Robust rules for prediction and description

Manuel Proenca, H.

4

Discovering predictive rule lists with CLASSY

In this chapter[1], we propose the CLASSY algorithm based on the MDL formulation of predictive rule lists given in Chapter 3. This algorithm uses a greedy heuristic to find good predictive rule lists and can be applied to supervised tabular datasets with univariate nominal targets, i.e., multiclass classification. In machine learning, a predictive rule list is an instance of interpretable machine learning models, as long as the number of rules is reasonably small. To validate our approach, we conduct an empirical comparison on 17 datasets against state-of-the-art-algorithms.

Note that in Chapter 3 we presented a formulation of predictive rule lists for single and multi-nominal classification and single and multi-regression problems. However, we only developed CLASSY for *discrete* explanatory variables and regular classification, hence, we only present results for this scenario in this chapter. For regression the greedy algorithm would require some changes in order to find good regression rule lists.

Recapitulation of predictive rule list definition and MDL encoding. In the previous chapters we defined what a predictive rule list is and gave its definition of optimality according to the MDL principle. We will now restate those definitions here.

First, let us recall from Chapter 2.4 the predictive rule lists model in Figure 4.1.

The best predictive rule list RL according to the MDL principle is the one that, given

[1] Parts of this chapter are based on Proença and van Leeuwen [96]

$$
\begin{array}{rlll}
s_1\text{:} & \text{IF} & a_1 \sqsubseteq \mathbf{x} \quad \text{THEN} & y \sim Cat(\hat{p}_{1|1}, \cdots, \hat{p}_{k|1}) \\
 & & \vdots & \\
s_\omega\text{:} & \text{ELSE IF} & a_\omega \sqsubseteq \mathbf{x} \quad \text{THEN} & y \sim Cat(\hat{p}_{1|\omega}, \cdots, \hat{p}_{k|\omega}) \\
\text{dataset:} & \text{ELSE} & & y \sim Cat(\hat{p}_{1|d}, \cdots, \hat{p}_{k|d})
\end{array}
$$

Figure 4.1: Generic rule list model RL for classification with ω rules $R = \{r_1, ..., r_\omega\}$ and one target variable distribution per rule. Note that the parameters of the default rule of a rule list are not fixed (in contrast to those of a subgroup list) and just describes the subset covered by it, i.e., D^d.

the dataset D, minimizes the two-part code defined in Chapter 3.1:

$$RL^* = \underset{RL \in \mathcal{RL}}{\arg\min}\, L(D, RL) = \underset{RL \in \mathcal{RL}}{\arg\min} \left[L(\mathbf{Y} \mid \mathbf{X}, RL) + L(RL) \right],$$

where $L(RL)$ is the length of encoding the predictive rule list model RL, and $L(\mathbf{Y} \mid \mathbf{X}, RL)$ is the length of encoding the target variables data given the predictive rule list RL and the explanatory variables $\mathbf{X}$. The *model encoding* is the same for predictive rule lists and subgroup lists, as they only differ in how the default rule encodes the data, and was defined in Chapter 3.2. Nonetheless, at the time these experiments were developed [96], the formulation was only done for discrete explanatory variables, and it is suboptimal compared to that of Eq. (3.3). Thus, the model encoding throughout this chapter is:

$$L(RL) = L_{\mathbb{N}}(|R|) + \sum_{a_i \in R} \left[L_{\mathbb{N}}(|a_i|) + |a_i| \log m \right],$$

where R is the list of predictive rules in RL, i.e., the model excluding the default rule. Compared with Eq. (3.3), we here used a uniform code for the variables in a_i, i.e., $|a_i| \log m$, which is suboptimal for unordered sets compared to $\binom{m}{|a_i|}$. Also, we do not require $\sum_{v \in a_i} L(v)$ for the different types of variables, as only binary explanatory variables were considered, and only their positive value, i.e., $x = 1$ is encoded.
Then, in the case of *classification*, where there is only one target variable Y, we use a categorical distribution and the Prequential Plug-in encoding defined in Chapter 3.4:

$$L(Y \mid \mathbf{X}, RL) = L_{\text{plug-in}}(Y^d) + \sum_{\rho_i \in R} L_{\text{plug-in}}(Y^i),$$

which is asymptotically optimal. Even though the Normalized Maximum Likelihood encoding is optimal for finite n, and thus, theoretically better than the prequential plug-in, we did not find very significant differences in preliminary experiments, except for a few edge cases.

Structure of the chapter. This chapter is organized as follows. First, in Section 4.1 the most relevant related work is covered, together with the main differences to our approach. After that, in Section 4.2 the CLASSY algorithm, a heuristic algorithm to mine predictive rule lists is defined. Then, in Section 4.3 we empirically validate our proposed method on 17 datasets when compared against the state-of-the-art algorithms for classification. Finally, in Section 4.4 the main conclusions are presented.

4.1 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 [91].

Table 4.1 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 [15], C4.5 [103], and RIPPER [22], lack a global optimisation criterion and thus rely on heuristics and hyperparameters to deal with overfitting. Fuzzy rule-based models [3, 59], here represented by FURIA [55] use rule sets instead of rule lists and lack probabilistic predictions. Recent Bayesian methods [122, 69, 122] are limited to small numbers of candidate rules and binary classification, limiting their usability, and are here represented by SBRL [125] (which is representative for all of them). A recent approach also using MDL and probabilistic rule lists (MRL) [7] is aimed at describing rather than classifying and cannot deal with multiclass problems or a large number of candidates. Interpretable decision sets (IDS) [69] and certifiable optimal rules (CORELS) [5] use similar rules but do not provide probabilistic models or predictions.

Note that methods that explain black-box models [104, 105], 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) using 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 [111].

Table 4.1: 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[69]	✓	-	✓	✓	-
CORELS [5]	-	-	✓	-	-
MRL[7]	-	✓	✓	-	✓
SBRL[125]	-	✓	✓	-	-
FURIA[55]	✓	-	-	✓	-
Others	✓	✓	-	✓	-

4.1.1 Rule-based classifiers

Rule lists have long been successfully applied for classification; RIPPER is one of the best-known algorithms [22]. Similarly, decision trees, which can easily be transformed to rule lists, have been used extensively; CART [15] and C4.5 [103] are probably the best-known representatives. These early approaches represent highly greedy algorithms that use heuristic methods and pruning to find the 'best' models.

Fuzzy rule-based 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 [55], using genetic algorithms to combine pre-mined fuzzy association rules [3], and doing a multiobjective search over accuracy and comprehensibility to find Pareto-optimal solutions [59]. Although these approaches are related, the rules are aggregated in a rule set, i.e., a set of independent *if* $\cdots$ *then* $\cdots$ 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.

Global optimization approaches. Over the past years, rule learning methods that go beyond greedy approaches have been developed, i.e., using probabilistic logic programming for independent rule-like models [11], greedy optimization of submodular problem formulation, or simulated annealing in the case of decision sets [69, 122], Monte-Carlo search for Bayesian rule lists [81, 125], and through branch-and-bound

with tight bounds for decision lists [5]. 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 optimality criteria. Also, only Bayesian rule lists [81, 125] and Bayesian decision sets [122] provide probabilistic predictions.

All previously mentioned algorithms share some similarities with CLASSY. In particular Bayesian rule lists [81, 125] 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 [108]—and therefore to different results; we will empirically compare the two approaches. Certifiable optimal rules [5] have a similar rule structure but do not provide probabilistic models or predictions. Decision sets [69] share the use of rules, but as opposed to (ordered) lists they consider (unordered) sets of rules.

4.1.2 Pattern mining

Association rule mining [2], 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 [85] and CMAR [82], but they tend to lack interpretability because they use large numbers of rules. Ensembles of association rules, such as Harmony [121] or classifiers based on emergent patterns [41], can increase classification performance when compared to the previous methods, however, they can only offer local interpretations.

Supervised pattern set mining [128]. 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.
Note that subgroup discovery, and specifically subgroup lists, are also related as they share the same model structure as predictive rule lists. For more details on that, we refer the interested reader to Chapter 4.

4.1.3 MDL-based data mining

In data mining, the MDL principle has been used to summarize different types of data, e.g., transaction data [120, 18], and two-view data [73].

In prediction, it has been previously used to deal with overfitting [22, 103] and in the selection of the best compressing pattern [127].
RIPPER and C4.5 [22, 103] use the MDL principles in their post-processing phase as a criterion for pruning, while we use it in a holistic way for model selection. Although Krimp has been used for classification [120], it was not designed for this: it outputs large pattern sets, one for each class, and does not give probabilistic predictions. DiffNorm [18] 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 [7], 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.

4.2 The CLASSY algorithm

Given our model class—predictive 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 predictive 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.

4.2.1 Separate-and-conquer greedy search

Greedy search is very commonly used for learning decision trees and predictive rule lists [103, 22, 39], as well as for pattern-based modelling using the MDL principle [120, 18, 73]. A few recent approaches use optimization techniques [125], 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 [34], 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 [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*.

4.2.2 Compression gain

The proposed heuristic is based on the compression gain that is obtained by adding a rule $\rho = (a, \hat{\Theta}^a)$ to a rule list RL, which will be denoted by $RL \oplus \rho$. Note that for categorical distributions $\hat{\Theta}^a = \{\hat{p}_{1|a}, \cdots \hat{p}_{k|a}\}$, which are just the conditional probability for each class label $c \in \{1, \cdots, k\}$, given the description a. 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 as positive if adding a rule represents a compression improvement, and negative vice-versa.

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

$$\begin{aligned} \Delta L(D, RL \oplus \rho) &= L(D, RL) - L(D, RL \oplus \rho) \\ &= \underbrace{L_{\text{plug-in}}(Y \mid \mathbf{X}, RL) - L_{\text{plug-in}}(Y \mid \mathbf{X}, RL \oplus \rho)}_{\Delta L(Y|\mathbf{X}, RL \oplus \rho)} \\ &+ \underbrace{L(R) - L(RL \oplus \rho)}_{\Delta L(RL)}, \end{aligned} \tag{4.1}$$

where $L_{\text{plug-in}}(\cdots)$ was used to refer that we use the prequential plug-in encoding of Eq. (3.14) in this chapter (instead of the Normalized Maximum Likelihood). Using Eq. (4) we show the *model gain* as:

$$\begin{aligned} \Delta L(RL \oplus \rho) = &L_{\mathbb{N}}(|R|) - L_{\mathbb{N}}(|R| + 1) \\ &- L_{\mathbb{N}}(|a|) - |a| \log m. \end{aligned} \tag{4.2}$$

Note that the 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 ρ to RL 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 ρ_d and the combination of the new pattern $a \in \rho$ with the new default rule ρ'_d. Using Equation (B.3) we obtain

$$\begin{aligned}\Delta L(Y \mid \mathbf{X}, RL \oplus \rho) = & \overbrace{\cancel{\sum_{i=1}^{\omega} L_{\text{plug-in}}(Y^i)} + L_{\text{plug-in}}(Y^d)}^{L_{\text{plug-in}}(Y|\mathbf{X},RL)} \\ & \underbrace{-\cancel{\sum_{i=1}^{\omega} L_{\text{plug-in}}(Y^i)} - L_{\text{plug-in}}(Y^{d'}) - L_{\text{plug-in}}(Y^a)}_{L_{\text{plug-in}}(Y|\mathbf{X},RL\oplus\rho)},\end{aligned} \tag{4.3}$$

where $Y^{d'}$ is the subset of the data covered by the new default rule (after ρ is added to RL) and $\omega = |R|$.
Normalized compression gain, denoted $\delta L(D, RL \oplus \rho)$, 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:

$$\delta L(Y \mid \mathbf{X}, RL \oplus \rho) = \frac{\Delta L(Y \mid \mathbf{X}, RL \oplus \rho)}{n_a} \tag{4.4}$$

By normalizing for the number of instances that a predictive 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, we hypothesize that normalized gain will lead to better predictive rule lists. We will empirically verify if this is indeed the case.

Note that the *absolute* and *normalized* gains are specific cases of the β-gain of Eq. (5.1) presented later in Chapter 5, where for these specific cases, they correspond to $\beta = 0$ and $\beta = 1$, respectively.

4.2.3 Candidate generation

Candidates are probabilistic rules of the form $\rho = (a, \hat{\Theta}^a)$ that are considered for addition to a predictive 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 [13], and then finding the corresponding consequent categorical distribution $\hat{\Theta}^a$ given the dataset, i.e., using the maximum likelihood estimate of Eq. (2.9). In practice, these mining algorithms have only two parameters: the minimum support threshold $n_{min.}$ and the maximum length d_{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 fewer conditions then b, i.e., $a \subset b$, implies that $n_a \geq n_b$. This property is also used to **remove strictly redundant rules** in CLASSY.
Given all candidates from the frequent pattern mining algorithm, if antecedent a is a strict subset of antecedent b, i.e., $a \subset b$, and they have equal support, $n_a = n_b$, we say that antecedent b is redundant and will never be selected. This is a consequence of their encoding, i.e., as $Y^a = Y^b \implies L_{plug-in}(Y^a) = L_{plug-in}(Y^b)$ 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 the model search, as the gain of a will always be greater.

4.2.4 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 3.4. The algorithm, outlined in Algorithm 4.1, 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 predictive rule list.
The first step of our algorithm is to remove the strictly redundant patterns as mentioned in Section 4.2.3 (Ln 1). After that, we initialize the predictive rule list with the default rule (Ln 2), which acts as the baseline model to start from. Then, while there is a predictive 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 previously added rule, and then the gain of adding the candidate can be updated. When no rule improves compression (negative gain)

the while loop stops and the rule list is returned.

Algorithm 4.1 The CLASSY algorithm

Input: Dataset D, candidate set $Cands$
Output: Multiclass probabilistic rule list R

1: $Cands \leftarrow RemoveRedundancy(Cands)$
2: $RL \leftarrow [\varnothing]$
3: **repeat**
4: $\quad \rho \leftarrow \arg\max_{\forall r' \in Cands} : \delta L(D, RL \oplus \rho')$
5: $\quad RL \leftarrow RL \oplus \rho$
6: $\quad \text{UpdateCandidates}(D, RL, Cands)$
7: **until** $\delta L(D, RL \oplus \rho') \leq 0, \forall \rho' \in Cands$
8: **return** RL

4.2.5 Time and space complexity

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

Initialization step. 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, in the worst case, it would cover the whole dataset, and it would take $\mathcal{O}(|D| + |D|)$. Doing this for all candidates gives $\mathcal{O}(|Cands||D|)$.

Iterative loop. After the initialization step, CLASSY iteratively finds the best rule to add for a total of $\omega = |R|$ runs, where RL is the predictive rule list 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 predictive rule lists 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 (3.14) and gains (4.4) 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 $\mathcal{O}(multi(|D|) + |D|)$ time, where $multi(*)$ is the complexity of the multiplication used; 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.

Memory complexity. 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

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

4.3 Empirical evaluation

In this section we empirically evaluate our approach[2], 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

[2]For the reproducibility of the experiments, please check `https://github.com/HMProenca/MDLRuleLists`

runtime.

Data. We use 17 varied discretised datasets (see Table F.1) from the LUCS/KDD[3] 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.

Candidate rules generation. Frequent pattern mining algorithms generate different candidate sets by setting different values for the minimum support per class threshold n_{min} and maximum pattern length d_{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 $d_{max} = 4$ and $n_{min} = 5\%$, 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 minutes while demonstrating that CLASSY can deal with large candidate set sizes. First, we chose $d_{max} = 4$ 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 n_{min} by requiring the runs for all datasets to strictly finish in under 10 minutes and for most datasets even under 1 minute, to be comparable to $CART$, $C5.0$, and $JRip$ in runtime, and also to have attained (empirical) convergence in terms of compression ratio on the training set—further lowering n_{min} would not increase compression—as can be seen from the vertical dashed lines in Figure 4.3.

Candidate patterns are mined using Borgelt's implementation of the well-known frequent pattern mining algorithm FP-growth [13]. The same candidate set was used for all experiments except when assessing its influence on CLASSY in Section 4.3.2. For that experiment we fixed $d_{max} = 4$ and varied the minimum support threshold per class from $n_{min} = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$.

Evaluation criteria. We evaluate and compare our approach based on classification performance, overfitting, interpretability, and runtime. Besides, 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).

MDL criterion quantification. To quantify how well a predictive rule list compresses the class labels and be able to compare the MDL score across datasets, we define

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

relative compression as

$$L\% = \frac{L(D, R)}{L(D \mid \hat{\Theta}^d)}, \tag{4.5}$$

where $L(D \mid \hat{\Theta}^d)$ is the encoding length of the data given the predictive 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. Classification performance is measured using three measures presented in Section 2.5: accuracy; balanced accuracy [17]; 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.
AUC, on the other hand, is not based on a fixed threshold and takes into account the probabilities associated with each prediction. In the case of multiclass datasets, we use *weighted AUC* [102], as it takes into account the class distribution in the dataset.

Interpretability. For interpretability, we follow the most commonly used measure, i.e., that smaller models are easier to understand [26]. 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 predictive 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. Overfitting is measured in terms of the absolute difference between the AUC performance in the training set and in the test set.

Runtime. 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.

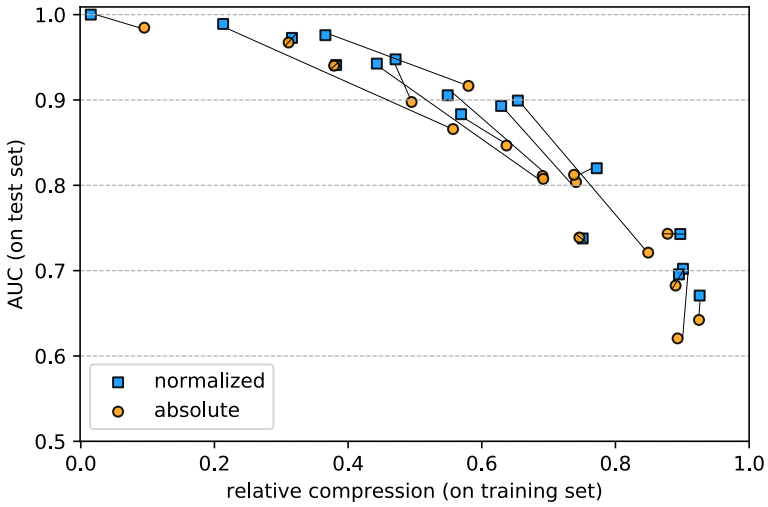


Figure 4.2: Relation between compression and AUC; better compression on the training set (lower relative compression) corresponds to better classification on the test set (higher AUC). Results obtained with CLASSY using normalized (squares) and absolute (circles) gain, on all 17 datasets; each point 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.

4.3.1 Compression versus classification

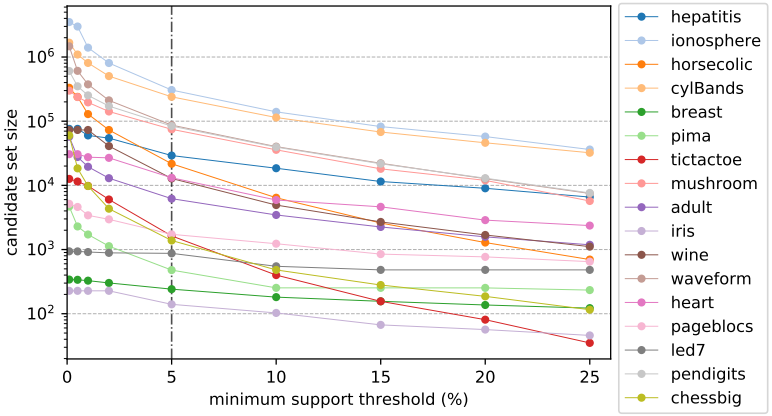
We first investigate the effect of using absolute (4.1) or normalized gain (4.4). To this end, Figure 4.2 depicts how the two heuristics perform for relative compression (on the training set) and AUC (on the test set).

The first observation is that better compression of the training data 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 predictive 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.

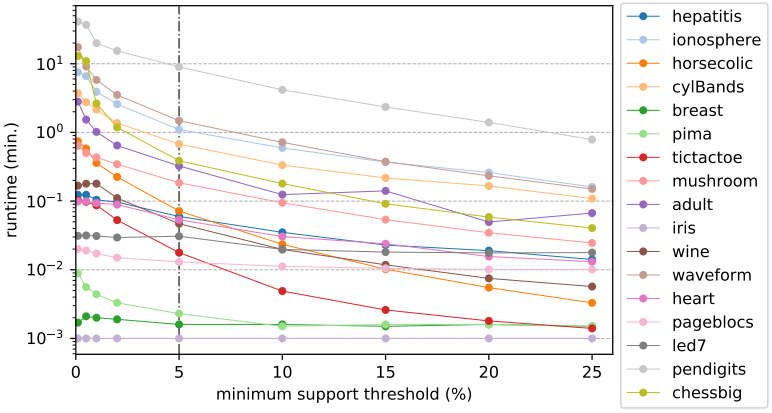
The second observation is that normalized gain performs better overall than absolute gain: AUC is higher in 15 out of 17 cases and relative compression is lower or equal in 11 out of 17 times. This confirms that normalized gain is, as we hypothesized, the best choice. We will therefore use *normalized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

4.3.2 Candidate set influence

In this set of experiments, we study the influence of the candidate set on CLASSY, which technically is its only "hyperparameter", 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 $n_{min.} = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$ and the maximum pattern length was fixed at $d_{max} = 4$, allowing the generation of large candidate sets.

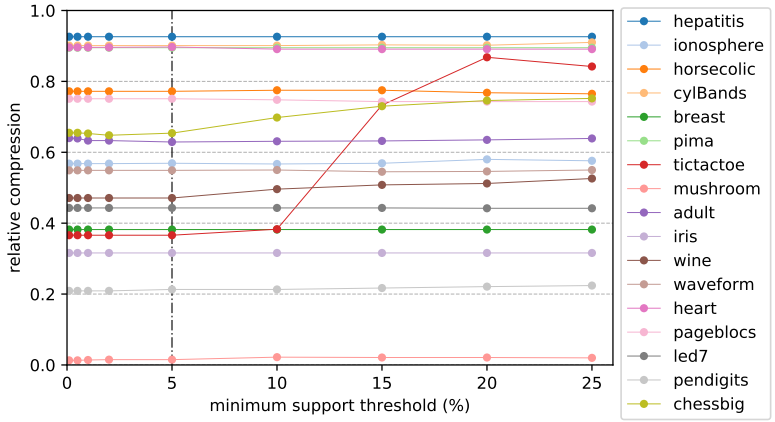


(a) candidate set size

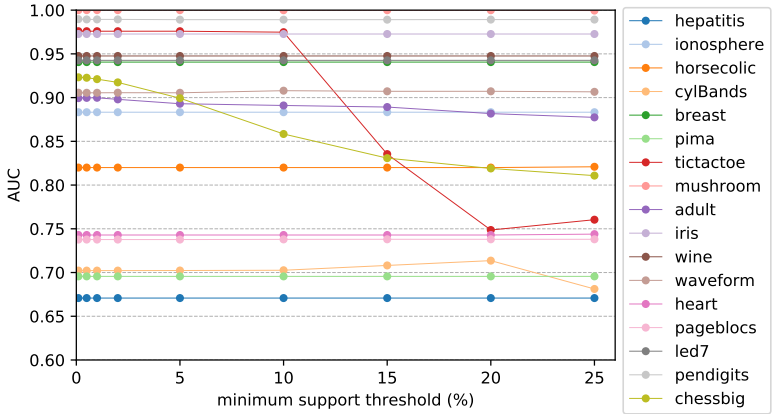


(b) runtime

The results can be seen in the set of Figures 4.3, which show the influence of the can-



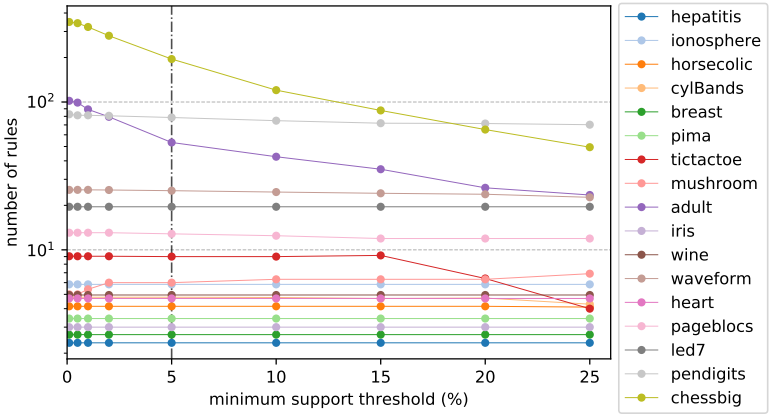
(c) compression in training set



(d) AUC in test set

didate set on CLASSY through: the size of candidates mined in Figure 4.3a; runtime in Figure 4.3b; compression on the training set in Figure 4.3c; AUC in the test set in Figure 4.3d; and the number of rules in a rule list in Figure 4.3e.

Minimum support. Figure 4.3a shows the growth 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 4.3b shows that in general, the runtime increases at a rate similar to the increase in candidate size of Figure 4.3a. This follows our analysis of time complexity in Section 4.2.5, which tells us that the time complexity of CLASSY grows proportionally to the dataset size times the candidate set size,



(e) number of rules

Figure 4.3: 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 $n_{min.} = \{0.1\%, 0.5\%, 1\%, 2\%, 5\%, 10\%, 15\%, 20\%, 25\%\}$. 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 in the experiments section for predictive rule lists (Section 4.3).

thus, given a fixed dataset size it becomes proportional only to the candidate set size.

Compression and classification. Figures 4.3c and 4.3d 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 predictive rule list 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.

Number of rules. Figure 4.3e 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.

4.3.3 Classification performance

We now compare the classification performance of CLASSY to Scalable Bayesian Rule Lists (SBRL) [125], JRip[4], FURIA[4], CART[5], C5.0[6], and Support Vector Machines[7] (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.

Hyperparameters tuning. CLASSY has no hyperparameters apart from the candidate set, which was generated using FP-growth with $n_{min} = 5\%$ and $d_{max} = 4$ for each dataset (as described at the beginning of Section 4.3). 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 the 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 [125]: 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

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

predictive 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.

Analyse of results. The results are presented in Table 4.2. The SVM models achieve the best ranking overall, but they do not belong to the class of interpretable models.
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 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 predictive 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).

4.3.4 Interpretability

The results are shown in Table 4.3, 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, and FURIA. On the other hand, in multiclass datasets, it achieves a much lower (=better) ranking than all the other algorithms.

Table 4.2: Classification performance average results (10 times repeated 10-fold cross-validation) 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 tends to find more compact models, with a 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 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 a small number of rules and conditions, the low variance between the reported values for the different datasets suggests that this strongly depends on the hyper-parameter 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.

4.3.5 Statistical significance testing

To analyze whether the results of Tables 4.2 and 4.3 are statistically different [25], we use two non-parametric multiple hypothesis tests, namely Friedman's test [37] and Iman and Davenport's test [57], on the rankings of the algorithms.
The results can be seen in the left side of Table 4.4, 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 [54] for pairwise comparisons with CLASSY as control algorithm.
The results of these pairwise comparisons can be seen in the right side of Table 4.4. For most of these tests the null hypothesis—stating that CLASSY and its competitor perform on par—can not 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

Table 4.3: Interpretability performance average results (10 times repeated 10-fold cross-validation) 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	*

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 then 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.

4.3.6 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 4.5. 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* where it got 13%, which was the best value after SVM. SBRL is very 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 4.2. 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.

4.3.7 Runtime

All runtimes are averages over ten times repetitions of 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 4.4. CART, C5.0, JRip, and FURIA are the fastest, with most runtimes under 1 minute with CLASSY being at a 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 Figures 4.3b and 4.3d.

4.3.8 Discussion

From the classification and interpretability results of Table 4.2 and 4.3, it can be seen that CLASSY can provide a good trade-off between classification performance and rule

Table 4.4: 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 Table 4.2 and 4.3 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 4.2 and AUC_{rules} is the AUC comparison of all tree- and rule-based algorithms (SVM excluded) of Table 4.3. 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 4.5: Overfitting average results (10 times repeated 10-fold cross-validation) using the absolute difference between AUC performance in training and test sets as a 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 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

list size. Particularly, in the case of 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*.

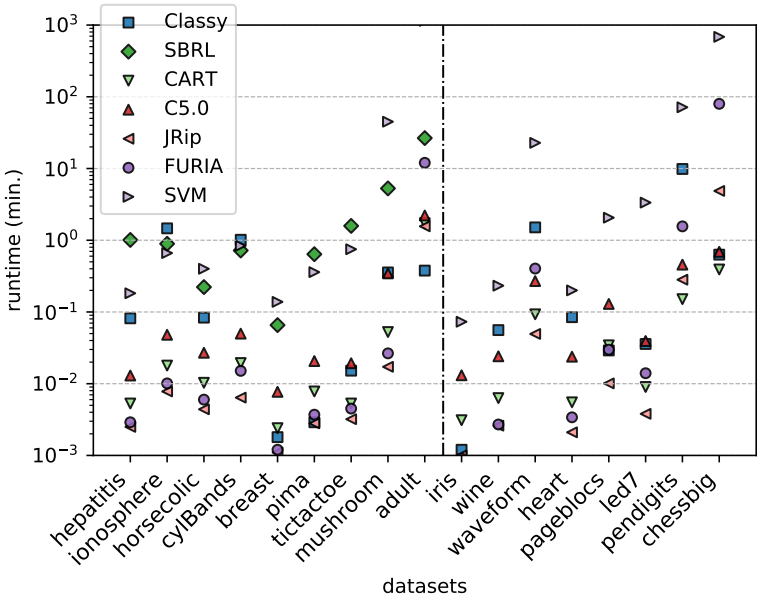


Figure 4.4: Average runtime per fold in minutes for each algorithm and each dataset. The datasets are ordered first by the number classes and then by number of samples (ascending). The vertical dashed line separates binary (to the left) from multiclass datasets. Note that SBRL does not have a runtime for multiclass datasets.

Candidate set influence. In the set of Figures 4.3, 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 hyperparameter—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 the case of small data: no data needs to be reserved for validation.

Compression and classification. From Figure 4.2 we can observe that better compression corresponds to better classification, which is a strong empirical validation of our formalization. As expected, the normalized gain is the best heuristic to use in combination with our greedy rule selection strategy, as it results in better classifiers for 88% of the datasets.

Runtime. From the runtimes of Figure 4.4, 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 F.1.

Classification and interpretability. In terms of classification and interpretability, comparing the average ranking with other rule- and tree-based methods in Table 4.3, it is shown that CLASSY performs equally well while also able to find rule lists with fewer conditions, *without any parameter tuning*. 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 4.2, 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.

Overfitting. In terms of overfitting, Table 4.5 shows that CLASSY tends 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.

4.4 Conclusions

We proposed CLASSY, a heuristic algorithm that finds good probabilistic rule lists for multiclass classification by greedily approximating our MDL-based formulation of the problem. CLASSY naturally trades off model complexity with predictive accuracy,

effectively avoiding overfitting with very few hyperparameters.
We empirically demonstrated, on a variety of datasets, that CLASSY finds predictive rule lists that perform on par with state-of-the-art interpretable classifiers for predictive accuracy, even though some form of hyperparameter tuning is done for all methods except for CLASSY. Moreover, the models found by our approach are more compact than those obtained by the other methods, which we expect to make them more understandable in practice. Finally, we show that compression strongly correlates with predictive accuracy, which can be regarded as an empirical validation of the MDL-based selection criterion.

Limitations. The CLASSY algorithm is restricted to discretized input variables and multiclass problems. Most machine learning problems in tabular data involve a mix of binary, nominal and numeric input variables. Nonetheless, the formulation for those problems is available in Chapter 3, and one could directly use the RSD algorithm of Chapter 5 for multiclass problems. However, extending it to regression would require adaptations of the greedy gain used, as the problem is not as well defined as classification. In terms of statistical properties, and contrary to RSD, each rule added to the list per iteration does not minimize a specific statistical test, except decreasing the overall MDL score.