 4 + 0 \log 4 + \log 1280 = 13\text{bits}$ ² then $L_{model}(\mathcal{R}) = 32\text{bits}$. Together with $L_{data}(\mathcal{D}|\mathcal{R}) = 1217\text{bits}$ computed in Example 4, the total coding length of \mathcal{R} is $L(\mathcal{R}) = 1217 + 32 = 1249\text{bits}$.

¹ All logarithms are to base 2 and by convention, we use $0 \log 0 = 0$.

² Note that by convention the size of the default rule is $m_2 = 0$.

Case $k > 2$: Assuming now a rule list $\mathcal{R} = \langle (I^{(1)}, p^{(1)}), (I^{(2)}, p^{(2)}), \dots, (I^{(k)}, p^{(k)}) \rangle$ with $k > 2$. For $k > 1$ we need to modify the definition of $L_{local\ data}$ such that it does not consider parts of the data covered by a previous itemset in the sequence. Hence,

$$L_{local\ data}(\mathcal{D}|I^{(j)}) = \mathcal{Q}(\Phi_j^+, \Phi_j^-) + \mathcal{Q}(\Phi_j^-, \Phi_j^+) \quad (7)$$

and the total coding length is the summation of local lengths:

$$L_{data}(\mathcal{D}|\mathcal{R}) = \sum_{j=1}^k L_{local\ data}(\mathcal{D}|I^{(j)}); \quad (8)$$

the coding length of the model is:

$$L_{model}(\mathcal{R}) = \log n + \sum_{j=1}^{k-1} \left(\log m + m_j \log m + \log n \right) \quad (9)$$

To encode the size of \mathcal{R} itself, we need $\log n$ bits. Because all rule list include the *default rule*, we omit these $\log m + \log n$ bits.

Example 6. Fig. 2 shows example rule lists with coding lengths.

4.3 Coding Length Related to Likelihood and Quality of Rule Lists

The coding length of the class labels given a model \mathcal{R} is the number of bits needed to encode the class labels with \mathcal{R} . As a consequence of our choice to use Shannon's theorem, this coding length corresponds to the (*-log*) *likelihood* of the class labels according to the model. In the other words, if we would minimize the coding length of the data only, we would maximize the likelihood of the data under the model. However, as stated earlier, in this work our aim is also to find small and interpretable rule lists. We choose our code such that a relatively large weight is given to the complexity of the model.

Assuming the database of Example 4, the size of the original data is $5 \times 256 = 1280$. Encoding this data with $\mathcal{R}_1 = \langle (\{A, B, C\}, 0.50), (\emptyset, 0.33) \rangle$ we obtained $L_{data}(\mathcal{D}|\mathcal{R}_1) = 1217\text{bits}$, $L_{model}(\mathcal{R}_1) = 32\text{bits}$ and in total $L(\mathcal{R}_1) = 1249\text{bits}$. Instead, when we encode this data with $\mathcal{R}_2 = \langle (\emptyset, 0.40) \rangle$ we obtain $L_{data}(\mathcal{D}|\mathcal{R}_2) = 1243\text{bits}$, $L_{model}(\mathcal{R}_2) = 6\text{bits}$ and in total $L(\mathcal{R}_2) = 1249\text{bits}$. Looking at likelihoods only, one can see that \mathcal{R}_1 is a better model for representing this data, as it captures more information than \mathcal{R}_2 . However, in total, it is not preferable over \mathcal{R}_2 , since it is more complex to encode. The model coding length penalizes the likelihood and ensures a simple model is preferred.

For our example, the only way to improve \mathcal{R}_1 is to add (if possible) a new rule that reduces the error made by \mathcal{R}_1 by assuming that the part not covered by $\{A, B, C\}$ is for the *default rule*. Thus, by adding the itemset $\{C\}$ to \mathcal{R}_1 , which covers all 0s still present, we obtain the best model $\mathcal{R} = \langle (\{A, B, C\}, \frac{1}{2}), (\{C\}, \frac{0}{2}), (\phi, \frac{1}{1}) \rangle$ with $L(\mathcal{R}) = 546\text{bits}$ since the *default rule* now covers only remaining 1s.

4.4 A Greedy Algorithm

The probabilistic rule list that minimizes the MDL score (2) can be constructed greedily, extending the list by one rule at each step. Greedy algorithms are known to be efficient and approximate optimal solutions well in other rule learning tasks.

Algorithm 1 shows a greedy algorithm that starts with empty rule list \mathcal{R} , and then iteratively finds within a given set of patterns the rule that minimizes the coding length. The local best rule is obtained by considering at each iteration the sub-problem of finding the optimal rule list with $k \leq 2$ on the remaining data. This corresponds to finding the itemset $I^{(1)}$ such that the *coding length* is smallest (Line 3). Once the local best rule is selected the rule list is updated in Line 6 and in Line 7; its coverage is removed from \mathcal{D} . The process is then run again until \mathcal{D} is empty or the *default rule* is selected.

Example 7. Assuming our running example, at the first iteration of the greedy algorithm, the minimum code-length $L(\langle\{A, B\}, \emptyset\rangle) = 722\text{bits}$ and then it is the greedy solution (See Fig. 2).

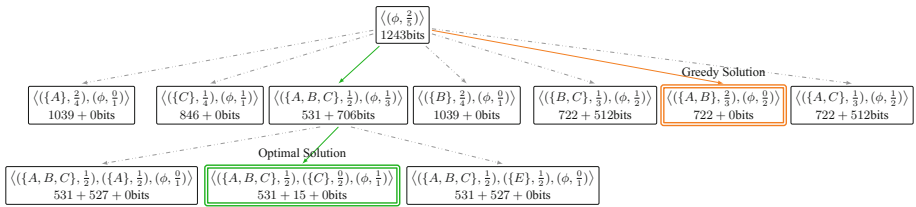


Fig. 2. Finding greedy and optimal solution base on the example of Fig. 1.

The greedy algorithm may be sub-optimal. For instance it fails to discover the $L(\langle\{A, B, C\}, \{C\}, \emptyset\rangle) = 546\text{bits}$ on our example.

4.5 Branch-and-Bound Algorithm

For finding solutions that are better than the greedy solution, we propose a best-first branch-and-bound algorithm that can prune away candidates based on a lower-bound on the MDL value. Each node in the search tree is a partial rule list, consisting of a sequence of rules *without* the *default rule*. The children of each node correspond to appending one additional rule from \mathcal{F} to the partial rule list.

Algorithm 2 shows the pseudo-code of this branch-and-bound expansion search. For clarity we omit the probabilities in the rule list representation. The algorithm receives as input a list of rule candidates \mathcal{F} and database \mathcal{D} . A priority queue is used to store the set of rule lists not yet expanded, ordered by the code-length obtained when extending the partial rule with the *default rule* (best-first

strategy). The initial best rule is the *default rule* (Line 2) and the empty rule list is added as initial search node. As long as the queue is not empty, the priority queue is dequeued and the returned partial rule list is expanded (Line 6). Each new partial rule list is evaluated as if it was completed with the *default rule* (\emptyset) and checked whether it is better than the current best rule list (Lines 7, 8).

Before adding the new partial rule list to the queue, a lower-bound on the code length is computed, that is, an optimistic estimate of the code length achievable (see next section). Only if the lower-bound is better than the current best value, the rule list is added to the queue (Lines 9, 10). If not, this part of the search tree is effectively pruned.

Algorithm 1: Greedy(\mathcal{F}, \mathcal{D})

```

1  $\mathcal{R} \leftarrow \langle \rangle$ 
2 do
3    $I^* \leftarrow \operatorname{argmin}_{I \in \mathcal{F}^*} L(\langle (I, p^{(1)}), (\emptyset, p^{(2)}) \rangle)$ 
4   if  $L(\langle (I, p^{(1)}), (\emptyset, p^{(2)}) \rangle) \geq L(\langle (\emptyset, p^{(1)}) \rangle)$  then
5      $I^* \leftarrow \emptyset$ 
6    $\mathcal{R} \leftarrow \mathcal{R} \cup (I^*, p^{(1)})$   $\triangleright$  Add this rule to the rule list
7    $\mathcal{D} \leftarrow \mathcal{D} \setminus \varphi(I^*)$ 
8 while  $I^* \neq \emptyset$ ;
9 return  $\mathcal{R}$ 

```

Algorithm 2: Branch-and-bound (\mathcal{F}, \mathcal{D})

```

1  $PQ$  : PriorityQueue  $\triangleright$  Partial rule lists ordered by code-length when
   adding default rule
2  $best\mathcal{R} \leftarrow \langle \emptyset \rangle$ ,  $best \leftarrow L(best\mathcal{R})$ 
3  $PQ.enqueue-with-priority(\langle \rangle, L(\langle \emptyset \rangle))$ 
4 while  $\mathcal{R} \leftarrow PQ.dequeue()$  do
5   for each  $I \in \mathcal{F} \setminus \mathcal{R}$  do
6      $\mathcal{R}' \leftarrow \langle \mathcal{R}, I \rangle$ 
7     if  $L(\langle \mathcal{R}', \emptyset \rangle) < best$  then
8        $best\mathcal{R} = \langle \mathcal{R}', \emptyset \rangle$ ,  $best \leftarrow L(best\mathcal{R})$ 
9     if  $lower-bound(\mathcal{R}') < best$  then
10       $PQ.enqueue-with-priority(\mathcal{R}', L(\langle \mathcal{R}', \emptyset \rangle))$ 
11 return  $best\mathcal{R}$ 

```

Lower-bound on a partial rule list A good lower-bound is difficult to compute since there is an exponential number of rules that can be added to the list. Because the rule list itself is already evaluated in the algorithm, we are seeking a lower-bound on any expansion of the rule list. The coding length is determined by $L(\mathcal{R}) = L_{model}(\mathcal{R}) + L_{data}(\mathcal{D}|\mathcal{R})$ according to (8) and (9).

The most optimistic expansion is hence achieved with the smallest possible expansion of the rule list yielding the greatest reduction of the coding

length for the data. In the best case, this is a rule of length one ($|I^{(j+1)}| = 1$) that perfectly separates the positives from the negatives. In this case, the additional code length of the rule list corresponds to a rule of length one: $L_{local\ model}(I^{(j+1)}) = \log m + 1 \log m + \log n$ and the addition to the code length of the data is: $L_{local\ data}(\mathcal{D}|I^{(j+1)}) = \mathcal{Q}(|\Phi_{j+1}^+|, 0) + \mathcal{Q}(0, |\Phi_{j+1}^-|) = 0$ with the data coding length of the *default rule* also being 0.

While such a rule expansion may not exist, the resulting value is a valid lower-bound on the code length achievable by any expansion of the partial rule list. This is because any expansion has to be greater than or equal in size to 1, and any expansion will achieve at best a data compression of 0.

Implementation details *Choice of \mathcal{F}* . The complexity of Algorithm 2 is $\mathcal{O}(|\mathcal{F}|^d)$ where d is the depth in the best-first search tree. The efficiency of the algorithm strongly depends on $|\mathcal{F}|$ since in the worst case the number of nodes is in $\mathcal{O}(|\mathcal{F}|^{|\mathcal{F}|})$.

To control the size of \mathcal{F} one can consider all frequent itemsets with a given minimum frequency threshold. Because we are interested in a small coding length, we propose to further restrict the set of patterns to the set of *frequent free itemsets* [9]. Known also as generators, a free itemset is the smallest itemset (in size) that does not contain a subset with the same cover: if I is free, $\nexists J \subset I$ s.t. $\varphi(I) = \varphi(J)$. In fact, there may be multiple free itemsets with the same cover and for our purposes just a single one of them is sufficient. In Fig. 1, all the itemsets in a double bordered rectangle are free.

Set representation as bitvectors. Each candidate itemset in \mathcal{F} is represented by the tuple (set of items, set of covered transactions). Operations on sets such as union, intersection, count, ... being at the core of our implementation, they must be implemented very effectively. For this, we represent each set by bitvectors and all the cover computation are bitwise operations on bitvectors. A rule list is represented by an array of *itemset indices* into \mathcal{F} . From the index, one can identify the itemset and its coverage. During the search process at each iteration, a new itemset I is added to the partial rule list (Line 6 of Algorithm 2). This operation involves updating the cover of the rule list computed using (1) which depends on all the transactions already covered. To do this effectively, we keep the transactions already covered in a single bitvector $T_{covered}^{(j)} = \varphi(I^{(1)}) \cup \varphi(I^{(2)}) \cup \dots \cup \varphi(I^{(j)})$. The coverage after the addition of a new itemset $I^{(j+1)}$ is then

$$\Phi(\mathcal{D}, \mathcal{R} \cup I^{(j+1)}, j+1) = -T_{covered}^{(j)} \cap \varphi(I^{(j+1)}).$$

5 Experiments

We evaluate our approach from three perspectives: (i) the quality of obtained solutions: how expressive and concise are the rule lists; what is the log-likelihood of the data given the lists; (ii) the accuracy and sensibility of our method under various parameters, evaluated using area under ROC curves (AUC), (iii) the predictive power of our method, using AUC as well.

Table 2. Benchmark features

name	anneal	car	australian-cr.	heart-cl.	krvskp	mushroom	primary-tu.	dermatology	gallup	door	soybean
$ \mathcal{D} $	812	1728	653	296	3196	8124	336	366	15734	3216	630
$ \mathcal{I} $	89	21	124	95	73	112	31	133	41	11	50
$\frac{ \mathcal{D}^+ }{ \mathcal{D} }$	0.77	0.7	0.55	0.54	0.52	0.52	0.24	0.2	0.19	0.16	0.15

Table 3. Total code lengths for several datasets (θ is the minimum support for \mathcal{F})

	anneal	car	australian-cr.	heart-cl.	krvskp	mushroom	primary-tu.	dermatology	gallup	door	soybean
θ	20	5	20	20	5	20	20	20	10	1	1
$ \mathcal{F} $	1361	22	2495	2024	65	1145	214	763	15	35	49
PRLg	587	710	386	262	2594	1978	249	39	10327	1876	356
PRLc	532	628	380	249	845	967	249	39	10163	1876	314

Note that we add a comparison with other classification methods to properly position our work; our aim is not to build a classification model that is more accurate on commonly used datasets.

Datasets. We use nine annotated datasets publicly available from the *CP4IM*³ and UCI⁴ repositories. We also used the *door* dataset as described in the introduction (Example 1). Furthermore, we used the *Gallup* dataset [2], from a project with the same name on migratory intentions. This data set is not publicly available, but can be purchased. Our objective here is to understand the migratory intentions between two countries by considering the socio-parameters of education, health, security and age. All these datasets have been preprocessed and their characteristics are given in Table 2.

Algorithms. We compare with popular tree-based classification methods such as Random Forests (*RF*) and decision trees (*CART*) from the *scikit-learn library*, as well as the rule-learning methods *JRIP* (Weka version of *RIPPER*) and *SBRL* [11] available in R CRAN (see Sect. 2). We run *SBRL* with the default setting (number of iterations set to 30.000, number of chains 10 and a lambda parameter of 10).

Protocols. All experiments were run in the JVM with maximum memory set to 8GB on PCs with Intel Core i5 64bits processor (2.7GHz) and 16GB of RAM running MAC OS 10.13.3. Our approach is called *PRL* (for probabilistic rule lists) and is implemented in Scala. The candidate itemsets \mathcal{F} are the frequent free itemsets. *PRL* name can be followed by *g* for greedy or *c* for complete branch-and-bound. Evaluation of AUC is done using *stratified 10-fold cross-validation*. For the reproducibility of results, all our implementations are open source and available online⁵.

³ <https://dtai.cs.kuleuven.be/CP4IM/datasets/>.

⁴ <http://archive.ics.uci.edu/ml/datasets.html>.

⁵ <https://projetsJOHN@bitbucket.org/projetsJOHN/mdlrulesets>.

Compression power of PRL. Table 3 gives the total code length obtained for the greedy *PRLg* and the complete branch-and-bound *PRLc* approaches. As can be observed, the *compression ratio* (total code length/size of the datasets) is substantial. For instance, it is 10% for the dermatology dataset. For 8/11 instances *PRLc* discovers a probabilistic rule list compressing better than the one obtained with *PRLg*. The gain obtained with *PRLc* is sometimes substantial, for instance on the *krvskp* and *mushroom* data sets.

Impact of the parameters. The set of possible itemsets \mathcal{F} to create the rule list is composed of the frequent free itemsets generated with a minimum support threshold θ . Fig. 3a reports the compression ratio for decreasing values of θ . As expected the compression ratio becomes smaller whenever θ decreases. The reason is that the set \mathcal{F} is growing monotonically, allowing more flexibility to discover a probabilistic rule list that compresses well.

Both the greedy and the complete branch-and-bound algorithms can easily limit the size of the probabilistic rule list they produce. This is done by stopping the expansion of the list beyond a given size limit k . Figure 3b reports the compression ratio for increasing values of k . As expected the compression ratio becomes smaller whenever k increases for *PRLc* and stabilizes at some point when the limit k becomes larger than the length of the optimal rule list. Surprisingly this is not necessarily the case for the greedy approach that is not able to take advantage of longer rule lists on this benchmark.

Regarding the execution time according to the size of the rules, as shown in Fig. 3c, with a time limit of 10 min, we can see that the greedy approach is more scalable. *PRLc* and *SBRL* execution time evolves exponentially, *PRLc* being faster than *SBRL* though. Note that as soon as the optimal solution is found, in the case of *PRLc*, the execution time does not increase so much anymore. The reason is that most of the branches are cut-off by the branch-and-bound tree exploration beyond that depth limit.

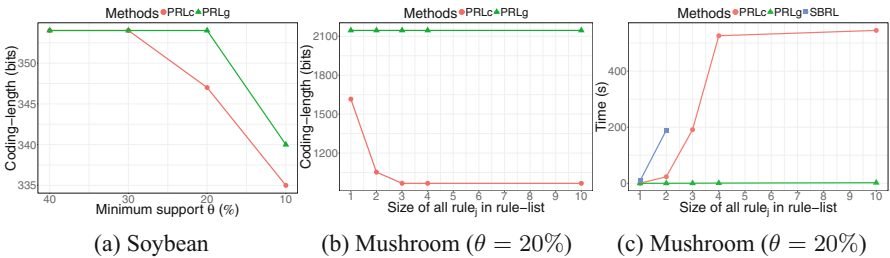


Fig. 3. Sensibility of PRL for several settings using mushroom and soybean datasets

Comparison of PRL with existing rule learning algorithms. We compare the rule list produced by our approaches (*PRLg* and *PRLc*) and by *SBRL* [11]. Figure 4a gives the code length for the model and for the data (class labels) for

various datasets for the different approaches. Note that the code length for the data corresponds to the *log-likelihood* of the class labels under the rule list. From the rule lists obtained using the training set, the probability (to be positive) of each transaction in the test set is predicted and the coding lengths are computed using the (8) and (9). The reported values are averaged over 10 folds. The model coding length represents the size of the encoding of the initial rule list.

One can see that the PRL approaches are competitive with SBRL. On Fig. 4a, it often obtains the smallest data coding length except for the *mushroom* dataset. The reason is that the test set of *mushroom* is classified perfectly by SBRL. The rule lists produced are arguably shorter with PRLg and PRLc than with SBRL.

The *mushroom* dataset is investigated further in Fig. 4b and 4c. The data coding length and the area under the ROC curve are computed for increasing prefixes of the lists. As we can see, at equal prefix size ($k < 5$) our approach obtains *better likelihood* and is *more accurate* than SBRL. Then beyond $k \geq 5$ SBRL continues to improve on accuracy while PRLg and PRLc stagnates. The lists indeed have reached their optimal length at $k = 5$. This evolution is a clear

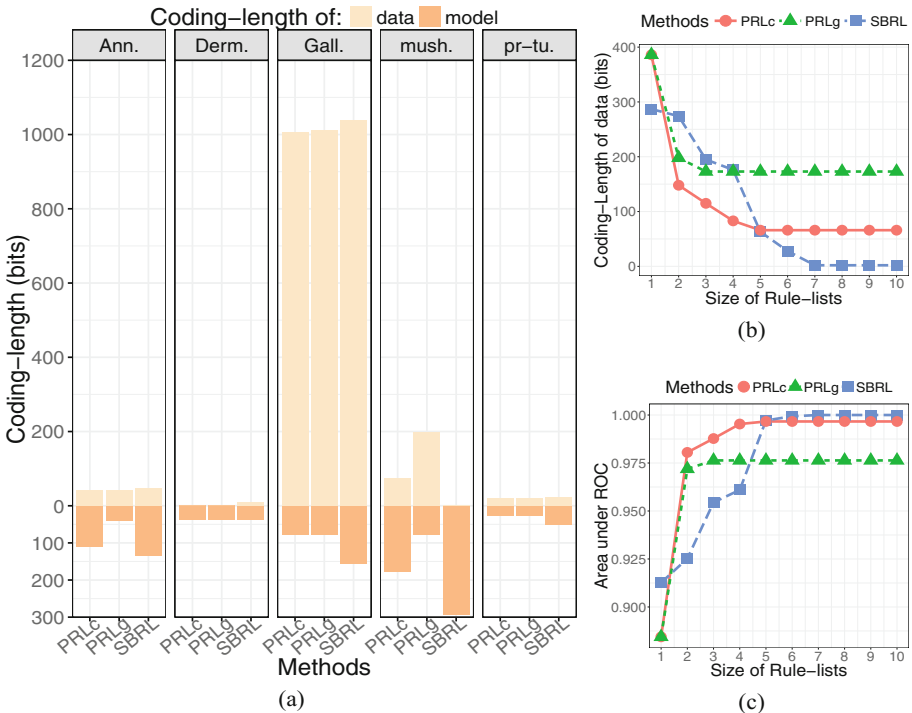


Fig. 4. (a) Comparison of coding length in average among *PRL* (g,c) and *SBRL* for different test datasets and (b and c) evolution of the coding length of *data only* (top) and the AUC (bottom) for several rule lists size, for *mushroom* dataset, for all 10-folds ($\theta = 10\%$, $|I| = 2$).

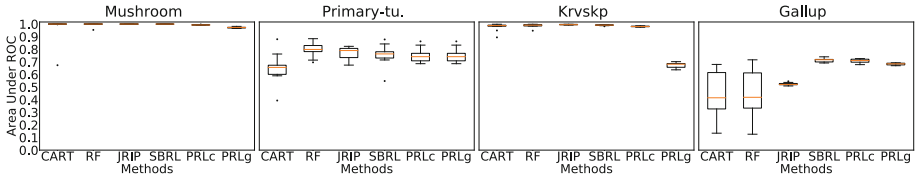


Fig. 5. Comparison of Area under ROC among different methods and four datasets, for all 10-folds ($\theta = 10\%$, $|I| = 1$).

illustration of the difference between the type of rule lists produced by SBRL and our approach. While SBRL lists are more focused on classification, MDL-based lists are a trade-off between the data-coding length (classification) and the complexity of lists (model code length).

Prediction power of PRL and other supervised learning approaches.

Although our approach is not designed to generate the best rule list for classification, we evaluate its prediction power in the light of well-known classification methods: *CART*, *RF*, *SBRL* and *JRIP* using 10-fold cross-validation and default settings. For *PRL* the classification is done by associating with each transaction the probability that its label is positive. This probability is that of the first rule of the rule list (obtaining from the training set) that matches with this transaction. The results are shown in Fig. 5.

In general, the AUC of our methods are greater than 0.6 and the optimal solution always has a greater or equal accuracy compared to the greedy approach. The difference becomes significant on databases like *Krvskp* where the difference in compression ratio is also high (Fig. 3).

State-of-the-art methods are often more accurate, except in unbalanced datasets (Gallup, primary-tu.) where our approaches are very competitive. One can see that rule based methods do better on very unbalanced databases like Gallup.

6 Conclusion

This work proposed a supervised rule discovery task focused at finding probabilistic rule lists that can concisely summarize a boolean target attribute, rather than accurately classify it. Our method is in particular applicable when the target attribute corresponds to rare events. Our approach is based on two ingredients, namely, the Minimum Description Length (MDL) principle, and a branch-and-bound search strategy. We have experimentally shown that obtained rule lists are compact and expressive. Future work will investigate the support of multivariate target attributes (> 2 classes) and new types of patterns, such as sequences.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. *Int. Conf. Manag. Data (SIGMOD)* **22**(2), 207–216 (1993)
2. Esipova, N., Ray, J., Pugliese, A.: Number of potential migrants worldwide tops 700 million. Gallup, USA (2018)
3. Fano, R.M.: The transmission of information. Massachusetts Institute of Technology, Research Laboratory of Electronics Cambridge, Cambridge (1949)
4. Fürnkranz, J., Gamberger, D., Lavrač, N.: Foundations of Rule Learning. Springer Publishing Company, Incorporated (2014)
5. Grünwald, P.D.: The minimum description length principle. MIT press, Cambridge (2007)
6. Guns, T., Nijssen, S., De Raedt, L.: k-Pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.* **25**(2), 402–418 (2013)
7. Lavrac, N., Kavsek, B., Flach, P.A., Todorovski, L.: Subgroup discovery with CN2-SD. *J. Mach. Learn. Res.* **5**, 153–188 (2004)
8. Rissanen, J.: Modeling by shortest data description. *Automatica* **14**(5), 465–471 (1978)
9. Szathmary, L., Napoli, A., Kuznetsov, S.O.: ZART: a multifunctional itemset mining algorithm. In: Eklund, P.W., Diatta, J., Liquiere, M. (eds.) *Proceedings of the 5th International Conference on Concept Lattices and Their Applications, CLA 2007*. vol. 331 (2007)
10. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.* **23**(1), 169–214 (2011)
11. Yang, H., Rudin, C., Seltzer, M.: Scalable bayesian rule lists. In: Precup, D., Teh, Y.W. (eds.) *Proceedings of the 34th International Conference on Machine Learning, ICML'17*. *Proceedings of Machine Learning Research*, vol. 70, pp. 3921–3930. PMLR (2017)
12. Zimmermann, A., Nijssen, S.: Supervised pattern mining and applications to classification. In: Aggarwal, C.C., Han, J. (eds.) *Frequent Pattern Mining*, pp. 425–442. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07821-2_17