Interpretable multiclass classification by MDL-based rule lists

Manuel Proença, H.; Leeuwen, M. van

Information Sciences 512 (2020) 1372–1393

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Interpretable multiclass classification by MDL-based rule lists

Hugo M. Proença*, Matthijs van Leeuwen

LIACS, Leiden University, the Netherlands

ARTICLE INFO

Article history:
Received 9 May 2019
Revised 10 August 2019
Accepted 25 October 2019
Available online 25 October 2019

Keywords:
Rule lists
Minimum Description Length principle
Interpretable models
Classification

ABSTRACT

Interpretable classifiers have recently witnessed an increase in attention from the data mining community because they are inherently easier to understand and explain than their more complex counterparts. Examples of interpretable classification models include decision trees, rule sets, and rule lists. Learning such models often involves optimizing hyperparameters, which typically requires substantial amounts of data and may result in relatively large models.

In this paper, we consider the problem of learning compact yet accurate probabilistic rule lists for multiclass classification. Specifically, we propose a novel formalization based on probabilistic rule lists and the minimum description length (MDL) principle. This results in virtually parameter-free model selection that naturally allows to trade-off model complexity with goodness of fit, by which overfitting and the need for hyperparameter tuning are effectively avoided. Finally, we introduce the Classy algorithm, which greedily finds rule lists according to the proposed criterion.

We empirically demonstrate that Classy selects small probabilistic rule lists that outperform state-of-the-art classifiers when it comes to the combination of predictive performance and interpretability. We show that Classy is insensitive to its only parameter, i.e., the candidate set, and that compression on the training set correlates with classification performance, validating our MDL-based selection criterion.

1. Introduction

Interpretable machine learning has recently witnessed a strong increase in attention [12], both within and outside the scientific community, driven by the increased use of machine learning in industry and society. This is especially true for applications domains where decision making is crucial and requires transparency, such as in health care [27,30] and societal problems [26,47].

While it is of interest to investigate how existing 'black-box' machine learning models can be made transparent [38], the trend towards interpretability also offers opportunities for data mining, or *Knowledge Discovery from Data* (KDD), as this field traditionally has a stronger emphasis on intelligibility.

In recent years several interpretable approaches have been proposed for supervised learning tasks, such as classification and regression. Those include approaches based on prototype vector machines [35], generalized additive models [32], decisions sets [25,44], and rule lists [30,46]. Restricting our focus to classification, we make two important observations. First,

* Corresponding author.
E-mail addresses: h.manuel.proenca@liacs.leidenuniv.nl (H.M. Proença), m.van.leeuwen@liacs.leidenuniv.nl (M. van Leeuwen).

https://doi.org/10.1016/j.ins.2019.10.050
0020-0255/

Rule		antecedent		consequent	usage
1	IF	$\{backbone = no\}$	THEN	$\Pr(invertebr.) = 0.55$	10
				$\Pr(bug) = 0.45$	8
2	ELSE IF	$\{breathes = no\}$	THEN	$\Pr(fish) = 0.93$	13
				$\Pr(reptile) = 0.07$	1
3	ELSE IF	$\{feathers = yes\}$	THEN	$\Pr(bird) = 1.00$	20
4	ELSE IF	$\{milk = no\}$	THEN	$\Pr(reptile) = 0.50$	4
				$\Pr(amphibian) = 0.50$	4
∅	ELSE	ELSE	THEN	$\Pr(mammal) = 1.00$	41

Fig. 1. Example of a Probabilistic Rule List (PRL) obtained by CLASSY on the *zoo* dataset, without the need for any parameter tuning. Test accuracy: 87%. The dataset contains 7 classes, 101 examples, and 35 binary variables. Usage refers to the number of examples covered by a certain rule and class label. Note that we did not apply Laplace smoothing here for clarity of presentation; Eq. (5) defines the actual probability estimates.

we observe that state-of-the-art algorithms [3,25,30,44,46] are designed for binary classification; no interpretable methods specifically aimed at multiclass classification have been proposed, in spite of being a common scenario in practice. Multiclass classification is more challenging because of 1) the increased complexity in model search, due to the uncertain consequences of favouring one class over the others, and 2) the lack of possibilities to prune the search such as commonly used when finding, e.g., decision lists [3] or Bayesian rule lists [46] for binary classification. Our second observation is that although recent methods based on rules [30,46] and decision sets [25,44] have been shown to be effective, they tend to have 1) a fair number of hyperparameters that need to be fine-tuned, and 2) limited scalability. Especially the need for hyperparameter tuning can be problematic in practice, as it requires significant amounts of computation power and data (i.e., not all data can be used for training, as a substantial part has to be reserved for validation).

To address these shortcomings, *we introduce a novel approach to finding interpretable, probabilistic multiclass classifiers that requires very few hyperparameters and results in compact yet accurate classifiers*. In particular, we will show that our method naturally provides a desirable trade-off between model complexity and classification performance without the need for parameter tuning, which makes the application of our approach very straightforward and the resulting models both adequate classifiers and easy to interpret.

We will use probabilistic rule lists, as both the antecedent of a rule (i.e., a *pattern*) and its consequent (i.e., a probability distribution) are interpretable [30]. Using a probabilistic model has the additional advantage that one cannot only provide a crisp prediction, but also make a statement about the (un)certainty of that prediction.

We show that, given a set of ordered patterns, we can trivially estimate the corresponding consequent probability distributions from the data. The remaining question, then, is how to select a set of patterns that together define a probabilistic rule list that is accurate yet does not overfit. This is not only important to ensure generalizability beyond the observed data, but also to keep the models as compact as possible: larger models are harder to interpret by a human analyst [21]. Recent optimization [25] and Bayesian [46] approaches heavily rely on hyperparameters to achieve this, but those need to be tuned by the analyst and we specifically aim to avoid this.

The solution that we propose is based on the minimum description length (MDL) principle [18,40], which has been successfully used to select small sets of patterns that summarize the data in the context of exploratory data mining [9,29,42]. The MDL principle can be paraphrased as "*induction by compression*" and roughly states that the best model is the one that best compresses the data. Advantages of the MDL principle include that it has solid theoretical foundations, avoids the need for hyperparameters, and automatically protects against overfitting by balancing model complexity with goodness of fit.

Our first main contribution is the formalization of the problem of selecting the optimal probabilistic rule list using the minimum description length principle. Although the MDL principle has been used for pattern-based classification before [42], we are the first to introduce a MDL-based problem formulation aimed at selecting rule lists for multiclass classification. Technically, our approach includes the use of the prequential plug-in code, a form of *refined MDL* that has only been used once in pattern-based modelling [9]. One advantage of our approach is that the resulting problem formulation is completely parameter-free.

Our second main contribution is CLASSY *, a heuristic algorithm for finding good probabilistic rule lists.* Inspired by the KRIMP algorithm [42], we select a good set of rules from a set of candidate patterns. We empirically demonstrate, by means of a variety of experiments, that CLASSY outperforms RIPPER, C5.0, CART, and Scalable Bayesian Rule Lists (SBRL) [46] when it comes to the combination of classification performance and interpretability, in particular when taking into account that it has much fewer hyperparameters.

Table 1
Our approach, Classy, does *Multiclass* classification, makes *Probabilistic* predictions, has a global optimisation *Criterion*, can handle large numbers of *cand*idate rules, and does not need hyperparameter *tuning*. "Others" denotes classical algorithms such as CART, CBA, C4.5, and RIPPER.

Method	Multiclass	Probabilistic	Criterion	$\gg$1K cand	No tuning
Classy	✔	✔	✔	✔	✔
IDS [25]	✔	-	✔	✔	-
CORELS [3]	-	-	✔	-	-
MRL [4]	-	✔	✔	-	✔
SBRL [46]	-	✔	✔	-	-
FURIA [20]	✔	-	-	✔	-
Others	✔	✔	-	✔	-

To illustrate this, Fig. 1 shows an example rule list that was found using Classy on the *zoo*[1] dataset, without any parameter tuning. Although it is not perfectly accurate, its accuracy (87%) is pretty good considering that there are seven classes and the list only has four rules. Moreover, it provides probabilistic predictions and is very easy to interpret.

The remainder of this paper is organized as follows. First, Section 2 discusses related work, after which we introduce probabilistic rule lists and MDL for such lists in Sections 3 and 4, respectively. Section 5 presents the Classy algorithm. After that we continue with experiments in Section 6 and conclude in Section 7.

2. Related work

We start by comparing the most important features of our algorithm to those of state-of-the-art algorithms, and then provide a brief overview of the most relevant literature, grouped into three topics: 1) rule-based models; 2) similar approaches in pattern mining; and 3) MDL-based data mining. For an in-depth overview of interpretable machine learning, we refer to Molnar [34].

Table 1 compares the most important features of our proposed approach, called Classy, to those of other rule-based classifiers, which will be described in the next subsections. Classical methods, such as CART [7], C4.5 [37] and RIPPER [10], lack a global optimisation criterion and thus rely on heuristics and hyperparameters to deal with overfitting. Fuzzy rule-based models [2,23], here represented by FURIA [20] use rule sets instead of rule lists and lack probabilistic predictions. Recent Bayesian methods [25,44,44] are limited to small numbers of candidate rules and binary classification, limiting their usability, and are here represented by SBRL [46] (which is representative for all of them). A recent approach also using MDL and probabilistic rule lists (MRL) [4] is aimed at describing rather than classifying and cannot deal with multiclass problems or a large number of candidates. Interpretable decision sets (IDS) [25] and certifiable optimal rules (CORELS) [3] use similar rules but do not provide probabilistic models or predictions.

Note that methods that explain black-box models [38,39], typically denoted by the term *explainable machine learning*, also aim to make the decisions of classifiers interpretable. However, they mostly focus on sample-wise (local interpretation) explanations, while we focus on explaining the whole dataset (global interpretation) by means of a single model. As these goals lead to clearly different problem formulations and thus different results, it would not be meaningful to empirically compare our approach to explainable machine learning methods.

2.1. Rule-based models

Rule lists have long been successfully applied for classification; RIPPER is one of the best known algorithms [10]. Similarly, decision trees, which can easily be transformed to rule lists, have been used extensively; CART [7] and C4.5 [37] are probably the most best-known representatives. These early approaches represent highly greedy algorithms that use heuristic methods and pruning to find the 'best' models.

Fuzzy rules have been extensively studied in the context of classification and interpretability. Several approaches to construct fuzzy rule-based models have been proposed, such as transforming the resulting model of another algorithm into a fuzzy model and posteriorly optimizing it [20], using genetic algorithms to combine pre-mined fuzzy association rules [2], and doing a multiobjective search over accuracy and comprehensibility to find Pareto-optimal solutions [23]. Although these approaches are related, the rules are aggregated in a rule set, i.e., a set of independent if rules that can be activated at the same time to classify one instance, contrary to one rule at the time for rules lists. This makes the comparison between both types of models difficult. Also, these fuzzy rule-based models do not provide probabilistic predictions.

Over the past years, rule learning methods that go beyond greedy approaches have been developed, i.e., by means of probabilistic logic programming for independent rule-like models [5], greedy optimization of submodular problem formulation or simulated annealing in the case of decision sets [25,44], by Monte-Carlo search for Bayesian rule lists [30,46], and

[1] http://archive.ics.uci.edu/ml/datasets/Zoo.

through branch-and-bound with tight bounds for decision lists [3]. Even though in theory these approaches could be easily extended to the multiclass scenario, in practice their algorithms do not scale with the higher dimensionality that arises from the search in multiclass space with an optimality criteria. Also, only Bayesian rule lists [30,46] and Bayesian decision sets [44] provide probabilistic predictions.

All previously mentioned algorithms share some similarities with CLASSY. In particular Bayesian rule lists [30,46] are closely related as they use the same type of models, albeit with a different formulation, based on Bayesian statistics. This difference leads to different types of priors—for example, we use the universal code of integers [41]—and therefore to different results; we will empirically compare the two approaches. Certifiable optimal rules [3] have a similar rule structure but do not provide probabilistic models or predictions. Decision sets [25] share the use of rules, but as opposed to (ordered) lists they consider (unordered) sets of rules.

2.2. Pattern mining

Association rule mining [1], a form of pattern mining, is concerned with mining relationships between itemsets and a target item, e.g., a class. One of its key problems is that it suffers from the infamous *pattern explosion*, i.e., it tends to give enormous amounts of rules. Several classifiers based on association rule mining have been proposed. Best-known are probably CBA [33] and CMAR [31], but they tend to lack interpretability because they use large numbers of rules. Ensembles of association rules, such as Harmony [43] or classifiers based on emergent patterns [16], can increase classification performance when compared to the previous methods, however they can only offer local interpretations. Subgroup discovery and similar approaches [24] are all relevant too, but they focus on finding descriptive patterns and not on finding global classification models. Approaches to supervised pattern mining based on significance testing [45] do not focus on finding global models either.

A last class of related methods is that of *supervised pattern set mining* [49]. The key difference is that these methods do not automatically trade-off model complexity and classification accuracy, requiring the analyst to choose the number of patterns k in advance.

2.3. MDL-based data mining

In data mining, the MDL principle has been used to summarize different types of data, e.g., transaction data [9,42], and two-view data [28].

In prediction it has been previously used to deal with overfitting [10,37] and in the selection of the best compressing pattern [48].

RIPPER and C4.5 [10,37] use the MDL principles in their post-processing phase as a criteria for pruning, while we use it in a holistic way for model selection. Although Krimp has been used for classification [42], it was not designed for this: it outputs large pattern sets, one for each class, and does not give probabilistic predictions. DiffNorm [9] creates models for combinations of classes and also uses the prequential plug-in code, but was designed for data summarization. Aoga et al. recently also proposed to use probabilistic rule lists and MDL [4], but 1) we propose a vastly improved encoding, which is tailored towards prediction (instead of summarization), 2) our solution does multiclass classification, and 3) our algorithm has better scalability.

3. Multiclass classification with rule lists

In this section we formalize the probabilistic rule list model and show how to estimate its parameters, i.e., the rule consequent probabilities. The notation most commonly used throughout this paper is summarized in Table 2.

Let $D = (X, Y) = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$ be a *supervised Boolean dataset*, i.e., a Boolean dataset X with a (multi)class label vector Y. Note that any categorical dataset can be trivially represented as a Boolean dataset using dummy variables, hence our methods apply to any categorical dataset as well. Each example forms a pair $(\mathbf{x}, y)$, which consists of an instance of Boolean variables $\mathbf{x}$ and a class label y.

An instance $\mathbf{x} = (x_1, x_2, ..., x_k)$ consists of $k = |V|$ binary values, with V the set of all Boolean variables in X. That is, $\mathbf{x}$ is an element of the set of all possible Boolean vectors of size $|V|$, i.e, $\mathbf{x} \in \mathcal{X} \doteq \{0, 1\}^{|V|}$. A pattern a is a logical conjunction of variable-value assignments over instance space $\mathcal{X}$, e.g., $a = [x_2 = 1 \wedge x_3 = 1]$. A pattern a *occurs* in instance $\mathbf{x}$, denoted $a \sqsubseteq \mathbf{x}$, iff $\mathbf{x}$ satisfies the predicate defined by pattern a. Thus, in our example, the pattern occurs in an instance iff $x_2 = 1$ and $x_3 = 1$; the values of the other variables do not influence the occurrence of this pattern. A pattern is said to have size $|a|$ equal to the number of conditions it contains; in this case $|a| = 2$. Each class label is an element of the set of all classes, i.e., $y_i \in \mathcal{Y}$, where $\mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$ and $|\mathcal{Y}|$ is the number of classes in the dataset.

We consider the problem of multiclass classification. That is, given training data D, the goal is to induce a classification model that accurately predicts class label $c \in \mathcal{Y}$ for any (possibly unseen) instance $\mathbf{x} \in \mathcal{X}$.

3.1. Probabilistic rule lists

A *probabilistic rule list* (PRL) R is an ordered list of k *rules* $r_1, \ldots, r_k$ ending with a *default rule* $r_\varnothing$, where each defines a probability distribution over the class labels. Note that this means that R has $|R| + 1$ rules in total. Each rule consists

Table 2
Table of commonly used notation.

Symbol	Definition
D	Dataset of examples
D^a	Subset of D where pattern a occurs
$D^{y=c}$	Subset of D where class label c occurs
$D^{(a,y=c)}$	Subset of D where a and c both occur
n	Total number of examples/instances
X	Dataset of instances of D (without class labels)
V	Set of all Boolean variables in D
$\|V\|$	Total number of Boolean variables in D
$\mathcal{X}$	Set of all possible binary vectors of size $\|V\|$
$\mathbf{x}$	Instance
x	Boolean variable
Y	Vector of class labels in D
$\mathcal{Y}$	Set of all class labels in D
$\|\mathcal{Y}\|$	Total number of class labels in D
y	Class label
$\mathcal{R}$	Set of all proabilistic rule lists
R	Proabilistic rule list
$\|R\|$	Number of rules in R (excluding the default rule)
r_i	ith rule of R
a_i	Pattern/rule antecedent of r_i
$\|a_i\|$	Number of logical conditions in pattern a_i
$supp(a)$	Number of instances where a occurs
U^{a_i}	Usage of a_i given R
$U^{(a_i,c)}$	Usage of a_i where class c also occurs given R
$\boldsymbol{\theta}_i$	Vector of probabilities of each class label of r_i
θ_i^c	Probability of the ith rule for class c

of a pair $r_i = (a_i, \boldsymbol{\theta}(a_i))$, where a pattern a_i is the antecedent and a categorical distribution (i.e., a generalized Bernoulli distribution) over the class labels $\boldsymbol{\theta}(a_i)$ is the consequent. Whenever clear from the context we use $\boldsymbol{\theta}_i$ as shortcut for the parameters associated with pattern a_i. Each categorical distribution is parameterized by individual class probabilities $\boldsymbol{\theta}_i = (\theta_i^{c_1}, \ldots, \theta_i^{c_{|\mathcal{Y}|}})$, such that $\theta_i^{c_j} > 0, \forall_{i,j}$ and $\sum_j \theta_i^{c_j} = 1, \forall_i$. That is, rule i is given by

$$a_i \rightarrow y \sim Categorical(\boldsymbol{\theta}_i). \tag{1}$$

The default rule $r_\varnothing$, which intuitively corresponds to a rule with the empty set as antecedent, is associated with a categorical distribution over the class labels denoted by $\boldsymbol{\theta}_\varnothing$. An example PRL with $|R| = 2$ rules is given by:

$$\text{rule 1} \ : if\ a_1 \sqsubseteq \mathbf{x} \ \text{ THEN } \ y \sim Categorical(\boldsymbol{\theta}_1) \tag{2}$$
$$\text{rule 2} \ : else\ if\ a_2 \sqsubseteq \mathbf{x}\ then\ y \sim Categorical(\boldsymbol{\theta}_2) \tag{2}$$
$$\text{default} : else\ y \sim Categorical(\boldsymbol{\theta}_\varnothing) \tag{2}$$
$$\tag{2}$$

Given a PRL R, an instance $\mathbf{x}$ is classified by going through the rule list top-down, i.e., $\mathbf{x}$ is classified according to $r_* = (a_*, \boldsymbol{\theta}_*)$, which is defined as the first rule in the list for which a_* occurs in $\mathbf{x}$ ($a_* \sqsubseteq \mathbf{x}$). If none of the patterns a_i, $\forall_{i \in 1,\ldots,|R|}$ occurs in a given instance, the classifier automatically falls back to the default rule $r_\varnothing$. From the pattern a_* that is activated for a given instance, the user obtains a corresponding probability distribution $\boldsymbol{\theta}_*$ over the class labels for instance $\mathbf{x}$. In case a crisp prediction is necessary, as for example when comparing a PRL with other classifiers, we follow the typical approach, which is to predict the class label that has the highest probability:

$$\hat{y} = \arg\max_{c \in \mathcal{Y}} \theta_*^c. \tag{3}$$

Note that contrary to decision lists, which only provide an associated class per rule, probabilistic rule lists provide a distribution over all class labels. This provides the user with extra information about the classification that is made, in the form of a probability of seeing each class label for a certain instance. This is especially relevant in the multiclass scenario where crisp classification implies a choice between more than two classes.

3.2. Parameter estimation

In the previous section the PRL is assumed to be given, while in practice we want to learn its parameters from the data. We defer the problem of selecting the patterns a_i to the next section and first describe how to estimate the parameters of the categorical distributions from data, i.e., how to estimate $\boldsymbol{\theta}_i$, for $i \in \{1, \ldots, |R|, \varnothing\}$, given an (ordered) set of patterns.

We first introduce some notation. The *support* of a pattern a_i is the number of times that the pattern occurs in (training) data D:

$$supp(a_i) = |\{\mathbf{x} \subset D \mid a_i \sqsubseteq \mathbf{x}\}| \tag{4}$$

The *usage* of $a_i \in R$ is the number of times it is activated in (training) data D. That is, it is the support of a_i minus the instances that were already covered by other patterns that come before a_i in R:

$$\text{usage}(a_i \mid R, D) = |\{\mathbf{x} \subset D \mid a_i \sqsubseteq \mathbf{x} \wedge \left(\bigwedge_{\forall_{j<i}} a_j \sqsubseteq \mathbf{x}\right)\}|$$

For ease of presentation, we abbreviate $\text{usage}(a_i \mid R, D)$ as U^{a_i} whenever D and R are clear from the context.

Next, we introduce class-specific usage as the number of times a pattern is activated on a training instance with class label c. We define a *class-conditioned dataset* as

$$D^{y=c} = \{(\mathbf{x}, y) \subset D \mid y = c\},$$

and *class-specific usage* as

$$U^{(a_i,c)} = \text{usage}(a_i \mid R, D^{y=c}).$$

Given the usages and class-specific usages, which are easy to compute, it is straightforward to define a maximum likelihood estimator for $\Pr(y = c \mid a_i)$, for any rule a_i and class c. We use a variant that is called a smoothed maximum likelihood estimator:

$$\hat{\theta}_i^c = \frac{U^{(a_i,c)} + \epsilon}{U^{a_i} + |\mathcal{Y}|\epsilon}. \tag{5}$$

Unlike the regular maximum likelihood estimator, this smoothed variant—known as Laplace smoothing—adds a (small) pseudocount ϵ to each class-specific usage even when that class has no counts. This avoids zero probabilities for any class label and corresponds to using a symmetric Dirichlet prior ϵ for each class [17]. Note that this estimate could be interpreted as the confidence of the nth rule in a list, accounting for what has been covered by previous rules.

4. MDL for multiclass classification

Having defined our models and parameter estimator, the remaining question is how to select adequate models. As we are interested in finding compact yet accurate rule lists that do not overfit, we resort to the minimum description length (MDL) [18,40] principle, which can be paraphrased as "*induction through compression*". The problem of selecting a concrete rule list from a large space of possible rule lists is a *point hypothesis selection* problem, for which we should use a two-part code [18].

In contrast to existing pattern-based modeling approaches (e.g., [29,42]), we deal with a *supervised* setting in which the goal is to learn a mapping from instances to class labels. This implies that we are not looking for structure *within* instance data X, but for structure in X that helps to *predict* Y.

That is, to induce a mapping from instances to class labels, we should consider the instance data X to be given as 'input' to the (classification) model and *only encode the class labels* Y. Clearly, the models that we consider are the probabilistic rule lists that we introduced in the previous section. Then, given the complete space of models $\mathcal{R}$, uniquely specified by all ordered sets of patterns over $\mathcal{X}$, the optimal model is the model $R \in \mathcal{R}$ that minimizes

$$L(D, R) = L(Y \mid X, R) + L(R), \tag{6}$$

where $L(Y|X, R)$ is the encoded length, in bits[2], of the class labels given data X and model R, and $L(R)$ is the encoded length, in bits, of the model. Eq. (6) represents a trade-off between how well the model fits the data, $L(Y|X, R)$ and the complexity of that model, $L(R)$. Note that, on a high level, two-part code in (6) is similar to the one that was recently used in the context of two-view data [28], but there the goal was summarization rather than classification and the details of the encodings are very different. The next subsections describe the two parts of the encoding, together with examples of how to compute the length of the encodings in practice.

4.1. Model encoding

Following the rule of parsimony associated with the MDL principle [18], the model encoding should result in larger code lengths for more complex models. To accomplish this we use only two types of codes for the different model components, the universal code for integers and the uniform code.

[2] To obtain lengths in bits, all logarithms in this paper are to the base 2.

The universal code for integers [41], also called the universal prior for integers, is given by $L_{\mathbb{N}}(i) = \log k_0 + \log^* i$, where $\log^* i = \log i + \log\log i + \ldots$ and $k_0 \approx 2.865064$. This code makes no *a priori* assumption about the maximum number i accepted by the model and a small assumption in terms of penalizing larger numbers, as it grows logarithmically with i and thus slower than the number of data instances n. This makes it quite different from the Poisson prior typically used in Bayesian approaches [46]: that prior more strongly penalizes integers that are further away from the expectation of the distribution, as defined by the user-chosen parameter. We use $L_{\mathbb{N}}(n)$ when we want to penalize the increase of elements in the model, such as the number of rules or the length of a pattern.

The uniform code avoids any bias by assigning code words of equal length to all elements and is therefore used when all elements are equal. E.g., to encode a variable x from a set of $|V|$ variables: $L_U(x) = -\log \frac{1}{|V|} = \log |V|$.

We will now show how to compute the total length of a model, i.e., a probabilistic rule list R over the variable space V:

$$L(R) = L_{\mathbb{N}}(|R|) + \sum_{a_i \subset R} L(a_i), \tag{7}$$

where first the number of rules is encoded using the universal code for integers, and then the individual patterns are encoded. The length of pattern a_i is given by

$$L(a_i) = L_{\mathbb{N}}(|a_i|) + |a_i| \log |V| \tag{8}$$

where the number of conditions in a_i is encoded with the universal code for integers, and then each of its conditions are encoded with a uniform code over V. Contrary to what is common in existing MDL-based pattern set mining approaches (e.g., [9,42]), which are aimed at summarization, we do not use normalized supports for encoding our patterns. That is, previous work typically uses codes based on the support of a pattern of size 1 (singleton), e.g., $a = [x_2 = 1]$, and normalizes this support by the sum of all their supports. Although this works well for summarization, in classification higher support does not necessarily imply better predictive power; hence we use the uniform code. In general, without prior knowledge the uniform code represents the best, unbiased choice [18].

Example 1 (part 1 of 2): We use the example rule list in Fig. 1 to show how to compute the length of the model encoding. The model contains 4 rules plus a default one, 1 condition per rule, over a dataset with 35 binary variables. According to Eq. (7) the length of the model encoding is:

$$L(R) = L_{\mathbb{N}}(4) + \sum_{i=1}^{4} L_{\mathbb{N}}(1) + \log 35 = 31.12 \text{ bits}$$

Note that for the purpose of model selection we are only concerned with the length of the encoding, not in materialised codes, hence the values should not be rounded to natural numbers.

4.2. Data encoding

For the encoding of the data we use the *prequential plug-in code*, because it is asymptotically optimal even without any prior knowledge on the probabilities [18]. Moreover, *the prequential plug-in code directly uses and gives us the smoothed maximum likelihood estimates* θ_i^c *for* $\Pr(y = c \mid a_i)$, as defined in Section 3.2, which makes it a natural choice.

Intuitively, the idea of the prequential plug-in code is that one starts with a pseudocount ϵ for each possible element, constructs a code using these pseudocounts, starts encoding/sending/decoding messages one by one, and then *updates the count of each element after sending/receiving each individual message*.

We apply this idea to encode the class labels, Y. Ignoring the rule list for a moment, initially each class label has a pseudocount of ϵ. Hence, when sending the first class label, y_1, we effectively use a uniform code, i.e., $-\log \frac{\epsilon}{|\mathcal{Y}|\epsilon}$. After that, however, we increase the count of that class label by one. Normalizing the updated counts results in a new categorical probability distribution—hence a new code: $-\log \frac{\epsilon+1}{|\mathcal{Y}|\epsilon+1}$. This code is the *best possible code given the data seen so far* and is equal to the smoothed maximum likelihood of Eq. (5). Formally, the plug-in code for encoding the class labels is defined as

$$\Pr_{\text{plug-in}}(y_i = c \mid Y_{i-1}) := \frac{|\{y \in Y_{i-1} \mid y = c\}| + \epsilon}{\sum_{k \in \mathcal{Y}} |\{y \in Y_{i-1} \mid y = k\}| + \epsilon}, \tag{9}$$

where y_i represents the ith class label, $Y_{i-1} = \{y_1, \ldots, y_{i-1}\}$ represents the sequence of the $i-1$ first class labels, and ϵ is the pseudocount necessary for $\Pr_{\text{plug-in}}(y_1 = c \mid y_0)$ to be valid. Choosing the uniform prior, i.e., $\epsilon = 1$, is a common choice for categorical distributions [46], and we will use that value henceforth.

We now show how the probabilistic rule lists can be used in the encoding of the class labels. By definition, only one rule is activated for each instance, hence each rule only activates in a unique part (subset) of the dataset. By realizing that the encoding of an example in a subset only depends on the rule that formed that subset, the encoding of the dataset can be simplified to the sum of the encoding of its subsets. We define the part covered by a rule antecedent $a_i \in R$ as

$$D^{a_i} = \{X^{a_i}, Y^{a_i}\} = \{(\mathbf{x}, y) \in D \mid a_i \sqsubseteq \mathbf{x} \wedge \left(\bigwedge_{\forall_{j<i}} a_j \sqsubseteq \mathbf{x} \right)\}.$$

Thus it is possible to define the encoding of the whole data as:

$$L(Y \mid D, R) = \sum_{a_i \in R} L(Y^{a_i} \mid X^{a_i}, R). \tag{10}$$

Inserting the prequential plug-in code (9) in (10) we obtain:

$$\begin{aligned} L(Y^{a_i} \mid X^{a_i}, R) &= -\log\left(\prod_{j=1}^{U^{a_i}} \Pr_{\text{plug-in}}(y_j \mid Y_{j-1}^{a_i})\right) \\ &= -\log\left(\frac{\prod_{c\in\mathcal{Y}}\prod_{j=0}^{U^{(a_i,c)}-1}(j+\epsilon)}{\prod_{j=0}^{U^{(a_i,c)}-1}(j+\epsilon C)}\right) \\ &= -\log\left(\frac{\prod_{c\in\mathcal{Y}}(U^{(a_i,c)}-1+\epsilon)!/(\epsilon-1)!}{(U^{a_i}-1+\epsilon|\mathcal{Y}|)!/(\epsilon|\mathcal{Y}|-1)!}\right) \\ &= -\log\left(\frac{\prod_{c\in\mathcal{Y}}\Gamma(U^{(a_i,c)}+\epsilon)/\Gamma(\epsilon)}{\Gamma(U^{a_i}+\epsilon|\mathcal{Y}|)/\Gamma(\epsilon|\mathcal{Y}|)}\right), \end{aligned} \tag{11}$$

where $Y_j^{a_i}$ is a sequence of class labels of length j in part D^{a_i}, and U^{a_i} and $U^{(a_i,c)}$ are the usage and class-specific usage of pattern a_i respectively. Further, Γ is the gamma function, an extension of the factorial to real and complex numbers that is given by $\Gamma(i) = (i-1)!$.

Even though the code was formulated for sequential data, the order in which the class labels are transmitted in *i.i.d* data does not affect the encoded length, as the probability distribution only depends on the *usage* of the patterns, not on the order, as can be seen in the last equation.

Example 1 (part 2 of 2): We continue our example of Fig. 1 with the computation of the length of the data encoding. For this we use $\epsilon = 1$, as we will also use in our experiments. First, we show how to compute the length of encoding the part covered by rule 1 using Eq. (11):

$$L(Y^{a_1} \mid X^{a_1}, R) = -\log\left(\frac{\Gamma(10+1)\Gamma(8+1)\Gamma(1)^5}{\Gamma(1)^7} \cdot \frac{\Gamma(7)}{\Gamma(18+7)}\right) = 32.46 \text{ bits}$$

We repeat this step (not shown) for the other parts of the dataset, which are covered by rules 2,3,4 and $\varnothing$ respectively. Summing the length of all five parts, following Eq. (10), we obtain 110.36 bits for the length of the data encoding given R, i.e., $L(Y|D, R)$.

Finally, we sum the results obtained for $L(Y|D, R)$ and $L(R)$ (in the first part of this example), and obtain 141.48 bits for the joint length of the data and model encoding, as defined by Eq. (6).

5. The Classy algorithm

Given our model class—probabilistic rule lists—and its corresponding MDL formulation, what remains is to develop an algorithm that—given the training data—finds the best model according to our MDL criterion. To this end, in this section, we present Classy, a greedy search based algorithm that iteratively finds the best rules to add to a rule list. This section is structured as follows. First, a brief description of separate-and-conquer greedy search is given. Then compression gain, i.e., the measure that uses compression to score candidate rules, is described. After that, the Classy algorithm is defined. Then, it is explained how individual rules—candidates for the model—are generated from the data. Finally, we analyse Classy's time and space complexity.

5.1. Separate-and-conquer greedy search

Greedy search is very commonly used for learning decision trees and rule lists [10,15,37], as well as for pattern-based modelling using the MDL principle [9,28,42]. A few recent approaches use optimization techniques [46], but these have the limitation that the search space must be strongly reduced, providing an exact solution to an approximate problem (as opposed to an approximate solution to an exact problem).

Global heuristics, such as evolutionary algorithms, have been extensively applied to fuzzy rule-based model learning [13], and although they could also be applied here, we found that the arguments in favor of a local search approach were stronger: 1) local heuristics have often been successfully applied for pattern-based modelling using the MDL principle, making it a natural approach to consider; 2) local heuristics are typically faster than global heuristics, as much fewer candidates need to be evaluated; 3) global heuristics typically require substantially more (hyper) parameters that need to be tuned (e.g., population size, selection and mutation operators, etc.), while local heuristics have very few.

Given the arguments presented here the algorithm that we propose is based on greedy search. More specifically, it is a heuristic algorithm that, starting from a rule list with just a default rule equal to the priors of the class labels in the data, adds rules according to the well-known *separate-and-conquer* strategy [15]: 1) iteratively find and add the rule that gives the

largest change in compression; 2) remove the data covered by that rule; and 3) repeat steps 1-2 until compression cannot be improved. This implies that *we always add rules at the end of the list, but before the default rule*.

5.2. Compression gain

The proposed heuristic is based on the compression gain that is obtained by adding a rule $r = (a, \boldsymbol{\theta})$ to a rule list R, which will be denoted by $R \oplus r$. We will argue—and demonstrate empirically later—that for the current task it is better to consider *normalized* gain rather than the typically used *absolute* gain. Note that the gains are defined positive if adding a rule represents a compression improvement, and negative vice-versa.

Absolute compression gain, denoted $\Delta L(D, R \oplus r)$, is defined as the difference in code length before and after adding a rule r to R. The gain can be divided in two parts: *data gain*, $\Delta L(Y|X, R \oplus r)$, and *model gain*, $\Delta L(R \oplus r)$. Together this gives

$$\begin{aligned}\Delta L(D, R \oplus r) &= L(D, R) - L(D, R \oplus r) \\ &= \underbrace{L(Y \mid X, R) - L(Y \mid X, R \oplus r)}_{\Delta L(Y|X, R \oplus r)} \\ &+ \underbrace{L(R) - L(R \oplus r)}_{\Delta L(R)}. \end{aligned} \tag{12}$$

Using Eq. (7) we show absolute gain as:

$$\begin{aligned}\Delta L(R \oplus r) &= L_{\mathbb{N}}(|R|) - L_{\mathbb{N}}(|R| + 1) \\ &- L_{\mathbb{N}}(|a|) - |a| \log |V|. \end{aligned} \tag{13}$$

Note that model gain is always negative, as adding a rule adds additional complexity to the model.

In the case of the data gain it should be noted that adding rule r to R only activates the part of the data previously covered by the default rule, as new rules are only added after the previous ones and before the default rule. This search strategy of adding rules assumes that the previous rules already cover their subset well, and that improvements only need to be made where no rule is activated, which corresponds to the region of the dataset covered by the default rule. Hence, we only need to compute the difference in length of using the previous default rule $\varnothing$ and the combination of the new pattern $a \in r$ with the new default rule $\varnothing'$. Using Eq. (10) we obtain

$$\begin{aligned}\Delta L(Y \mid X, R \oplus r) = &\overbrace{\cancel{\sum_{a_i \in R} L(Y^{a_i} \mid X^{a_i}, R)} + L(Y^{\varnothing} \mid X^{\varnothing}, R)}^{L(Y|X,R)}. \\ &\underbrace{-\cancel{\sum_{a_i \in R} L(Y^{a_i} \mid X^{a_i}, R)} - L(Y^{\varnothing'} \mid X^{\varnothing'}, R \oplus r) - L(Y^{a} \mid X^{a}, R \oplus r)}_{L(Y|X, R \oplus r)} \end{aligned} \tag{14}$$

Normalized compression gain, denoted $\delta L(D, X \oplus r)$, is defined as the absolute gain normalized by the number of instances that are activated by pattern $a \in r$, which can be obtained by dividing absolute gain by the usage of a:

$$\delta L(Y \mid X, R \oplus r) = \frac{\Delta L(Y \mid X, R \oplus r)}{U^a} \tag{15}$$

By normalizing for the number of instances that a rule covers, normalized gain *favors rules that cover fewer instances but provide more accurate predictions* compared to absolute gain. When greedily covering the data, it is essential to prevent choosing large but moderately accurate rules in an early stage; this is likely to lead to local optima in the search space, from which it could be hard to escape. As this is bound to happen when using absolute gain, our hypothesis is that normalized gain will lead to better rule lists. We will empirically verify if this is indeed the case.

5.3. Candidate generation

Candidates are probabilistic rules of the form $r = (a, \boldsymbol{\theta})$ that are considered for addition to a rule list for a dataset D. The candidates are generated by first mining a rule antecedent/pattern a using a standard frequent pattern mining algorithm, e.g., FP-growth [6], and then finding the corresponding consequent categorical distribution $\boldsymbol{\theta}$ given the dataset, i.e., using Eq. (5). In practice these mining algorithms have only two parameters: the minimum support threshold m_s and the maximum length l_{max} of a pattern. Mining frequent patterns can be done efficiently due to the anti-monotone property of their support, i.e., given a pattern a and b, if a has less conditions then b, i.e., $a \subset b$, implies that $supp(a) \geq supp(b)$. This property is also used to **remove strictly redundant rules** in Classy. Given all candidates from the frequent pattern mining algorithm, if the antecedent a is a strict subset of antecedent b, i.e., $a \subset b$, and they have equal support, $supp(a) = supp(b)$, we say that antecedent b is redundant and will never be selected. This is a consequence of their encoding, i.e., $L_{plug-in}(Y^a \mid X^a, R) = L_{plug-in}(Y^b \mid X^b, R)$ in the case they are being considered for the same position, and that the model encoding length of b will always be larger than a, i.e., $L(a) < L(b)$. From this we can conclude that b will never be preferred over a during model search, as the gain of a will always be greater.

Algorithm 1 The Classy algorithm.

Input: Dataset D, candidate set $Cands$
Output: Multiclass probabilistic rule list R
1: $Cands \leftarrow RemoveRedundancy(Cands)$
2: $R \leftarrow [\varnothing]$
3: **repeat**
4: $\quad r \leftarrow \text{argmax}_{\forall r' \in Cands} : \delta L(D, R \oplus r')$
5: $\quad R \leftarrow R \oplus r$
6: $\quad \text{UpdateCandidates}(D, R, Cands)$
7: **until** $\delta L(D, R \oplus r') \leq 0, \forall r' \in Cands$
8: **return** R

5.4. Finding good rule lists

We are now ready to introduce Classy, a greedy algorithm for finding good solutions to the MDL-based multiclass classification problem as formalized in Section 4. The algorithm, outlined in Algorithm 1, expects as input a (supervised) training dataset D and a set of candidate patterns, e.g., a set of frequent itemsets mined from D, and returns a probabilistic rule list.

The first step of our algorithm, line 1 is to remove the strictly redundant patterns as mentioned in Section 5.3. After that, in line 2 we initialize the rule list with the default rule, which acts as the baseline model to start from. Then, while there is a rule that improves compression (Ln 7), we keep iterating over three steps: 1) we select the best rule to add (Ln 4)—we here use normalized gain for ease of presentation, but this can be trivially replaced by absolute gain; 2) we add it to the rule list (Ln 5); and 3) we update the usage, and gain of the candidate list (Ln 6). To update the usage of a candidate it is necessary to remove from its usage the instances that it has in common with the previous added rule, and then the gain of adding the candidate can be updated. When there is no rule that improves compression (negative gain) the while loop stops and the rule list is returned.

5.5. Complexity

In this section we analyze the time and space complexity of Classy. In terms of time complexity, Classy can be divided in two parts: 1) an initialization step, and 2) an iterative loop where one rule is added to a PRL in each iteration.

The time complexity of the initialization step is dominated by sorting the candidates (ascending by length) obtained after running the frequent pattern mining algorithm, and the computation of their instance ids, i.e., the indexes of the instances where each candidate is present. Sorting all candidates takes $\mathcal{O}(|Cands| \log |Cands|)$ time. To compute the instance ids of the candidates, Classy first computes the presence of each singleton condition, i.e., $x_i = 1$ is tested for each variable, in each instance, and then stores them as a bitset in a hash table. As this is done for the whole dataset, it takes $\mathcal{O}(|D||V|)$ time. Then, for candidates of size equal or greater than two and given a sorted array of candidates, it sequentially computes the instance ids of each candidate a based on its decomposition in two candidates of one less condition, i.e., it computes the ids of a based on two candidates b_1 and b_2 for which $b_1 \cup b_2 = a$ of length $|b_1| = |b_2| = |a| - 1$. The ids of a are obtained by the intersection of the sets of instance ids of the smaller length candidates and has a complexity of $\mathcal{O}(|(b_1)_{ids}| + |(b_2)_{ids}|)$. As this is done for each class and in a worst case it would cover the whole dataset, it takes $\mathcal{O}(|D| + |D|)$. Doing this for all candidates gives $\mathcal{O}(|Cands||D|)$.

After the initialization step Classy iteratively finds the best rule to add for a total of $|R|$ runs, where R is the PRL that Classy outputs at the end. The time complexity of this loop is dominated by the removal of the instance ids that all candidates have in common with the last added rule. Using again the fact that the intersection of instance ids is upper bounded by the dataset size $|D|$, the removal of instance ids takes at most $\mathcal{O}(|R||Cands||D|)$ time. Given that the rule list can grow at most to the size of the dataset, an upper bound on this complexity is $\mathcal{O}(|Cands||D|^2)$. Joining everything together, Classy has a worst case time complexity of

$$\mathcal{O}(|Cands||D|^2),$$

which is really a worst case scenario, because in general MDL will obtain PRLs that are much smaller than the dataset size, i.e., $|R| \ll |D|$, making it possible to treat it as a constant. Making this assumption we obtain a more realistic worst case time complexity of

$$\mathcal{O}(|Cands| \log |Cands| + |Cands||D|).$$

Note that the time complexity associated with the Gamma function used in the computation of lengths (10) and gains (15) of data encoding is not problematic when compared with the other terms. This is due to its recursive computation for $|D|$ values, which can be stored in a dictionary. In total this takes $\mathcal{O}(M(|D|) + |D|)$ time, where $M(*)$ is the complexity of the used multiplication; in the case of our Python implementation this is the Katsuraba multiplication. From then on, the lookup of a value only takes $\mathcal{O}(1)$ time.

Table 3
Dataset properties: number of {samples, binary variables, classes, average number of candidate patterns per fold for Classy with $m_s = 5\%$ and $l_{max} = 4$}. The datasets are ordered first by number of classes and then by the number of samples.

Dataset	$\|D\|$	$\|V\|$	$\|\mathcal{Y}\|$	$\|Cands\|$
hepatitis	155	48	2	39137
ionosphere	351	155	2	332560
horsecolic	368	81	2	23552
cylBands	540	120	2	304749
breast	699	14	2	299
pima	768	34	2	543
tictactoe	958	26	2	1907
mushroom	8124	84	2	79602
adult	48842	96	2	7231
iris	150	14	3	144
wine	178	63	3	13439
waveform	5000	96	3	86889
heart	303	46	5	21876
pageblocks	5473	39	5	2902
led7	3200	22	10	2507
pendigits	10992	81	10	107001
chessbig	28056	54	18	1384

In terms of memory complexity, Classy has to store for each candidate for each class: their instance ids $\mathcal{O}(|(a)_{ids}||\mathcal{Y}|)$, their support $\mathcal{O}(|\mathcal{Y}|)$, and their score $\mathcal{O}(|\mathcal{Y}|)$. It is easy to see that $|(a)_{ids}||\mathcal{Y}|$ is upper bounded by the dataset size $|D|$ and that all other memory requirements will be dominated by this part. Also, the storage of the gamma function for each integer up to $|D|$ is only $\mathcal{O}(|D|)$, which gets dwarfed by the instance storage, thus obtaining a worst case memory complexity of

$$\mathcal{O}(|D||Cands|).$$

6. Experiments

In this section we empirically evaluate our approach[3], first in terms of its sensitivity to the candidate set provided and the relationship between compression and classification performance, and second in comparison to a set of representative, state-of-the-art baselines in terms of classification performance, interpretability, overfitting, and runtime.

Data. We use 17 varied datasets (see Table 3) from the LUCS/KDD[4] repository, all of which are commonly used in classification papers. They were selected to be diverse, ranging from 150 to 48 842 samples, from 16 to 157 Boolean variables, and from 2 to 18 classes.

Candidates. Frequent pattern mining algorithms generate different candidate sets by setting different values for the minimum support per class threshold m_s and maximum pattern length l_{max}. To demonstrate that Classy is insensitive to the exact settings of these parameters, we fix a single set of parameter values for all experiments on all datasets (except when we investigate the influence of the candidate set). Specifically, we use $l_{max} = 4$ and $m_s = 5\%$, to obtain a desirable trade-off between candidate set size, convergence, and runtime.

These values were objectively derived based on two criteria: making each run finish within 10 min while demonstrating that Classy can deal with large candidate set sizes. First we chose $l_{max} = 4$ because this potentially results in very large candidate sets with many redundant rules (i.e., rules that are very similar / strongly overlapping). We then fixed m_s by requiring the runs for all datasets to strictly finish in under 10 min and for most datasets even under 1 min, so as to be comparable to *CART, C*5.0 and *JRip* in runtime, and also to have attained (empirical) convergence in terms of compression ratio on the training set—further lowering m_s would not increase compression—as can be seen from the vertical dashed lines in Fig. 3c.

Candidate patterns are mined using Borgelt's implementation of the well-known frequent pattern mining algorithm FP-growth [6]. The same candidate set was used for all experiments except when assessing its influence on Classy in Section 6.2. For that experiment we fixed $l_{max} = 4$ and varied the minimum support threshold per class from $m_s = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$.

Evaluation criteria. We evaluate and compare our approach based on classification performance, overfitting, interpretability, and runtime. In addition, we assess the influence of the candidate set on our algorithm and whether better compression corresponds to better classification. All results presented are averages obtained using 10 times repeated 10-fold cross-validation (with different seeds).

[3] Implementation available on https://github.com/HMProenca/MDLRuleLists.
[4] http://cgi.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html.

To quantify how well a rule list compresses the class labels, we define *relative compression* as

$$L\% = \frac{L(D, R)}{L(D, \{\varnothing\})}, \tag{16}$$

where $L(D, \{\varnothing\})$ is the compressed size of the data given the rule list with only a default rule, i.e., with only the dataset priors for each class. We measure relative compression on the training data, as we use that for model selection.

Classification performance is measured using three measures: accuracy: balanced accuracy [8]; and *Area Under the ROC Curve* (AUC). Each measure portrays different aspects of the classifier performance. Accuracy shows the total number of correct classifications. Balanced accuracy, or averaged class accuracy, takes into account the imbalance of class distributions in the dataset and gives the same importance to each class. It is obtained by averaging the recall associated with each class label; informally:

$$bAcc = \frac{1}{|\mathcal{Y}|} \sum_{c \in \mathcal{Y}} \text{recall}(c)$$

AUC, on the other hand, is not based on a fixed threshold and takes into account the probabilities associated with each prediction. In case of multiclass datasets we use *weighted AUC* [36], as it takes into account the class distribution in the dataset. Weighted AUC is obtained by weighing per-class 'binary' AUCs (one-versus-all) with the marginal class frequencies:

$$AUC_{weighted} = \sum_{c \in \mathcal{Y}} AUC(c) \frac{supp(c)}{|D|}, \tag{17}$$

where $AUC(c)$ is the one-versus-all AUC for class label c and $\frac{supp(c)}{|D|}$ is the frequency of that same class label.

For interpretability we follow the most commonly used interpretation, i.e., that smaller models are easier to understand [12]. With this in mind, we assess: the number of rules and the number of conditions per rule; in all cases, fewer is better. When analyzing decision trees, the number of leaves is given as the number of rules (which includes the default rule), and the average depth of the leaves (except for the longest—assumed the default rule) is given as the number of conditions per rule. Although rule lists derived from decision trees can often be simplified, we here choose not to do this because these directly measures how it would be read by humans.

Overfitting is measured in terms of the absolute difference between the AUC performance in the training set and in the test set. Finally, for runtime, wall clock time in minutes is measured; no parallelization was used.

Note on standard deviations. Given that the number of values reported both in figures and tables is large, and that standard deviations are usually 2+ orders of magnitude smaller than their corresponding averages, we choose not to report them—to avoid unnecessarily cluttering the presentation. We did analyse them though and explicitly comment on the few cases where relevant.

6.1. Compression versus classification

We first investigate the effect of using absolute (12) or normalized gain (15). To this end Fig. 2 depicts how the two heuristics perform with respect to relative compression (on the training set) and AUC (on the test set).

The first observation is that better compression of the training data clearly corresponds to better classification performance on the test data. This is backed by a correlation of -0.92 and a corresponding p-value lower than 0.0001 for the independence test between both variables for the normalized gain data. This is a crucial observation, as it constitutes an independent, empirical validation of using the MDL principle for rule list selection. Moreover, it also shows that MDL successfully protects against overfitting: using normalized gain leads to models that not only compress the training data better, but also provides accurate predictions on the test data.

The second observation is that normalized gain performs better overall than absolute gain: AUC is higher in 15 out of 17 cases and relative compression is lower or equal in 11 out of 17 times. This confirms that normalized gain is, as we hypothesized, the best choice. We will therefore use *normalized gain* for the remaining experiments.

6.2. Candidate set influence

In this set of experiments we study the influence of the candidate set on Classy, which technically is its only "parameter", as it is the only part that can influence its output given the same dataset. In order to vary the candidate set objectively, the minimum support threshold ranges over $m_s = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$ and the maximum pattern length was fixed at $l_{max} = 4$, allowing the generation of large candidate sets.

The results can be seen in the set of Fig. 3, which show the influence of the candidate set on Classy through: the size of candidates mined in Fig. 3a; runtime in Fig. 3b; compression on the training set in Fig. 3c; AUC in the test set in Fig. 3d; and the number of rules in a rule list in Fig. 3e.

Fig. 3a shows the growth of of the candidate set size with the minimum support threshold used, and that, as expected, its growth is exponential with the change in minimum support. Fig. 3b shows that in general the runtime increases at a rate similar to the increase in candidate size of Fig. 3a. This is in accordance with our analysis of time complexity in Section 5.5,

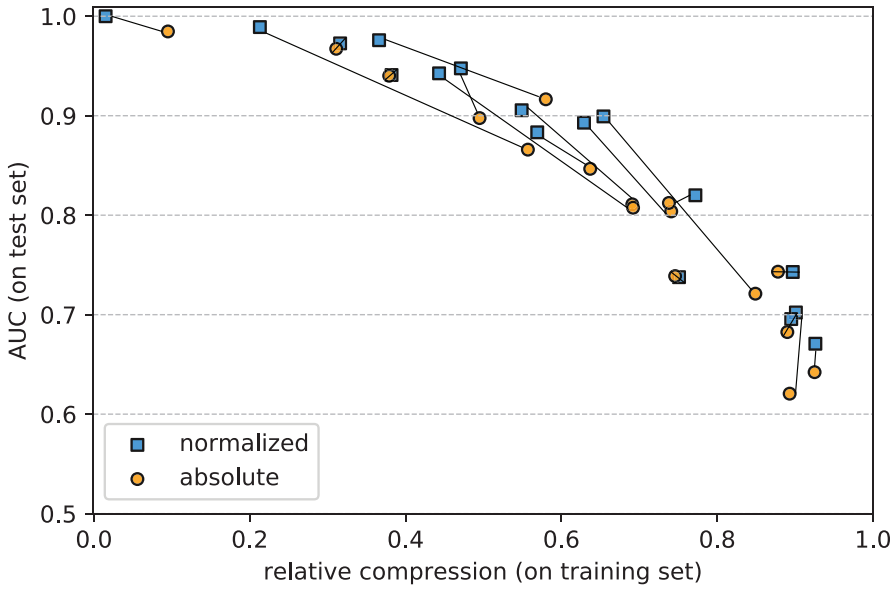


Fig. 2. Relation between compression and AUC; better compression on the training set (lower relative compression) corresponds to better classification on the test set (higher AUC). Obtained with Classy using normalized (squares) and absolute (circles) gain, on all 17 datasets; each point represents the 10 times repeated 10-fold average for one dataset with one type of gain; each connected pair represents the same dataset, for the two types of gain.

which tells us that the time complexity of Classy grows proportionally to the dataset size times the candidate set size, thus, given a fixed dataset size it becomes proportional only to the candidate set size.

Fig. 3c and d show how Classy performs in classification in terms of compression in the training set and AUC in the test set, respectively. The values for both plots remain constant for most cases, and when a value deteriorates in terms of compression (increase in compression ratio) for smaller candidate sets, it also deteriorates accordingly in terms of AUC (decrease in AUC) in the test set. We make two important observations: 1) the minimum in compression is achieved at the minimum support used for 13 out of 17 datasets, and in the cases where it does not happen the difference in relative compression is below 1%, which tells us that Classy can find a good description of the data using large candidate sets, without too greedily using rules that only cover few instances; 2) the minimum in compression and maximum in AUC are achieved for the same support value for 12 out 17 cases, and in the other cases, the difference is usually smaller than 2% in both measures, revealing the robustness of our MDL formulation at obtaining models that generalize well. The main exception is *ionosphere*, where the best AUC is found at the minimum support threshold of 10%, while the lowest threshold finds a PRL with 3% lower AUC, without almost any change in compression. This can be explained by the relatively small number of examples of *ionosphere* (351 instances) combined with its peculiar structure.

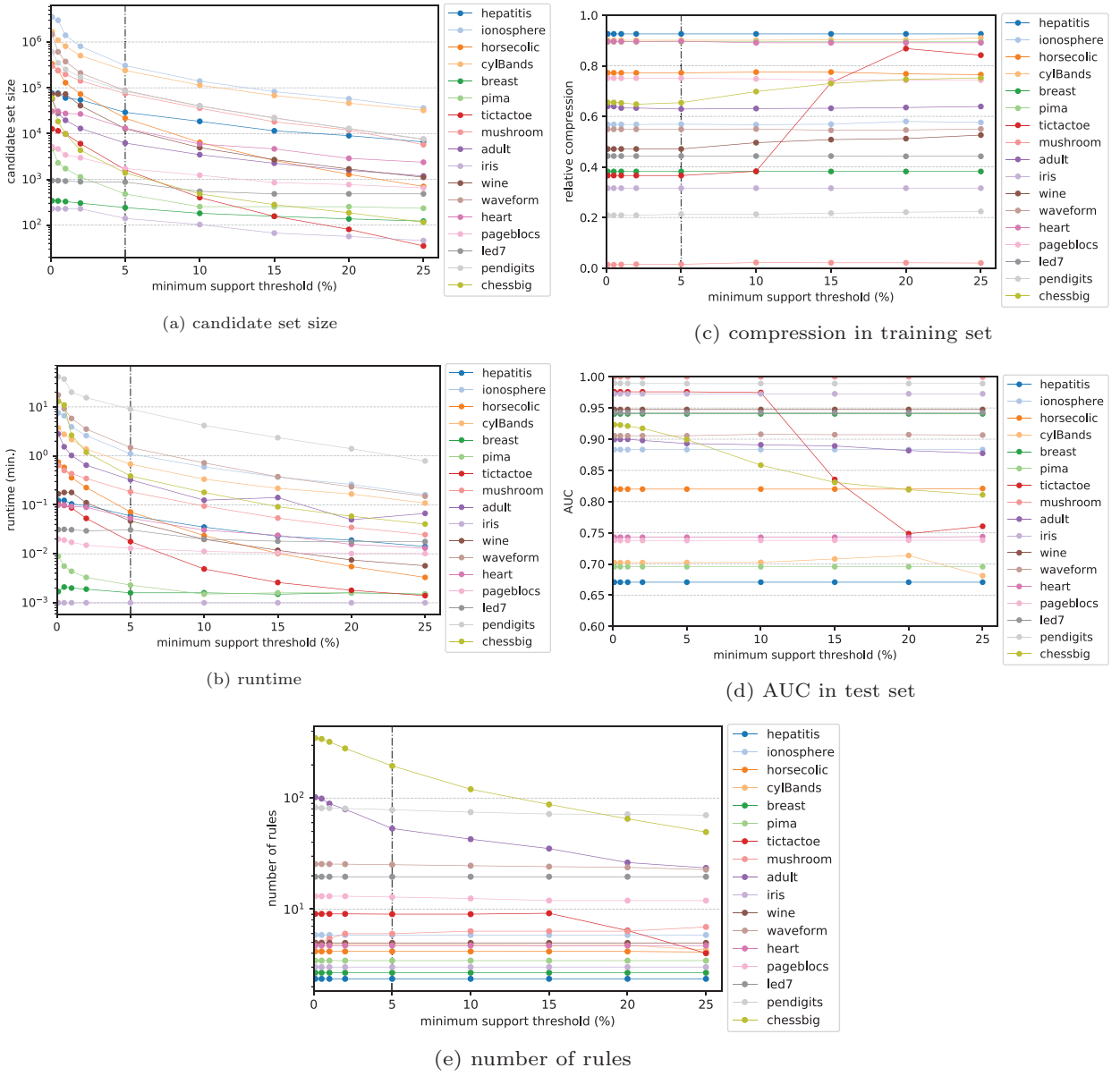
Fig. 3e shows the number of rules selected based on the candidate set. As expected the number of rules selected only decreases or remains constant with the candidate set, except for *mushroom*. Upon closer inspection, we observe that this is due to the disappearance of a rule with good performance but low coverage from the candidate set, which has to be replaced by a combination of other rules. The cases where many more rules are selected for lower minimum support thresholds, such as *chessbig* and *adult*, have lower compression and higher AUC values for these large number of rules, which makes these selections sustainable.

6.3. Classification performance

We now compare the classification performance of Classy to Scalable Bayesian Rule Lists (SBRL) [46], JRip[5], FURIA[5], CART[6], C5.0[7], and Support Vector Machines[8] (SVM). These methods are state-of-the-art classifiers, and SBRL, CART, C5.0, JRip, and FURIA—a fuzzy unordered rule induction algorithm—are clearly related to our approach. C5.0 is a newer version of C4.5, and JRip is a Java-implementation of RIPPER.

Classy has no parameters apart from the candidate set, which was generated using FP-growth with $m_s = 5\%$ and $l_{max} = 4$ for each dataset (as described at the beginning of Section 6). We tuned CART by selecting the best performing model on the training set from the models generated with the following complexity parameters: {0.001; 0.003; 0.01; 0.03; 0.1}. The

[5] https://cran.r-project.org/package=RWeka.
[6] https://cran.r-project.org/package=rpart.
[7] https://cran.r-project.org/package=C50.
[8] https://cran.r-project.org/package=e1071.



(a) candidate set size

(c) compression in training set

(b) runtime

(d) AUC in test set

(e) number of rules

Fig. 3. Influence of the minimum support threshold on {candidate set size; runtime (in minutes); relative compression on the training set; AUC in the test set; number of rules} for a maximum rule length of 4 and a minimum support threshold per class of $m_s = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$. The values were averaged over 10 times repeated 10-fold crossvalidation and each dataset is connected by a line to aid visualization. The vertical dashed line represents the selected minimum support of 5% used to compare against the other algorithms.

same was done for C5.0, with confidence factors: {0.05; 0.15; 0.25; 0.35; 0.45}. The SVM, with radial kernel, was tuned using 3-fold cross-validation and a grid search on $\gamma = \{2^{-6:0}\}$ and $c = \{2^{-4:4}\}$ within the training set. JRip and FURIA were tuned by setting their hyperparameters to 3 folds, a minimum weight of 2, and 2 optimization runs.

SBRL was trained using the guidelines provided by the authors [46]: the number of chains was set to 25; iterations to 5000; η, representing the average size of patterns in a rule, to 1; and λ, representing the average number of rules, to 5. The algorithm was first run on the training set and then re-run with λ changed to the number of rules obtained. In an attempt to follow their guidelines to use around 300 candidate rules, minimum and maximum itemset length were set to 1 and 2 (or 3 if possible) respectively, while the minimum threshold for the negative and positive classes was set to one of {5%, 10%, 15%}. Note that we initially attempted a fair comparison by using the same candidates for SBRL as for CLASSY, but due to the limitations on the number of rules that SBRL could practically handle this unfortunately turned out to be infeasible.

The results are presented in Table 4. The SVM models achieves the best ranking overall, but they do not belong to the class of interpretable models.

Table 4
Classification performance average results (10 times repeated 10-fold crossvalidation) measured through {accuracy; balanced accuracy; Area Under the ROC Curve (AUC) (weighted AUC for multiclass datasets)}, per dataset {binary;multiclass} for each algorithm. At the bottom of the binary and multiclass datasets the average rank of the algorithm for each is presented. The rank of 1 is given for the best value (highest classification performance) and 7 and 6 for the worst in binary and multiclass, respectively. Note that SBRL cannot do multiclass classification, hence only being present in the binary ranking and the blank spaces.

	Accuracy							Balanced accuracy							AUC						
datasets	Classy	SBRL	JRip	CART	C5.0	FURIA	SVM	Classy	BRL	JRip	CART	C5.0	FURIA	SVM	Classy	BRL	JRip	CART	C5.0	FURIA	SVM
hepatitis	0.83	0.78	0.79	0.79	0.79	0.79	0.83	0.66	0.59	0.65	0.67	0.65	0.61	0.70	0.67	0.62	0.64	0.72	0.68	0.70	0.85
ionosphere	0.89	0.88	0.90	0.91	0.90	0.90	0.92	0.87	0.86	0.89	0.89	0.89	0.88	0.91	0.88	0.88	0.89	0.92	0.92	0.91	0.96
horsecolic	0.80	0.84	0.81	0.83	0.83	0.83	0.84	0.78	0.82	0.80	0.81	0.82	0.81	0.82	0.82	0.83	0.81	0.85	0.85	0.84	0.88
cylBands	0.70	0.68	0.73	0.73	0.74	0.76	0.81	0.65	0.65	0.72	0.71	0.73	0.74	0.80	0.70	0.73	0.74	0.78	0.78	0.79	0.88
breast	0.93	0.94	0.93	0.93	0.94	0.94	0.94	0.94	0.95	0.94	0.94	0.95	0.95	0.95	0.94	0.95	0.96	0.95	0.95	0.95	0.96
pima	0.73	0.74	0.73	0.73	0.73	0.74	0.74	0.66	0.69	0.68	0.67	0.67	0.68	0.67	0.70	0.69	0.68	0.71	0.69	0.68	0.75
tictactoe	0.98	0.81	0.98	0.92	0.94	0.99	0.99	0.98	0.75	0.97	0.91	0.93	0.98	0.99	0.98	0.86	0.97	0.97	0.98	1.00	1.00
mushroom	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
adult	0.85	0.85	0.85	0.85	0.85	0.80	0.86	0.75	0.75	0.74	0.75	0.76	0.74	0.76	0.89	0.88	0.74	0.88	0.87	0.76	0.86
rank	4.4	4.6	4.8	4.9	4.1	3.3	1.9	4.9	4.3	4.4	4.7	3.6	3.8	2.3	4.6	4.8	5.4	3.9	3.8	3.8	1.7
iris	0.95		0.94	0.93	0.93	0.93	0.94	0.95		0.94	0.93	0.93	0.93	0.94	0.97		0.97	0.97	0.97	0.95	0.99
wine	0.89		0.88	0.87	0.89	0.93	0.95	0.90		0.89	0.87	0.90	0.92	0.95	0.95		0.93	0.93	0.95	0.96	1.00
waveform	0.75		0.77	0.76	0.76	0.78	0.80	0.75		0.77	0.76	0.76	0.78	0.80	0.91		0.87	0.90	0.90	0.86	0.94
heart	0.56		0.54	0.57	0.54	0.57	0.59	0.30		0.22	0.31	0.30	0.27	0.30	0.74		0.54	0.76	0.72	0.69	0.84
pageblocs	0.93		0.93	0.93	0.92	0.92	0.93	0.51		0.52	0.52	0.48	0.52	0.52	0.74		0.72	0.74	0.69	0.73	0.72
led7	0.74		0.72	0.75	0.75	0.74	0.76	0.74		0.72	0.75	0.75	0.74	0.76	0.94		0.92	0.94	0.94	0.88	0.95
pendigits	0.92		0.95	0.92	0.96	0.97	0.98	0.92		0.95	0.92	0.96	0.97	0.99	0.99		0.98	0.99	1.00	0.99	1.00
chessbig	0.50		0.52	0.49	0.78	0.62	0.93	0.44		0.61	0.41	0.81	0.67	0.92	0.90		0.81	0.87	0.96	0.67	1.00
rank	4.0		3.4	3.2	2.6	1.6	1.1	4.1		4.4	4.1	4.1	3.2	1.3	2.6		5.1	3.8	3.4	4.6	1.5

Classy performs on par with most tree- and rule-based models in terms of accuracy and balanced accuracy, worst than FURIA for these two measures, and better than these in terms of AUCs for multiclass datasets. The better performance of FURIA can be explained by the fact that it uses fuzzy rule sets rather than probabilistic rule lists; this allows for multiple rules to be activated and aggregated for a single classification, which improves predictive performance but makes interpretability less straightforward. This also means that the number of rules and conditions cannot be directly compared: a FURIA rule set consisting of 5 rules actually translates to up to 32 unique rules in the rule list setting. Further, FURIA does not provide probabilistic predictions, unlike our approach.

Comparing to other rule list models, such as SBRL and JRip, Classy performs better for most of the measures used. When viewed against the tree-based models, we can see that our method performs on par with CART for most measures and slightly worse than C5.0, except for AUC in the multiclass scenario. Also, as we will show later, C5.0 tends to obtain equivalent rule lists that are much bigger than the ones produced by Classy, which makes them perform better in general (but not always).

6.4. Interpretability

The results are shown in Table 5, where we use AUC, the number of rules, and the number of conditions to compare the trade-off between AUC and model complexity of the tree- and rule-based models. Note that we choose AUC for predictive performance as it agrees with our goal of using the probabilities output of Classy to explain the decisions made. Also note that we intentionally removed FURIA from the rankings of the number of rules and conditions as its models are rule sets—not rule lists.

For binary datasets Classy is in a middle ranking, better than SBRL and JRip, and worst than CART, C5.0 nd FURIA. On the other hand, in multiclass datasets it achieves a much lower (=better) ranking than all the other algorithms.

Classy tends to find more compact models, with similar number of rules and fewer logical conditions in total, than C5.0, CART, and JRip, that are as accurate or better than these. This can be clearly seen by its average rank of 2 and 1.9 for rules and 1.7 and 1.5 for the total number of conditions, for binary and multiclass datasets respectively. It also can be seen that for most datasets it obtained the lowest number of conditions of all tree- and rule-based classifiers. Although SBRL also finds very compact rule lists, with small number of rules and conditions, the low variance between the reported values for the different datasets suggests that this strongly depends on the hyperparameter settings, which penalize too strongly the number of rules not around the user defined expected average number of rules. Indeed, the compact rule lists exhibit subpar classification performance for some datasets (i.e., *hepatitis* and *tictactoe*). This suggests that without additional (computation-intensive) tuning of these hyperparameters, the recommended procedure for SBRL may lead to underfitting. As expected, C5.0, with its tendency to maximize the classification performance as much as possible, tends to create overgrown models, such as the almost 3000 rules for *chessbig*, that do not necessarily generalize well, such is the case in *adult*, where it obtained the same number of rules as Classy but with a 2% lower AUC, and for *pendigits* were it obtained a number of rules around 4 times higher than Classy and CART for the same performance.

6.5. Statistical significance testing

To analyze whether the results Tables 4 and 5 are statistically different [11], we use two non-parametric multiple hypothesis tests, namely Friedman's test [14] and Iman and Davenport's test [22], on the rankings of the algorithms.

The results can be seen in the left side of Table 6, which divides the datasets into two groups, for binary and multiclass datasets respectively. The results show that there are significant differences for most measures (significance level 0.05). The only exceptions are balanced accuracy in the binary case, AUC of rule-based models in the binary case, and the number of rules for the multi-class case.

For those cases where the null hypothesis—stating that the algorithms perform on par—is rejected we proceed with a post-hoc Holm's test [19] for pairwise comparisons with Classy as control algorithm.

The results of these pairwise comparisons can be seen in the right side of Table 6. For most of these tests the null hypothesis—stating that Classy and its competitor perform on par—cannot be rejected. This can be mostly explained by the relatively small number of datasets; the power of the tests is not very high. We therefore cannot draw strong conclusions from these results, but this is not necessarily a negative outcome: we aimed at showing that Classy performs as well as other rule- and tree-based algorithms while obtaining simpler models. The results show that Classy does use significantly fewer conditions than C5.0 for both multiclass and binary datasets, and than CART for the binary case. Also, as expected the SVM obtained always better results than Classy except for multiclass AUC. FURIA was better in terms of accuracy but worse in terms of AUC.

6.6. Overfitting

To study overfitting, we compared the averages of the absolute difference between the AUC values in the training and test set over 10 times repeated 10-folds for each algorithm. The results can be seen in Table 7. In general Classy, together with SVM, seem to be the most consistent algorithms in obtaining the lowest values. The usual performance of Classy is 5% or lower, 12 out of 17 times, except in the case of *hepatitis* were it got 13%, which was the best value after SVM. SBRL is very

Table 5
Interpretability performance average results (10 times repeated 10-fold crossvalidation) of tree- and rule-based models measured through {Area Under the ROC Curve (AUC) (weighted AUC for multiclass datasets); number of rules; number of conditions }, per dataset {binary; multiclass} for each algorithm. Rank gives the average rank of each algorithm for binary and multiclass datasets. The rank of 1 is given for the best value (highest AUC or lowest number of rules/conditions). Note that SBRL cannot do multiclass classification, hence only being present in the binary ranking and the blank spaces. * The number of rules and conditions used by FURIA are presented as a reference, as they are not directly comparable to those of the other methods as they form rule sets (and cannot be trivially translated to rule lists). FURIA was therefore also not included in the rankings of those criteria.

	AUC						Number of rules						Number of conditions					
datasets	CLASSY	SBRL	JRip	CART	C5.0	FURIA	CLASSY	SBRL	JRip	CART	C5.0	FURIA	CLASSY	SBRL	JRip	CART	C5.0	FURIA
hepatitis	0.67	0.62	0.64	0.72	0.68	0.70	2	2	3	4	9	4*	1	2	5	6	38	7*
ionosphere	0.88	0.88	0.89	0.92	0.92	0.91	6	3	6	5	12	7*	4	4	9	10	58	12*
horsecolic	0.82	0.83	0.81	0.85	0.85	0.84	4	2	4	6	16	6*	3	2	7	10	76	9*
cylBands	0.70	0.73	0.74	0.78	0.78	0.79	5	3	7	20	59	11*	4	4	17	111	897	18*
breast	0.94	0.95	0.96	0.95	0.95	0.95	3	3	4	5	6	5*	3	5	11	13	15	13*
pima	0.70	0.69	0.68	0.71	0.69	0.68	3	2	3	10	11	2*	3	3	3	23	47	2*
tictactoe	0.98	0.86	0.97	0.97	0.98	1.00	9	6	10	24	42	10*	21	15	27	66	254	17*
mushroom	1.00	1.00	1.00	1.00	1.00	1.00	6	5	5	8	9	8*	7	8	7	25	35	17*
adult	0.89	0.88	0.74	0.88	0.87	0.76	53	13	17	22	52	21*	114	25	81	106	630	34*
rank	3.8	4.0	4.4	3.0	2.9	2.9	2.6	1.1	2.8	3.7	4.9	*	1.7	1.7	2.7	3.9	5.0	*
iris	0.97		0.97	0.97	0.97	0.95	3		3	3	3	2*	2		2	3	3	3*
wine	0.95		0.93	0.93	0.95	0.96	5		5	5	8	4*	4		7	6	24	6*
waveform	0.91		0.87	0.90	0.90	0.86	25		23	46	81	15*	65		108	125	683	33*
heart	0.74		0.54	0.76	0.72	0.69	5		3	11	39	2*	4		6	28	299	2*
pageblocs	0.74		0.72	0.74	0.69	0.73	13		8	10	9	3*	13		9	20	37	4*
led7	0.94		0.92	0.94	0.94	0.88	20		19	29	28	3*	46		74	43	138	6*
pendigits	0.99		0.98	0.99	1.00	0.99	78		107	66	266	74*	221		439	30	2934	133*
chessbig	0.90		0.81	0.87	0.96	0.67	195		418	118	2874	281*	483		2688	25	48826	803*
rank	1.8		4.3	2.9	2.4	3.8	2.3		2.0	2.2	3.5	*	1.5		2.4	2.1	4.0	*

Table 6
Statistical significance testing of differences between the algorithms. Results for Friedman (χ^2 statistic), and Iman and Davenport (F statistic) tests for all the measures presented in Tables 4 and 5 for binary and multiclass datasets with a significance level of 0.05. In case the null hypothesis for the differences is rejected, the post-hoc Holm's procedure is used for pairwise comparisons with CLASSY as control. AUC_{all} is the AUC comparison with all algorithms (SVM included) of Table 4 and AUC_{rules} is the AUC comparison of all tree- and rule-based algorithms (SVM excluded) of Table 5. p represents the p-value obtained for each specific test, **R** the rejection of $\mathcal{H}_0$—null hypothesis. Note that not all algorithms can be tested for all measures, thus k shows the number of algorithms tested for each measure.

			Difference tests						Holm's post-hoc procedure (CLASSY as control)											
			Friedman			Iman and Davenport			SBRL		JRip		CART		C5.0		FURIA		SVM	
Measures	Classes	k	χ^2	p	$\mathcal{H}_0$	F	p	$\mathcal{H}_0$	p	$\mathcal{H}_0$	p	$\mathcal{H}_0$	p	$\mathcal{H}_0$	p	$\mathcal{H}_0$	p	$\mathcal{H}_0$	p	$\mathcal{H}_0$
Acc	binary	7	13.36	0.038	**R**	2.63	0.028	**R**	0.827	–	0.703	–	0.585	–	0.743	–	0.300	–	0.014	**R**
	multi	6	17.34	0.004	**R**	5.36	<0.001	**R**			0.504	–	0.385	–	0.142	–	0.009	**R**	0.002	**R**
bAcc	binary	7	8.94	0.177	–	1.59	0.171	–												
	multi	6	15.71	0.008	**R**	4.53	0.003	**R**			0.738	–	1.000	–	1.000	–	0.350	–	0.003	**R**
AUC_{all}	binary	7	16.00	0.014	**R**	3.37	0.007	**R**	0.827	–	0.445	–	0.300	–	0.413	–	0.413	–	0.005	**R**
	multi	6	20.00	0.001	**R**	7.00	<0.001	**R**			0.008	**R**	0.229	–	0.423	–	0.033	–	0.229	–
AUC_{rules}	binary	6	5.70	0.337	–	1.16	0.346	–												
	multi	5	13.10	0.011	**R**	4.85	0.004	**R**			0.002	**R**	0.155	–	0.429	–	0.011	**R**		
Rules	binary	5	28.18	<0.001	**R**	28.82	<0.001	**R**	0.053	–	0.766	–	0.136	–	0.002	**R**				
	multi	4	6.64	0.084	–	2.68	0.073	–												
Conditions	binary	5	29.40	<0.001	**R**	35.64	<0.001	**R**	1.000	–	0.205	–	0.011	**R**	<0.001	**R**				
	multi	4	16.35	0.001	**R**	14.96	<0.001	**R**			0.175	–	0.333	–	<0.001	**R**				

Table 7
Overfitting average results (10 times repeated 10-fold crossvalidation) using the absolute difference between AUC performance in training and test sets as measure, per fold, for each algorithm and each dataset. Rank gives the average rank of each algorithm for binary and multiclass datasets. Note that SBRL does not have a values for multiclass datasets.

	$\lvert AUC_{train} - AUC_{test} \rvert$						
datasets	Classy	SBRL	JRip	CART	C5.0	FURIA	SVM
hepatitis	0.13	0.16	0.19	0.14	0.21	0.22	0.13
ionosphere	0.06	0.05	0.07	0.04	0.05	0.08	0.04
horsecolic	0.06	0.06	0.08	0.06	0.09	0.08	0.10
cylBands	0.07	0.06	0.09	0.11	0.18	0.13	0.12
breast	0.02	0.02	0.02	0.02	0.02	0.02	0.02
pima	0.06	0.05	0.05	0.06	0.07	0.05	0.05
tictactoe	0.01	0.03	0.01	0.02	0.02	0.00	0.00
mushroom	0.00	0.00	0.00	0.00	0.00	0.00	0.00
adult	0.01	0.00	0.01	0.00	0.01	0.01	0.03
rank	3.6	2.7	4.4	4.2	5.2	4.3	3.6
iris	0.02		0.02	0.02	0.02	0.04	0.01
wine	0.03		0.05	0.04	0.04	0.03	0.00
waveform	0.02		0.02	0.02	0.03	0.02	0.02
heart	0.05		0.04	0.07	0.15	0.04	0.06
pageblocs	0.00		0.00	0.00	0.00	0.00	0.00
led7	0.01		0.01	0.01	0.01	0.01	0.01
pendigits	0.00		0.01	0.00	0.00	0.01	0.00
chessbig	0.00		0.02	0.00	0.02	0.00	0.00
rank	2.4		4.8	4.4	4.1	3.6	1.8

consistent, clearly achieving the lowest values for binary datasets, however this can be explained by its more conservative choice of rules and thus lower AUC on the test set as shown in Table 4. Comparing with all rule- and tree-based models, Classy obtained the lowest ranking for multiclass datasets, being, from these ones, the algorithm that less overfits overall.

6.7. Runtime

All runtimes are averages over ten times repetitions of ten folds, run on a 64-bit Windows Server 2012*R*2, with Intel Xeon E5-2630v3 CPU at 2.4GHz and 512GB RAM. Runtimes include parameter tuning where applicable, and candidate mining for Classy and SBRL.

The results are depicted in Fig. 4. CART, C5.0, JRip and FURIA are the fastest, with most runtimes under 1 min with Classy being at maximum one order of magnitude slower. Comparing to SBRL, Classy is 10 times faster, even though it considers around 100 times more candidates than this and performs better in terms of AUC. The worst runtimes were obtained for SVM, due to its costly grid search.

It should be noticed that reducing the candidate set size of Classy would have an exponential reduction in its runtimes without much deterioration of its classification performance, as can be seen in Fig. 3b and d.

6.8. Discussion

From the classification and interpretability results of Tables 4 and 5 it can be seen that Classy is able to provide a good trade-off between classification performance and rule list size. This is particularly the case for multiclass datasets such as *chessbig*, where classical algorithms like JRip find a model with double the number of rules, or in the case of *mushroom*, where CART and C5.0 find more complex models with the same performance. It is interesting to notice that Classy performs better in terms of AUC than accuracy. This shows that when it makes a wrong prediction it does so with a small probability, which is reassuring. Moreover, Classy has only one hyperparameter — its candidate set — which its tuning is hardly needed as the algorithm has no problem in dealing with large numbers of candidates. This is quite different from the extensive tuning done for the other methods. It is important to observe that *all methods except for* Classy *were tuned*.

More importantly, in the set of Fig. 3, it is shown that larger candidate sets do not result in worse models, as our formalization in terms of the MDL principle is well-suited to avoid overfitting without the need for cross-validation and/or parameter tuning. In other words, Classy is insensitive to its only parameter — its candidate set — making it virtually parameter-free. This is a big advantage, as one can simply run Classy on all training data with as many candidates as possible, without worrying about any parameters. It also means that all training data can be used for training, which is important in case of small data: no data needs to be reserved for validation.

From Fig. 2 we can observe that better compression corresponds to better classification, which is a strong empirical validation of our formalization. As expected, normalized gain is clearly the best heuristic to use in combination with our greedy rule selection strategy, as it results in better classifiers for 88% of the datasets.

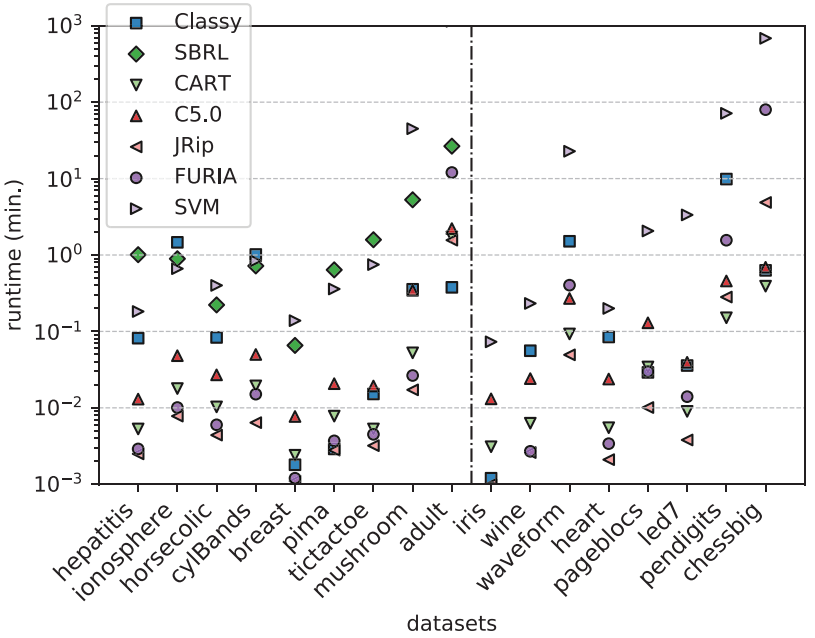


Fig. 4. Average runtime per fold in minutes for each algorithm and each dataset. The datasets are ordered first by the number classes and then by number of samples (ascending). The vertical dashed line separates binary (to the left) from multiclass datasets. Note that SBRL does not have a runtime for multiclass datasets.

From the runtimes of Fig. 4, it can be seen that Classy runtimes are slower by an order of magnitude than other (fast) algorithms, such as C5.0, CART, and JRip, and similar to SBRL. This is expected for the size of candidate sets used in our experiments, as can be seen in Table 3.

In terms of classification and interpretability, comparing the average ranking with other rule- and tree-based methods in Table 5, it is shown that Classy performs equally well while also able to find rule lists with less conditions, *without any parameter tunning*. CART creates models with fewer rules that have more conditions per rule, while C5.0 has a high AUC at the expense of over-complex rules. FURIA has a better performance in terms of both standard and balanced accuracy, and worst in terms of AUC, which is expected as it is not a probabilistic classifier. Also, it is hard to compare its interpretability as all its rules can interact with each other, generating a much larger equivalent rule list than Classy. SBRL on the other hand seems to be able to find simple models that underperform in terms of AUC compared with Classy, which can be either a result of its formalization or because it cannot use larger candidate sets. The experiments also revealed that the Poisson distribution used as prior in SBRL, for the number of conditions per rule and the number of rules, creates tight constraints from which the results hardly deviate. Our results suggest that if the 'optimal' values for these hyperparameters are not known in advance, the best model may not be found. An indicative example of this is the *tictactoe* dataset in Table 4, a deterministic dataset for which SBRL can only find the right amount of rules and logical conditions per rule when given these exact values in advance. The results obtained with Classy demonstrate that using the universal prior for integers alleviates this strong dependence on hyperparameter tuning.

In terms of overfitting, Table 7 shows that Classy has a tendency to select models that generalize well and that are not overconfident in the training set. It obtains low differences between training and test compared with the other rule- and tree-based models.

7. Conclusions and future work

We proposed a novel formalization of the multiclass classification problem using probabilistic rule lists and the minimum description length (MDL) principle. Our problem formulation allows for parameter-free model selection and naturally trades off model complexity with predictive accuracy, effectively avoiding overfitting. To find solutions to this problem, we introduced the heuristic Classy algorithm, which greedily constructs rule lists using the MDL-based criterion.

We empirically demonstrated, on a variety of datasets, that Classy finds probabilistic rule lists that perform on par with state-of-the-art interpretable classifiers with respect to predictive accuracy, despite the fact that some form of hyperparameter tuning is done for all methods except for Classy. Moreover, the models found by our approach were shown to be more compact than those obtained by the other methods, which is expected to make them more understandable in practice. Finally, compression was shown to strongly correlate with predictive accuracy, which can be regarded as an empirical validation of the MDL-based selection criterion.

Directions for future work include, for instance, the following:

- Bridge the gap between both kinds of search methods used to learn rule lists from data in a principled way, namely optimal strategies — accurate but slow — and greedy methods — fast but imperfect.
- Extend our MDL formulation to other types of data and/or tasks, such as continuous data and regression problems.

Declaration of Competing Interest

None.

Acknowledgment

This work is part of the research programme Indo-Dutch Joint Research Programme for ICT 2014 with project number 629.002.201, SAPPAO, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

References

[1] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: ACM sigmod record, vol. 22, ACM, 1993, pp. 207–216.
[2] J. Alcala-Fdez, R. Alcala, F. Herrera, A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning, IEEE Trans. Fuzzy Syst. 19 (5) (2011) 857–872.
[3] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, C. Rudin, Learning certifiably optimal rule lists, KDD'17, ACM, 2017.
[4] J.O.R. Aoga, T. Guns, S. Nijssen, P. Schaus, Finding probabilistic rule lists using the minimum description length principle, DS'18, 2018.
[5] E. Bellodi, F. Riguzzi, Structure learning of probabilistic logic programs by searching the clause space, Theory Pract. Logic Program. 15 (2) (2015) 169–212.
[6] C. Borgelt, Efficient implementations of Apriori and eclat, in: FIMI03: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.
[7] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and Regression Trees, CRC press, 1984.
[8] K.H. Brodersen, C.S. Ong, K.E. Stephan, J.M. Buhmann, The balanced accuracy and its posterior distribution, in: 2010 20th International Conference on Pattern Recognition, IEEE, 2010, pp. 3121–3124.
[9] K. Budhathoki, J. Vreeken, The difference and the norm – characterising similarities and differences between databases, in: ECMLPKDD'15, Springer, 2015, pp. 206–223.
[10] W.W. Cohen, Fast effective rule induction, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 115–123.
[11] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.
[12] F. Doshi-Velez, B. Kim, Towards a rigorous science of interpretable machine learning, arXiv:1702.08608 (2017).
[13] A. Fernandez, V. Lopez, M.J. del Jesus, F. Herrera, Revisiting evolutionary fuzzy systems: Taxonomy, applications, new trends and challenges, Knowl.-Based Syst. 80 (2015) 109–121.
[14] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Am. Stat. Assoc. 32 (200) (1937) 675–701.
[15] J. Fürnkranz, D. Gamberger, N. Lavrač, Foundations of Rule Learning, Springer Science & Business Media, 2012.
[16] M. García-Borroto, J.F. Martínez-Trinidad, J.A. Carrasco-Ochoa, A survey of emerging patterns for supervised classification, Artif. Intell. Rev. 42 (4) (2014) 705–721.
[17] A. Gelman, H.S. Stern, J.B. Carlin, D.B. Dunson, A. Vehtari, D.B. Rubin, Bayesian Data Analysis, Chapman and Hall/CRC, 2013.
[18] P.D. Grünwald, The Minimum Description Length Principle, MIT press, 2007.
[19] S. Holm, A simple sequentially rejective multiple test procedure, Scand. J. Stat. (1979) 65–70.
[20] J. Hühn, E. Hüllermeier, Furia: an algorithm for unordered fuzzy rule induction, Data Min. Knowl. Discovery 19 (3) (2009) 293–319.
[21] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, B. Baesens, An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models, Decis. Support Syst. 51 (1) (2011) 141–154.
[22] R.L. Iman, J.M. Davenport, Approximations of the critical region of the fbietkan statistic, Commun. Stat. 9 (6) (1980) 571–595.
[23] F. Jiménez, G. Sánchez, J.M. Juárez, Multi-objective evolutionary algorithms for fuzzy classification in survival prediction, Artif. Intell. Med. 60 (3) (2014) 197–219.
[24] P. Kralj Novak, N. Lavrač, G. Webb, Supervised descriptive rule discovery: a unifying survey of contrast set, emerging pattern and subgroup mining, J. Mach. Learn. Res. 10 (2009) 377–403.
[25] H. Lakkaraju, S.H. Bach, J. Leskovec, Interpretable decision sets: a joint framework for description and prediction, KDD'16, ACM, 2016.
[26] H. Lakkaraju, C. Rudin, Learning cost-effective and interpretable treatment regimes for judicial bail decisions, NIPS 2016, 2016.
[27] H. Lakkaraju, C. Rudin, Learning cost-effective and interpretable treatment regimes, Artificial Intelligence and Statistics, 2017.
[28] M. van Leeuwen, E. Galbrun, Association discovery in two-view data, IEEE Trans. Knowl. Data Eng. 27 (12) (2015).
[29] M. van Leeuwen, J. Vreeken, Mining and using sets of patterns through compression, in: Frequent Pattern Mining, Springer, 2014, pp. 165–198.
[30] B. Letham, C. Rudin, T.H. McCormick, D. Madigan, et al., Interpretable classifiers using rules and Bayesian analysis: building a better stroke prediction model, Ann. Appl. Stat. 9 (3) (2015) 1350–1371.
[31] W. Li, J. Han, J. Pei, CMAR: accurate and efficient classification based on multiple class-association rules, in: Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, IEEE, 2001, pp. 369–376.
[32] Y. Lou, R. Caruana, J. Gehrke, Intelligible models for classification and regression, in: KDD'12, ACM, 2012, pp. 150–158.
[33] B.L.W.H.Y. Ma, B. Liu, Integrating classification and association rule mining, KDD'98, 1998.
[34] C. Molnar, Interpretable machine learning, A Guide for Making Black Box Models Explainable, 2018.
[35] I. Polaka, E. Gašenko, O. Barash, H. Haick, M. Leja, Constructing interpretable classifiers to diagnose gastric cancer based on breath tests, Procedia Comput. Sci. 104 (2017).
[36] F. Provost, P. Domingos, Well-trained pets: Improving probability estimation trees (2000).
[37] J.R. Quinlan, C4. 5: Programs for Machine Learning, Elsevier, 2014.
[38] M.T. Ribeiro, S. Singh, C. Guestrin, Why should i trust you?: Explaining the predictions of any classifier, in: KDD'16, ACM, 2016, pp. 1135–1144.
[39] M.T. Ribeiro, S. Singh, C. Guestrin, Anchors: high-precision model-agnostic explanations, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI), 2018.
[40] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978).
[41] J. Rissanen, A universal prior for integers and estimation by minimum description length, Ann. Stat. (1983) 416–431.
[42] J. Vreeken, M. van Leeuwen, A. Siebes, Krimp: mining itemsets that compress, Data Min. Knowl. Discovery 23 (1) (2011) 169–214.
[43] J. Wang, G. Karypis, Harmony: efficiently mining the best rules for classification, in: Proceedings of the 2005 SIAM International Conference on Data Mining, SIAM, 2005, pp. 205–216.

[44] T. Wang, C. Rudin, F. Velez-Doshi, Y. Liu, E. Klampfl, P. MacNeille, Bayesian rule sets for interpretable classification, in: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, 2016, pp. 1269–1274.
[45] G.I. Webb, Discovering significant patterns, Mach. Learn. 68 (1) (2007) 1–33.
[46] H. Yang, C. Rudin, M. Seltzer, Scalable Bayesian rule lists, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 3921–3930.
[47] J. Zeng, B. Ustun, C. Rudin, Interpretable classification models for recidivism prediction, J. R. Stat. Soc. 180 (3) (2017).
[48] X. Zhang, G. Dong, K. Ramamohanarao, Information-based classification by aggregating emerging patterns, in: IDEAL, Springer, 2000, pp. 48–53.
[49] A. Zimmermann, S. Nijssen, Supervised pattern mining and applications to classification, Frequent Pattern Mining, Springer, 2014.

Hugo M. Proença is a PhD candidate at the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University. His research project is part of a joint Dutch-Indian project in collaboration with GE India Technology Center in Bangalore and IIT Roorkee, which aims at optimizing the accuracy and reliability of predicting scheduled flight times. His research interests include interpretable machine learning, the Minimum Description Length (MDL) principle, and pattern mining.

Dr. Matthijs van Leeuwen is assistant professor and group leader of the Explanatory Data Analysis group at LIACS, Leiden University, the Netherlands. His primary research interest is exploratory data mining: how can we enable domain experts to explore and analyse their data, to discover structure and ultimately novel knowledge? Van Leeuwen was awarded several grants and best paper awards, co-organised international conferences and workshops, and is on the editorial board of DAMI and the guest editorial board of the ECML PKDD Journal Track. He was guest editor of a TKDD special issue on Interactive Data Exploration and Analytics.