Interpretable multiclass classification by MDL-based rule lists

Hugo M. Proença[a,*], *Matthijs van Leeuwen*[a]

[a]*LIACS, Leiden University, The Netherlands*

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.

Keywords: Rule lists; Minimum Description Length principle; Interpretable

[*]Corresponding author
Email addresses: `h.manuel.proenca@liacs.leidenuniv.nl` (Hugo M. Proença), `m.van.leeuwen@liacs.leidenuniv.nl` (Matthijs van Leeuwen)

models; Classification

1. Introduction

Interpretable machine learning has recently witnessed a strong increase in attention [1], 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 [2, 3] and societal problems [4, 5].
While it is of interest to investigate how existing 'black-box' machine learning models can be made transparent [6], 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 [7], generalized additive models [8], decisions sets [9, 10], and rule lists [2, 22]. Restricting our focus to classification, we make two important observations. First, we observe that state-of-the-art algorithms [9, 10, 2, 22, 11] 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 [11] or Bayesian rule lists [22] for binary classification. Our second observation is that although recent methods based on rules [2, 22] and decision sets [9, 10] 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 [2]. 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 [12]. Recent optimization [9] and Bayesian [22] 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 [13, 14], which has been successfully used to select small sets of patterns that summarize the data in the context of exploratory data mining [15, 16, 17]. 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 [15], 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 before in pattern-based modelling [17]. 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 [15], 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) [22] when it comes to the combination of classification performance and interpretability, in particular when taking into account that it has much fewer hyperparameters.

$$
\begin{array}{rl}
\textsc{If}\ \{\mathit{backbone} = \mathit{no}\}\ \textsc{then} & \Pr(\mathit{invertebr.}) = 0.55 \\
 & \Pr(\mathit{bug}) = 0.45 \\
\textsc{else if}\ \{\mathit{breathes} = \mathit{no}\}\ \textsc{then} & \Pr(\mathit{fish}) = 0.93 \\
 & \Pr(\mathit{reptile}) = 0.07 \\
\textsc{else if}\ \{\mathit{feathers} = \mathit{yes}\}\ \textsc{then} & \Pr(\mathit{bird}) = 1.00 \\
\textsc{else if}\ \{\mathit{milk} = \mathit{no}\}\ \textsc{then} & \Pr(\mathit{reptile}) = 0.50 \\
 & \Pr(\mathit{amphibian}) = 0.50 \\
\textsc{else}\quad \textsc{then} & \Pr(\mathit{mammal}) = 1.00
\end{array}
$$

Figure 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 and 101 examples.

To illustrate this, Figure 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 Classy, the proposed algorithm, to 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 to interpretable machine learning, please refer to the work of Molnar [18].
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 [19], C4.5 [20] and RIPPER [21], lack a global optimisation criterion and thus rely on heuristics and hyperparameters to deal with overfitting. Recent Bayesian methods [10, 9, 10] are limited to small numbers of candidate rules and binary classification, limiting their usability, and are here represented by SBRL [22] (which is representative for all of them). A recent approach also using MDL and probabilistic rule lists (MRL) [23] is aimed at describing rather than classifying and cannot deal with multiclass problems or a large number of candidates. Interpretable decision sets (IDS) [9] and certifiable optimal rules (CORELS) [11] use similar rules but do not provide probabilistic models or predictions.

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

Method	Multiclass	Probabilistic	Criterion	$\gg$1K cand	No tuning
Classy	✔	✔	✔	✔	✔
IDS[9]	✔	-	✔	✔	-
CORELS [11]	-	-	✔	-	-
MRL[23]	-	✔	✔	-	✔
SBRL[22]	-	✔	✔	-	-
Others	✔	✔	-	✔	-

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.

Note that methods that explain black-box models [6, 24], 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 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 [21]. Similarly, decision trees, which can easily be transformed to rule lists, have been used extensively; CART [19] and C4.5 [20] 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 also been extensively studied in the context of classification [25] and even though they offer some level of interpretability, their functionality is limited to the continuous domain.

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 [26], greedy optimization of submodular problem formulation or simulated annealing in the case of decision sets [9, 10], by Monte-Carlo search for Bayesian rule lists [2, 22], and through branch-and-bound with tight bounds for decision lists [11]. 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 [2, 22] and Bayesian decision sets [10] provide probabilistic predictions.
All previously mentioned algorithms share some similarities with Classy. In particular Bayesian rule lists [2, 22] 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 [27]—and therefore to different results; we will empirically compare the two approaches. Certifiable optimal rules [11] have a similar rule structure but do not provide probabilistic models or predictions. Decision sets [9] share the use of rules, but they are not used in the form of ordered rule lists.

2.2. Pattern mining

Association rule mining [28], 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 [29] and CMAR [30], but they tend to lack interpretability because they use large numbers of rules. Ensembles of association rules, such as Harmony [31] or classifiers based on emergent patterns [32], can increase classification performance when compared to the previous methods, however they can only offer local interpretations. Subgroup discovery and similar approaches [33] 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

[34] do not focus on finding global models either.
A last class of related methods is that of *supervised pattern set mining* [35]. 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 [15, 17], and two-view data [36].
In prediction it has been previously used to deal with overfitting [21, 20] and in the selection of the best compressing pattern [37].
RIPPER and C4.5 [21, 20] 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 [15], it was not designed for this: it outputs large pattern sets, one for each class, and does not give probabilistic predictions. DiffNorm [17] 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 [23], 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. 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 $\mathcal{Y} = \{1, \dots, |\mathcal{Y}|\}$ 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* $r_1, \dots, r_k$ 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 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 $\boldsymbol{\theta}_i = (\theta_i^{c_1}, \dots, \theta_i^{c_{|\mathcal{Y}|}})$, 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

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

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:

$$\begin{aligned} \text{rule 1 } &: \text{IF } a_1 \sqsubseteq \mathbf{x} \text{ THEN } y \sim Categorical(\boldsymbol{\theta}_1) \\ \text{rule 2 } &: \text{ELSE IF } a_2 \sqsubseteq \mathbf{x} \text{ THEN } y \sim Categorical(\boldsymbol{\theta}_2) \\ \text{default } &: \text{ELSE } y \sim Categorical(\boldsymbol{\theta}_\varnothing) \end{aligned} \tag{2}$$

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 $a_i, \forall_{i \in 1,\dots,|R|}$ 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 $i \in \{1, \dots, |R|, \varnothing\}$, 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:

$$\text{usage}(a_i \mid R, D) = |\{\mathbf{x} \subset D \mid a_i \sqsubseteq \mathbf{x} \wedge \left(\bigwedge_{\forall j < i} a_j \not\sqsubseteq \mathbf{x} \right) \}|$$

For ease of presentation, we abbreviate 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 [38].

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) [13, 14] 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 [14].

In contrast to existing pattern-based modeling approaches (e.g., [15, 16]), 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 $L(Y \mid X, R)$ 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. Equation (6) represents a trade-off between how well the model fits the data, $L(Y \mid X, R)$ 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 [36], 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.

4.1. Model encoding

Following the rule of parsimony associated with the MDL principle [14], the model encoding should result in larger code lengths for more complex mod-

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

els. To accomplish this we use only two types of codes for the different model components, the universal code for integers and the uniform code.

The universal code for integers [27], 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 [22]: 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., [15, 17]), which are aimed at summarization, we do not use normalized supports for our codes, i.e. codes are based on the support of all patterns of size 1 (singletons), as e.g. $a = [x_2 = 1]$, and normalized by the sum of all their supports. The reason for our use of the uniform code is that, in classification, higher support does not necessarily imply better predictive power. In general, without prior knowledge the uniform code represents the best, unbiased choice [14].

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 [14]. 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 Subsection 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 i^{th} class label, $Y_{i-1} = \{y_1, ..., y_{i-1}\}$ 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 [22], 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 sbusets. We define the part covered by a rule antecedent $a_i \in R$ as

$$D^{a_i} = \{X^{a_i}, Y^{a_i}\} = \{(\mathbf{x}, y) \in D \mid a_i \sqsubseteq \mathbf{x} \wedge \left(\bigwedge_{\forall j<i} a_j \not\sqsubseteq \mathbf{x} \right)\}.$$

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:

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

where $Y^{a_i}_j$ 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 and 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.

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	i^{th} 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 i^{th} rule for class c

Table 2: Table of commonly used notation.

5. The Classy algorithm

Given our model class—probabilistic rule lists—and its corresponding MDL formulation, what remains is to define an algorithm that given the training data finds the best model according to the MDL criteria that was defined. In this section we present such an algorithm: Classy; a greedy search based algorithm that iteratively finds the best rules to add to a rule list. This sections is divided as follows. First, a brief description of separate-and-conquer greedy search is done. Then compression gain, i.e, the measure that ranks the quality of the rules to add is described. After that, the Classy algorithm is defined. Then, it is explained how individual rules—candidates to the model—are generated from the data. Finally, the time and space complexity of Classy is investigated.

5.1. Separate-and-conquer greedy search

Greedy search is very commonly used for learning decision trees and rule lists [20, 21, 39], as well as for pattern-based modelling using the MDL principle [15, 17, 36]. A few recent approaches use optimization techniques [22], 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).

Given its ability to scale well, combined with its success when associated with MDL, 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 piors of the class labels in the data, adds rules according to the well-known *separate-and-conquer* strategy [39]: 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 *relative* 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*, $\Delta L(Y \mid X, R \oplus r)$, and *model gain*, $\Delta L(R \oplus r)$. Together this gives

$$\begin{aligned}\Delta L(D, R \oplus r) &= L(D, R) - L(D, R \oplus r) \\ &= \underbrace{L(Y \mid X, R) - L(Y \mid X, R \oplus r)}_{\Delta L(Y \mid X, R \oplus r)} \\ &+ \underbrace{L(R) - L(R \oplus r)}_{\Delta L(R)}.\end{aligned} \tag{11}$$

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

$$\begin{aligned}\Delta L(R \oplus r) =& L_{\mathbb{N}}(|R|) - L_{\mathbb{N}}(|R| + 1) \\ & - L_{\mathbb{N}}(|a|) - |a| \log |V|.\end{aligned} \tag{12}$$

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 Equation (10) we obtain

$$\Delta L(Y \mid X, R \oplus r) = \overbrace{\cancel{\sum_{a_i \in R} L(Y^{a_i} \mid X^{a_i}, R)} + L(Y^{\varnothing'} \mid X^{\varnothing'}, R \oplus r)}^{L(Y|X,R)} . \\ \underbrace{- \cancel{\sum_{a_i \in R} L(Y^{a_i} \mid X^{a_i}, R)} - L(Y^{\varnothing} \mid X^{\varnothing}, R) - L(Y^{a} \mid Y^{a}, R \oplus r)}_{L(Y|X,R \oplus r)} \tag{13}$$

Normalized 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:

$$\delta L(Y \mid X, R \oplus r) = \frac{\Delta L(Y \mid X, R \oplus r)}{U^a} \tag{14}$$

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 the probabilistic rules of the form: $r = (a, \boldsymbol{\theta})$; that are considered to be added to the rule list, given the dataset D. The candidates are generated by first mining a rule antecedent/pattern a using a standard frequent pattern mining algorithm, as for e.g. FP-growth [40], and then finding the correspondent consequent categorical distribution $\boldsymbol{\theta}$ given the dataset, i.e. using (5). In practice this mining algorithms have only two parameters, the minimum support 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. $L(a) < L(b)$. Shortly, at no point during model search will b be preferred over a, as the gain of a will always be greater.

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.

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:     $r \leftarrow \arg\max_{\forall r' \in Cands} : \delta L(D, R \oplus r')$
5:     $R \leftarrow R \oplus r$
6:     UpdateCandidates$(D, R, Cands)$
7: until $\delta L(D, R \oplus r') \leq 0, \forall r' \in Cands$
8: return $R$
```

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: an initialization step; and an iterative loop where one rule is added to a PRL at each iteration.

The time complexity of the initialization step is dominated by an ascending length sorting of the candidates 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. The sorting of 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 sets 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 2 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 scenario 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, thus obtaining 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 (14) of data encoding is not problematic when compared with the other terms. This is due to its recursive computation for $|D|$ values, that are then stored in a dictionary. This takes in total $\mathcal{O}(M(|D|) + |D|)$ time, with $M(*)$ as the multiplication complexity of the algorithm used, which in the case of python is the Katsuraba multiplication. From then on, the lookup of a value only takes $\mathcal{O}(1)$ time.

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 the 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 against a set of representative baselines in terms of classification performance, interpretability, overfitting and scalability.

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 two to 18 classes.

Candidates. Frequent pattern mining algorithms can generate different candidate sets by combining different values for the minimum support threshold per class m_s and maximum pattern length l_{max}. When comparing Classy other algorithms, we fix $l_{max} = 4$ and $m_s = 5\%$, as a trade-off between candidate size and running time of Classy. This value was selected to make an impartial decision based on two criteria: making each run below 10 minutes and to demonstrate that Classy can deal with large candidate sizes. To fix the candidate set generation parameters, first l_{max} was fixed, as a large value that should generate a large number of patterns, as can be seen in the Cands column of Table 3, and then m_s was selected accordingly to achieve the runtime imposed. Candidate patterns are mined from D using the frequent pattern algorithm FP-growth [40]. When testing the influence of the candidate set, the only "parameter" of Classy, we again fix $l_{max} = 4$, and vary the minimum support threshold per class from

[3]Code: `https://github.com/HMProenca/MDLRuleLists`

[4]`http://cgi.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html`

$m_s = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$.

The same candidate set was used for all experiments except when assessing its influence in Classy in Section 6.2.

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

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

$$L\% = \frac{L(D,R)}{L(D,\{\varnothing\})}, \tag{15}$$

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 the *Area Under the ROC Curve* (AUC), as we want to evaluate how well the probabilistic models separate the classes. For multiclass datasets we use *weighted AUC* [41], obtained by weighing per-class binary AUCs with the marginal class frequencies.

For interpretability we follow the most commonly used interpretation, i.e., that smaller models are easier to understand [1]. 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 scalability, wall clock time in minutes is measured; no parallelization

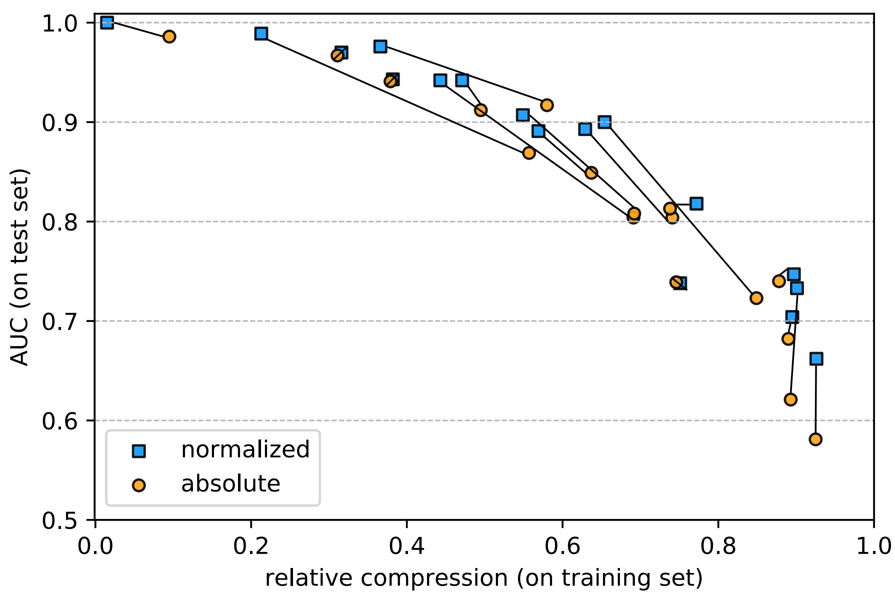


Figure 2: Relation between compression and AUC; better compression on the training set (lower relative compression) corresponds to better classification on the test set (higher AUC). With Classy using normalized and absolute gain, on all 17 datasets; each connected pair represents a dataset.

was used.

Note on standard deviations. Given the large amount of values reported either on figures or tables, and also that standard deviations are usually 2 or more orders of magnitude smaller than its corresponding averages, we feel that reporting them would lower the clarity and thus decided to exclude them from the analysis. For the few cases where its values are relevant, we will do so by explicitly commenting on them.

6.1. Compression vs classification

We first investigate the effect of using absolute (11) or normalized gain (14). To this end Figure 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

Dataset	$\lvert D\rvert$	$\lvert V\rvert$	$\lvert\mathcal{Y}\rvert$	$\lvert Cands\rvert$
adult	48842	96	2	9531
breast	699	14	2	1144
chessbig	28056	54	18	1387
cylBands	540	120	2	384688
heart	303	46	5	21932
hepatitis	155	48	2	39149
horsecolic	368	81	2	23530
ionosphere	351	155	2	385502
iris	150	14	3	144
led7	3200	22	10	2525
mushroom	8124	84	2	95122
pageblocks	5473	39	5	2904
pendigits	10992	81	10	107 031
pima	768	34	2	544
tictactoe	958	26	2	1906
waveform	5000	96	3	86903
wine	178	63	3	13446

Table 3: Dataset properties; number of {samples, binary variables, classes, average number of candidate patterns per fold for Classy with $m_s = 5\%$ and $l_{max} = 4$.}

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 compress better, but also predict better.

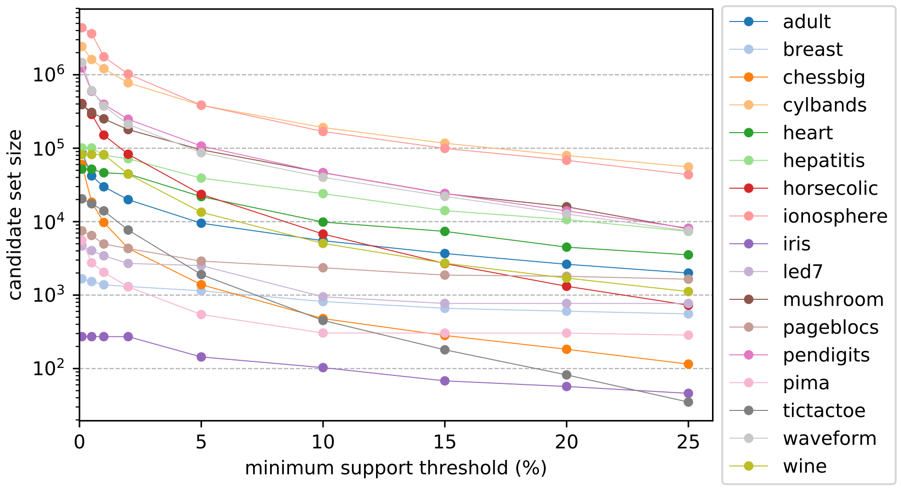
The second observation is that normalized gain performs better overall than absolute gain: AUC is higher in 15 out of 17 cases and relative compression is lower or equal in 13 out of 17 times. This confirms that normalized gain is, as we hypothesized, the best choice. We will therefore use *normalized gain* for the remaining experiments.

6.2. Candidate set influence

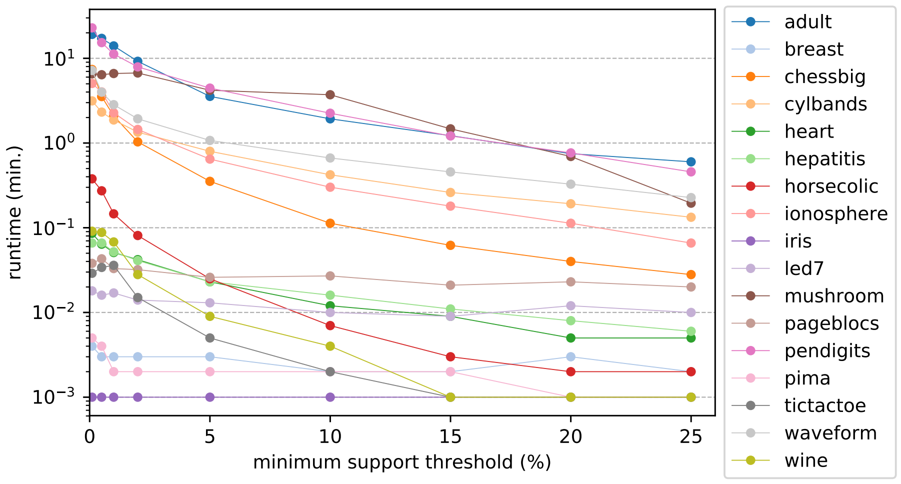
In this set of experiments we study the influence of the candidate set on Classy, which technically its is only "parameter", as it is the only part that can influence its output given the same dataset. In order to vary the candidate set objectively, the minimum support threshold ranges over $m_s = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$ 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 Figures 3, which show the influence of the candidate set on Classy through: the size of candidates mined in Figure 3a; runtime in Figure 3b; compression on the training set in Figure 3c; AUC in the test set in Figure 3d; and number of rules in a rule list in Figure 3e.

Figure 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. Figure 3b shows that in general the runtime increases at a rate similar to the increase in candidate size of figure 3a. This is in accordance with our analysis of time complexity in Section 5.5, 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. *Pima* and *breast* seem to be the



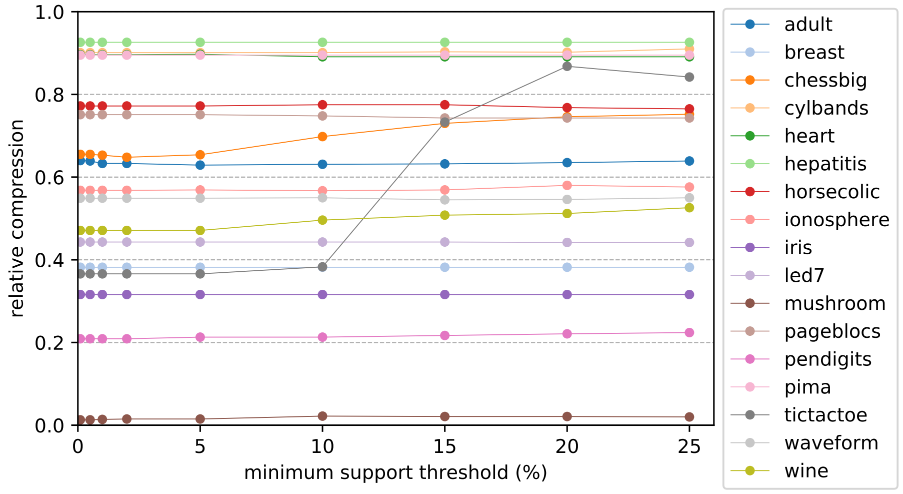
(a) candidate set size



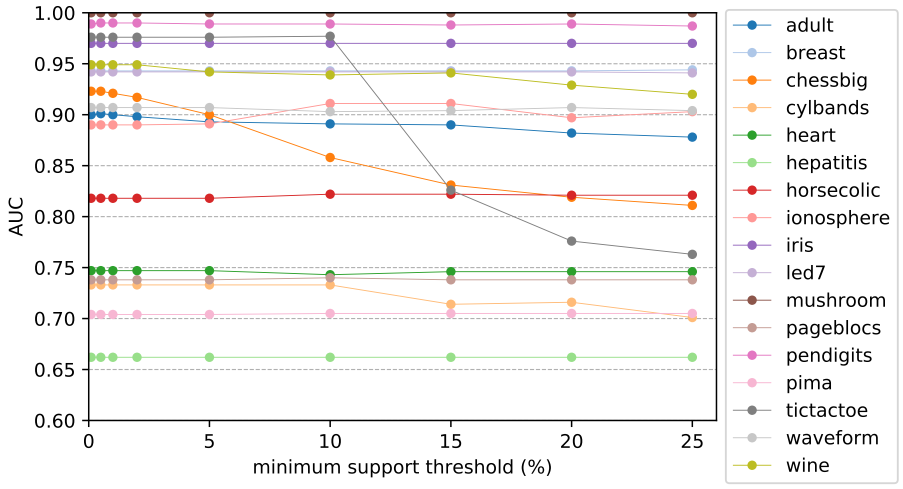
(b) runtime

major exceptions to this trend, with an increase in runtime for smaller candidate set sizes, which can be explained by the total number of rules selected increasing by 1 for larger minimum support, and by the fact that for their dataset size the behavior is not asymptotic.

Figure 3c and 3d show how Classy performs in classification in terms of com-

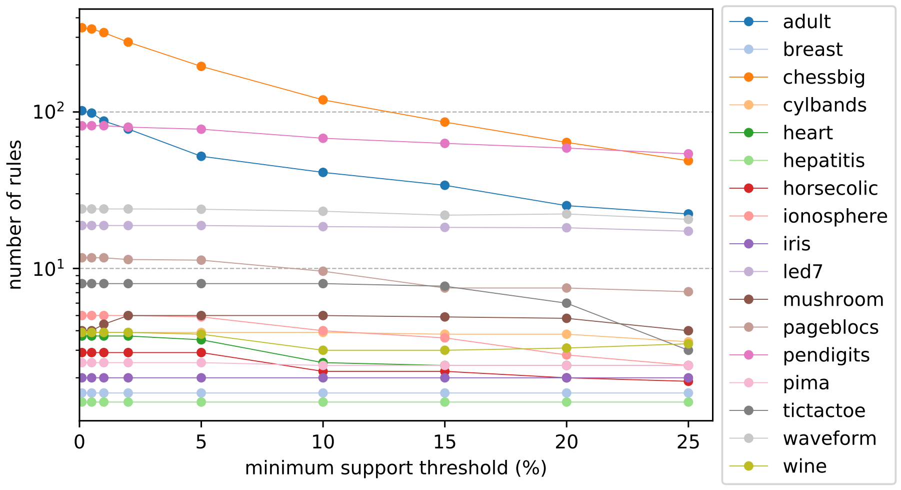


(c) compression in training set



(d) AUC in test set

pression in the training set and AUC in the test set, respectively. It should be noted that the values for both graphics remain, for most cases constant, and that if 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. There are two reassuring facts in these



(e) number of rules

Figure 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-folds and each dataset is connected by a line to aid visualization.

figures: 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 (compressive) of the data for big candidate sets, without being too greedy at using small covering rules; 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 small number of examples of *ionosphere* (351 instances) which does not allow our algorithm to distinguish a better generalizing rule from another less so.

Figure 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*. On a closer inspection, this is due to the disappearance of a rule with good performance but low coverage from the candidate set that has to be replaced by a combination of other rules. In the cases where much more rules were selected for lower minimum support thresholds, such as *chessbig* and *adult*, have an associated 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) [22], JRip[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, and JRip are clearly related to our approach. C5.0 is a newer version of C4.5, and JRip is a Java-implementation of RIPPER.

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), Classy has no parameters. 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 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.

SBRL was trained using the guidelines provided by the authors [22]: the number of chains was set to 25; iterations to 5000; η, representing the average size of

[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`

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 the AUC columns of Table 4. The SVM models perform better overall, but they do not belong to the class of interpretable models. Classy has the best overall performance of all rule- and tree-based classifiers with respect to AUC as shown by its overall average ranking of 2.1 out of 4, obtaining among them a higher or equal AUC for 8 out o 17 datasets. When looking only at binary datasets, Classy unperformed with an average ranking of 3.0 out of 5, coming after CART and C5.0. On the other hand, in multiclass datasets, it obtained the best rank of 1.6, outperforming all rule- and tree-based algorithms. This was achieved without any parameter tuning for Classy. Comparing with SBRL, Classy has a better ranking for binary datasets and outperforms in 7 out of 9 datasets. In the two cases that SBRL performed better it also found a more compact set of rules, which can be associated with its more thorough search method or due to its smaller candidate set. Also, the fact that it cannot handle so many candidates, makes it impossible to find some of the better performing solutions of Classy. CART and C5.0 have comparative performance to Classy achieving similar rankings. JRip typically has a lower AUC than the other classifiers.

6.4. Interpretability

The other columns in Table 4 show that Classy tends to find more compact models, with fewer rules and logical conditions per rule, than C5.0, CART, and RIPPER, that are as accurate or better than these. This can be clearly seen

Dataset	Classy			CART			C5.0			Ripper			SBRL			SVM
	AUC	$\lvert R\rvert$	$\lvert I\rvert$	AUC	$\lvert R\rvert$	$\lvert I\rvert$	AUC	$\lvert R\rvert$	$\lvert I\rvert$	AUC	$\lvert R\rvert$	$\lvert I\rvert$	AUC	$\lvert R\rvert$	$\lvert I\rvert$	AUC
adult	0.89	52	2	0.88	22	5	0.87	51	12	0.75	16	5	0.88	13	2	0.86
breast	0.94	2	4	0.96	6	3	0.96	6	3	0.96	3	4	0.96	3	3	0.85
chessbig	0.90	196	3	0.87	119	6	0.97	2866	17	0.81	396	6	–	–	–	1.00
cylBands	0.73	4	1	0.78	21	6	0.78	65	16	0.74	7	3	0.75	3	2	0.80
heart	0.75	4	2	0.77	11	3	0.72	38	8	0.54	3	3	–	–	–	0.83
hepatitis	0.66	1	1	0.69	4	2	0.68	10	5	0.62	4	2	0.60	2	2	0.85
horsecolic	0.82	3	1	0.85	6	2	0.85	16	5	0.78	4	2	0.82	2	3	0.79
ionosphere	0.89	5	1	0.91	5	2	0.91	12	5	0.89	6	2	0.88	3	2	0.87
iris	0.97	2	1	0.97	3	2	0.97	3	2	0.97	3	1	–	–	–	0.99
led7	0.94	19	3	0.94	29	3	0.94	28	5	0.92	19	4	–	–	–	0.95
mushroom	1.00	5	2	1.00	8	3	1.00	9	4	1.00	5	2	1.00	5	2	0.90
pageblocks	0.74	11	1	0.74	10	2	0.70	10	5	0.73	8	1	–	–	–	0.72
pendigits	0.99	78	3	0.99	68	4	1.00	262	11	0.99	107	4	–	–	–	1.00
pima	0.70	3	2	0.71	9	2	0.69	13	5	0.68	3	1	0.69	2	3	0.69
tictactoe	0.98	8	3	0.97	25	3	0.99	43	6	0.97	10	3	0.87	7	3	1.00
waveform	0.91	24	3	0.90	50	5	0.90	78	8	0.87	26	5	–	–	–	0.94
wine	0.94	4	1	0.94	5	1	0.94	8	3	0.92	5	2	–	–	–	1.00
$rank_{binary}$	3.0	2.1	**1.8**	**2.3**	3.7	3.3	2.4	4.9	4.8	3.6	3.1	2.8	3.8	**1.2**	2.3	–
$rank_{multi}$	**1.6**	**1.7**	**1.1**	2.2	2.4	2.2	2.3	3.4	3.9	3.7	2.1	2.6	–	–	–	–
$rank_{all}$	**2.1**	**1.5**	**1.3**	2.2	2.6	2.4	2.3	3.6	3.8	3.4	2.1	2.4	–	–	–	–

Table 4: Results per dataset. Area Under the ROC Curve (AUC) (weighted AUC for multi-class datasets), average number of rules $|R|$ (including the default rule), and average number of logical conditions per rule ($|I|$). At the bottom the average rank of rule- and tree-based algorithms {Classy, CART, C5.0, RIPPER, SBRL} is shown for *binary*, *multi*class, and *all* datasets. The rank of 1 is given for the best value—highest AUC; lowest $|R|$; lowest $|I|$—and 4 for the worst in *multi*class and *all*, and 5 for *binary*. The best average rank for each measure is shown in bold. Note that SBRL cannot do multiclass classification, hence only being present in the binary ranking and the −s.

by its overall average rank of 1.5 for rules and 1.3 for conditions per rule, which shows that for most datasets it obtained the lowest number of rules and conditions of each tree- and rule-based classifier. Although SBRL also finds very compact rule lists, with small number of rules and average number of conditions per rule, the low variance between the reported values for the different datasets suggests that this strongly depends on the hyperparameter settings, which penalize too strongly 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 overcomplex 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. Overfitting

To study overfitting, we compared the averages of the absolute difference between the AUC values in the training and test set over 10-folds for each algorithm. The results can be seen in Figure 4. In general Classy, together with SVM, seem to be the most consistent algorithms in obtaining the lowest values. The usual performance of Classy is around 5% or lower, except in the case of *hepatitis* were it got 14%, which was the best value after SVM. SBRL is very consistent, clearly achieving the lowest value for 3 datasets, namely *breast*, *cylbands* and *horsecolic*, 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 or similar value for 11 out of 17 datasets, being, from these ones, the algorithm that less overfits overall. C5.0 is clearly the algorithm that more times overestimates in

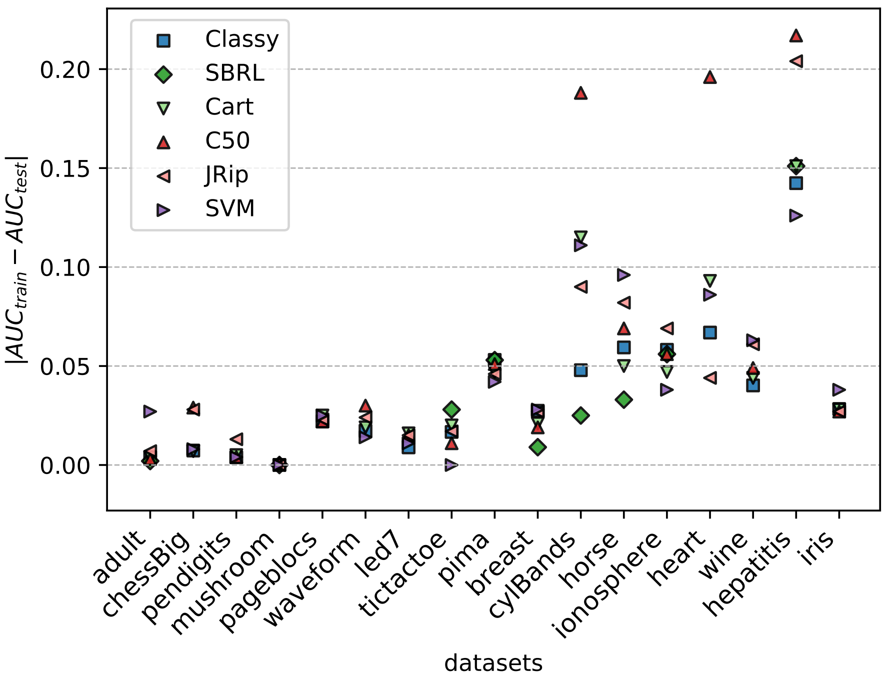


Figure 4: Average absolute difference between AUC performance in training and test sets, per fold, for each algorithm and each dataset. The datasets are ordered by the number of instances (descending). The standard deviations for these values tend to be the same order of magnitude as their corresponding averages. We decided to not report them in the figure as it does not impact our analysis and to ease the visualization. Note that SBRL does not have a values for multiclass datasets.

the training set, followed by JRip and then CART.

6.6. Scalability

All runtimes are averages over ten folds, run on a 64-bit Windows Server 2012R2, 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 Figure 5. CART, C5.0, and JRip are the fastest, with most runtimes under 1 minute with Classy being one order of magnitude slower than these. Comparing to SBRL, Classy has a similar runtime, even though it considers around 100 times more candidates than this and performs

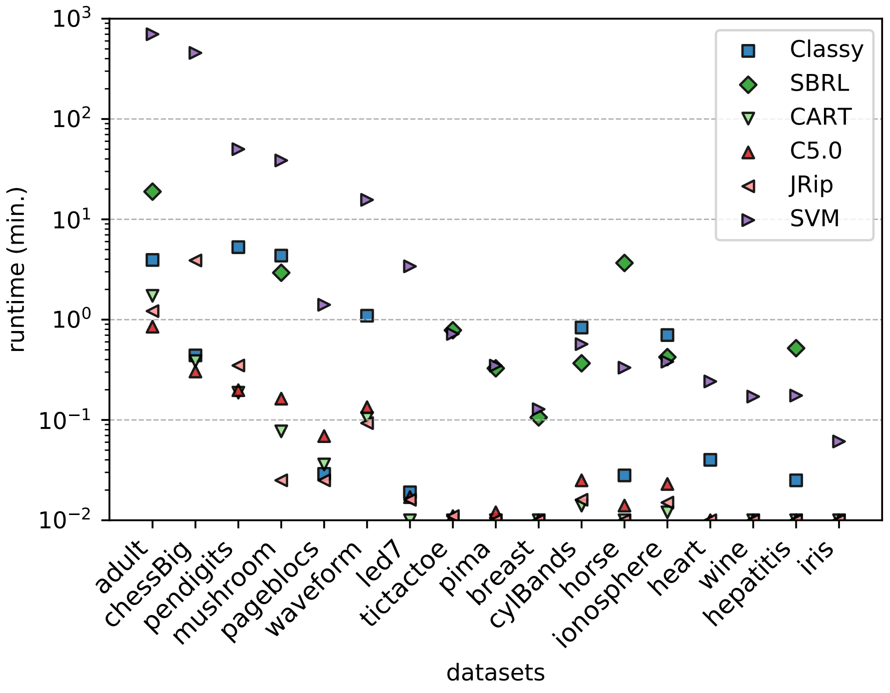


Figure 5: Average runtime per fold in minutes for each algorithm and each dataset. The datasets are ordered by the number of transactions (descending). Note that SBRL does not have a runtime for multiclass datasets.

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 Figures 3b and 3d.

6.7. Discussion

From the classification and rule list size results in Table 4 it can be seen that Classy is able to provide a good trade-off between AUC performance and rule list size. This is particularly the case for multiclass datasets such as *chessbig*, where classical algorithms like CART and C5.0 tend to find models that are either too sparse or too complex, or in the case of *mushroom*, where CART and C5.0 find more complex models with the same performance. 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 Figures 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 Figure 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.

From the runtimes of Figure 5, 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 4, 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. 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, Figure 4 shows that Classy has a tendency to select models that generalize well and that are not overconfident in the training set. It obtains the lowest difference 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.

References

[1] F. Doshi-Velez, B. Kim, Towards a rigorous science of interpretable machine learning, arXiv:1702.08608 [stat.ML] (2017).

[2] B. Letham, C. Rudin, T. H. McCormick, D. Madigan, et al., Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model, The Annals of Applied Statistics 9 (3) (2015) 1350–1371 (2015).

[3] H. Lakkaraju, C. Rudin, Learning cost-effective and interpretable treatment regimes, in: Artificial Intelligence and Statistics, 2017 (2017).

[4] H. Lakkaraju, C. Rudin, Learning cost-effective and interpretable treatment regimes for judicial bail decisions, in: NIPS 2016, 2016 (2016).

[5] J. Zeng, B. Ustun, C. Rudin, Interpretable classification models for recidivism prediction, Journal of the Royal Statistical Society: Series A (Statistics in Society) 180 (3) (2017).

[6] 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 (2016).

[7] I. Polaka, E. Gašenko, O. Barash, H. Haick, M. Leja, Constructing interpretable classifiers to diagnose gastric cancer based on breath tests, Procedia Computer Science 104 (2017).

[8] Y. Lou, R. Caruana, J. Gehrke, Intelligible models for classification and regression, in: KDD'12, ACM, 2012, pp. 150–158 (2012).

[9] H. Lakkaraju, S. H. Bach, J. Leskovec, Interpretable decision sets: A joint framework for description and prediction, in: KDD'16, ACM, 2016 (2016).

[10] 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 (2016).

[11] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, C. Rudin, Learning certifiably optimal rule lists, in: KDD'17, ACM, 2017 (2017).

[12] 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, Decision Support Systems 51 (1) (2011) 141–154 (2011).

[13] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978).

[14] P. D. Grünwald, The minimum description length principle, MIT press, 2007 (2007).

[15] J. Vreeken, M. van Leeuwen, A. Siebes, Krimp: mining itemsets that compress, Data Mining and Knowledge Discovery 23 (1) (2011) 169–214 (2011).

[16] M. van Leeuwen, J. Vreeken, Mining and using sets of patterns through compression, in: Frequent Pattern Mining, Springer, 2014, pp. 165–198 (2014).

[17] K. Budhathoki, J. Vreeken, The difference and the norm – characterising similarities and differences between databases, in: ECMLPKDD'15, Springer, 2015, pp. 206–223 (2015).

[18] C. Molnar, Interpretable machine learning, A Guide for Making Black Box Models Explainable (2018).

[19] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, Classification and regression trees, CRC press, 1984 (1984).

[20] J. R. Quinlan, C4. 5: programs for machine learning, Elsevier, 2014 (2014).

[21] W. W. Cohen, Fast effective rule induction, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 115–123 (1995).

[22] 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 (2017).

[23] J. O. R. Aoga, T. Guns, S. Nijssen, P. Schaus, Finding probabilistic rule lists using the minimum description length principle, in: DS'18, 2018 (2018).

[24] 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 (2018).

[25] J. Hühn, E. Hüllermeier, Furia: an algorithm for unordered fuzzy rule induction, Data Mining and Knowledge Discovery 19 (3) (2009) 293–319 (2009).

[26] E. Bellodi, F. Riguzzi, Structure learning of probabilistic logic programs by searching the clause space, Theory and Practice of Logic Programming 15 (2) (2015) 169–212 (2015).

[27] J. Rissanen, A universal prior for integers and estimation by minimum description length, The Annals of statistics (1983) 416–431 (1983).

[28] 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 (1993).

[29] B. L. W. H. Y. Ma, B. Liu, Integrating classification and association rule mining, in: KDD'98, 1998 (1998).

[30] 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 (2001).

[31] 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 (2005).

[32] M. García-Borroto, J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, A survey of emerging patterns for supervised classification, Artificial Intelligence Review 42 (4) (2014) 705–721 (2014).

[33] P. Kralj Novak, N. Lavrač, G. Webb, Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining, Journal of Machine Learning Research 10 (2009) 377–403 (2009).

[34] G. I. Webb, Discovering significant patterns, Machine Learning 68 (1) (2007) 1–33 (2007).

[35] A. Zimmermann, S. Nijssen, Supervised pattern mining and applications to classification, in: Frequent Pattern Mining, Springer, 2014 (2014).

[36] M. van Leeuwen, E. Galbrun, Association discovery in two-view data, IEEE Transactions on Knowledge and Data Engineering 27 (12) (2015).

[37] X. Zhang, G. Dong, K. Ramamohanarao, Information-based classification by aggregating emerging patterns, in: IDEAL, Springer, 2000, pp. 48–53 (2000).

[38] A. Gelman, H. S. Stern, J. B. Carlin, D. B. Dunson, A. Vehtari, D. B. Rubin, Bayesian data analysis, Chapman and Hall/CRC, 2013 (2013).

[39] J. Fürnkranz, D. Gamberger, N. Lavrač, Foundations of rule learning, Springer Science & Business Media, 2012 (2012).

[40] C. Borgelt, Efficient implementations of apriori and eclat, in: FIMI03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations, 2003 (2003).

[41] F. Provost, P. Domingos, Well-trained pets: Improving probability estimation trees (2000).