Information-theoretic partition-based models for interpretable machine learning

Yang, L.

Chapter 3

Probabilistic Truly Unordered Rule Sets

This chapter consists of a paper titled *Probabilistic Truly Unordered Rule Sets* (submitted to JMLR), which describes a refined version of the TURS model (in comparison to Chapter 2).
For being self-contained, Chapter 3 inevitably contains some repeated content from Chapter 2, including notation descriptions, basic definitions, and some identical related work discussions. However, the definition, model selection criterion, and algorithm for learning a TURS model are all refined based on Chapter 2. The differences between Chapter 2 and 3 are briefly discussed at the end of Section 3.1, and more thoroughly in the Appendix of this chapter.

Chapter Abstract

Rule set learning has been frequently revisited because of its interpretability. Existing methods have several shortcomings though. First, most existing methods impose orders among rules, either explicitly or implicitly, which makes the models less comprehensible. Second, due to the difficulty of handling conflicts caused by overlaps (i.e., instances covered by multiple rules), existing methods often do not consider *probabilistic* rules. Third, learning classification rules for multi-class target is understudied, as most existing methods focus on binary classification or multi-class classification via the "one-versus-rest" approach.

To address these shortcomings, we propose TURS[1], for Truly Unordered Rule Sets. To resolve conflicts caused by overlapping rules, we propose a novel model that exploits the probabilistic properties of our rule sets, with the intuition of only allowing rules to overlap if they have similar probabilistic outputs. We next formalize the problem of learning a TURS model based on the MDL principle and develop a carefully designed heuristic algorithm. We benchmark against a wide range of rule-based methods and demonstrate that our method learns rule sets that have lower model complexity and highly competitive predictive performance. In addition, we empirically show that rules in our model are empirically "independent" and hence truly unordered.

[1] A refined version based on Chapter 2.

3.1 Introduction

Despite the great success of black-box models in a wide range of tasks, intrinsically interpretable machine learning models have also received a lot of attention due to their transparency and hence their applicability to sensitive real-world scenarios, such as health care and judicial systems (Rudin 2019). We particularly focus on modelling and learning probabilistic rule sets for multi-class classification.

A probabilistic rule is in the form of **IF** X **meets certain conditions, THEN** $P(Y) = \hat{P}(Y)$, in which X represents the feature variables, Y the target variable, and $\hat{P}$ the associated class probability estimator.

Rule-based methods have the unique advantage that they are not only accessible and interpretable to statisticians and data scientists but also to domain experts, since rules can be directly read. While a single rule summarizes a local pattern from the data and hence only describes a subset of the instances, existing rule-based methods adopt various approaches to put individual rules together to form a global predictive model.

For instance, rule lists (or decision lists) (Fürnkranz et al. 2012) connect all individual rules by the "**IF ... (possibly multiple) ELSE IF ... ELSE ...**" statement, which is equivalent to specifying an explicit order for each rule. This approach is compatible with the very efficient divide-and-conquer algorithms: when a rule is induced from data, the covered instances (i.e., instances satisfying the condition of the rule) can be removed and hence iteratively simplify the search space. While this approach is very efficient, it comes at the cost of interpretability. As the condition of each rule depends on all preceding rules, comprehending a single rule requires going over (the negations of) all preceding rules' conditions, which is impractical when the rule list becomes large.

On the other hand, rule set models put rules together without specifying explicit orders. In this case, an instance can be covered by one single rule or simultaneously by multiple rules. When the instances covered by two or more rules have intersections, we say that these rules *overlap*. Although existing rule set methods claim that individual rules in rule sets are unordered (Clark and Boswell 1991; Kotsiantis 2013; Van Leeuwen and Knobbe 2012), we argue that they are not truly unordered. In fact, when one instance is covered by multiple rules at the same time, different rules may give conflicting (probabilistic) predictions. As a

result, ad-hoc schemes are widely used to resolve the conflicts, typically by ranking the involved rules with certain criteria (e.g., accuracy) and always selecting the highest ranked rule (Lakkaraju et al. 2016; Zhang and Gionis 2020). This approach, however, imposes implicit orders among rules, making rules entangled instead of truly unordered.

Implicit orders in rule sets severely harm interpretability, especially from the perspective of comprehensibility. While no agreement has been reached on the precise definition of interpretability of machine learning models (Molnar 2020; Murdoch et al. 2019), we specifically treat interpretability with domain experts in mind. In particular, to explain a single prediction for an instance to domain experts when implicit orders exist, it is insufficient to only provide the rules that the instance satisfies, because other *higher-ranked* rules that the instance does *not* satisfy are also a necessary part of the explanation. For example, imagine a patient is predicted to have *Flu* because they have *Fever*. If the model also contains the higher-ranked rule *"Blood in stool* $\rightarrow$ *Dysentery"*, the explanation should include the fact that *"Blood in stool"* is not true, because otherwise the prediction would change to *Dysentery*. If the model contains many rules, however, it becomes impractical to go over all higher-ranked rules for each prediction.

Additionally, decision trees, which can broadly be viewed as a rule-based approach, often have rules (root-to-leaf paths) that share multiple attributes due to their inherent structure. This can result in overly lengthy rules (as we will also empirically demonstrate in the Experiment section), since some internal nodes may not contribute to the classification itself but only serve to maintain the tree structure. Thus, decision trees are often less compact than decision rules, and consequently it is more difficult for domain experts to grasp the internal decision logic, and hence also the explanations for single predictions.

Given these shortcomings of existing rule-based models, we introduce *truly unordered rule sets (TURS)*, with the following properties. First, unlike most recently proposed rule sets/lists methods that only predict labels as outputs (Dash et al. 2018; Wang et al. 2017; Yang et al. 2021; Yang et al. 2017), we aim for formalizing a set of rules as a probabilistic model in a principled way. Since rule-based methods are potentially most applicable in sensitive areas, probabilistic predictions are much more useful for decision making and knowledge discovery, especially when domain experts are responsible for taking actions, such as in

health care. Probabilistic rules also allow us to directly apply our model for multi-class classification tasks, without leveraging the commonly used "one-versus-rest" paradigm (Clark and Boswell 1991; Hühn and Hüllermeier 2009). Second, we aim to develop a method to learn a set of probabilistic rules without implicit orders: to achieve this, we "allow" rules to overlap only if they have similar probabilistic outputs. In this case, when one instance is covered by multiple rules, it does not matter much even if we randomly pick one of these rules for prediction, since the differences of the prediction given by each individual rule is controlled. Thus, each rule becomes "self-standing" and can be used for explaining the predictions alone.

Particularly, we formally define a *truly unordered rule set* (TURS) as a probabilistic model, i.e., given a TURS model denoted as M and a dataset D, the likelihood of the target values conditioned on the feature values is defined. Notably, without putting implicit orders among rules, instances covered by multiple rules are modelled in a subtle manner such that the resulting likelihood is "penalized" if these overlapping rules have very different probabilistic outputs. Thus, we leverage our formal definition of TURS model and incorporate the probabilistic output differences into the goodness-of-fit of our probabilistic model, *without* introducing any hyper-parameter to control the probabilistic output differences of overlapping rules. Further, we treat the problem of learning a TURS model from data as a probabilistic model selection task, and hence further design a model selection criterion based on the minimum description length (MDL) principle (Grünwald 2007; Grünwald and Roos 2019), which does not require a regularization parameter to be tuned.

We resort to heuristics for optimization as the search space combined with the model selection criterion do not allow efficient search. Yet, we carefully and extensively extend the common heuristic approach for learning decision rules from data, in the following aspects. First, we consider a "learning speed score" heuristic, i.e., the decrease of our optimization score (to be minimized) *per extra covered instance* as the quality measure for searching the next "best" rule. Second, we take a novel beam search approach, such that 1) the degree of "patience" is considered by using a diverse beam search approach, and 2) an auxiliary beam together with a "complementary" score is proposed, in order to resolve the challenge that rules that have been added to the rule set may become obstacles for new rules. This challenge comes along with the fact that, unlike existing rule set methods, we

do consider overlaps of rules in the process of learning rules from data. Third, we propose an *MDL-based local testing method* in order to characterize whether the "left out" instances during the process of refining a rule can be well covered by rules we search for later. That is, while existing heuristics in rule learning only characterize the "quality" of the individual rules in different ways, our local testing criterion can be regarded as a look-ahead strategy.

In summary, our main contributions in this chapter are as follows:

1. In contrast to most recently proposed rule lists/sets methods that focus on non-probabilistic modelling and binary classification, we propose a principled way to formalize rule sets as probabilistic models that arguably provides more transparency and uncertainty information to domain experts in sensitive areas such as health care. It can also handle multi-class classification naturally, without resorting to the one-versus-rest scheme.

2. While existing rule sets methods adopt an ad-hoc approach to deal with conflicts caused by overlaps, often by always following the rule that scores the best according to a pre-defined criterion (e.g., accuracy or F-score), we identify that this approach puts implicit orders among rules that can severely harm interpretability. To tackle this issue, we propose to only "allow" overlaps that are formed by rules with similar probabilistic outputs. We formally define the TURS model, for Truly Unordered Rule Sets, in a way such that the probabilistic output difference among overlapping rules is incorporated in the goodness-of-fit as measured by the likelihood.

3. We formalize the problem of learning a TURS model from data as a probabilistic model selection task. We further propose an MDL-based model selection criterion that automatically handles the trade-off between the goodness-of-fit and model complexity, without any hyper-parameters to be tuned by the time-consuming cross-validation.

4. We develop a heuristic optimization algorithm with considerable algorithmic innovations. We benchmark our model TURS together with the proposed algorithm with extensive empirical comparisons against a wide range of rule-based methods. We show that TURS has superior performance in the following aspects: 1) it has very competitive predictive performance (measured by ROC-AUC); 2) it can *empirically* learn truly unordered rules: the

probabilistic conflicts caused by overlaps are negligible, in the sense that the influence is little even if we predict for instances covered by multiple rules by randomly picking one single rule from these rules; 3) TURS learns a set of rules with class probability estimates that can generalize well to unseen data; and 4) it produces simpler models in comparison to competitor algorithms.

Comparison with our previous work. This chapter is based on the previous chapter (Chapter 2), with vast extensions and modifications in all components, including probabilistic modelling, model selection criterion, algorithmic approach, and experiments. We summarize the key difference points between this chapter and the previous chapter as follows. First of all, we developed a completely new algorithm, with 1) a learning-speed-score heuristic motivated by the "normalized gain" used in the CLASSY algorithm for rule lists (Proença and Leeuwen 2020); 2) a diverse beam search approach with diverse "patience", in which the concept of patience is taken from the PRIM method (Friedman and Fisher 1999) for regression rules; 3) an innovative extension to the normal beam search approach, in the sense that we propose to use an auxiliary beam together with the "main" beam (and hence we simultaneously keep two beams); and 4) an MDL-local-test that serves as a look-ahead strategy for instances that are not covered for now. Further, we substantially extend the experiments in various aspects, and we now demonstrate that we can empirically treat the rule sets induced from data as truly unordered, in the sense that if a instance is covered by multiple rules we can now randomly pick one single rule for prediction, with negligible influence on the predictive performance (measured by ROC-AUC). Lastly, we also make a moderate modification to our optimization score. We discuss all these differences more in detail in the Appendix.

Organization of the chapter. The remainder of the chapter is structured as follows. In Section 3.2 we review related work. In Section 3.3 we present how to formalize a rule set as a probabilistic model, with the key component of how to model the instances covered by overlaps, i.e., by multiple rules at the same time. In Section 3.4, we discuss our model selection approach for learning a the truly unordered rule set, and formally define the model selection criterion based on the minimum description length (MDL) principle. In Section 3.5, we motivate and discuss our heuristics for learning the rule sets, and next present our proposed algorithm. Finally, we discuss our experiment setup and demonstrate

our experiment results in Section 3.6.

Algorithm	Model type	Rule learning strategy	Probabilistic	Handle overlap conflicts
CBA	ordered rule list	divide and conquer	✓	explicit order
CN2-ordered	ordered rule list	divide and conquer	✓	explicit order
PART	ordered rule list	divide and conquer	✓	explicit order
CLASSY	ordered rule list	divide and conquer	✓	explicit order
RIPPER	ordered list of rule sets	divide and conquer	×	explicit order
C4.5 rules	ordered list of rule sets	one-versus-rest	×	explicit order
BRS	rule set (binary target)	rules for positive class only	×	no conflict
CG	rule set (binary target)	rules for positive class only	×	no conflict
Submodular	rule set (binary target)	rules for positive class only	×	no conflict
CN2-unordered	rule set	one-versus-rest	✓	ad-hoc (weighted average)
FURIA	fuzzy rule set	one-versus-rest	✓	fuzzy (weighted average)
CMAR	rule set	association rule mining	×	ad-hoc (implicit orders, χ^2)
CPAR	rule set	association rule mining	×	ad-hoc (implicit orders, accuracy)
IDS	rule set	optimization for multi-class target	×	ad-hoc (implicit orders, F1-score)
DRS	rule set	optimization for multi-class target	×	ad-hoc (implicit orders, accuracy)
TURS (ours)	**truly unordered rule set**	**optimization for multi-class target**	✓	**Not needed**

Table 3.1: Summary of the algorithms' key properties.

3.2 Related Work

We next review the related work and we categorize them as follows. First, we discuss rule list methods, in which no overlap among rules exists by definition. Second, we review previous methods that learn rules for a single class labels, and based on it, the one-versus-rest rule learning methods. Last, we discuss rule sets methods for multi-class targets, as well as several different but related methods such as association rule mining. We summarize the key properties of closely related methods in Table 3.1.

Rule lists. Rules in a rule list are connected by IF-THEN-ELSE statements, and hence are with explicit orders. When classifying an instance, rules in the rule list are checked sequentially: once a rule is found of which the condition is satisfied by the instance, that single rule is used for prediction. Existing methods include CBA (Liu et al. 1998), ordered CN2 (Clark and Niblett 1989), PART (Frank and Witten 1998), and the more recently proposed CLASSY (Proença and Leeuwen 2020) and Bayesian rule list (Yang et al. 2017). Although these methods are often efficient by leveraging the divide-and-conquer (i.e., sequential covering) approach, rule lists are more difficult to interpret than rule sets because of their explicit orders. To comprehend the conditions of each rule, conditions in all preceding rules must also be taken into account; thus, the condition of each individual rule may not be meaningful when domain experts examine it separately (except for the first one).

One-versus-rest rule learning. This category focuses on only learning rules for a single class label, i.e., the "positive" class, which is already sufficient for binary classification (Dash et al. 2018; Quinlan 1990; Wang et al. 2017; Yang et al. 2021). That is, if an instance satisfies at least one of the induced rules, it can be classified as "positive", and otherwise negative. As all rules output the "positive" class, no prediction conflicts exist by definition. Recently, this line of research focuses on adopting discrete optimization techniques with provably better theoretical properties than heuristic algorithms; however, they suffer from the following drawbacks. First, these methods are non-probabilistic by definition, and hence it is not clear how to estimate the class probability for the instances covered by multiple rules (i.e., in the overlap). Second, no explicit explanation exists for those instances that are classified into the "negative" class; instead, the explanations for the negative class depend on the negation of all rules for the positive class, which can be overly complicated to comprehend when the number of rules is large. Third, these methods require discretizing and binarizing the feature matrix, and hence can only afford rather coarse search granularity for continuous-valued features, due to the high memory cost.

Further, learning rules for a single class label can be extended to multi-class classification, through the one-versus-rest paradigm. Existing methods mostly take the following two approaches to achieve this. The first, taken by RIPPER (Cohen 1995) and the C4.5 decision rule method (Quinlan 2014), is to learn each class in a certain order. After all rules for a single class have been learned, all covered instances are removed (or those with this class label). The resulting model is essentially an ordered list of rule sets, and hence is more difficult to interpret than a rule set.

The second approach does no impose an order among the classes; instead, it learns a set of rules for each class against all other classes. The most well-known are unordered-CN2 and FURIA (Clark and Boswell 1991; Hühn and Hüllermeier 2009). FURIA avoids dealing with conflicts of overlaps by essentially using all (fuzzy) rules for predicting unseen instances; i.e., the rules' outputs are weighted by the so-called "certainty factor". As a result, it cannot provide a single rule to explain its prediction. Unordered-CN2, on the other hand, handles overlaps by estimating the class probability as the weighted average of the class probability estimates for all individual rules involved in the overlap. That is, unlike our

method, CN2 adopts a post-hoc conflict resolving scheme, and as a result the issue of probabilistic conflicts is ignored during the training phase of CN2.

Multi-class rule sets. Very few methods exist for formalizing learning rules for multi-class classification as an optimization task directly (like our method), which leads to algorithmically more challenging tasks than the one-versus-rest paradigm, as the separate-and-conquer strategy is not applicable. To the best of our knowledge, the only existing methods are IDS (Lakkaraju et al. 2016) and DRS (Zhang and Gionis 2020). Both are neither probabilistic nor truly unordered. To handle conflicts of overlaps, IDS follows the rule with the highest F1-score, and DRS uses the most accurate rule. As we elaborated in Section 3.1, this approach imposes implicit orders and thus harms the comprehensibility of the model.

Decision trees and association rules. Other related approaches include the following. To begin with, decision tree based methods such as CART (Breiman et al. 1984) and C4.5 (Quinlan 2014) produce rules that are forced to share many "attributes" and hence are longer than necessary, as we will empirically demonstrate in Section 3.6.

Besides, a large category of methods is associative rule classification, which is to build rule-based classifiers based on existing association rule mining algorithms (Abdelhamid and Thabtah 2014). Association rule mining is known to have the problem of inducing redundant rules (Chen et al. 2006), hence a single instance can be easily covered by potentially many rules at the same time. As a result, various ad-hoc schemes have been proposed for handling the prediction conflicts of rules.

For instance, CMAR (Li et al. 2001) first groups rules based on their (different) predicted class labels for a given instance, and next constructs a contingency table for the whole dataset based on 1) whether an instance is covered by the group of rules and 2) the class label of each instance. Then the group of rules (and hence the conflicting class labels) is ranked with the χ^2 statistic. Moreover, CPAR (Yin and Han 2003) extends the sequential covering approach taken by FOIL (Quinlan 1990): instead of removing covered instances, the weights of covered instances are downgraded, in order to guide the search algorithm to focus on uncovered instances, and then resolves the prediction conflicts based on ranking the rules with the expected accuracy.

Lastly, the 'lazy learning' approach for associative rule classification, which

focuses on learning a single rule for every test (unseen) instance separately with a given training set of instances, can also avoid the conflicts of overlaps (Veloso et al. 2006). As a result, the lazy learning approach will not construct a global rule set model that describes the whole dataset, and hence provide less transparency for domain experts than our method.

3.3 Truly Unordered Rule Sets

We first formalize individual rules as *local* probabilistic models, and then define rule sets as *global* probabilistic models. The key challenge lies in how to define $P(Y = y|X = x)$ for an instance (x, y) that is covered by multiple rules.

3.3.1 Probabilistic rules

Denote the input random variables by $X = (X_1, \ldots, X_d)$, where each X_i is a one-dimensional random variable representing one dimension of X, and denote the categorical target variable by Y together with its domain $\mathscr{Y}$ that contains all unique class labels. Further, denote the dataset from which the rule set can be induced as $D = \{(x_i, y_i)\}_{i\in[n]}$, or (x^n, y^n) for short. Each (x_i, y_i) is an instance. Then, a probabilistic rule S is written as

$$(X_1 \in R_1 \wedge X_2 \in R_2 \wedge \ldots) \rightarrow P_S(Y), \tag{3.1}$$

where each $X_i \in R_i$ is called a *literal* of the *condition* of the rule. Specifically, each R_i is an interval (for a quantitative variable) or a set of categorical levels (for a categorical variable).

A probabilistic rule of this form describes a subset S of the full sample space of X, such that for any $x \in S$, the conditional distribution $P(Y|X = x)$ is approximated by the probability distribution of Y conditioned on the event $\{X \in S\}$, denoted as $P(Y|X \in S)$. Since in classification Y is a discrete variable, we can parametrize $P(Y|X \in S)$ by a parameter vector $\vec{\beta}$, in which the jth element β_j represents $P(Y = j|X \in S)$, for all $j \in \mathscr{Y}$. We therefore denote $P(Y|X \in S)$ as $P_{S,\vec{\beta}}(Y)$, or $P_S(Y)$ for short. To estimate $\vec{\beta}$ from data, we adopt the maximum likelihood estimator, denoted as $P_{S,\hat{\vec{\beta}}}(Y)$, or $\hat{P}_S(Y)$ for short.

Further, if an instance (x, y) satisfies the condition of rule S, we say that

(x, y) is *covered* by S. Reversely, the *cover* of S denotes the instances it covers. When clear from the context, we use S to *both represent the rule itself and/or its cover*, and define the number of covered instances $|S|$ as its *coverage*.

3.3.2 The TURS model

We aim for defining a rule set model with the following properties. First, each individual rule can be regarded as a reliable local pattern and generalizable probabilistic model that can serve as an explanation for the model's predictions. Second, if certain rules overlap with each other, i.e., some instances are covered by multiple rules simultaneously, then the probabilistic outputs of these rules "must be similar enough", in the sense that the likelihood of a TURS model given a fixed dataset incorporates (and penalizes) the differences of probabilistic outputs of overlapping rules.

Given a rule set with K individual rules, denoted as $M = \{S_i\}_{i \in [K]}$, any instance (x, y) falls into one of three cases: 1) exactly one rule covers x; 2) at least two rules cover x; and 3) no rule in M covers x. We formally define the model M as follows.

Covered by one rule only. Given a single rule denoted as S, when $x \in S, S \in M$ and $x \notin M \setminus S$, we define

$$P(Y|X = x) = P(Y|X \in S) = P_S(Y), \ \ \forall x \in S, x \notin M \setminus S \tag{3.2}$$

in which $P_S(Y)$ can be estimated from data. That is, we use $P_S(Y)$ to "approximate" the conditional probability $P(Y|X = x)$. To estimate $P_S(Y)$ we adopt the maximum likelihood (ML) estimator based on all instances covered by S. We define the ML estimator as $\hat{P}_S(Y)$, and let

$$\hat{P}_S(Y = j) = \frac{|\{(x, y) : x \in S, y = j\}|}{|S|}, \forall j \in \mathscr{Y}. \tag{3.3}$$

Note that we intentionally *do not exclude* instances in S that are also covered by other rules (i.e., in overlaps) for estimating $P_S(Y)$. Hence, the probability estimation for each rule is independent of other rules; as a result, each rule is *self-standing*, which forms the foundation of a truly unordered rule set.

Covered by multiple rules. For the second case when $x \in \bigcap_{i \in I} S_i, I \subseteq [K]$, we

define

$$P(Y|X = x) = P(Y|X \in \bigcup_{i \in I} S_i), \quad \forall x \in \bigcap_{i \in I} S_i, I \subseteq [K] \tag{3.4}$$

in which $P(Y|X \in \bigcup_{i \in I} S_i)$ is to be estimated from data with the ML estimator, defined and denoted as

$$\hat{P}(Y = j|X \in \bigcup_{i \in I} S_i) = \frac{|\{(x, y) : x \in \bigcup_{i \in I} S_i, y = j\}|}{|\bigcup_{i \in I} S_i|}, \forall j \in \mathscr{Y}. \tag{3.5}$$

Note that we take the union $\bigcup_{i \in I} S_i$ for the instances covered by the overlap (i.e., intersection) $\bigcap_{i \in I} S_i$. As counter-intuitive as it may seem at first glance, this subtle definition plays a key role in our modelling: with this novel definition, the likelihood of the data given the model—as the measure of the model's goodness-of-fit—automatically incorporates the differences between the rules' probabilistic outputs if they form an overlap.

Without loss of generalization, consider two rules denoted as S_i and S_j. When $P_{S_i}(Y)$ and $P_{S_j}(Y)$ are very similar, the conditional probability conditioned on the event $\{S_i \cup S_j\}$, denoted as $P(Y|S_i \cup S_j)$, will also be similar to both $P_{S_i}(Y)$ and $P_{S_j}(Y)$. In this case, it does not matter which of these three (i.e., $P_{S_i}(Y)$, $P_{S_j}(Y)$, or $P(Y|S_i \cup S_j)$) we use to model $P(Y|X = x), \forall x \in S_i \cap S_j$, in the sense that the "goodness-of-fit" measured by the likelihood of all instances covered by the overlap $S_i \cap S_j$ would be all similar.

On the other hand, when $P_{S_i}(Y)$ and $P_{S_j}(Y)$ are very different, the goodness-of-fit would be poor when using $P(Y|S_i \cup S_j)$ for estimating $P(Y|X = x)$ for $x \in S_i \cap S_j$. Thus, we leverage this property to penalize "bad" overlaps by incorporating the probabilistic goodness-of-fit in our model selection criterion that will be discussed in detail in Section 3.4.

Covered by no rule. When no rule in M covers x, we say that x belongs to the so-called "else rule" that is part of every rule set and equivalent to $x \notin \bigcup_i S_i$. Thus, we approximate $P(Y|X = x)$ by $P(Y|X \notin \bigcup_i S_i)$. We denote the else rule by S_0, which is the only rule in every rule set that depends on the other rules and is therefore not self-standing; however, it will also have no overlap with other rules by definition.

TURS as a probabilistic model. Building upon our definition for modelling the conditional class probability and the maximum likelihood estimators, we can

now formally define truly unordered rule sets as probabilistic models. Formally, a rule set M as a probabilistic model is a family of probability distributions, denoted $P_{M,\theta}(Y|X)$ and parametrized by θ. Specifically, θ is a parameter vector representing all necessary probabilities of Y conditioned on events $\{X \in G\}$, where G is either a single rule (including the else-rule) or the union of multiple rules. θ is estimated from data by estimating each $P(Y|X \in G)$.

The resulting estimated vector is denoted as $\hat{\theta}$ and contains $\hat{P}(Y|X \in G)$ for all $G \in \mathscr{G}$, where $\mathscr{G}$ consists of all individual rules and the unions of overlapping rules in M. To simplify the notation, we denote $(x, y) \vdash G$, for the following two cases: 1) when G is a single rule (including the else rule), then $(x, y) \vdash G \iff x \in G$; and 2) when G is a union of multiple rules, $G = \bigcup S_i$, then $(x, y) \vdash G \iff x \in \bigcap S_i$. By assuming the dataset $D = (x^n, y^n)$ to be i.i.d., we have

$$P_{M,\theta}(y^n|x^n) = \prod_{G \in \mathscr{G}} \prod_{(x,y) \vdash G} P(Y = y|X \in G). \tag{3.6}$$

3.3.3 Predicting for a new instance

When an unseen instance x' comes in, we predict $P(Y|X = x')$ depending on whether x' is covered by one rule, multiple rules, or no rule. An important question is whether we always need access to the training data, i.e., whether the probability estimates we obtain from the training data points are sufficient for predicting $P(Y|X = x')$, especially when x' is covered by multiple rules by which no instance in the training data is covered simultaneously.

For instance, if x' is covered by two rules S_i and S_j, $P(Y|X = x')$ is then predicted as $\hat{P}(Y|X \in S_i \cup S_j)$. However, if there are no training data points covered both by S_i and S_j, then we would not obtain $\hat{P}(Y|X \in S_i \cup S_j)$ in the training phase. Nevertheless, in this case we have $|S_i \cup S_j| = |S_i| + |S_j|$, and hence

$$\hat{P}(Y|X \in S_i \cup S_j) = \frac{|S_i|\hat{P}(Y|X \in S_i) + |S_j|\hat{P}(Y|X \in S_j)}{|S_i| + |S_j|}. \tag{3.7}$$

By contrast, when x' is covered by one rule only or no rule, the corresponding class probability estimation is already obtained during the training phase. Thus, we conclude that access to the training data is not necessary for prediction.

3.4 Rule Set Learning as Probabilistic Model Selection

Exploiting the formulation of rule sets as probabilistic models, we define the task of learning a rule set as a probabilistic model selection problem. Specifically, we use the minimum description length (MDL) principle for model selection.

The MDL principle is one of the best off-the-shelf model selection methods and has been widely used in machine learning and data mining (Galbrun 2022; Grünwald and Roos 2019). Although rooted in information theory, it has been recently shown that MDL-based model selection can be regarded as an extension of Bayesian model selection (Grünwald and Roos 2019).

The principle of MDL-based model selection is to pick the model, such that the code length (in bits) needed to encode the data given the model, together with the model itself, is minimized. We begin with discussing Normalized Maximum Likelihood (NML) distributions for calculating the bits for encoding the data given the model, followed by the calculation of the code length for encoding the model itself.

3.4.1 Normalized Maximum Likelihood Distributions for Rule Sets

As the Kraft inequality connects code length and probability[2], the core idea of the (modern) MDL principle is to assign a single probability distribution to the data given a rule set M (Grünwald and Roos 2019), the so-called *universal distribution* denoted by $P_M(Y^n|X^n = x^n)$. Informally, $P_M(Y^n|X^n = x^n)$ should be a representative of the rule set model—as a family of probability distributions—$\{P_{M,\theta}(y^n|x^n)\}_\theta$. The theoretically optimal "representative" is defined to be the one that has minimax regret, i.e.,

[2]Note that Section 3.4.1 — 3.4.2 describe the definitions of NML distributions and our proposed approximation for it, which were already introduced in Chapter 2 (Section 2.4.1 — 2.4.2). We deliberately keep the repeated content so that Chapter 3 is self-contained in describing the refined method for learning TURS models.

$$\arg\min_{P_M} \max_{z^n \in \mathscr{Y}^n} \Bigg[-\log_2 P_M(Y^n = z^n | X^n = x^n) \\ - \Big(-\log_2 P_{\hat{\theta}(x^n, z^n)}(Y^n = z^n | X^n = x^n) \Big) \Bigg]. \tag{3.8}$$

We write the parameter estimator as $\hat{\theta}(x^n, z^n)$ to emphasize that it depends on the values of (X^n, Y^n). The unique solution to P_M of Equation (3.8) is the so-called normalized maximum likelihood (NML) distribution (Grünwald 2007),:

$$P_M^{NML}(Y^n = y^n | X^n = x^n) = \frac{P_{M,\hat{\theta}(x^n, y^n)}(Y^n = y^n | X^n = x^n)}{\sum_{z^n \in \mathscr{Y}^n} P_{M,\hat{\theta}(x^n, z^n)}(Y^n = z^n | X^n = x^n)}. \tag{3.9}$$

That is, we "normalize" the distribution $P_{M,\hat{\theta}}(.)$ to make it a proper probability distribution, which requires the sum of all possible values of Y^n to be 1. Hence, we have $\sum_{z^n \in \mathscr{Y}^n} P_M^{NML}(Y^n = z^n | X^n = x^n) = 1$ (Grünwald and Roos 2019).

3.4.2 Approximating the NML Distribution

A crucial difficulty in using the NML distribution in practice is the computation of the normalizing term $\sum_{z^n} P_{\hat{\theta}(x^n, z^n)}(Y^n = z^n | X^n = x^n)$. Efficient algorithms almost only exist for exponential family models (Grünwald and Roos 2019), hence we approximate the term by the product of the normalizing terms for the individual rules.

NML distribution for a single rule. For an individual rule $S \in M$, we write all instances covered by S as (x^S, y^S), in which y^S can be regarded as a realization of the random vector of length $|S|$, denoted as Y^S. Y^S takes values in $\mathscr{Y}^{|S|}$, the $|S|$-ary Cartesian power of $\mathscr{Y}$. Consequently, following the definition of the NML distribution in Equation (3.9), the NML distribution for $P_S(Y)$ equals

$$P_S^{NML}(Y^S = y^S | X^S = x^S) = \frac{\hat{P}_S(Y^S = y^S | X^S = x^S)}{\sum_{z^S \in \mathscr{Y}^{|S|}} \hat{P}_S(Y^S = z^S | X^S = x^S)}. \tag{3.10}$$

Note that $\hat{P}_S$ depends on the values of z^S. As $\hat{P}_S(Y)$ is a categorical distribution,

it has been shown (Mononen and Myllymäki 2008) that the normalizing term can be written as $\mathcal{R}(|S|, |\mathscr{Y}|)$, a function of $|S|$—the rule's coverage—and $|\mathscr{Y}|$—the number of unique values that Y can take:

$$\mathcal{R}(|S|, |\mathscr{Y}|) = \sum_{z^S \in \mathscr{Y}^{|S|}} \hat{P}_S(Y^S = z^S | X^S = x^S), \tag{3.11}$$

and it can be efficiently calculated in sub-linear time (Mononen and Myllymäki 2008).

The approximate NML distribution. We propose to approximate the normalizing term of the NML distribution for rule set model P_M^{NML} as the product of the normalizing terms of P_S^{NML} for all $S \in M$:

$$P_M^{apprNML}(Y^n = y^n | X^n = x^n) = \frac{P_{M,\hat{\theta}(x^n,y^n)}(Y^n = y^n | X^n = x^n)}{\prod_{S \in M} \mathcal{R}(|S|, |\mathscr{Y}|)}. \tag{3.12}$$

Note that the sum over all $S \in M$ *does* include the "else rule" S_0. The rationale of using the approximate-NML distribution is as follows. First, it is equal to the NML distribution for a rule set without any overlap, as follows.

Proposition 1. *Given a rule set M in which for any $S_i, S_j \in M$, $S_i \cap S_j = \emptyset$, then $P_M^{NML}(Y^n = y^n | X^n = x^n) = P_M^{apprNML}(Y^n = y^n | X^n = x^n)$.*

Second, when overlaps exist in M, approximate-NML puts a small extra penalty on overlaps, which is desirable to trade-off overlap with goodness-of-fit: when we sum over all instances in each rule $S \in M$, the instances in overlaps are "repeatedly counted". Third, approximate-NML behaves like the Bayesian information criterion (BIC) asymptotically, which follows from the next proposition.

Proposition 2. *Assume M contains K rules in total, including the else rule. Under the mild assumption that $|S|$ grows linearly as the sample size n for all $S \in M$, then $\log\left(\prod_{S \in M} \mathcal{R}(|S|, |\mathscr{Y}|)\right) = \frac{K(|\mathscr{Y}|-1)}{2} \log n + \mathcal{O}(1)$, where $\mathcal{O}(1)$ is bounded by a constant w.r.t. to n.*

The proofs of these two propositions are shown in the Appendices of Chapter 2.

3.4.3 Code length of model

To obtain the final MDL based score, we next describe how to calculate the code length of the model, denoted as $L(M)$. The code length needed to encode the model depends on the encoding scheme we choose. Given the Kraft's inequality (Grünwald 2007), this can be practically treated as putting prior distributions on the model class. We describe the encoding scheme in a hierarchical manner due to the complexity of the model class.

Integer code for the number of rules. First, we encode the number of rules in the rule set, for which we use the standard Rissanen's integer universal code (Rissanen 1983). The code length needed for encoding an integer K is equal to

$$L_{rissanen}(K) = c + \log_2(K) + \log_2(\log_2(K)) + \log_2(\log_2(\log_2(K))) + \ldots;$$

the summation continues until a certain precision is reached (which we set as 10^{-5} in our implementation), and $c \approx 2.865$ is a constant.

Encoding individual rules. Next, we encode the each individual rule separately. For a given rule with k literals, we first encode k, the number of literals, by a *uniform code*: as k's range is bounded by the number of columns of the dataset, denoted by K_{col}, the code length needed to encode k is equal to

$$L_{num\_literal} = \log_2 K_{col}. \tag{3.13}$$

As each literal contains one unique variable, given the number of literals k, we further specify which are these k variables among all K_{col} variables, again with a uniform code. Thus, the code length needed to specify which these k variables are is equal to

$$L_{which\_vars} = \log_2 \binom{K_{col}}{k}. \tag{3.14}$$

Further, we sequentially encode the *operator* (i.e., '$\geq$' and/or '$<$') and the *value* of each literal. Specifically, for numeric variables, the literal is in either of the two forms: 1) $X \geq$ (or $<$) v, and 2) $v_1 \leq X < v_2$. As a result, we first need to encode the which form the literal is, which cost $L_{form} = 1$ bit. Next, to encode the values v [or $(v_1, v2)$], we need to know in advance the search space of v [or $(v_1, v2)$], which are chosen as quantiles in our algorithm implementation.

The number of candidate values (quantiles) for each numeric feature variable is a hyper-parameter, which we argue should be chosen based on the task at hand: it should be large enough without loss too much information for the prediction, while at the same time the computational budget and the prior knowledge on what is useful for interpreting the rules should also be taken into account in practice.

Denote the number candidate cut points *after excluding those that result in a rule with coverage equal to 0* as K_{value}. Depending on whether the literal contains one or two splits, we can further calculate the code length needed to encode the operator and value(s) in the literal, denoted as L_{value_op}, as

$$L_{value\_op} = L_{operator} + L_{form} + \log_2 K_{value}, \text{or} \tag{3.15}$$

$$L_{value\_op} = \log_2 \binom{K_{value}}{2} + L_{form}, \tag{3.16}$$

since for the former case we also need to encode the operator in the literal, i.e., "$\geq$" or "$<$", which cost $L_{operator} = 1$ bit. In contrast, the latter case has only one possibility for the operators, and hence requires 0 bit to encode it.

Next, for categorical variables with $\mathcal{L}$ levels, encoding a subset of l levels requires $L_{value\_op} = \log_2 \mathcal{L} + \log_2 \binom{\mathcal{L}}{l}$ bits; the former term, $\log_2 \mathcal{L}$, is needed for encoding the number l itself, and the latter one is code length needed to specify these l levels from $\mathcal{L}$ in total. For simplicity, in our implementation we assume all categorical features are one-hot encoded, and hence $L_{value\_op} = 1$.

To sum up, the number of bits needed for encoding an individual rule S, denoted as $L(S)$, is equal to

$$L(S) = L_{num\_literal} + L_{which\_vars} + \sum L_{value\_op}, \tag{3.17}$$

in which the term $\sum L_{value\_op}$ denotes the summation of the code length needed to encode the operator and value for each single literal.

Note that $2^{-L(S)}$ can be interpreted as a prior probability mass for S among all possible individual rules (Grünwald 2007). Moreover, because of the way we determine K_{value} (i.e., by excluding those candidate cut points that lead to rules with coverage equal to 0), the code length needed to encode a single rule does depend on the order of encoding each literal in the condition of the rule. This turns out to be desirable because of our algorithmic approach, which will be

described in Section 3.5.

Encoding the rule set. Based on the code length needed for single rules, we can now define the code length needed to encode the whole rule set. Given a rule set M with K rules, the total bits needed to encode M is

$$L(M) = L_{rissanen}(K) + \sum_{i=1}^{K} L(S) - \log_2(K!), \tag{3.18}$$

in which the last term is to eliminate the redundancy caused by the fact that the order of the rules in a rule set does not matter.

To see the rationale of introducing the term $(-\log_2(K!))$, consider the prior probability of each rule denoted as $P(S_i) = 2^{-L(S_i)}$. Then, the prior probability of the set of rules $\{S_1, ..., S_K\}$, conditioned on the fixed K, can be defined as

$$P(\{S_1, ..., S_K\}) = \sum \prod_{i=1}^{K} P(S_i) = (K!) \prod_{i=1}^{K} P(S_i), \tag{3.19}$$

in which the sum goes over all permutations of $\{S_1, ..., S_K\}$. Thus, we have $L(M) = L_{rissanen}(K) - \log_2 P(\{S_1, ..., S_K\})$, which connects the definition of $L(M)$ to the prior probability of M and hence justifies the introduction of the term $(-\log_2(K!))$ in Equation (3.18).

3.4.4 MDL-based model selection

After the describing the approximate normalized maximum likelihood distributions and the code length (number of bits) needed to specifying a model in the model class, we can now formulate the task of learning truly unordered rule sets as a model selection problem. That is, our goal is to search for the rule set, denoted as M^*, among all possible rule sets $\mathcal{M}$, such that

$$M^* = \arg\min_M L\left((x^n, y^n), M\right) := \arg\min_M \left[-\log_2 P_M^{apprNML}(Y^n = y^n | X^n = x^n) + L(M)\right], \tag{3.20}$$

in which $P_M^{apprNML}(Y^n = y^n | X^n = x^n)$ is defined in Equation (3.12) and $L(M)$ in Equation (3.18).

We refer to the proposed optimization function $L((x^n, y^n), M)$ as our model selection criterion; for a fixed model M and a (training) dataset (x^n, y^n), we refer to the value of $L((x^n, y^n), M)$ as the *MDL-based score for the rule set model M*.

3.5 Learning Truly Unordered Rules from Data

Given the combinatorial nature of the search space, learning rule sets from data is an extremely difficult task. Notably, although recently proposed algorithms can obtain provably optimal rule lists (Angelino et al. 2017) and decision trees (Hu et al. 2019), their branch-and-bound approaches are not applicable to learning TURS models due to the following reasons. First, our model class (and hence also search space) is different than that of rule lists and decision trees, since our TURS model allows for overlaps of rules. Second, the output of the TURS model is probabilistic while the optimal trees/lists algorithms learn rule-based models with non-probabilistic (or just binary) output. Third, our model selection criterion, although requiring no hyper-parameter for regularization, does not allow efficient search for the global optimum, as like most existing MDL-based approaches (Galbrun 2022). Hence, we cannot easily apply the branch-and-bound approaches as employed by the optimal tree/list algorithms.

As for rule set methods, traditional algorithms focus on defining heuristics that try to characterize the "quality" of individual rules in different ways, often without a global optimization score (Fürnkranz and Flach 2005; Fürnkranz et al. 2012). In addition, recently proposed ones mostly rely on randomized techniques: DRS (Zhang and Gionis 2020) is based on heuristic-based randomized algorithm, IDS (Lakkaraju et al. 2016) on stochastic local search, BRS (Wang et al. 2017) on simulated annealing, and CG (Dash et al. 2018) on (randomized) integer programming. However, BRS and CG are only suitable for binary target and non-probabilistic rules, while DRS and IDS turn out to have unsatisfactory predictive performance as shown in Section 3.6.

Therefore, we develop a heuristic-based algorithm for iteratively learning single rules with extensive innovations in comparison to traditional heuristic algorithms.

3.5.1 Learning a rule set

In the following, we start by describing the process of iteratively learning a rule set, followed by discussing the heuristic of defining the "best" single rule given the current status of the rule set. Then, we discuss how to learn a single rule in Section 3.5.2, in which we introduce a *diverse-patience dual-beam search*

algorithm, together with a novel look-ahead strategy that we propose based on the analogy between the MDL principle and hypothesis testing (Grünwald 2007, Chapter 14.3), which we hence name "MDL-based local testing".

Iteratively learning a rule set

Algorithm 3: Iteratively Learning a Rule Set

Data: dataset $D = (x^n, y^n)$
Result: rule set M

```
1 Initialize M                                          // Empty rule set.
2 while TRUE do
3     S ← Learn-Single-Rule(M, D)        // Described in Algorithm 4
4     if L(D, M ∪ {S}) < L(D, M) then
5         M ← M ∪ {S}     // The ``else-rule" updates accordingly
6     else
7         return rule set M
```

The process of learning a rule set iteratively, rule by rule, is shown in Algorithm 3. The algorithm starts with an empty rule set (in which all instances are covered by the "else-rule") [Line 1]. Then, the "best" single rule, defined as the one that maximizes what we call the *learning-speed-score* heuristic that is discussed in detail next, is learned from data [Line 3]. This single rule is added to the rule set if adding it to the rule set decreases the MDL-based model selection criterion defined Equation (3.20) [Lines 4-5]. This process is repeated until no new rule can be found that further optimizes our model selection criterion [Lines 2-9].

Heuristic score for a single rule

Consider the search space of all possible rule sets, adding one single rule to the rule set can be considered as one single "step" towards another "point" in the search space. As it is obviously meaningless to add a new rule that does not cover any previously uncovered instance, such a step always leads to a monotonic increase for the *coverage* of the rule set (excluding the else rule).

Therefore, we propose a heuristic that leads to the next rule (step) with the

steepest descent with respect to the increase in the coverage of rule set; that is, the next "best" single rule (step) is defined as the one that maximizes *the decrease of the MDL-based score per extra covered instance*. We hence name this heuristic as the *learning speed score*. Formally, given a rule set denoted as M, the learning speed score for a single rule S to be added to M is defined as

$$r(S) = \frac{L\left((x^n, y^n), M\right) - L\left((x^n, y^n), M \cup \{S\}\right)}{|M \cup \{S\}| - |M|}, \tag{3.21}$$

in which $M \cup \{S\}$ denotes the rule set obtained by adding the single rule S to M. Further, $|M|$ and $|M \cup \{S\}|$ respectively denotes the coverage before and after adding S to the rule set M (excluding the else-rule).

We next discuss how to search for the next best rule that optimizes $r(S)$.

3.5.2 Learning a single rule

For describing our algorithm for learning a single rule, we start with describing the general paradigm of applying beam search in learning a single rule, and then move forward to describe our three algorithmic innovations. Last, we put everything together and describe our proposed algorithm in detail.

Preliminary: Beam Search for Learning a single rule

Recall that the condition of a rule S can be written as the *conjunction* of literals, in which each literal takes the form of $\{X_i \in R_i\}$, with R_i representing an interval if X_i is a quantitative variable and a set of categorical levels if X_i is a categorical variable.

When applying a beam search in learning a single rule, we start with an *empty rule containing no literal that hence covers all instances*. Next, we enumerate all feature variables X_i to construct the search space of all possible single literals: for continuous-valued X_i, we pick quantiles as *splits points* and combine it with the operator ('$\geq$' or '$<$') to construct a literal, in which the "search granularity" (i.e., the number of quantiles) is a hyper-parameter that depends on the task at hand, as previously discussed in Section 3.4.3; for categorical variables, we assume they are all one-hot encoded for simplicity, and hence the possible literals are just $(X_i = 1)$ or $(X_i = 0)$. After enumerating all possible single literals, given a beam

width W, we rank these literals with a predetermined criterion, and then pick the top-W literals to be the W candidate rules of length one.

Next, for each of these W candidate rule of length one, we repeat the process of enumerating all possible single literals to append to this rule. We refer to these possible rules obtained by adding one more literal to a given rule as the *rule growth results*. Among all *rule growth results* of these W length-one candidate rules, we again pick the top-W length-two candidate rules, according to the predetermined criterion.

We can repeat this process until some stopping criterion is met, e.g., no rule growth result that can further optimize the model selection criterion can be found (or this has happened consecutively for a number of times). Lastly, among all these candidate rules with different lengths, we return the rule based on the heuristic that defines the "best" next rule (i.e., the learning speed score $r(.)$ in our case).

Note that we build our *diverse-patience dual-beam search* algorithm upon this general paradigm of applying beam search to learning a single rule with significant algorithmic innovations, as follows: 1) instead of using one single heuristic for searching for the next "best" rule, we introduce a look ahead strategy in the rule growth process; 2) instead of simply keeping the top-W rule growth results in the beam, we also monitor the diversity of "patience"; and 3) instead of a single beam, we introduce another auxiliary beam with a complementary score and we simultaneously keep two beams. The complementary score is proposed as we observe that allowing overlaps in rule sets leads to the algorithmic challenge that existing rules in the rule set may become obstacles to searching for new rules to be added to the rule set. We next describe these three heuristics in depth.

MDL-based local testing

When growing a rule S by adding a literal and obtaining its growth result S', we essentially leave out the instances covered by S *but not* S' to be covered potentially by rules we may obtain later. Existing rule learning heuristics often neglect this left-out part but focus only on characterizing the quality of the rule growth result S' itself. In contrast, we introduce a local test that can be viewed as a way of assessing whether it is better to model the instances in $\{S \setminus S'\}$ by the rule S (and hence discard S' and stop growing S), or to leave out the instances

in $\{S \setminus S'\}$ for "future" rules that we may obtain later.

Formally, consider a rule S, its growth result S', and the potentially left-out part, defined and denoted as $S_l = S \setminus S'$. We only proceed to consider S' as an appropriate rule growth candidate if

$$-\log_2 P_S^{NML}(y^S|x^S) > -\log_2 P_{S'}^{NML}(y^{S'}|x^{S'}) - \log_2 P_{S_l}^{NML}(y^{S_l}|x^{S_l}) + L_{split}, \tag{3.22}$$

in which $P_S^{NML}(y^S|x^S)$ is the NML-distribution when viewing a single rule as a local probabilistic model, defined in Equation (3.10). Further, L_{split} denotes the code length needed to encode the condition that splits S into S' and S_l. This requires specifying 1) the variable of the literal and 2) the numeric threshold or the categorical levels (which depends on the variable type), both with the uniform code as described in Section 3.4.3. That is, we only allow rule growth that satisfies the local test defined in Equation (3.22).

Intuitively, this is equivalent to building a depth-one decision tree for instances covered by S only, in which the left and right nodes are S' and S_l respectively. We then compare whether S on itself or S' *together with* S_l is a better local model, according to the MDL principle (Grünwald 2007). Recall that MDL-based model selection picks the model that minimizes the code length needed to encode the data together with the model; thus, if the local test is satisfied, we prefer the depth-one decision tree with nodes S' and S_l over the single-node tree with the only node S, and vice versa.

The rationale of the local test is that, by explicitly considering the local model for the left out part S_l, we incorporate the potential carried by the instances in S_l. That is, the local test we introduce can exclude those rule growth results of S that may leave out a subset of instances that are hard to model later. We empirically show in Section 3.5.2 that without MDL-based local testing, the learning speed score can be too greedy and hence the algorithm fails to reveal the ground-truth rule set model even in a simple simulated case.

Beam search with "patience" diversity

We now present the beam search with the patience diversity. For the simplicity of presentation, we now focus on describing the beam search with the "main" beam that adopts the learning speed score $r(S)$ as the heuristic, after which the

description of the complementary-score-assisted auxiliary beam immediately follows in Section 3.5.2.

Assuming the beam width is W, we start with a rule with empty condition which all instances satisfy. Next, we go over all possible rule growth results by adding one single literal. Furthermore, we keep the top-W rule growth results, with the following properties: 1) they satisfy the MDL-based local testing, defined previously in Section 3.5.2; 2) they have the highest learning speed score defined by $r(S)$ in Equation (3.21); and 3) they satisfy the patience diversity constraint, which we discuss below.

Motivation for "patience" diversity. While we aim to iteratively search for the rule with the best learning speed score $r(S)$ (Equation 3.21), it may be too greedy to directly use $r(S)$ to search for the next best literal (as a rule can contain multiple literals). Denote a rule as S and its growth result as S', we empirically observe that the coverage of S' can shrink drastically in comparison to that of S when directly using $r(S)$ for learning the next literal. However, a more "patient" search procedure with a moderate change in the coverage may be desirable in some cases, as a moderate decrease in coverage leaves many possibilities for adding more literals later. This concept of "patience" was first introduced in PRIM (Friedman and Fisher 1999), and we are the first to combine it with a beam search approach.

Specifically, we propose to use the beam search approach to keep the diversity of the patience, i.e., to have a variety of rule growth results, with diverse coverage *relative to the rule from which the rule growth result is obtained*.

Beam search with patience diversity. Given a potentially incomplete rule S, we search all candidate rules $\{S'\}$ that can be obtained by adding a literal to S (excluding those not satisfying the MDL-based local test).

Given a beam width W, we categorize all candidate rules, denoted as $\{S'\}$, into W clusters according to their coverage: the wth cluster is defined as:

$$\{S'\}_w = \{S' \in \{S'\} : \frac{|S'|}{|S|} \in \left[\frac{w-1}{W}, \frac{w}{W}\right)\}, \quad w \in \{1, ..., W\}; \tag{3.23}$$

i.e., all candidate rule growth results in $\{S'\}_w$ must satisfy the condition that its coverage divided by the coverage of S is in the interval $[(w-1)/W, w/W)$.

For each cluster, we search for the best growth result by optimizing the learning speed score $r(S')$. In this way, our beam search is diverse with regard to

the degree of "patience": when the coverage decreases by a small ratio only, the optimization is "patient" (by leaving a lot of possibilities for adding more literals); on the other hand, when the coverage decreases by a large ratio, the optimization is greedy (by leaving out little room for further refinement). We empirically show that adopting patience diversity improves the prediction performance of our method in Section 3.6.6.

Auxiliary beam with a complementary score

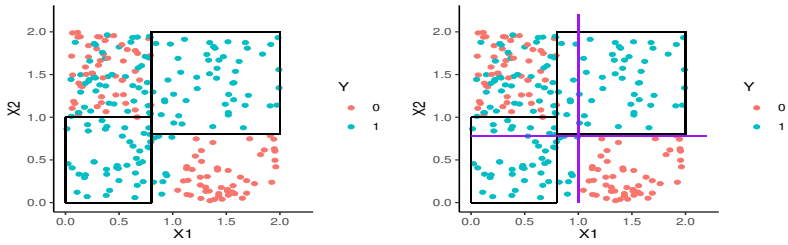

Figure 3.1: (Left) Simulated data with a rule set containing two rules (black outlines). (Right) Growing a rule to describe the bottom-right instances will create conflicts with existing rules. E.g., adding either $X_1 > 1$ (vertical purple line) or $X_2 < 0.8$ (horizontal purple line) would create a huge overlap that deteriorates the likelihood.

We now describe the auxiliary beam in our dual-beam approach. We start with the motivation for having an auxiliary beam, and next describe in detail the complementary score, as well as how we incorporate the auxiliary beam in the beam search algorithm.

Motivation for auxiliary beam. Recall that the learning speed score $r(S)$ evaluates the decrease of the MDL-based optimization score per extra covered instance when S is added to the rule set; thus, to maximize $r(S)$ we aim for obtaining a rule S that 1) improves the likelihood of the instances not covered by the rule set so far, and 2) has similar class probability estimates to those rules in the rule set that overlap with S. However, when iteratively searching for the next literal, the single literals we consider may not be able to contribute to both aims simultaneously.

Consider an illustrative example with data and a rule set with two rules (in black) in Figure 3.1 (left). If we want to grow a rule that covers the *bottom-right* instances, the existing rules form a blockade: the right plot shows how adding

either $X_1 > 1$ or $X_2 < 0.8$ to the empty rule (shown in purple) would create a large overlap with the existing rules, with significantly different probability estimates.

Our auxiliary beam is useful in cases like this, to keep literals like $X_1 > 1$ or $X_2 < 0.8$, which solely contributes to the first goal we discussed above, i.e., it improves the likelihood of the instances not covered by the rule set so far but creates a "bad" overlap with a large class probability difference. As a rule's class probability estimation is still up to change during the growing process, we can potentially "correct" bad overlaps by adding more literals later.

Thus, we propose an auxiliary beam together with a complementary score, informally defined as the *learning speed score calculated by ignoring the overlap created by the new rule that is being grown.* We next formally define the complementary score.

Complementary score. Formally, given a rule set M and a new rule S (i.e., $S \notin M$), the complementary learning speed after adding S to M, denoted as $R(S)$, is defined as

$$R(S) = \frac{L\left((x^n, y^n), M\right) - L\left((x^n, y^n), M \cup \{S \setminus M\}\right)}{|M \cup \{S \setminus M\}| - |M|} \tag{3.24}$$

in which $S \setminus M$ can be regarded as a "hypothetical" rule with the cover equal to the instances covered by rule S excluding the instances covered by rules already in M, and hence $L\left((x^n, y^n), M \cup \{S \setminus M\}\right)$ denotes the MDL-based score (as defined in Equation 3.20) after adding $S \setminus M$ to the rule set M.

Complementary-score-assisted beam search. We simultaneously keep two beams, both with a beam width W. Apart from the beam that keeps the top-W literals according to the learning speed score $r(.)$, we additionally keep an auxiliary beam that keeps the top-W literals according to the complementary score $R(.)$.

Further, the auxiliary beam must also satisfy the MDL-based local testing defined in Section 3.5.2, with the NML distribution calculated based on instances *excluding* those covered by the rule set. That is, consider a rule set M, a rule S with its growth result S', and the left-out part $S \setminus S' := S_l$, the local test for the

auxiliary beam is defined as

$$\begin{aligned} &- \log_2 P_S^{NML}(y^{S\setminus M}|x^{S\setminus M}) > \\ &\quad - \log_2 P_{S'\setminus M}^{NML}(y^{S'\setminus M}|x^{S'\setminus M}) - \log_2 P_{S_l\setminus M}^{NML}(y^{S_l\setminus M}|x^{S_l\setminus M}) + L_{split}. \end{aligned} \tag{3.25}$$

Additionally, the auxiliary beam must satisfy the patience diversity, as described in Section 3.5.2. The only difference is that the coverage of each rule is calculated based on $S \setminus M$ instead of S; i.e., instances that are already covered by the rule set M are ignored.

Algorithm description

We now put all heuristics together and describe in full our algorithm for finding the next rule, of which the pseudo code is provided in Algorithm 4.

With a rule set M that either contains no rule or some existing rules, we always start with an *Empty Rule* that contains no literals for its condition, and we initialize the "rules-for-next-iter" **[Line 2]** as an array containing the empty rule only.

For each iteration, we initialize a new beam an a new auxiliary beam **[Line 4-5]** with beam width W. The beam keeps the top-W rule growth results using the learning speed score defined in Equation (3.21); in contrast, the auxiliary beam keeps the W best rule growth results using the complementary score by ignoring the rule's overlap with M, as discussed in detail in Section 3.5.2.

Next, we use every rule in the "rules-for-next-iter" array as a "base" for growing **[Line 6-18]**. Specifically, given a rule, we first generate its candidate growth *by adding one literal only* **[Line 7]**. That is, we go over all feature variables in the dataset, and for each variable, we generate candidate literals with numeric thresholds (quantiles) or with categorical levels, based on the variable type.

Further, we cluster the generated candidates by their coverage (for the beam), as well as their coverage excluding the instances already covered by M (for the auxiliary beam) **[Line 8 & 12]**. We next filter out the candidates in "categorized-candidates" and "categorized-candidates-auxiliary" with the MDL-based local test defined in Section 3.5.2 **[Line 9 & 13]**. Further, we search for the best candidate in each cluster of the beam using $r(.)$, and each cluster of the auxiliary beam using $R(.)$ **[Line 10-11 & 14-15]**.

To check whether the growing process should be stopped after this iteration, we take a budget denoted as K_{stop}: we stop the beam search when this is the K_{stop}-th time in a row that both beams (the beam and the auxiliary beam) produce rules with worse scores ($r(.)$ for the "beam" and $R(.)$ for the "auxiliary-beam") than the previous beams [**Line 19**].

If the stopping criterion is not met, we first filter the beam and auxiliary beam to reduce the number of rules in each beam to be equal to the beam width W, as both of them now contain ($W * \text{length(rules-for-next-iter)}$) rules [**Line 22-23**]. Specifically, we sort all rules in the beam based on their coverage and categorize them into W clusters; next, for each cluster, we keep the top-W rules with the highest $r(.)$ for the "beam" and highest $R(.)$ for the "auxiliary-beam", as the base for rule growing for the next iteration. Last, we update "all-candidate-rules" and "rules-for-next-iter" [**Line 24-25**], and continue to the next iteration [**Line 3**]. The former is the pool we use for finally selecting the next best rule to be potentially added to M, and the latter contains all "base rules" for the next rule growth iteration, which contains all rules in the beam and the auxiliary beam.

Finally, if the stopping criterion is met, we return the rule S among "all-candidate-rules" with the best (largest) learning speed score $r(.)$ [**Line 20**].

3.6 Experiments

We extensively benchmark our diverse-patience dual-beam algorithm and we study the truly unordered rule sets (TURS) model learned from data in the following aspects:

1. Does the TURS model learned from data achieve on-par or better classification performance in comparison to other rule-based methods, especially rule set methods that allow (implicit) orders among rules?

2. Can rules in the TURS model learned from data be empirically treated as *truly unordered*?

3. Do the class probability estimates from rules in the induced TURS model generalize well to unseen (test) instances, such that these probability estimates are reliable to serve as part of the explanations for the (probabilistic) predictions?

4. Is the model complexity of the TURS model learned from data smaller than that of the rule-based models learned by competitor methods?
5. What are the effects of our proposed heuristics, including the beam search with patience diversity and the MDL-based local test?

3.6.1 Setup

Datasets. We conduct an extensive experiments with 31 datasets, summarized in Table 3.2. Our multi-class datasets are from the UCI repository, while the binary-class datasets are from both the UCI repository (Dua and Graff 2017) and the ADBench Github repository (Han et al. 2022). ADBench is a benchmark toolbox for anomaly detection (including imbalanced classification), and all datasets from it are marked in *italics* in Table 3.2.

Competitors. We compare against a wide ranges of methods, summarized as follows. First, we compare with *unordered* CN2 (Clark and Boswell 1991), which adopts the one-versus-rest strategy. As CN2 does not impose an implicit order among rules, it is conceptually the closest competitor to our method. Second, we compare with DRS (Zhang and Gionis 2020) and IDS (Lakkaraju et al. 2016), as they are the only two multi-class rule set methods without first learning rules for individual class labels and then leveraging the one-versus-rest strategy, to the best of our knowledge. Further, similar to us, they also incorporate the properties of overlaps in their optimization scores: DRS aims to minimize the size of overlaps, while IDS optimizes a linear combination of seven scores, one of which explicitly penalizes the size of overlaps. Third, we compare with CLASSY, a recently proposed ordered rule list method, as it uses a similar model selection approach based on the MDL principle. Fourth, since the MDL principle is conceptually related to Bayesian modelling, we also compare with BRS (Wang et al. 2017) as a representative method under the Bayesian framework, which also adopts an non-heuristic simulated annealing approach. Last, we include RIPPER (Cohen 1995), CART (Breiman et al. 1984), and C4.5 decision trees (Quinlan 2014), due to their wide use in practice.

Implementation details. For TURS, we set the beam width as 10, and the number of candidate cut points for numeric features as 20[3]. For competitor al-

[3]We observe that further increasing the number of candidate cut points for numeric features to 100, as well as the beam width to 20, makes no big difference on the predictive performance in general.

gorithms, we use CN2 from Orange (Demšar et al. 2013), IDS from a third-party implementation with proven scalability (Filip and Kliegr 2019), RIPPER and C4.5 from Weka (Hall et al. 2009) and its R wrapper, CART from Python's Scikit-Learn package (Pedregosa et al. 2011), and finally, DRS, BRS, CLASSY from the original authors' implementations. Competitors algorithms' configurations are set to be the same as the default as in the paper and/or in original authors' implementations. We make the code public for reproducibility[4].

All reported results in this section are based on five-fold stratified cross-validation, unless mentioned otherwise.

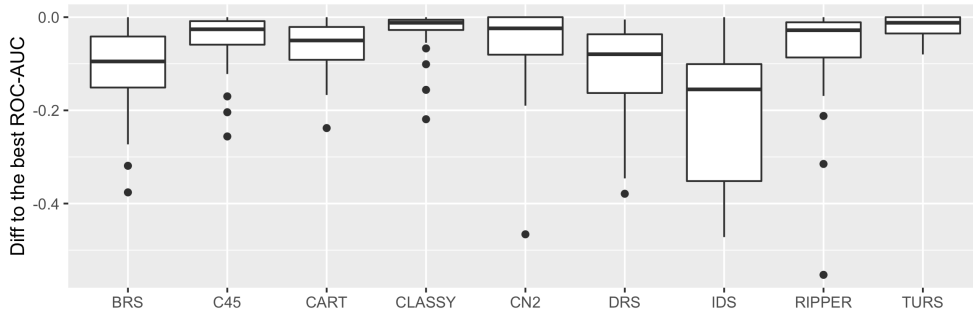

Figure 3.2: For each algorithm, we calculate for every individual dataset the difference between its ROC-AUC score and the best ROC-AUC scores. The differences to the best ROC-AUC scores for each algorithm is illustrated by a box-plot.

3.6.2 Classification performance

To investigate the classification performance for the TURS model learned from data, we report in Table 3.3 average ROC-AUC scores on the test sets obtained using five-fold *stratified* cross-validation. For multi-class classification, we report the "macro" one-versus-rest AUC scores, as "macro" AUC treats all class labels equally and hence can characterize how well the classifiers predict for the minority classes.

Note that BRS (Wang et al. 2017) can only be applied to binary datasets. Further, we fail to obtain the results of DRS on three datasets because the implementation of DRS makes it incapable of handling datasets with very large number

[4] `https://github.com/ylincen/TURS2`.

of columns[5]. We also fail to obtain the result of IDS on one dataset as it exceeds the predetermined time limit: 10 hours for one single fold of one dataset.

We show that TURS is very competitive in comparison to its competitors in the following aspects. First, TURS performs the best in 11 out of the total 31 datasets, and performs the best in 6 out of 11 multi-class datasets. We denote the best ROC-AUC for each dataset in bold. Second, we report the difference between TURS's ROC-AUC scores and the best ROC-AUC scores for each individual dataset, in the bracket in the table. This shows the gap between TURS and the best competitor for each individual dataset.

We further calculate the ROC-AUC scores of each competitor algorithm for each dataset, minus the best ROC-AUC score for each individual dataset, which measures the "gaps to the best" for each competitor algorithm. We compare these gaps-to-best scores for all competitor algorithms in Figure 3.2. The boxplots demonstrate that TURS is very stable for all 31 datasets we have tested, and in comparison to its competitors the gaps-to-best scores are much smaller.

Third, among all rule set methods (CN2, DRS, IDS, TURS), TURS shows substantially superior performance against DRS and IDS. As DRS and IDS both aim to reduce the size of overlaps, our results indicate that simply minimizing the sizes of overlaps may impose a too restricted constraint and hence lead to sub-optimal classification performance. On the other hand, CN2 is competitive in terms of obtaining the best AUCs, especially for binary datasets, as shown in Table 3.3. However, as shown in Figure 3.2, CN2 has in general larger gaps to the best AUCs than TURS does. Further, more comparison between TURS and CN2 will be presented in the following paragraphs.

3.6.3 Prediction with 'random picking' for overlaps

Recall that in our definition of the truly unordered rule set (TURS) model, we estimate the class probabilities for overlaps by considering the "union" of the covers of all involved rules. Thus, the next question we study empirically is whether our formalization of rule sets as probabilistic models can indeed lead to overlaps only formed by rules with similar probabilistic estimates.

Therefore, we compare the probabilistic predictions of our TURS models

[5]The key issue is that their implementation involves transforming a binary vector to an integer, and they use the "numpy" package for this, which does not support "arbitrarily large integers".

against the probabilistic predictions by what we call "random picking" for overlaps: when an unseen instance is covered by multiple rules, we randomly pick one of these rules, and use its estimated class probabilities (estimated from the training set) as the probabilistic prediction for this instance.

Intuitively, if the overlaps are formed only by rules with similar probabilistic output, we expect the probabilistic prediction performance by TURS and by "TURS with random-picking" (abbreviated as TURS-RP) to be very close. We report the ROC-AUC of TURS and TURS-RP in Table 3.4, together with the percentage of instances covered by more than one rules (the "%overlaps" column). The ROC-AUC scores are obtained using five-fold cross-validation, and specifically, for each fold, the "random picking" ROC-AUC is obtained by averaging the ROC-AUC scores obtained by 10 random picking probabilistic predictions.

We benchmark the ROC-AUC scores against those of CN2 (IDS and DRS are excluded due to their sub-optimal performance in general). We have shown that the differences between the ROC-AUC of TURS and TURS-RP are all negligible up to the second decimal (i.e., smaller than 0.01), while the differences between the ROC-AUC of CN2 and CN2-RP are mostly larger than 0.01 (shown in bold), among which eight are larger than 0.05.

We can hence conclude that, while CN2 relies heavily on its conflict resolving schemes for overlaps, TURS produces overlaps only formed by probabilistic rules with very similar probability estimates. This indicates our probabilistic rules can be viewed as *truly unordered in the sense that, when an instance is covered by multiple rules, the rule chosen to predict class probabilities has little effect on the prediction performance.*

3.6.4 Generalizability of local probabilistic estimates

While rule-based models are commonly considered to be intrinsically explainable models, we argue that only rules with probability estimates that generalize well can serve as trustworthy explanations. Thus, we next examine the difference between individual rules' probability estimates on the train and test sets.

Specifically, given a rule set induced from a specific dataset, we look at each individual rule's probability estimates, estimated from the training and test set respectively, by the maximum likelihood estimator. Finally, we report the weighted averages of the probability estimates differences for all rules, weighted by the

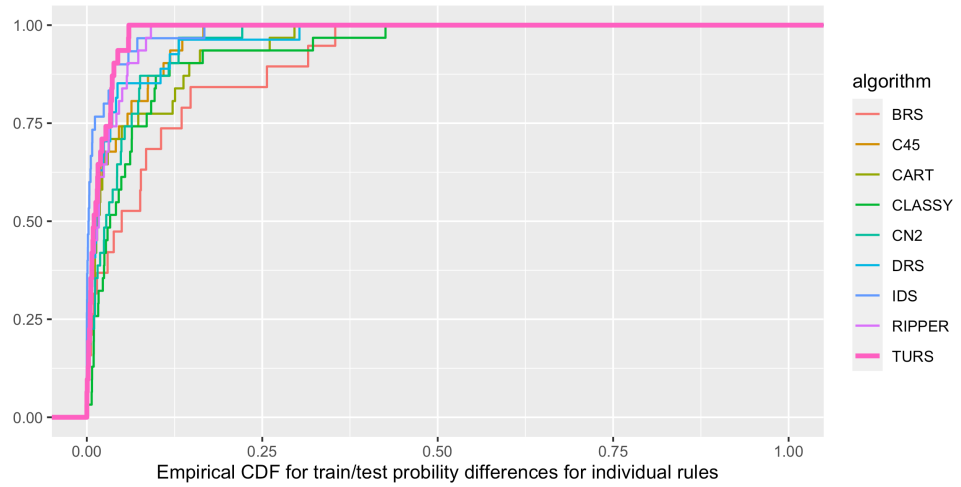

Figure 3.3: The weighted average of the differences between the class probability estimates of every individual rule for training and test sets, shown as the empirical cumulative distribution function, in which the weight is defined as the coverage of each rule for the training set.

coverage of each rule on the training set.

Formally, given a rule set with K rules, $M = \{S_1, ..., S_K\}$, denote the probability estimates of all rules by $(\mathbf{p}_1, ..., \mathbf{p}_K)$ and $(\mathbf{q}_1, ..., \mathbf{q}_K)$, respectively estimated from the training and test set. Assume each probability estimate has length C (i.e., $C = 2$ for binary target and $C > 2$ for multi-class target), we measure how well the individual rules generalize by

$$g = \frac{1}{K} \sum_j |S_j| \left(\sum_c \frac{1}{C} |p_{jc} - q_{jc}| \right) \tag{3.26}$$

in which p_{jc} (q_{jc}) is the c-th element of vector $\mathbf{p_j}$ ($\mathbf{q_j}$). Note that each individual rule is treated separately in calculating the g-score above, and hence the overlaps do not play a role here.

We calculate this score for all algorithms and all datasets, averaged using the five-fold stratified cross-validation, and we present the results with empirical cumulative density functions (ECDF) in Figure 3.3. Since the position of the curve towards the upper-left shows that the corresponding algorithm has small probability estimate differences between training and test sets, we observe that

TURS (the bold curve) dominates rule sets learned by the rest of the algorithms, with IDS the only close competitor.

For some datasets, IDS learns rule sets that have smaller probability estimation differences than the TURS model (shown by the fact that part of the corresponding blue curve is above the curve of TURS in bold). However, this indicates that IDS has serious "under-fitting" if we take into consideration IDS's suboptimal predictive performance as discussed in Section 3.6.2. That is, IDS produces rules with too large coverage, and hence is not specific and refined enough for classification, although rules with large coverage have probability estimates that generalize well.

Thus, in conclusion, rules in the TURS model learned by our algorithm are equipped with more reliable and trustworthy class probability estimates, in comparison to the other eight tree- and rule-based models.

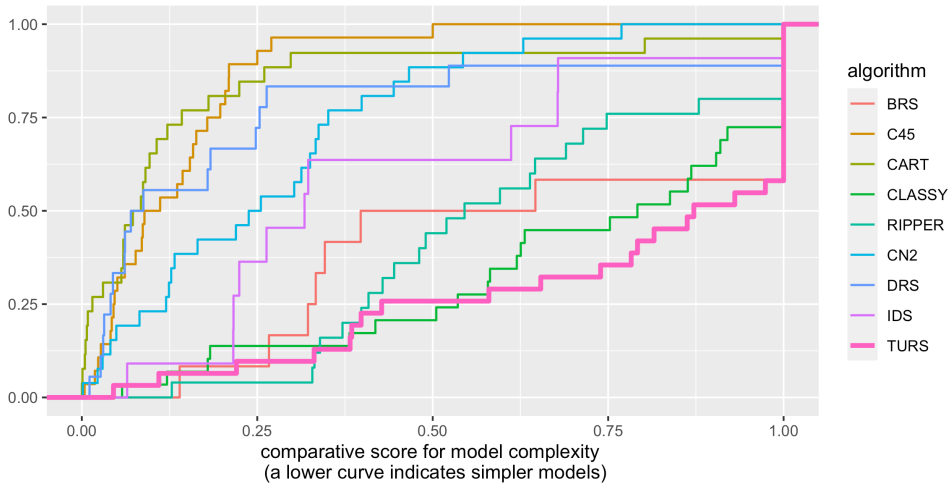

Figure 3.4: Empirical cumulative distribution function for the comparative score for model complexity. Curves towards the bottom-right indicate larger comparative scores and simpler models.

3.6.5 Model complexity

We study next whether TURS empirically leads to more complex rule sets given that it allows overlaps formed by rules with similar probabilistic outputs only.

We measure the model complexity by the number of total literals for each model: i.e., summing up the lengths of rules in a rule set, rule list, or decision tree (by treating each tree path as rule), which directly indicates the workload for a domain expert if they read the rules. We report this measure in Table 3.5, and specifically, we mark the results from the models with substantially worse ROC-AUC scores than those of TURS by denoting them in *smaller* font sizes. Precisely, for a given dataset, all competitor models with more than 0.1 smaller ROC-AUC scores than that of TURS are marked. Excluding the results from these models, we observe that TURS produces the simplest model for 13 out 31 datasets. The model complexity of all simplest models are denoted in bold in Table 3.5.

Further, to illustrate the differences between the number of literals across all algorithms, we calculate a comparative score as follow: for each individual dataset, we divide the minimum total number of literals by the total number of literals of each algorithm. This score show that, for each pair of dataset and algorithm, what is the ratio of the minimum number of literals for this dataset, over the number of literals for the algorithm-dataset pair, i.e., *larger scores indicate simpler models as the minimum number of literals is the numerator*. We plot the ECDF of these comparative scores in Figure 3.4, excluding the comparative scores obtained from models with substantially worse ROC-AUC scores than that of TURS, same as above. We observe that TURS lies at the most bottom-right, dominating the other competitors, since curves towards the bottom-right indicate larger comparative scores and hence simpler models.

3.6.6 Ablation study 1: diverse patience beam search

We study the effect of using the beam search with the "diverse patience", by replacing it with a "normal" (non-diverse) beam search. Suppose the beam width is W, we then pick the top-W rule growth candidates *without categorizing rule growth candidates by their coverage*. That is, we "turn off" the diverse coverage constraints both for updating the beam and the auxiliary beam.

As shown in Figure 3.5, when using the diverse coverage heuristic, the ROC-AUC on the test sets (points and curve in orange) becomes better on 25 out of 31 datasets, demonstrating the benefits for predictive performance.

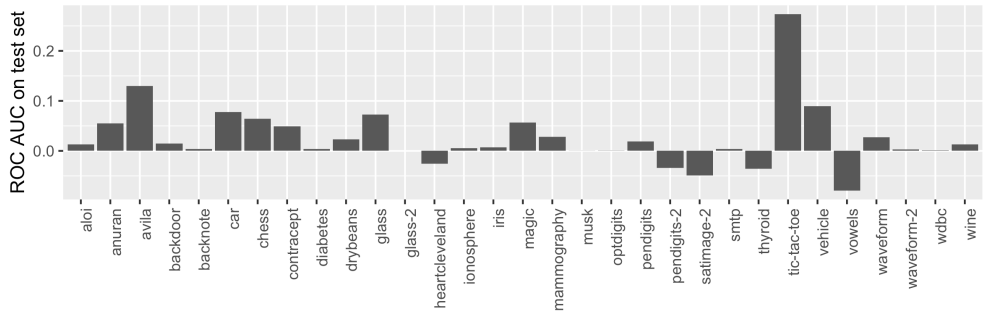

Figure 3.5: The differences between the ROC-AUC scores on the test sets with and without the diverse patience.

3.6.7 Ablation study 2: MDL-based local testing

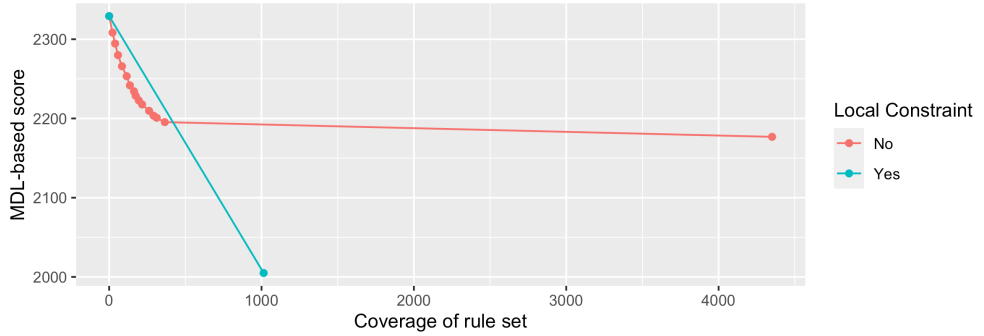

Figure 3.6: The process of adding rules to the rule set, with and without the local testing heuristics, using the first dataset among the 100 simulated datasets. Each point represents the status after a single rule is added, with the x-axis representing the coverage of the (potentially incomplete) rule set after adding this rule, and the y-axis representing the MDL-based score.

Recall that the MDL-based local test is used for evaluating the "potential" in the left out instances when growing a rule. Thus, from the perspective of optimization, it is used for looking ahead to prevent ending up in a local minimum when optimizing our MDL-based model selection criterion as defined in Equation (3.20).

We next illustrate that, without the local test, our algorithm would fail to reveal the ground-truth rule set model even for a very simple simulated dataset. Instead, it would learn a much more complicated model, and consequently, our optimization algorithm would end up at an inferior minimum.

Consider a simulated dataset generated by a known ground-truth rule set model with one rule only as follows. The feature variables are denoted as $X = (X_1, ..., X_{50})$, which are assumed to be all binary. We sample $X_1 \sim Ber(0.2)$, $X_i \sim Ber(0.5)(i = 2, ..., 50)$, in which $Ber(.)$ denotes the Bernoulli distribution. Further, we consider binary target variable Y and sample $Y|X_1 = 1 \sim Ber(0.7)$ and $Y|X_1 = 0 \sim Ber(0.95)$. That is, $X_1 = 1$ (or $X_1 = 0$) is the only "true rule" in this simulated dataset.

We simulate the dataset with sample size 5 000 for 100 times, and run TURS with and without local testing. As shown in Table 3.6, without local testing we achieve a worse (larger) score for our optimization function (i.e., the MDL-based score).

Notably, although the ROC-AUC scores are similar for using and not using the local testing, the ground truth model is only found when local testing is used. When the local testing is disabled, the number of rules and the rule lengths are both not consistent with the "true" model, as irrelevant variables are picked when growing the rules. We have two perspectives to explain the inconsistency.

To begin with, as shown in Table 3.6, when the local testing is *not* used, the difference between the class probabilities estimated from the training and test dataset is larger than the difference when the local testing is imposed, which indicates that the rules as local probabilistic models generalize worse when the local test heuristic is turned off. In other words, we observe overfitting locally.

Further, as we wrote as motivation in Section 3.5.2, the local testing heuristic is designed to prevent leaving out instances that are difficult to cover for 'future' rules, and we do notice this phenomenon empirically. Specifically, for a single run of TURS on the simulated dataset, we plot in Figure 3.6 the procedure of iteratively searching for the next best rule: each point represents the status of the rule set after a single rule is added, with the x-axis representing the coverage of the rule set (i.e., the number of instances covered by at least one of the rules excluding the else-rule), and the y-axis representing the MDL-based score for the rule set as a whole model. Thus, our learning speed score, defined in Section 3.5.1, basically tries to iteratively find the next point (i.e., the next rule) in Figure 3.6 with the steepest slope.

However, without the local test heuristic, although the learning speed scores (shown by the red curve in Figure 3.6) are in the first place larger than that of

the blue curve (for the case when the local testing is used), the red curve achieves an inferior optimization result in the end. That is, without local testing, the instances that are left out are simply ignored during the process of rule growing, which leads to a worse optimization result.

3.6.8 Runtime

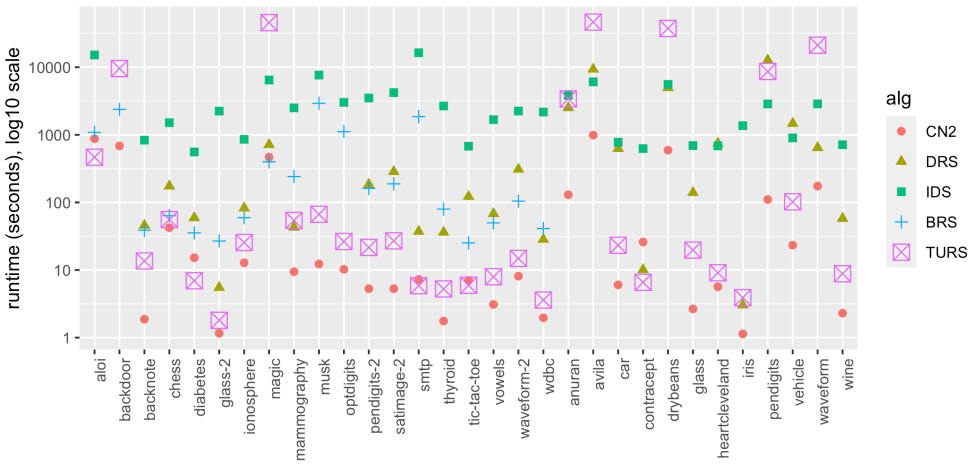

Figure 3.7: Average runtime for five rule set methods. The y-axis is scaled by $log_{10}(\cdot)$.

Last, we report the runtime of TURS, together with rule set competitor methods only, as decision trees/lists methods from mature software (Weka and Python Scikit-Learn) are highly optimized in speed and are known to be very fast.

We illustrate average the runtime (in seconds) obtained using cross-validation in Figure 3.7. In general, the runtime of TURS is competitive among all rule set methods except for CN2. CN2 seems faster in general and scales better to larger datasets, which can be caused both by a more efficient implementation (from the software "Orange3"), and by its algorithmic properties (a greedy and separate-and-conquer approach). However, as we saw in Section 3.6.3, rule sets learned from data by CN2 cannot be empirically treated as truly unordered.

3.7 Conclusion

We studied the problem of learning truly unordered rule sets from data. While existing rule set methods adopt post-hoc schemes to resolve conflicts caused by overlapping rules, we proposed the intuitive idea of only "allowing" rules to overlap if they have similar probabilistic output. Building upon this, we formally defined a truly unordered rule set (TURS) model: given a set of rules and a dataset (assumed i.i.d.), the TURS model defines the likelihood of the class labels given the feature values.

Further, we formalized the problem of learning such TURS model from data as a probabilistic model selection problem, by leveraging the minimum description length (MDL) principle. Our MDL-based model selection criterion can strike a balance between the goodness-of-fit and the model complexity without any regularization parameter.

We further proposed a carefully designed dual-beam diverse-patience algorithm to learn the TURS model from data. We showed that our algorithm can induce rules with competitive performance with respect to the following aspects. First, we benchmarked our algorithm using a large number of datasets and showed that the learned TURS model has very competitive predictive performance measured by the ROC-AUC. Specifically, in comparison to other multi-class rule set methods (CN2, DRS, IDS), the TURS model learned by our algorithm shows clear superiority with respect to the ROC-AUC scores. Second, uniquely, we showed that the TURS model learned by our algorithm is empirically truly ordered, in the sense that the predictive performance is hardly affected when predicting instances covered by multiple rules through a randomly picked rule among these multiple rules. Third, the learned TURS model contains single rules with reliable and trustworthy class probability estimates that can generalize well to the unseen instances. Fourth, the model complexity of the learned TURS model is also competitive in comparison to other rule-based methods.

For future work, we consider using TURS as a building block towards designing interactive rule learning algorithms with humans in the loop, since rules being truly unordered instead of entangled are more comprehensible and easier to edit. That is, comprehending and editing single rules in the TURS model does not require domain experts or data analysts to consider other (potentially many,

higher-ranked) rules. In sensitive area like health care, this may help build trust between the data-driven models and domain experts.

In addition, extending truly unordered rule sets to other machine learning tasks such as feature construction, subgroup discovery, regression with uncertainty, and explaining black-box models are all promising directions.

3.8 Appendix: Comparison to the Previous Work

As this chapter is based on the previous chapter, we hereby summarize the main changes and additions as follows. First, while working on the follow-up real-world case study in health care, we noticed an unsatisfactory prediction performance of our previous method. After careful investigation, we realized it was the algorithmic heuristics that could be further improved. Specifically, the previous method used a heuristic motivated by the FOIL's information gain (Fürnkranz et al. 2012), i.e., an MDL-based Foil-like compression gain; however, we later noticed extending the FOIL's information gain to multi-class situations will cause problem when using it to guide the search for rule growth, since it can be proven that the FOIL's information gain will only lead to rules with lower empirical entropy than the rules in the previous step. This specifically can cause problems when the dataset is noisy (in the sense that the Bayes-optimal classifier cannot achieve a perfect or near-perfect classification) and/or imbalanced. Therefore, we now implement a *learning-speed* heuristic, motivated by the "normalized gain" used in the CLASSY algorithm for rule lists (Proença and Leeuwen 2020); however, as we observe "normalized gain" often shrinks of the rule's coverage (the number of instances covered by the rule) too fast, we further introduced a diverse beam search with diverse "patience", in which the concept of patience is motivated from the PRIM method (Friedman and Fisher 1999), one of the first pioneer works for regression rules.

Second, one unique challenge of learning truly unordered rules is to both evaluate the quality of individual rules and the quality of the overlaps (i.e., whether the rules that form the overlap do not have similar enough outputs). However, this makes existing rules obstacles for the following search for more rules, as we elaborate in Section 3.5.2. In our previous work, we adopted a "two-stage" algorithm: in the first stage, the existing rules are ignored when calculating the heuristics, and next we use the results of the first stage as "seeds" for the second stage, in which the existing rules are considered in order to calculate the MDL-based score for evaulating the rules. However, we noticed that the first stage can output rules of which the number of covered instances is too small to be further refined when incorporating its overlaps with existing rules in the second stage. Therefore, we now combine these two stages by always keeping two "beams" in

the beam search, with one beam using the heuristic score that ignores the existing rules and the other incorporating the existing rules.

Third, given the necessity to evaluate the "potential" for the instances that are not covered by any rule so far during the rule set learning process, which is closely related to the claim made in the previous work (Fürnkranz and Flach 2005) that evaluating incomplete rule sets are a challenging and unresolved issue in rule learning in general, in our previous work we proposed to use a surrogate CART decision tree model to assess the potential for the uncovered instances. However, this approach turned out to be not very stable for this purpose in general, as we cannot afford the computational time for tuning the regularization parameter for the post-pruning for CART; in addition, when the dataset is very imbalanced, the performance of CART is sub-optimal and hence does not provide a satisfactory assessment. To resolve this issue, in this chapter we introduce a local constraint based on the local MDL compression gain, as discussed in Section 3.5.2.

Besides the algorithmic improvements, we substantially extended the experiments for the purpose of studying the truly unordered rules in detail. That is, the purpose of the experiments in the previous chapter was to show that, with the (soft) constraints of only allowing rules with similar outputs to overlap, truly unordered rule sets can achieve on-par predictive performance in comparison to rule sets methods that adopt ad-hoc schemes for conflicts caused by overlaps. However, in this chapter, we aim for studying 1) the predictive performance on a large scale of datasets, 2) whether the induced rules from data can be empirically regarded as truly unordered, in the sense that how large is the effect if we randomly pick one rule for predicting an instance covered by multiple rules, 3) whether the probabilistic estimates of individual rules can generalize to unseen (test) instances, such that the individual rules can be used as reliable and trustworthy explanations to the predictions, and 4) whether our rule sets need to sacrifice model complexity for being "truly unordered", given that our search space is essentially much more complicated in comparison to i) non-probabilistic rules, ii) rules for binary targets only, and iii) methods with the separate-and-conquer strategy that simplifies the search space by iteratively removing covered instances.

Moreover, we also made a moderate modification to our optimization score. If we simply regard the (vanilla definition of) MDL-based model selection criterion as a score based on the penalized maximum likelihood, the penalty consists of two

terms: 1) the code length of model and 2) the regret. However, it is well-known that, firstly by the implementation of C4.5 rules (Quinlan 2014), the code length of the model (the first term in the penalty) does not consider the redundancy in the model class of all possible rule sets, which can cause under-fitting. Specifically, during the implementation of our previous work, we simply exclude this "code length of the model" term, since we noticed that when not including this term, the predictive performance is in general better (at the cost of higher model complexity though). However, with the improved algorithm we propose in this chapter, we can now include the code length of model term for obtaining simpler models without sacrificing predictive performance.

Finally, we now formally defined TURS as a probabilistic model, while the previous chapter was not very precise in this regard. Also, we unified the nested overlap (i.e., one rule fully cover the other rule) and non-nested overlap of rules in the previous chapter, without using separate modelling schemes for the two cases respectively. Empirically, checking whether an overlap is a nested overlap is computationally expensive, while the empirical results show that the final model learned from the data rarely contains such nested overlap.

Algorithm 4: Learn a single rule

Input: Rule set M, dataset (x^n, y^n), beam width W
Output: The next rule S

all_candidate_rules ← [ ]
rules_for_next_iter ← [$\emptyset$] `// Initialize the rule with an ``empty" condition`
while *TRUE* **do**
  beam ← [ ] `// Initialize the beam for the beam search`
  auxiliary_beam ← [ ] `// Initialize the auxiliary beam (Section 3.5.2)`
  for *rule in rules_for_next_iter* **do**
    rule_candidates ← generate_candidates(rule) `// Enumerate literals and append to rule`
    categorized_candidates ← categorize(rule_candidates)
     `// Categorize into clusters by coverage (Section 3.5.2)`
    categorized_candidates ←
     MDL_local_testing(categorized_candidates) `// Defined in Section 3.5.2`
    top_W_candidates ← The candidate with the highest $r(.)$
                in each category of categorized_candidates
    categorized_candidates_auxiliary ← categorize(rule_candidates)
     `// Categorize into clusters by coverage excluding the instances covered by` M `(Section 3.5.2)`
    categorized_candidates_auxiliary ←
     MDL_local_testing(categorized_candidates_auxiliary)
    top_W_auxiliary ← The best candidate with the highest $R(.)$
              in each category of categorized_candidates_auxiliary
    beam.append(top_W_candidates)
    auxiliary_beam.append(top_W_auxiliary)
  if *stopping_criterion_is_met* **then**
    return the rule with the highest $r(.)$ in all_candidate_rules
  else
    beam ← top-W candidates in beam with the highest $r(.)$
     `// Reduce the number of rules to` W
    auxiliary_beam ←
     top-W candidates in auxiliary_beam with the highest $r(.)$
    all_candidate_rules.append(beam)
    rules_for_next_iter ← beam ∪ auxiliary_beam

Data	# rows	# columns	# classes	max. class prob.	min. class prob.
aloi	49534	28	2	0.970	0.030
backdoor	95329	197	2	0.976	0.024
backnote	1372	5	2	0.555	0.445
chess	3196	37	2	0.522	0.478
diabetes	768	9	2	0.651	0.349
glass-2	214	8	2	0.958	0.042
ionosphere	351	35	2	0.641	0.359
magic	19020	11	2	0.648	0.352
mammography	11183	7	2	0.977	0.023
musk	3062	167	2	0.968	0.032
optdigits	5216	65	2	0.971	0.029
pendigits-2	6870	17	2	0.977	0.023
satimage-2	5803	37	2	0.988	0.012
smtp	95156	4	2	1.000	0.000
thyroid	3772	7	2	0.975	0.025
tic-tac-toe	958	10	2	0.653	0.347
vowels	1456	13	2	0.966	0.034
waveform-2	3443	22	2	0.971	0.029
wdbc	367	31	2	0.973	0.027
anuran	7195	24	4	0.614	0.009
avila	20867	11	12	0.411	0.001
car	1728	7	4	0.700	0.038
contracept	1473	10	3	0.427	0.226
drybeans	13611	17	7	0.261	0.038
glass	214	11	6	0.355	0.042
heartcleveland	303	14	5	0.541	0.043
iris	150	5	3	0.333	0.333
pendigits	7494	17	10	0.104	0.096
vehicle	846	19	4	0.258	0.235
waveform	5000	22	3	0.339	0.329
wine	178	14	3	0.399	0.270

Table 3.2: Datasets for binary (top) and multi-class classification (bottom), publicly available on the UCI repository and the ADBench Python package; datasets from the latter are marked in *Italic*. We use the maximum and minimum of the marginal class probabilities to indicate the degree of class imbalance.

Appendix: Comparison to the Previous Work

Data	BRS	C45	CART	CLASSY	Ripper	CN2	DRS	IDS	TURS(diff to best)
aloi	0.519	0.398	0.621	**0.654**	0.485	0.569	0.500	0.509	0.619 (-0.035)
backdoor	0.917	0.990	0.979	0.996	0.976	**0.997**	—	—	0.995 (-0.002)
backnote	0.957	0.987	0.983	0.990	0.982	**0.993**	0.988	0.765	0.981 (-0.012)
chess	0.957	**0.998**	0.995	0.992	0.995	0.532	0.809	0.677	0.994 (-0.004)
diabetes	0.725	0.710	0.667	0.737	0.641	0.709	0.727	0.595	**0.750** (0)
glass-2	0.676	0.890	0.790	0.730	0.793	0.941	0.926	0.912	**0.949** (0)
ionosphere	0.802	0.882	0.851	0.886	0.911	**0.941**	0.712	0.786	0.904 (-0.037)
magic	0.767	0.869	0.799	**0.888**	0.819	0.698	0.774	0.507	0.887 (-0.001)
mammography	0.644	0.817	0.730	0.890	0.582	0.891	0.857	0.535	**0.897** (0)
musk	**1.000**	0.995	**1.000**	**1.000**	**1.000**	**1.000**	—	**1.000**	**1.000** (0)
optdigits	0.897	0.959	0.942	0.986	0.966	**0.992**	—	0.960	0.977 (-0.015)
pendigits-2	0.938	0.986	0.964	0.974	0.973	**0.996**	0.948	0.914	0.955 (-0.041)
satimage-2	0.922	0.914	0.915	0.929	**0.964**	**0.964**	0.699	0.867	0.909 (-0.055)
smtp	0.596	0.930	0.965	0.905	0.950	0.853	0.889	0.879	**0.972** (0)
thyroid	0.897	0.972	0.950	0.983	0.989	**0.998**	0.921	0.960	0.961 (-0.037)
tic-tac-toe	**1.000**	0.878	0.918	0.978	0.972	0.932	0.992	0.599	0.965 (-0.035)
vowels	0.854	0.693	0.773	0.796	0.758	**0.897**	0.813	0.748	0.817 (-0.08)
waveform-2	0.567	0.716	0.648	0.847	0.333	**0.886**	0.540	0.774	0.832 (-0.054)
wdbc	0.836	**0.999**	0.896	0.843	0.899	0.836	0.620	0.942	0.947 (-0.052)
anuran	—	0.995	0.944	0.968	**0.996**	0.962	0.945	0.602	0.973 (-0.023)
avila	—	**0.999**	0.977	0.987	0.993	0.920	0.729	0.617	0.990 (-0.009)
car	—	0.956	0.939	0.978	0.931	0.885	0.935	0.831	**0.980** (0)
contracept	—	**0.680**	0.597	0.653	0.607	0.598	0.598	0.549	0.658 (-0.022)
drybeans	—	0.970	0.943	0.977	0.979	0.929	0.975	0.591	**0.989** (0)
glass	—	0.970	**0.984**	0.975	0.940	0.937	0.926	0.793	0.967 (-0.017)
heartcleveland	—	0.603	0.572	**0.721**	0.509	0.694	0.611	0.513	0.695 (-0.026)
iris	—	0.960	0.975	0.970	0.962	0.977	0.954	0.810	**0.981** (0)
pendigits	—	0.982	0.974	0.986	0.983	0.991	0.967	0.522	**0.994** (0)
vehicle	—	0.856	0.789	0.870	0.859	0.858	0.764	0.579	**0.882** (0)
waveform	—	0.842	0.814	0.910	0.880	0.803	0.654	0.517	**0.915** (0)
wine	—	0.937	0.906	0.960	0.937	**0.973**	0.909	0.854	0.952 (-0.021)

Table 3.3: Average ROC-AUC scores obtained using cross-validation. BRS can only be applied to binary datasets, and DRS and IDS fail to get results on a few datasets, denoted as "—" in the table. Best ROC-AUC for each dataset is shown in bold and. The difference between the best ROC-AUC for each dataset and the ROC-AUC of TURS for the same dataset is shown in bracket.

data	TURS	TURS-RP	Diff.	%overlap	CN2	CN2-RP	Diff.	%overlap
aloi	0.619	0.62	-0.001	4%	0.569	0.578	-0.009	97%
anuran	0.973	0.969	0.004	28%	0.962	0.913	**0.048**	90%
avila	0.99	0.989	0.001	17%	0.92	0.915	0.004	45%
backdoor	0.995	0.995	0	0%	0.997	0.976	**0.021**	96%
backnote	0.981	0.98	0.001	20%	0.993	0.973	**0.019**	60%
drybeans	0.989	0.986	0.004	34%	0.929	0.908	**0.021**	94%
glass-2	0.949	0.949	0	0%	0.941	0.839	**0.102**	33%
heartcleveland	0.695	0.687	0.008	8%	0.694	0.663	**0.031**	61%
mammography	0.897	0.897	0	5%	0.891	0.806	**0.084**	86%
musk	1	1	0	0%	1	1	0	0%
optdigits	0.977	0.977	0	0%	0.992	0.972	**0.02**	92%
pendigits-2	0.955	0.955	0	0%	0.996	0.972	**0.024**	88%
satimage-2	0.909	0.909	0	0%	0.964	0.909	**0.055**	89%
smtp	0.972	0.972	0	0%	0.853	0.795	**0.058**	51%
thyroid	0.961	0.961	0	0%	0.998	0.941	**0.056**	87%
vehicle	0.882	0.878	0.004	15%	0.858	0.826	**0.033**	77%
vowels	0.817	0.817	0	1%	0.897	0.838	**0.059**	71%
waveform-2	0.832	0.832	0	9%	0.886	0.754	**0.132**	92%
wdbc	0.947	0.947	0	0%	0.836	0.596	**0.241**	69%
car	0.98	0.98	0.001	22%	0.885	0.794	**0.091**	91%
chess	0.994	0.994	0	23%	0.532	0.551	**-0.019**	95%
contracept	0.658	0.657	0.001	3%	0.598	0.572	**0.026**	100%
diabetes	0.75	0.748	0.002	11%	0.709	0.676	**0.033**	82%
glass	0.967	0.965	0.002	2%	0.937	0.937	0	0%
ionosphere	0.904	0.904	0	15%	0.941	0.895	**0.046**	55%
iris	0.981	0.98	0.001	5%	0.977	0.977	0	0%
magic	0.887	0.887	0	38%	0.698	0.738	**-0.04**	92%
pendigits	0.994	0.991	0.003	40%	0.991	0.982	0.009	76%
tic-tac-toe	0.965	0.965	0	7%	0.932	0.925	0.007	49%
waveform	0.915	0.905	0.009	51%	0.803	0.84	**-0.037**	77%
wine	0.952	0.952	0	0%	0.973	0.971	0.002	2%

Table 3.4: Average ROC-AUC for the predictions with and without "random picking", both for TURS and CN2. The difference between the two ROC-AUC scores are shown in bold if the difference is larger than 0.01. We further show the percentage of instances covered by more than one rule, denoted as %*overlap*.

Appendix: Comparison to the Previous Work

Data	BRS	C45	CART	CLASSY	RIPPER	CN2	DRS	IDS	TURS
aloi	**3**	2659.1	26952.8	52.3	26.2	2116	0	14	66.5
backdoor	**13**	701.5	2460.5	72.6	101.5	259.6	—	—	59.1
backnote	35.8	79.6	116.7	22.7	22.3	39.5	54	12.5	**14.2**
chess	**19.2**	250.2	340.5	33	57.9	297	54.2	14.5	58
diabetes	15.6	107.6	700.5	**5**	6.6	165.4	82.5	13.2	6.8
glass-2	10.8	19.7	6.7	3	5.9	1.8	37.2	15.5	**1**
ionosphere	31.2	59	86.6	5.6	12.5	25.1	440.7	12	**5.1**
magic	43.5	3211.8	20053.1	228.4	**87.1**	3351.9	44.7	18.5	227.8
mammography	3	273.7	1126.8	38.7	30.5	199.2	**24.4**	14	37.4
musk	6	4	**2**	**2**	**2**	**2**	—	9.3	**2**
optdigits	54.5	46.6	78.7	9.6	15.8	16.7	—	11.2	**7.6**
pendigits-2	14.8	48.3	66.9	**9.5**	12.7	19.9	136.5	14.1	12
satimage-2	15	14.8	17.8	7.9	9	9.9	524.8	12.4	**4**
smtp	3	20.9	34.4	5.6	7	19.2	16.7	13.9	**3**
thyroid	**3**	18.9	28.1	16.4	4.2	9.7	73.1	13.4	7.8
tic-tac-toe	**25.2**	411.9	410.8	29.2	42.3	81.1	101.7	13.5	28.9
vowels	25.2	58.9	147.8	14.1	13.5	22.6	141.9	14.3	**8.7**
waveform-2	4.2	165.6	406.3	19.3	20.5	79.4	522.2	**11.2**	12
wdbc	9	8	2.5	3.2	**2**	4.4	337.6	7.6	**2**
anuran	—	80.9	1284.3	90.7	**11**	264.8	346.1	12.4	100.2
avila	—	3460.4	7795.9	939.8	**707.2**	1297.4	181.3	14	726
car	—	659.8	776.1	**56.4**	171.7	72	307.6	13.5	132
contracept	—	1268.1	5536.2	12	14.7	221.8	**5**	16.8	12.6
drybeans	—	2481.6	7039.4	**105.8**	153.3	1280.6	202.3	13.2	182.4
glass	—	24.8	20	**5.2**	14	7.8	171	11.3	6
heartcleveland	—	245.7	410.2	6.6	5.5	42.1	522.2	14	**5.7**
iris	—	15.2	13	2.8	5.9	8.4	26.7	10.4	**2.3**
pendigits	—	1274.9	1689.7	**142.3**	260.9	430.4	561.7	15.6	174.5
vehicle	—	560.6	760.3	25.4	46.9	175.5	741.1	11.6	**23**
waveform	—	2782.9	3520	**125.7**	143	970.5	75.6	14.4	160.6
wine	—	22	15.5	5	8.9	7.5	103.8	14.4	**4.6**

Table 3.5: Total number of literals in the rule set, rule list, or decision tree. Smaller fonts indicate that the model learned by a certain algorithm gives the ROC-AUC score substantially worse than TURS does, and the results in bold indicate the smallest total number of literals (excluding those models with substantially worse ROC-AUC scores).

Local testing	# rules	rule length	ROC-AUC	MDL-based score	train/test prob. diff.
No	12.48(±1.56)	5.597(±0.42)	0.722(±0.02)	2191.189(±65.91)	0.049(±0.01)
Yes	1(±0)	1(±0)	0.724(±0.01)	2050.087(±68.88)	0.007(±0)

Table 3.6: Results of ablation study on local testing. We report the mean (± standard deviation) over 100 repetitions.