Response Letter

We thank the reviewers for their useful comments and feedback on improving the manuscript and the presentation. In the following, we summarize changes reflected in the revised manuscript.

Distribution shifts in life-long learning. We propose Hamming distance heuristics in IMLI with the assumption that distribution remains same across batches. One way to account for distribution shifts is to consider last p (> 1) batches instead of the (single) previous batch in the objective function in mini-batch learning. For a feature variable B^i_j, we consider its majority assignment in last p classification rules and encode as a soft clause to retain the majority assignment in the current batch. Moreover, we can reweigh the soft clause by prioritizing assignments of B^i_j in recent batches. We mention this in Section 3.4, page 28.

Discussions of related works in experiments. IMLI scales well than existing interpretable rule-based classifiers because of the incremental solving approach. Most existing works rely on rule-mining of potential classification rules followed by an optimization algorithm such as Bayesian optimization and branch and bound algorithm. In experiments, we observe the incremental learning of IMLI to be more scalable than the state-of-the-art methods. We elaborate this in Section 3.6.2, page 39.

How does Justicia outperform a direct approach by learning a Bayesian Network on limited samples? We agree with the reviewer that learning a Bayesian Network on limited samples does not always result in a better robustness (e.g., less standard deviation of fairness metrics estimation) of Justicia than the direct approach of estimating fairness on a dataset.

Elaborately, in Chapter 5 Figure 5.3, Justicia demonstrates higher robustness than the direct approach, where we consider a specific distribution of non-sensitive features conditioning only on sensitive features. Thus, Figure 5.3 does not involve experiments with Bayesian Network capturing correlations of all features, which we indeed introduce later in Chapter 6. In our revised experiment, we observe that Justicia with Bayesian network, called FVGM, exhibits less robustness due to Bayesian network learning.

Clarification on Fairness Influence Function (FIF). Our additive axiom for FIFs is based on the idea of decomposing the total unfairness of the classifiers among different subsets of features [1, 2]. We consider that the sum of FIFs of all subsets of non-sensitive features is equal *to the resultant unfairness of the classifier*, where unfairness is a real number in $[0, 1]$, such as statistical parity, equalized odds, and predictive parity.

Clarification on Bayesian Network. We consider a Bayesian network as an input distribution to express the conditional dependencies and independencies among features. In Chapter A, we demonstrate the encoding of probabilistic inference into SSAT via additional Boolean variables and clauses [89]. We also discuss the complexity of the SSAT encoding in terms of the complexity of the Bayesian network (ref. 2.3.3, page 15).

Interpretability and Fairness in Machine Learning: A Formal Methods Approach

Bishwamittra Ghosh

To my parents

Abstract

Interpretability and Fairness in Machine Learning: A Formal Methods Approach

by

Bishwamittra Ghosh

The significant success of machine learning in past decades has witnessed a host of applications of algorithmic decision-making in different safety-critical domains. The high-stake predictions of machine learning in medical, law, education, transportation and so on have far-reaching consequences on the end-users. Consequently, researchers call for the regulation of machine learning by defining and improving the interpretability, fairness, robustness, and privacy of predictions. In this thesis, we focus on the interpretability and fairness aspects of machine learning, particularly on *learning interpretable rule-based classifiers*, *verifying fairness*, and *identifying the sources of unfairness.* Prior studies aimed for these problems are limited by either scalability or accuracy or both. To alleviate these limitations, we apply formal methods in interpretable and fair machine learning and provide scalable and accurate solutions to the underlying problems.

In interpretable machine learning, rule-based classifiers are particularly effective in representing the decision boundary using a set of rules. The interpretability of rule-based classifiers is generally related to the size of the rules, where smaller rules with higher accuracy are preferable in practice. As such, interpretable classification learning becomes a combinatorial optimization problem suffering from poor scalability in large datasets. To this end, we propose an incremental learning framework, called IMLI, which extends the classification problem to million-size datasets through an iterative solving of MaxSAT (maximum satisfiability) queries in mini-batch learning. Building on incremental learning and MILP (mixed integer linear programming) solving, we propose another interpretable learning framework, called CRR, for learning a more expressible classification rule by relaxing logical formulas, obtaining higher accuracy and less rule-size than existing interpretable classifiers.

Fairness in machine learning centers on quantifying and mitigating the bias/unfairness of machine learning. In the presence of multiple fairness definitions and fairness algorithms, we propose a probabilistic fairness verifier, called Justicia, based on SSAT (stochastic satisfiability) with the goal of verifying whether the classifier achieves the desired level of fairness given the distribution of features. We also extend Justicia to consider feature correlations represented as a Bayesian Network, resulting in an accurate and scalable verification of fairness. Finally, we propose a framework for fairness influence functions (FIF) with an aim of quantifying the influence of features as their contribution on the bias of a classifier. FIF analysis interprets fairness by revealing potential features or subset of features attributing highly to the bias. Building on global sensitivity analysis, we propose

an algorithm, called FairXplainer, for computing FIFs of features, resulting in an accurate estimate of the bias of the classifier decomposed to input features.

List of Publications

This thesis is based on the following publications.

1. "How Biased is Your Feature?": Computing Fairness Influence Functions with Global Sensitivity Analysis
   Bishwamittra Ghosh, Debabrota Basu, Kuldeep S. Meel
   Under review, 2022

2. Efficient Learning of Interpretable Classification Rules
   Bishwamittra Ghosh, Dmitry Malioutov, Kuldeep S. Meel
   In Proceedings of JAIR, 2022

3. Algorithmic Fairness Verification with Graphical Models
   Bishwamittra Ghosh, Debabrota Basu, Kuldeep S. Meel
   In Proceedings of AAAI, 2022

4. Justicia: A Stochastic SAT Approach to Formally Verify Fairness
   Bishwamittra Ghosh, Debabrota Basu, Kuldeep S. Meel
   In Proceedings of AAAI, 2021

5. Classification Rules in Relaxed Logical Form
   Bishwamittra Ghosh, Dmitry Malioutov, Kuldeep S. Meel
   In Proceedings of ECAI, 2020

6. IMLI: An Incremental Framework for MaxSAT-Based Learning of Interpretable Classification Rules
   Bishwamittra Ghosh, Kuldeep S. Meel
   In Proceedings of AIES, 2019

Acknowledgment

Contents

List of Algorithms

List of Figures

List of Tables

Part I

Prologue

Chapter 1

Introduction

The last decades have witnessed significant progress in machine learning with a host of applications of algorithmic decision-making in different safety-critical domains, such as medical [3–5], law [6, 7], education [8], and transportation [9, 10]. In high-stake domains, machine learning predictions have far-reaching consequences on the end-users [11]. With the aim of applying machine learning for societal goods, there have been increasing efforts to regulate machine learning by imposing interpretability [12], fairness [13], robustness [14], and privacy [15] in predictions. In this thesis, we focus on the interpretability and fairness aspects of machine learning. The thesis establishes a close integration of formal methods with machine learning and proposes efficient algorithmic solutions for problems arising in interpretability and fairness in machine learning.

Towards responsible machine learning, we propose two research themes in this thesis: interpretability and fairness of machine learning classifiers. *In interpretable machine learning*, rule-based classifiers effectively represent the decision boundary using a set of rules comprising input features. Interpretable rule-based classifiers not only interpret the decision function but also be applied to explain the prediction of black-box classifiers [16–20], a fundamental research question in explainable artificial intelligence (XAI). In this thesis, we propose efficient algorithms based on incremental learning for interpretable rule-based classifiers. In another research theme of *fairness in machine learning*, classifiers tend to exhibit bias/unfairness to certain demographic groups in the data unless the classifiers are trained with a fairness objective. Consequently, the research on fairness centers on quantifying bias using multiple fairness definitions and mitigating bias based on multiple fairness algorithms. In this thesis, we study fairness verification problems to assess the fairness of different classifiers under the lens of multiple fairness definitions and algorithms. In addition, we study the sources of the unfairness of classifiers by identifying features attributing highly to bias. To summarize our thesis, in interpretable and fair machine learning, we prioritize improving the *scalability* and the *accuracy* of solutions—either or both of which prior approaches fail to achieve.

1.1 Interpretable Machine Learning

The problem in interpretable machine learning is to learn a classifier making interpretable predictions to the end-users. To achieve the interpretability of predictions, decision functions in the form of classification rules such as decision trees, decision lists, decision sets, etc. are particularly effective [21–31]. At this point, it is important to acknowledge that interpretability is an informal notion, and formalizing it in full generality is challenging. In our context of rule-based classifiers, we use *sparsity of rules* (that is, fewer rules each having fewer Boolean literals), which has been considered a proxy of interpretability in various domains, specifically in the medical domain [27, 32–35].

In this thesis, we study two interpretable rule-based classifiers, characterized by their expressiveness. At first, we study classifiers represented as formulas in propositional logic. In propositional logic, Conjunctive Formal Form (CNF) and Disjunctive Normal Form (DNF) are useful representations of Boolean formulas. Popular interpretable rule-based classifiers such as decision tree, decision lists, and decision sets share the logical structure of CNF/DNF in their representation of the decision function. Thereby, we propose to learn an interpretable CNF classifier, wherein its interpretability is defined by the number of Boolean literals that the formula contains. Compared to CNF, Boolean cardinality constraints are more expressive as they allow numerical bounds on Boolean literals [36]. Relying on the concept of cardinality constraints to increase expressiveness, our second interpretable rule-based classifier is a logical relaxation of CNF/DNF classifiers, namely *relaxed*-CNF.

The problem of learning rule-based classifiers is known to be computationally intractable. The earliest tractable approaches for classifiers such as decision trees and decision lists relied on heuristically chosen objective functions and greedy algorithmic techniques [37–39]. In these approaches, the size of rules is controlled by early stopping, ad-hoc rule pruning, etc. In recent approaches, the interpretable classification problem is reduced to an optimization problem, where the accuracy and the sparsity of rules are optimized jointly [26, 28]. Different optimization solvers such as linear programming [34], sub-modular optimizations [26], Bayesian optimizations [27], and MaxSAT [40] are then deployed to find the best classifier with maximum accuracy and minimum rule-size. The discrete combinatorial nature of learning rule-based classifiers leads to the intractability of the problem and suffers from scalability issues in large datasets. Therefore, we propose an incremental learning approach by wrapping traditional optimization solvers such as MaxSAT and MILP (mixed integer linear programming) to efficiently learn rule-based classifiers in a mini-batch learning setting.

The contributions of this thesis on interpretable machine learning are summarized in the following.

Scalability via Incremental Learning

We propose an incremental learning framework, called IMLI [41, 42], based on MaxSAT for synthesizing interpretable classification rules expressible in proposition logic. IMLI considers a joint objective function to optimize the accuracy and the interpretability of classification rules and learns an optimal rule by solving an appropriately designed MaxSAT query. Despite the progress of MaxSAT solving in the last decade, the straightforward MaxSAT-based solution cannot scale to practical classification datasets containing thousands to millions of samples. Therefore, we incorporate an efficient incremental learning technique inside the MaxSAT formulation by integrating mini-batch learning and iterative rule-learning. The resulting framework learns a classifier by iteratively covering the training data, wherein in each iteration, it solves a sequence of smaller MaxSAT queries corresponding to each mini-batch. In our experiments, IMLI achieves the best balance among prediction accuracy, interpretability, and scalability. For instance, IMLI attains a competitive prediction accuracy and interpretability w.r.t. existing interpretable classifiers and demonstrates impressive scalability on large datasets where both interpretable and non-interpretable classifiers fail. As an application, we deploy IMLI in learning popular interpretable classifiers such as decision lists and decision sets.

Expressiveness via Logical Relaxation

We extend our incremental learning framework to learn a more relaxed representation of classification rules with higher expressiveness [43]. Elaborately, we consider relaxed definitions of standard OR/AND operators in Boolean logic by allowing exceptions in the construction of a clause and also in the selection of clauses in a rule. Building on these relaxed definitions, we introduce relaxed-CNF classification rules motivated by the popular usage of checklists in the medical domain and Boolean cardinality constraints in logic. Relaxed-CNF generalizes widely employed rule representations including CNF, DNF, and decision sets. While the combinatorial structure of relaxed-CNF rules offers exponential succinctness, the naïve learning techniques are computationally expensive. To this end, we propose an incremental mini-batch learning procedure, called CRR, that employs advances in MILP solvers to efficiently learn relaxed-CNF rules. Our experimental analysis demonstrates that CRR can generate relaxed-CNF rules, which are more accurate and sparser compared to the alternative rule-based models.

1.2 Fairness in Machine Learning

As a technology machine learning is oblivious to societal good or bad. The success of machine learning as an accurate predictor, however, finds applications in high-stake decision-making, such as college admission [44], recidivism prediction [45], job applications [46] etc. In such applications, the deployed classifier often demonstrates bias towards certain

sensitive demographic groups involved in the data [47]. For example, a classifier deciding the eligibility of college admission may offer more admission to White-male candidates than to Black-female candidates—possibly because of the historical bias in the admission data, or the accuracy-centric learning objective of the classifier, or a combination of both [48–50]. Following such phenomena, multiple fairness metrics, such as *statistical parity*, *equalized odds*, *predictive parity* etc, have been proposed to quantify the bias of the classifier. For example, if the classifier in college admission demonstrates a statistical parity of 0.6, it means that White-male candidates are offered admission 60% more than Black-female candidates [51–53]. To this end, different fairness enhancing algorithms have been devised to improve fairness with respect to one or multiple fairness metrics. These algorithms try to rectify and mitigate bias in three ways: *pre-processing* the data [54–56], *in-processing* the classifier [57], and *post-processing* the outcomes of a classifier [58, 59]. Researchers also study fairness attack algorithms to worsen the fairness of a classifier, such as by adding poisoned data samples [60]. In the presence of multiple fairness metrics and algorithms, in this thesis, we contribute to two fundamental problems in fairness: (i) probabilistic verification of fairness and (ii) identification of sources of unfairness.

Probabilistic Fairness Verification

The problem in probabilistic fairness verification is to verify the bias of a classifier given the distribution of input features. The early works on fairness verification focused on measuring fairness metrics of a classifier for a given dataset [61]. Naturally, such techniques were limited in enhancing confidence of users for wide deployment. Consequently, recent verifiers seek to achieve verification beyond finite dataset and in turn focus on the probability distribution of features [62, 63]. More specifically, the input to the verifier is a classifier and the probability distribution of features, and the output is an estimate of fairness metrics that the classifier obtains given the distribution.

In order to solve the fairness verification problem, existing works have proposed two principled approaches. Firstly, [62] propose a formal method approach to reduce the verification problem into the weighted volume computation of an SMT formula. Secondly, [63] propose a sampling approach that relies on extensively enumerating the conditional probabilities of prediction given different sensitive features and thus, incurs high computational cost. Additionally, existing works assume feature independence of non-sensitive features and consider correlated features within a limited scope, such as conditional probabilities of non-sensitive features w.r.t. sensitive features and ignore correlations among non-sensitive features. As a result, the *scalability* and *accuracy* of existing verifiers remain major challenges.

In this thesis, we propose an efficient fairness verification framework for two classes of machine learning classifiers, classifiers represented as Boolean formulas and linear classifiers. Based on stochastic satisfiability (SSAT) [64], our proposed verifier Justicia verifies the fairness of Boolean classifiers such as decision tree by solving appropriately designed SSAT

formulas. Justicia also extends verification to compound sensitive groups, which are a combination of multiple categorical sensitive features such as race ∈ {White, Black} and gender ∈ {male, female}. Because SSAT encoding allows separate quantification to each sensitive feature without any restriction on the number of features. In experiments, Justicia is more scalable than the existing probabilistic verifiers [62, 63], and more robust than the sample-based empirical verifiers [61].

Linear classifiers have attracted significant attention from researchers in the context of fair algorithms [65–68]. Existing fairness verifier suffers from two-fold limitations for verifying linear classifiers: (i) poor scalability due to applying SSAT/SMT or sampling based techniques and (ii) inaccuracy due to ignoring feature correlations. Consequently, we propose a fairness verification framework for linear classifiers, namely FVGM, for an accurate and scalable fairness verification. FVGM proposes a novel *stochastic subset-sum* encoding for linear classifiers with an efficient pseudo-polynomial solution using dynamic programming. To address feature-correlations, FVGM considers a graphical model, particularly a Bayesian Network to represent the conditional dependence (and independence) among features in the form of a Directed Acyclic Graph (DAG). Experimentally, FVGM is more accurate and scalable than existing fairness verifiers; FVGM can verify group and causal fairness metrics for multiple fairness algorithms. We also demonstrate two novel applications of FVGM as a fairness verifier: (a) detecting fairness attacks, and (b) computing fairness influences of a subset of features on shifting the incurred bias of the classifiers from the original bias.

Identification of Sources of Unfairness

While fairness metrics globally quantify bias, they cannot detect or explain the sources of bias [1, 2, 69]. In order to identify the sources of bias and also the effect of affirmative/punitive actions to alleviate/deteriorate bias, it is important to understand *which factors contribute how much to the bias of a classifier on a dataset*. To this end, we follow a feature-attribution approach to understand the sources of bias [1, 2], where we relate the *influences* of input features towards the resulting bias of the classifier. Particularly, we define and compute *Fairness Influence Function* (FIF) that quantifies the contribution of individual and subset of features to the resulting bias. FIFs do not only allow practitioners to identify the features to act up on but also to quantify the effect of various affirmative [55–59, 70, 71] or punitive actions [60, 72, 73] on the resulting bias. We also instantiate an algorithm, FairXplainer, that uses variance decomposition among the subset of features and a local regressor to compute FIFs accurately, while also capturing the intersectional effects of the features. Our experimental analysis validates that FairXplainer captures the influences of both individual features and higher-order feature interactions, estimates the bias more accurately than existing local explanation methods, and detects the increase/decrease in bias due to affirmative/punitive actions in the classifier.

1.3 Tools

We develop following open-source tools for interpretable and fair machine learning.

- IMLI for interpretable machine learning
- Justicia for fairness verification and FIF computation

1.4 Outline

We organize the thesis in the following way. We discuss preliminaries in Chapter 2. In the first part of the thesis, we discuss an incremental learning framework based on MaxSAT solving for learning interpretable classifiers. We conclude this part by applying incremental learning on a more expressible but interpretable rule-based classifier in Chapter 4. In the second part of this thesis, we discuss fairness verification for classifiers represented as Boolean formulas in Chapter 5 and linear classifiers in Chapter 6. We conclude this part by discussing the identification of sources of unfairness of machine learning classifiers in Chapter 7. Finally, we conclude our thesis in Chapter 8.

Chapter 2

Preliminaries

We represent sets/vectors by bold letters, and the corresponding distributions by calligraphic letters. We express random variables in uppercase, and the valuation or an assignment of a random variable in lowercase.

2.1 Formal Methods

2.1.1 Propositional Satisfiability

Let ϕ be a Boolean formula defined over a set of Boolean variables $\mathbf{B} = \{B_1, B_2, \ldots, B_m\}$. A literal V is a variable B or its complement $\neg B$, and a clause C is a disjunction ($\vee$) or a conjunction ($\wedge$) of literals.[1] ϕ is in Conjunctive Normal Form (CNF) if $\phi \triangleq \bigwedge_i C_i$ is a conjunction of clauses where each clause $C_i \triangleq \bigvee_j V_j$ is a disjunction of literals. In contrast, ϕ is in Disjunctive Normal Form (DNF) if $\phi \triangleq \bigvee_i C_i$ is a disjunction of clauses where each clause $C_i \triangleq \bigwedge_j V_j$ is a conjunction of literals. We use σ to denote an assignment of variables in $\mathbf{B}$ where $\sigma(B_i) \in \{\text{true, false}\}$. A *satisfying assignment* σ^* of ϕ is an assignment that evaluates ϕ to true and is denoted by $\sigma^* \models \phi$. The propositional satisfiability (SAT) problem finds a satisfying assignment σ^* to a CNF formula ϕ such that $\forall i, \sigma^* \models C_i$, wherein $\sigma^* \models C_i$ if and only if $\exists V_j \in C_i, \sigma^*(V_j) = \text{true}$. Informally, σ^* satisfies at least one literal in each clause of a CNF.

2.1.2 Relaxation of Logical Formulas

A hard-OR or a soft-AND clause is a tuple (C, η) with an extra parameter η, where η is the threshold on the literals in C. Let $\mathbb{1}[\text{true}] = 1$ and $\mathbb{1}[\text{false}] = 0$. Motivated by Boolean cardinality constraints, we propose relaxed-CNF formulas, where in addition to ϕ, we have two more parameters η_c and η_l. We say that (ϕ, η_c, η_l) is in relaxed-CNF and σ^* is its satisfying assignment if and only if $\sigma^* \models (\phi, \eta_c, \eta_l)$ whenever $\sum_{i=1}^{k} \mathbb{1}[\sigma^* \models (C_i, \eta_l)] \geq \eta_c$,

[1] While *term* is reserved to denote disjunction of literals, we use clause for both disjunction and conjunction.

where $\sigma^* \models (C_i, \eta_l)$ if and only if $\sum_{V \in C_i} \mathbb{1}[\sigma^* \models V] \geq \eta_l$. Informally, σ^* satisfies a clause (C_i, η_l) if at least η_l literals in C_i are set to true by σ^* and σ^* satisfies (ϕ, η_c, η_l) if at least η_c clauses out of all $\{(C_i, \eta_l)\}_{i=1}^k$ clauses are true.

Theorem 1 ([74]). *Let (C, η) be a clause in a relaxed-CNF where C has m literals and $\eta \in \{1, \ldots, m\}$ is the threshold on literals. An equivalent compact encoding of (C, η) into a CNF formula $\phi = \bigwedge_i C_i$ requires $\binom{m}{m-\eta+1}$ clauses where each clause is distinct and has $m - \eta + 1$ literals of (C, η). Therefore, the total number of literals in ϕ is $(m-\eta+1)\binom{m}{m-\eta+1} = m$ (equal succinctness) when $\eta = 1$, otherwise $(m-\eta+1)\binom{m}{m-\eta+1} > m$ (exponential succinctness).*

2.1.3 Inner Product

In the thesis, we use true as 1 and false as 0 interchangeably. Between two vectors $\mathbf{U}$ and $\mathbf{V}$ of the same length defined over Boolean variables or constants (such as 0 and 1), we define $\mathbf{U} \circ \mathbf{V}$ to refer to their inner product. Formally, $\mathbf{U} \circ \mathbf{V} \triangleq \bigvee_i (U_i \wedge V_i)$ is a disjunction of element-wise conjunction where U_i and V_i denote the i^{th} variable/constant of $\mathbf{U}$ and $\mathbf{V}$, respectively. In this context, the conjunction "$\wedge$" between a variable and a constant follows the standard interpretation: $B \wedge 0 = 0$ and $B \wedge 1 = B$.

For example, let us consider a vector of variables $\mathbf{U} = [U_1, U_2, U_3]$ and a vector of constants of the same length $\mathbf{v} = [0, 1, 1]$. Then, $\mathbf{U} \circ \mathbf{v} = U_2 \vee U_3$. Applying numerical interpretation, the inner product can also be expressed as $\mathbf{U} \circ \mathbf{V} \triangleq \sum_i (U_i \cdot V_i)$. Hence, for the running example $\mathbf{U} = [U_1, U_2, U_3]$ and $\mathbf{v} = [0, 1, 1]$, we derive $\mathbf{U} \circ \mathbf{v} = u_2 + u_3$. In the thesis, we use Boolean interpretation of the inner product for learning CNF classification rules in Chapter 3 and numerical interpretation for learning relaxed-CNF classification rules in Chapter 4.

2.1.4 MaxSAT

The MaxSAT problem is an optimization analog to the SAT problem that finds an optimal assignment satisfying the maximum number of clauses in a CNF. In interpretable machine learning, we consider a *weighted* variant of the MaxSAT problem—more specifically, a weighted-partial MaxSAT problem—that optimizes over a set of hard and soft constraints in the form of a weighted-CNF formula. In a weighted-CNF formula, a weight $W(C_i) \in \mathbb{R}^+ \cup \{\infty\}$ is defined over each clause C_i, wherein C_i is called a *hard* clause if $W(C_i) = \infty$, and C_i is a *soft* clause, otherwise. To avoid notational clutter, we overload $W(\cdot)$ to denote the weight of an assignment σ. In particular, we define $W(\sigma)$ as the sum of the weights of *unsatisfied clauses* in a CNF, formally,

$$W(\sigma) = \sum_{i | \sigma \not\models C_i} W(C_i)\,.$$

Given a weighted-CNF formula $\phi \triangleq \bigvee_i C_i$ with weight $W(C_i)$, the weighted-partial MaxSAT problems finds an optimal assignment σ^* that achieves the minimum weight.

Formally, $\sigma^* = \textsc{MaxSAT}(\phi, W(\cdot))$ if $\forall \sigma \neq \sigma^*, W(\sigma^*) \leq W(\sigma)$. The optimal weight of the MaxSAT problem $W(\sigma^*)$ is infinity (∞) when at least one hard clause becomes unsatisfied. Therefore, the weighted-partial MaxSAT problem finds σ^* that satisfies all hard clauses[2] and as many soft clauses as possible such that the total weight of unsatisfied soft clauses is minimum. In Chapter 3, we reduce the classification problem to the solution of a weighted-partial MaxSAT problem. The knowledge of the inner workings of MaxSAT solvers is not required for this chapter.

2.1.5 Stochastic Boolean Satisfiability (SSAT)

The stochastic Boolean satisfiability (SSAT) problem [64] is a counting analog of the SAT problem concerning with the probability of satisfaction of the formula. The SSAT problem computes the probability of satisfaction of a CNF formula ϕ defined on the ordered set of *quantified* Boolean variables $\mathbf{B}$. An SSAT formula is of the form

$$\Phi = Q_1 B_1, \ldots, Q_m B_m,\ \phi, \tag{2.1}$$

where $Q_i \in \{\exists, \forall, \mathfrak{R}^{p_i}\}$ is either of the existential ($\exists$), universal ($\forall$), or randomized ($\mathfrak{R}^{p_i}$) quantifiers on B_i and ϕ is a quantifier-free CNF formula. In case of a randomized quantifier $\mathfrak{R}^{p_i}$, $p_i \triangleq \Pr[B_i = 1] \in [0, 1]$ is the probability of B_i being assigned to 1. In the SSAT formula Φ, the quantifier part $Q_1 B_1, \ldots, Q_m B_m$ is known as the *prefix* of the formula ϕ. Let B be the outermost variable in the prefix. The semantics of SSAT formulas are defined recursively in the following.

1. $\Pr[\text{true}] = 1$, $\Pr[\text{false}] = 0$,
2. $\Pr[\Phi] = \max_B\{\Pr[\Phi|_B], \Pr[\Phi|_{\neg B}]\}$ if B is existentially quantified ($\exists$),
3. $\Pr[\Phi] = \min_B\{\Pr[\Phi|_B], \Pr[\Phi|_{\neg B}]\}$ if B is universally quantified ($\forall$),
4. $\Pr[\Phi] = p \Pr[\Phi|_B] + (1-p) \Pr[\Phi|_{\neg B}]$ if B is randomized quantified ($\mathfrak{R}^p$) with probability p of being true,

where $\Phi|_B$ and $\Phi|_{\neg B}$ denote the SSAT formulas derived by eliminating the outermost quantifier of B by substituting the value of B in the CNF ϕ with 1 and 0, respectively. In this thesis, we focus on two specific types of SSAT formulas: *random-exist* (RE) SSAT and *exist-random* (ER) SSAT. In the ER-SSAT (resp. RE-SSAT) formula, all existentially (resp. randomized) quantified variables are followed by randomized (resp. existentially) quantified variables in the prefix.

Remark. *The decision problem of ER-SSAT is* NP^{PP} *whereas RE-SSAT problem is* PP^{NP}*-complete [64].*

[2] In our formulation, we assume that there is a satisfying assignment to a CNF formula containing all hard clauses.

The problem of SSAT and its variants have been pursued by theoreticians and practitioners for over three decades [75–77]. We refer the reader to [78, 79] for a detailed survey. It is worth remarking that the past decade has witnessed a significant performance improvements of SSAT solving, thanks to the close integration of techniques from SAT solving with advances in weighted model counting [80–82].

2.1.6 Stochastic Subset Sum Problem (S3P)

Let $\mathbf{B} \triangleq \{B_i\}_{i=1}^{|\mathbf{B}|}$ be a set of Boolean variables and $W_i \in \mathbb{Z}$ be the weight of B_i. Given a constraint of the form $\sum_{i=1}^{|\mathbf{B}|} W_i B_i = \tau$, for a constant threshold $\tau \in \mathbb{Z}$, the subset-sum problem seeks to compute an assignment $\mathbf{B} \in \{0,1\}^{|\mathbf{B}|}$ such that the constraint evaluates to true when $\mathbf{B}$ is substituted with $\mathbf{b}$. Subset sum problem is known to be a NP-complete problem and well-studied in theoretical computer science [83]. The *counting* version of the subset-sum problem counts all $\mathbf{b}$'s for which the above constraint holds; and this problem belongs to the complexity class #P. In this thesis, we consider the counting problem for the constraint $\sum_{i=1}^{|\mathbf{B}|} W_i B_i \geq \tau$ where variables B_i's are stochastic. More precisely, we define a *stochastic subset-sum* problem, namely S3P, that computes $\Pr[\sum_{i=1}^{|\mathbf{B}|} W_i B_i \geq \tau]$. Details of S3P are in Chapter 6.1.1.

2.2 Interpretable Machine Learning

We consider a standard binary classification problem where the dataset $\mathbf{D}$ is a collection of n samples $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ generated from an underlying distribution $\mathcal{D}$. Here, the feature valuation vector $\mathbf{x} = [x_1, \ldots, x_m] \in \{0,1\}^m$ is a vector of m Boolean features with $y \in \{0,1\}$ being the binary class label. Thus, $(\mathbf{x}, y)$ is called a positive sample if $y = 1$, and a negative sample otherwise. We use $(\mathbf{X}, Y)$ to denote the random variables corresponding to $(\mathbf{x}, y)$. Hence, each feature $X_j \in \mathbf{X}$ is sampled from a Bernoulli distribution $\mathcal{X}_j$, and $\mathcal{D} = \prod_{j=1}^{m} \mathcal{X}_j$ is the product distribution of all features.

2.2.1 Rule-based Classification

A classifier $\mathcal{R} : \mathbf{X} \to \widehat{Y} \in \{0,1\}$ is a function that takes a feature vector as input and outputs the predicted class label $\widehat{Y}$. In a rule-based classifier such as a CNF formula, we view $\mathcal{R}$ as a propositional formula defined over Boolean features $\mathbf{X}$ such that, if $\mathcal{R}$ becomes true for an input, the prediction $\widehat{Y} = 1$ and $\widehat{Y} = 0$, otherwise. The goal is to design $\mathcal{R}$ not only to approximate the training data, but also to generalize to unseen samples arising from the same distribution. Additionally, we prefer to learn a *sparse* $\mathcal{R}$ in order to favor interpretability. We define the structural complexity of $\mathcal{R}$, referred to as *rule-size*, in terms of the number of literals it contains. Let $\mathsf{clause}(\mathcal{R}, i)$ denote the i^{th} clause of $\mathcal{R}$ and $|\mathsf{clause}(\mathcal{R}, i)|$ be the number of literals in $\mathsf{clause}(\mathcal{R}, i)$. Thus, the size of the classifier is $|\mathcal{R}| = \Sigma_i |\mathsf{clause}(\mathcal{R}, i)|$, which is the sum of literals in all clauses in $\mathcal{R}$.

Example 2.2.1. Let $\mathcal{R} \triangleq (X_1 \vee X_3) \wedge (\neg X_2 \vee X_3)$ be a CNF classifier defined over three Boolean features $[X_1, X_2, X_3]$. For an input $[0, 1, 1]$, the prediction of $\mathcal{R}$ is 1, whereas for an input $[1, 1, 0]$, the prediction is 0 because $\mathcal{R}$ is true and false in these two cases, respectively. Moreover, $|\mathcal{R}| = 2 + 2 = 4$ is the size of $\mathcal{R}$.

2.2.2 Decision Lists.

A decision list is a rule-based classifier consisting of "if-then-else" rules. Formally, a decision list [29] is an ordered list $\mathcal{R}_L$ pairs $(C_1, V_1), \ldots, (C_k, V_k)$ where the rule C_i is a conjunction of literals (alternately, a single clause in a DNF formula) and $V_i \in \{0, 1\}$ is a Boolean class label[3]. Additionally, the last clause $C_k \triangleq$ true, thereby V_k is the default class. A decision list is defined as a classifier with the following interpretation: for an input feature vector $\mathbf{x}$, $\mathcal{R}_L(\mathbf{x})$ is equal to V_i where i is the least index such that the feature vector satisfies the rule, $\mathbf{x} \models C_i$. Since the last clause is true, $\mathcal{R}_L(\mathbf{x})$ always exists. Intuitively, whichever clause (starting from the first) in $\mathcal{R}_L$ is satisfied for an input, the associated class-label is considered as its prediction.

2.2.3 Decision sets.

A decision set is a set of *independent* "if-then" rules. Formally, a decision set $\mathcal{R}_S$ is a set of pairs $\{(C_1, V_1), \ldots, (C_{k-1}, V_{k-1})\}$ and a default pair (C_k, V_k), where C_i is—similar to a decision list—a conjunction of literals and $V_i \in \{0, 1\}$ is a Boolean class label. In addition, the last clause $C_k \triangleq$ true and V_k is the default class. For a decision set, if an input $\mathbf{x}$ satisfies one clause, say C_i, then the prediction is V_i. If $\mathbf{x}$ satisfies no clause, then the prediction is the default class V_k. Finally, if $\mathbf{x}$ satisfies ≥ 2 clauses, $\mathbf{x}$ is assigned a class using a tie-breaking [26].

2.3 Fairness in Machine Learning

In this section, we discuss preliminaries in fairness in machine learning, followed by methods to verify fairness and identify the sources of unfairness.

2.3.1 Dataset and Distribution

We consider a dataset $\mathbf{D}$ as a collection of n triples $\{(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)})\}_{i=1}^{n}$ generated from an underlying distribution $\mathcal{D}$. Each non-sensitive data point $\mathbf{x}^{(i)}$ consists of k features $[x_1^{(i)}, \ldots, x_k^{(i)}]$. Each sensitive data point $\mathbf{a}^{(i)}$ consists of m categorical features $[a_1^{(i)}, \ldots, a_m^{(i)}]$. $y^{(i)} \in \{0, 1\}$ is the binary class corresponding to $(\mathbf{x}^{(i)}, \mathbf{a}^{(i)})$.

We use $(\mathbf{X}, \mathbf{A}, Y)$ to denote the random variables corresponding to $(\mathbf{x}, \mathbf{a}, y)$. Each non-sensitive feature $X_i \in \mathbf{X}$ is sampled from a continuous probability distribution $\mathcal{X}_i$,

[3] In a more practical setting, $V_i \in \{0, \ldots, N\}$ can be multi-class for $N \geq 1$.

and each categorical sensitive feature $A_j \in \mathbf{A}$ is sampled from a discrete probability distribution $\mathcal{A}_j$. Thus, $\mathcal{D}$ is the product distribution of all features, $\mathcal{D} \triangleq \prod_{i=1}^{k} \mathcal{X}_i \prod_{j=1}^{m} \mathcal{A}_j$.

For sensitive features, a valuation vector $\mathbf{a} = [a_1, .., a_m]$ is called a *compound sensitive group*. For example, consider $\mathbf{A}$ = [race, sex] where race $\in$ {Asian, Color, White} and sex $\in$ {female, male}. Thus $\mathbf{a}$ = [Asian, female] is a compound sensitive group.

We represent a binary classifier trained on the dataset $\mathbf{D}$ as $\mathcal{M} : (\mathbf{X}, \mathbf{A}) \rightarrow \widehat{Y}$. Here, $\widehat{Y} \in \{0, 1\}$ is the class predicted for $(\mathbf{X}, \mathbf{A})$. Given this setup, we discuss different fairness metrics to compute the bias in the prediction of a classifier [52, 59, 84].

2.3.2 Fairness Metrics

A classifier $\mathcal{M}$ that solely optimizes accuracy, i.e., the average number of times $\widehat{Y} = Y$, may discriminate certain compound sensitive groups over others [85]. In the following, we describe two well-known fairness definitions: group fairness and causal fairness. To this end, we use $f(\mathcal{M}, \mathcal{D})$ to quantify the fairness of the classifier $\mathcal{M}$ given the distribution of features $\mathcal{D}$. Alternatively, a fairness metric can be computed on a finite dataset instead of on the distribution. In that case, we use the notation $f(\mathcal{M}, \mathbf{D})$, which is the bias of the classifier $\mathcal{M}$ on a dataset $\mathbf{D}$.

Statistical Parity (SP). Statistical parity belongs to *independence* measuring group fairness metrics, where the prediction $\widehat{Y}$ is statistically independent of sensitive features $\mathbf{A}$ [52]. The statistical parity of a classifier is measured as

$$f_{\mathsf{SP}}(\mathcal{M}, \mathcal{D}) \triangleq \max_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}] - \min_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}],$$

which is the difference between the maximum and minimum conditional probability of positive prediction the classifier for different sensitive groups.

Disparate impact (DI). Disparate impact also belongs to independence measuring group fairness metrics. Disparate impact measures the ratio between the minimum and the maximum probability of positive prediction of the classifier over all sensitive groups, and prescribe the ratio to be close to 1 [52]. Formally, the disparate impact of a classifier is

$$f_{\mathsf{DI}}(\mathcal{M}, \mathcal{D}) \triangleq \frac{\min_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]}{\max_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]}.$$

Equalized Odds (EO). *Separation* measuring group fairness metrics such as equalized odds constrain that $\widehat{Y}$ is independent of $\mathbf{A}$ given the ground class Y [59]. Formally, for $Y \in \{0, 1\}$, equalized odds is

$$f_{\mathsf{EO}}(\mathcal{M},\mathcal{D}) \triangleq \max(\max_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a},Y=0] - \min_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a},Y=0],$$
$$\max_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a},Y=1] - \min_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a},Y=1]).$$

Predictive Parity (PP). *Sufficiency* measuring group fairness metrics such as predictive parity constrain that the ground class Y is independent of $\mathbf{A}$ given the prediction $\widehat{Y}$ [86]. Formally,

$$f_{\mathsf{PP}}(\mathcal{M},\mathcal{D}) \triangleq \max(\max_{\mathbf{a}} \Pr[Y=1|\mathbf{A}=\mathbf{a},\widehat{Y}=0] - \min_{\mathbf{a}} \Pr[Y=1|\mathbf{A}=\mathbf{a},\widehat{Y}=0],$$
$$\max_{\mathbf{a}} \Pr[Y=1|\mathbf{A}=\mathbf{a},\widehat{Y}=1] - \min_{\mathbf{a}} \Pr[Y=1|\mathbf{A}=\mathbf{a},\widehat{Y}=1]).$$

Path-specific Causal Fairness (PCF). Let $\mathbf{a}_{\max} \triangleq \arg\max_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a}]$. We consider mediator features $\mathbf{Z} \subseteq \mathbf{X}$ sampled from the conditional distribution $\mathcal{Z}_{|\mathbf{A}=\mathbf{a}_{\max}}$. This emulates the fact that mediator variables have the same sensitive features $\mathbf{a}_{\max}$. To this end, the path-specific causal fairness, abbreviated as PCF, of a classifier is

$$f_{\mathsf{PCF}}(\mathcal{M},\mathcal{D}) \triangleq \max_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a},\mathbf{Z}] - \min_{\mathbf{a}} \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a},\mathbf{Z}].$$

Therefore, path-specific causal fairness constrains that the prediction $\widehat{Y}$ is not directly dependent of sensitive features $\mathbf{A}$ while $\mathbf{A}$ may indirectly affects $\widehat{Y}$ only through mediator features $\mathbf{Z}$. Hence, path-specific causal fairness is a variation of counterfactual fairness and causal fairness without mediator features [63].

Example 2.3.1. Following [63], we consider a classifier that decides the hiring of employees based on three features: gender (sensitive), years of experience (non-sensitive), and college-participation (mediator). It is practical to consider that gender $\in$ {male, female} can affect the college-participation of individuals, and all three features are determining factors for the hiring process. Let 'male' be the most favored group by the classifier, for instance. Path-specific causal fairness ensures that a female candidate should be given a job offer with similar probability as a male candidate. She, however, went to (participated in) college as if she were a male candidate while other non-mediator features such as 'years of experience' are the same. Therefore, path-specific causal fairness measures the effect of gender on job offer, but ignores the effect of gender on whether candidates went to college.

Fairness Certification. We certify the fairness of a classifier by comparing $f(\mathcal{M},\mathcal{D})$ with a fairness threshold, denoted by $\epsilon \in [0,1]$, that quantifies the desired level of fairness. In particular, a classifier is ϵ-fair with respect to statistical parity, equalized odds, predictive parity, and path-specific causal fairness if and only if $f(\mathcal{M},\mathcal{D}) \leq \epsilon$. In contrast, a classifier achieves $(1-\epsilon)$-disparate impact if and only if $f(\mathcal{M},\mathcal{D}) \geq 1-\epsilon$. In all above fairness metrics, a lower value of ϵ refers to higher fairness of the classifier.

2.3.3 Bayesian Network

In general, a Probabilistic Graphical Model [87], and specifically a *Bayesian network* [88,89], encodes the dependencies and conditional independence between a set of random variables. In fairness verification with correlated features, we leverage an access to a Bayesian network on sensitive and non-sensitive features $\mathbf{X} \cup \mathbf{A}$ that represents the joint distribution on them. A Bayesian network is denoted by a pair (G, θ), where $G \triangleq (\mathbf{V}, \mathbf{E})$ is a DAG (Directed Acyclic Graph), and θ is a set of parameters encoding the conditional probabilities induced by the joint distribution under investigation. Each vertex $V_i \in \mathbf{V}$ corresponds to a random variable. Edges $\mathbf{E} \in \mathbf{V} \times \mathbf{V}$ imply conditional dependencies among variables. For each variable $V_i \in \mathbf{V}$, let $\mathrm{Pa}(V_i) \subseteq \mathbf{V} \setminus \{V_i\}$ denote the set of parents of V_i. Given $\mathrm{Pa}(V_i)$ and parameters θ, V_i is independent of its other non-descendant variables in G. Thus, for the assignment v_i of V_i and $\mathbf{u}$ of $\mathrm{Pa}(V_i)$, the aforementioned semantics of a Bayesian network encodes the joint distribution of $\mathbf{V}$ as:

$$\Pr[V_1 = v_1, \ldots, V_{|\mathbf{V}|} = v_{|\mathbf{V}|}] = \prod_{i=1}^{|\mathbf{V}|} \Pr[V_i = v_i | \mathrm{Pa}(V_i) = \mathbf{u}; \theta]. \tag{2.2}$$

The complexity $C(G)$ of a Bayesian network (G, θ) with discrete random variables $\mathbf{V}$ is defined as the number of independent parameters used to define the probability distribution of $\mathbf{V}$ [90]. Let $\mathsf{Card}(V)$ denote the number of distinct values that a random variable V can take. As such, for each conditional probability $\Pr[V_i = v_i | \mathrm{Pa}(V_i) = \mathbf{u}; \theta]$, we need $\mathsf{Card}(\mathrm{Pa}(V_i))(\mathsf{Card}(V_i) - 1)$ independent parameters. Thus, the total complexity of a Bayesian network is:

$$C(G) = \sum_{i=1}^{|\mathbf{V}|} \mathsf{Card}(\mathrm{Pa}(V_i))(\mathsf{Card}(V_i) - 1) \tag{2.3}$$

2.3.4 Global Sensitivity Analysis: Variance Decomposition

Global sensitivity analysis is a field that studies how the global uncertainty in the output of a function can be attributed to the different sources of uncertainties in the input while considering the whole input domain [91]. Sensitivity analysis is an essential component for quality assurance and impact assessment of models in EU [92], USA [93], and research communities [94]. *Variance-based sensitivity analysis* is a form of global sensitivity analysis, where variance is considered as the measure of uncertainty [95, 96]. To illustrate, let us consider a real-valued function $g(\mathbf{Z})$, where $\mathbf{Z}$ is a vector of k input variables $\{Z_1, \ldots, Z_k\}$. Now, we decompose $g(\mathbf{Z})$ among the subsets of inputs, such that:

$$g(\mathbf{Z}) = g_0 + \sum_{i=1}^{k} g_{\{i\}}(Z_i) + \sum_{i<j}^{k} g_{\{i,j\}}(Z_i, Z_j) + \cdots + g_{\{1,2,\ldots,k\}}(Z_1, Z_2, \ldots, Z_k)$$
$$= g_0 + \sum_{\mathbf{S}\subseteq[k]\setminus\emptyset} g_{\mathbf{S}}(\mathbf{Z}_{\mathbf{S}}) \tag{2.4}$$

In this decomposition, g_0 is a constant, $g_{\{i\}}$ is a function of Z_i, $g_{\{i,j\}}$ is a function of Z_i and Z_j, and so on. Here, $[k] \triangleq \{1, 2, \ldots, k\}$ and $\mathbf{S}$ is an ordered subset of $[k] \setminus \emptyset$. We denote $\mathbf{Z}_{\mathbf{S}} \triangleq \{Z_i | i \in \mathbf{S}\}$ as the input of $g_{\mathbf{S}}$, where $\mathbf{Z}_{\mathbf{S}}$ is a set of variables with indices belonging to $\mathbf{S}$. The standard condition of this decomposition is the orthogonality of each term in the right-hand side of Eq. (2.4) [95]. $g_{\mathbf{S}}(\mathbf{Z}_{\mathbf{S}})$ is the effect of varying all the features in $\mathbf{Z}_{\mathbf{S}}$ simultaneously. For $|\mathbf{S}| = 1$, it quantifies an individual variable's effect. For $|\mathbf{S}| > 1$, it quantifies the higher-order interactive effect of variables.

Now, if we assume g to be square integrable, we obtain the decomposition of the variance of the output [95].

$$\mathsf{Var}[g(\mathbf{Z})] = \sum_{i=1}^{k} V_{\{i\}} + \sum_{i<j}^{k} V_{\{i,j\}} + \cdots + V_{\{1,2\ldots,k\}} = \sum_{\mathbf{S}\subseteq[k]\setminus\emptyset} V_{\mathbf{S}} \tag{2.5}$$

where $V_{\{i\}}$ is the variance of $g_{\{i\}}$, $V_{\{i,j\}}$ is the variance of $g_{\{i,j\}}$ and so on. Formally,

$$V_{\mathbf{S}} \triangleq \mathsf{Var}_{\mathbf{Z}_{\mathbf{S}}}\left[\mathbb{E}_{\mathbf{Z}/\mathbf{Z}_{\mathbf{S}}}[g(\mathbf{Z}) \mid \mathbf{Z}_{\mathbf{S}}]\right] - \sum_{\mathbf{S}'\subset\mathbf{S}/\emptyset} V_{\mathbf{S}'}.$$

Here, $\mathbf{S}'$ denotes all the ordered proper subsets of $\mathbf{S}$. This variance decomposition shows how the variance of $g(\mathbf{Z})$ can be decomposed into terms attributable to each input, as well as the interactive effects among them. Together all terms sum to the total variance of the model output.

Part II

Interpretable Machine Learning

In this part of the thesis, we discuss an incremental learning framework for interpretable classification rules in Chapter 3, and extend incremental learning to more expressible classification rules in Chapter 4.

Chapter 3

Efficient Learning of Interpretable Classification Rules

Machine learning models are presently deployed in safety-critical and high-stake decision-making arising in medical [3–5], law [6, 7], and transportation [9, 10]. In safety-critical domains, a surge of interest has been observed in designing interpretable, robust, and fair models instead of merely focusing on an accuracy-centric learning objective [97–104]. In this chapter, we focus on learning an interpretable machine learning classifier [12], where the rules governing the prediction are explicit (by design) in contrast to black-box models.

In order to achieve the interpretability of predictions, decision functions in the form of classification rules such as decision trees, decision lists, decision sets, etc. are particularly effective [21–31]. Such models are used to either learn an interpretable model from the start or as proxies to provide post-hoc explanations of pre-trained black-box models [16–20]. In our context of rule-based classifiers, we use the sparsity of rules as a measure of interpretability, which is adopted in various domains, specifically in the medical domain [27, 32–35].

In this chapter, we study decision functions expressible in *propositional logic*. In particular, we focus on learning logical formulas from data as classifiers and define interpretability in terms of the number of Boolean literals that the formula contains. In the following, we illustrate an interpretable classifier that decides if a tumor cell is malignant or benign based on different features of tumors.

A tumor is malignant if
[(compactness SE < 0.1) **OR** $\neg(0.1 \leq$ concave points $< 0.2)$] **AND**
[$\neg(0.2 \leq$ area $< 0.3)$ **OR** $\neg(0.1 \leq$ largest symmetry $< 0.2)$]

Example 3.0.1. We learn an interpretable classification rule in CNF for predicting the classification of a tumor cell. We consider WDBC (Wisconsin Diagnostic Breast Cancer) dataset [105] to learn a CNF formula with two *clauses* and four Boolean *literals*. In the formula, clauses are connected by Boolean 'AND', where each clause contains *literals*

connected by Boolean 'OR'. Informally, a tumor cell is malignant if at least one literal in each clause becomes true.

The problem of learning rule-based classifiers is known to be computationally intractable. The earliest tractable approaches for classifiers such as decision trees and decision lists relied on heuristically chosen objective functions and greedy algorithmic techniques [37–39]. In these approaches, the size of rules is controlled by early stopping, ad-hoc rule pruning, etc. In recent approaches, the interpretable classification problem is reduced to an optimization problem, where the accuracy and the sparsity of rules are optimized jointly [26,28]. Different optimization solvers such as linear programming [34], sub-modular optimizations [26], and Bayesian optimizations [27] are then deployed to find the best classifier with maximum accuracy and minimum rule-size. In our study, we propose an alternate optimization approach that fits particularly well to rule-learning problems. Particularly, we propose a *maximum satisfiability* (MaxSAT) solution for learning interpretable rule-based classifiers.

The MaxSAT problem is an optimization analog to the satisfiability (SAT) problem, which is complete for the class FP^{NP}. Although the MaxSAT problem is NP-hard, dramatic progress has been made in designing solvers that can handle large-scale problems arising in practice. This has encouraged researchers to reduce several optimization problems into MaxSAT such as optimal planning [106], automotive configuration [107], group-testing [108], data analysis in machine learning [109], and automatic test pattern generation for cancer therapy [110].

Contributions. The primary contribution of this chapter is a MaxSAT-based formulation, called IMLI (**I**ncremental **M**axSAT-based **L**earning of **I**nterpretable classification rules), for learning interpretable classification rules expressible in propositional logic. For the simplicity of exposition, our initial focus is on learning classifiers expressible in CNF formulas; we later discuss how the CNF learning formulation can be extended to popular interpretable classifiers such as decision lists and decision sets. In this chapter, IMLI provides precise control to jointly optimize the accuracy and the size of classification rules. IMLI constructs and solves a MaxSAT query of $\mathcal{O}(kn)$ size (number of clauses) to learn a k-clause CNF formula on a dataset containing n samples. Naturally, the naïve formulation cannot scale to large values of n and k. To scale IMLI to large datasets, we propose an incremental learning technique along with the MaxSAT formulation. Our incremental learning is an integration of mini-batch learning and iterative rule-learning, which are studied separately in classical learning problems. In the presented incremental approach, IMLI learns a k-clause CNF formula using an iterative separate-and-conquer algorithm, where in each iteration, a single clause is learned by covering a part of the training data. Furthermore, to efficiently learn a single clause in a CNF, IMLI relies on mini-batch learning, where it solves a sequence of smaller MaxSAT queries corresponding to each mini-batch.

In our experimental evaluations, IMLI demonstrates the best balance among prediction accuracy, interpretability, and scalability in learning classification rules. In particular,

IMLI achieves competitive prediction accuracy and interpretability w.r.t. state-of-the-art interpretable classifiers. Besides, IMLI achieves impressive scalability by classifying datasets even with 1 million samples, wherein existing classifiers, including those of the non-interpretable ones, either fail to scale or achieve poor prediction accuracy. Finally, as an application, we deploy IMLI in learning interpretable classifiers such as decision lists and decision sets.

3.1 Related Work

The progress in designing interpretable rule-based classifiers finds its root in the development of decision trees [21, 111, 112], decision lists [29], classification rules [113] etc. In early works, the focus was to improve the efficiency and scalability of the model rather than designing models that are interpretable. For example, decision rule approaches such as C4.5 rules [39], CN2 [37], RIPPER [113], and SLIPPER [38] rely on heuristic-based branch pruning and ad-hoc local criteria e.g., maximizing information gain, coverage, etc.

Recently, several optimization frameworks have been proposed for interpretable classification, where both accuracy and rule-size are optimized during training. For example, [34] proposed exact learning of rule-based classifiers based on Boolean compressed sensing using a linear programming formulation. [114] presented two-level Boolean rules, where the trade-off between classification accuracy and interpretability is studied. In their work, the Hamming loss is used to characterize prediction accuracy, and sparsity is used to characterize the interpretability of rules. [30] proposed a Bayesian optimization framework for learning falling rule lists, which is an ordered list of if-then rules. Other similar approaches based on Bayesian analysis for learning classification rules are [27, 115]. Building on custom discrete optimization techniques, [116] proposed an optimal learning technique for decision lists using a branch-and-bound algorithm. In a separate study, [26] highlighted the importance of decision sets over decision lists in terms of interpretability and considered a sub-modular optimization problem for learning a near-optimal solution for decision sets. Our proposed method for interpretable classification, however, relies on the improvement in formal methods over the decades, particularly the efficient CDCL-based solution for satisfiability (SAT) problems [117].

Formal methods, particularly SAT and its variants, have been deployed in interpretable classification in recent years. In the context of learning decision trees, SAT and MaxSAT-based solutions are proposed by [28, 118–120]. In addition, researchers have applied SAT for learning explainable decision sets [121–124]. In most cases, SAT/MaxSAT solutions are not sufficient in solving large-scale classification tasks because of the NP-hardness of the underlying problem. This observation motivates us in combining MaxSAT with more practical algorithms such as incremental learning.

Incremental learning has been studied in improving the scalability of learning problems, where data is processed in parts and results are combined to use lower computation

overhead. In case of non-interpretable classifiers such as SVM, several solutions adopting incremental learning are available [125,126]. For example, [127] proposed an online recursive algorithm for SVM that learns one support-vector at a time. Based on radial basis kernel function, [128] proposed a local incremental learning algorithm for SVM. In the context of deep neural networks, stochastic gradient descent is a well-known convex optimization technique—a variant of which includes computing the gradient on mini-batches [129–131]. Federated learning, on the other hand, decentralizes training on multiple local nodes based on local data samples with only exchanging learned parameters to construct a global model in a central node [132, 133]. Another notable technique is Lagrangian relaxation that decomposes the original problem into several sub-problems, assigns Lagrangian multipliers to make sure that sub-problems agree, and iterates by solving sub-problems and adjusting weights based on disagreements [134–136]. To the best of our knowledge, our method is the first method that unifies incremental learning with MaxSAT based formulation to improve the scalability of learning rule-based classifiers.

Classifiers that are interpretable by design can be applied to improve the explainability of complex black-box machine learning classifiers. There is rich literature on extracting decompositional and pedagogical rules from non-linear classifiers such as support vector machines [137–141] and neural networks [142–147]. In recent years, local model-agnostic approaches for explaining black-box classifiers are proposed by learning surrogate simpler classifiers such as rule-based classifiers [148–151]. The core idea in local approaches is to use a rule-learner that can classify synthetically generated neighboring samples with class labels provided by the black-box classifier. To this end, our framework IMLI can be directly deployed as an efficient rule-learner and can explain the inner-working of black-box classifiers by generating interpretable rules.

3.2 Problem Formulation

Given

1. a dataset $\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ of n samples, where feature vector $\mathbf{x}^{(i)} \in \{0,1\}^m$ contains m features and class label $y^{(i)} \in \{0,1\}$,

2. a positive integer $k \geq 1$ denoting the number of clauses to be learned in the classification rule, and

3. a regularization parameter $\lambda \in \mathbb{R}^+$,

we learn a classifier $\mathcal{R}$ represented as a k-clause CNF formula separating samples of class 1 from class 0.

Our goal is to learn classifiers that balance two goals: of being accurate but also interpretable. Various notions of interpretability have been proposed in the context of classification problems. A common proxy for interpretability in the context of decision rules

is the sparsity of rule. For instance, a rule involving fewer literals is highly interpretable. In this work, we minimize the total number of literals in all clauses, which motivates us to find $\mathcal{R}$ with minimum $|\mathcal{R}|$. Let $\mathcal{R}$ classify all samples correctly during training. Among all the classification rules that classify all samples correctly, we choose the sparsest (most interpretable) such $\mathcal{R}$.

$$\min_{\mathcal{R}} |\mathcal{R}| \text{ such that } \forall i, y^{(i)} = \mathcal{R}(\mathbf{x}^{(i)})$$

In practical classification tasks, perfect classification is unlikely. Hence, we need to balance interpretability with classification error. Let $\mathcal{E}_{\mathbf{D}} = \{(\mathbf{x}^{(i)}, y^{(i)}) | y^{(i)} \neq \mathcal{R}(\mathbf{x}^{(i)})\}$ be the set of samples in $\mathbf{D}$ that are misclassified by $\mathcal{R}$. Therefore, we balance between classification-accuracy and rule-sparsity and optimize the following function.[1]

$$\min_{\mathcal{R}} \ |\mathcal{E}_{\mathbf{D}}| + \lambda |\mathcal{R}| \tag{3.1}$$

Higher values of λ generate a rule with a smaller rule-size but of more training errors, and vice-versa. Thus, λ can be tuned to trade-off between accuracy and interpretability for a rule-based classifier, which we experiment extensively in Section 3.6.

3.3 MaxSAT-based Interpretable Classification Rule Learning

In this section, we discuss a MaxSAT-based framework for optimal learning of an interpretable rule-based classifier, particularly a CNF classifier $\mathcal{R}$. We first describe the decision variables in Section 3.3.1 and present the MaxSAT encoding in Section 3.3.2. Our MaxSAT formulation assumes binary features as input. We conclude this section by learning $\mathcal{R}$ with non-binary features in Section 3.3.3 and discussing more flexible interpretability constraints of $\mathcal{R}$ in Section 3.3.4.

3.3.1 Description of Variables

We initially preprocess feature vector $\mathbf{x}$ to account for the negation of Boolean features while learning a classifier.[2] In the preprocessing step, we negate each feature in $\mathbf{x}$ to a new feature and append it to $\mathbf{x}$. For example, if "age ≥ 25" is a Boolean feature, we add another feature "age < 25" in $\mathbf{x}$ by negating the feature "age ≥ 25". Hence, in the rest of the chapter, we refer m as the modified number of features in $\mathbf{x}$. We next discuss the variables in the MaxSAT problem.

[1] In our formulation, it is straightforward to add class-conditional weights (e.g., to penalize false-alarms more than mis-detects), and to allow instance weights (per sample).

[2] This preprocessing is similarly applied in [34].

We consider two types of Boolean variables: (i) *feature* variables B corresponding to input features and (ii) *error* variables ξ corresponding to the classification error of samples. We define a Boolean variable B_j^i that becomes true if feature X_j appears in the i^{th} clause of $\mathcal{R}$, thereby contributing to an increase in the rule-size of $\mathcal{R}$, and B_j^i is assigned false, otherwise. Moreover, we define an error variable ξ_l to attribute to whether the l^{th} sample $(\mathbf{x}^{(l)}, y^{(l)})$ is classified correctly or not. Specifically, ξ_l becomes true if $(\mathbf{x}^{(l)}, y^{(l)})$ is misclassified, and becomes false otherwise. We next discuss the MaxSAT encoding to solve the classification problem.

3.3.2 MaxSAT Encoding

We consider a partial-weighted MaxSAT formula, where we encode the objective function in Eq. (3.1) as soft clauses and the learning constraints as hard clauses. We next discuss the MaxSAT encoding in detail.

- **Soft clauses for maximizing training accuracy:** For each training sample, IMLI constructs a soft unit[3] clause $\neg\xi_l$ to account for a penalty for misclassification. Since the penalty for misclassification of a sample is 1 in Eq. (3.1), the weight of this soft clause is also 1.

$$E_l := \neg\xi_l; \qquad W(E_l) = 1 \tag{3.2}$$

  Intuitively, if a sample is misclassified, the associated error variable becomes true, thereby dissatisfying the soft clause E_l.

- **Soft clauses for minimizing rule-sparsity:** To favor rule-sparsity, IMLI tries to learn a classifier with as few literals as possible. Hence, for each feature variable B_j^i, IMLI constructs a unit clause as $\neg B_j^i$. Similar to training accuracy, the weight for this clause is derived as λ from Eq. (3.1).

$$S_j^i := \neg B_j^i; \qquad W\big(S_j^i\big) = \lambda \tag{3.3}$$

- **Hard clauses for encoding constraints:** In a MaxSAT problem, constraints that must be satisfied are encoded as hard clauses. In IMLI, we have a learning constraint that, if the error variable is false, the associated sample must be correctly classified, and vice-versa. Let $\mathbf{B}_i = \{B_j^i \mid j \in \{1, \ldots, m\}\}$ be a vector of feature variables corresponding to the i^{th} clause in $\mathcal{R}$. Then, we define the following hard clause.

$$H_l := \neg\xi_l \rightarrow \Big(y^{(l)} \leftrightarrow \bigwedge_{i=1}^{k} \mathbf{x}^{(l)} \circ \mathbf{B}_i\Big); \qquad W(H_l) = \infty \tag{3.4}$$

[3]A unit clause has a single literal.

In the hard clause, $\bigwedge_{i=1}^{k} \mathbf{x}^{(l)} \circ \mathbf{B}_i$ is a CNF formula including variables B_j^i for which the associated feature-value is 1 in $\mathbf{x}^{(l)}$. Since $y^{(l)} \in \{0,1\}$ is a constant, the constraint to the right of the implication "$\rightarrow$" is either $\bigwedge_{i=1}^{k} \mathbf{x}^{(l)} \circ \mathbf{B}_i$ or its complement. Therefore, the hard clause enforces that if the sample is correctly classified (using $\neg\xi_l$), either $\bigwedge_{i=1}^{k} \mathbf{x}^{(l)} \circ \mathbf{B}_i$ or its complement is true depending on the class-label of the sample. We highlight that the single-implication "$\rightarrow$" in the hard clause H_l acts as a double-implication "$\leftrightarrow$" due to the soft clause E_l. Because, according to the definition of "$\rightarrow$", the left constraint $\neg\xi_l$ can be false while the right constraint of "$\rightarrow$" is true. This, however, incurs unnecessarily dissatisfying the soft clause E_l, which is a sub-optimal solution and hence this solution is not returned by the MaxSAT solver.

We next discuss the translation of soft and hard clauses into a CNF formula, which can be invoked by any MaxSAT solver.

Translating E_l, S_j^i, H_l to a CNF formula. The soft clauses E_l and S_j^i are unit clauses and hence, no translation is required for them. In the hard clause, when $y^{(l)} = 1$, the simplification is $H_l := \neg\xi_l \rightarrow \bigwedge_{i=1}^{k} \mathbf{x}^{(l)} \circ \mathbf{B}_i$. In this case, we apply the equivalence rule in propositional logic $(A \rightarrow B) \equiv (\neg A \vee B)$ to encode H_l into CNF. In contrast, when $y_i = 0$, we simplify the hard clause as $H_l := \neg\xi_l \rightarrow \neg(\bigwedge_{i=1}^{k} \mathbf{x}^{(l)} \circ \mathbf{B}_i) \Rightarrow \neg\xi_l \rightarrow \bigvee_{i=1}^{k} \neg(\mathbf{x}^{(l)} \circ \mathbf{B}_i)$. Since $\mathbf{x}^{(l)} \circ \mathbf{B}_i$ constitutes a disjunction of literals, we apply Tseytin transformation to encode $\neg(\mathbf{x}^{(l)} \circ \mathbf{B}_i)$ into CNF. More specifically, we introduce an auxiliary variable $z_{l,i}$ corresponding to the clause $\neg(\mathbf{x}^{(l)} \circ \mathbf{B}_i)$. Formally, we replace $H_l := \neg\xi_l \rightarrow \bigvee_{i=1}^{k} \neg(\mathbf{x}^{(l)} \circ \mathbf{B}_i)$ with $\bigwedge_{i=0}^{k} H_{l,i}$, where $H_{l,0} := (\neg\xi_l \rightarrow \bigvee_{i=1}^{k} z_{l,i})$ and $H_{l,i} := z_{l,i} \rightarrow \neg(\mathbf{x}^{(l)} \circ \mathbf{B}_i)$ for $i = \{1, \ldots, k\}$. Finally, we apply the equivalence of $(A \rightarrow B) \equiv (\neg A \vee B)$ on $H_{l,i}$ to translate them into CNF. For either value of $y^{(l)}$, the weight of each translated hard clause in the CNF formula is ∞.

Once we translate all soft and hard clauses into CNF, the MaxSAT query Q is the conjunction of all clauses.

$$Q := \bigwedge_{l=1}^{n} E_l \wedge \bigwedge_{i=1,j=1}^{i=k,j=m} S_j^i \wedge \bigwedge_{l=1}^{n} H_l$$

Any off-the-shelf MaxSAT solver can output an optimal assignment σ^* of the MaxSAT query $(Q, W(\cdot))$. We extract σ^* to construct the classifier $\mathcal{R}$ and compute training errors as follows.

Construction 2. *Let $\sigma^* = \textsc{MaxSAT}(Q, W(\cdot))$. Then $X_j \in \mathsf{clause}(\mathcal{R}, i)$ if and only if $\sigma^*(B_j^i) = 1$. Additionally, $(\mathbf{x}^{(l)}, y^{(l)})$ is misclassified if and only if $\sigma^*(\xi_l) = 1$.*

Complexity of the MaxSAT query. We analyze the complexity of the MaxSAT query in terms of the number of Boolean variables and clauses in the CNF formula Q.

Proposition 3. *To learn a k-clause CNF classifier for a dataset of n samples over m boolean features, the MaxSAT query Q defines $km + n$ Boolean variables. Let n_{neg} be the number of negative samples in the training dataset. Then the number of auxiliary variables in Q is kn_{neg}.*

Proposition 4. *The MaxSAT query Q has $km + n$ unit clauses corresponding to the constraints E_l and S^i_j. For each positive sample, the hard clause H_l is translated into k clauses. For each negative sample, the CNF translation requires at most $km+1$ clauses. Let n_{pos} and n_{neg} be the number of positive and negative samples in the training set. Therefore, the number of clauses in the MaxSAT Query Q is $km+n+kn_{pos}+(km+1)n_{neg} \approx \mathcal{O}(kmn)$ when $n_{pos} = n_{neg} = \frac{n}{2}$.*

Example 3.3.1. *MaxSAT Encoding.*

We illustrate the MaxSAT encoding for a toy example consisting of four samples and two binary features. Our goal is to learn a two clause CNF classifier that can approximate the training data.

$$\mathbf{D}_{\mathsf{orig}} = \begin{array}{c} \begin{matrix} X_1 & X_2 & Y \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \end{array} \implies \mathbf{D} = \begin{array}{c} \begin{matrix} X_1 & \neg X_1 & X_2 & \neg X_2 & Y \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

In the preprocessing step, we negate the features $\{X_1, X_2\}$ in $\mathbf{D}_{\mathsf{orig}}$ and add complemented features $\{\neg X_1, \neg X_2\}$ in $\mathbf{D}$. In the MaxSAT encoding, we define 8 feature variables (4 features $\times$ 2 clauses in the classifier) and 4 error variables. For example, for feature X_2, introduced feature variables are $\{B^1_3, B^2_3\}$ and for feature $\neg X_2$, introduced variables are $\{B^1_4, B^2_4\}$. For four samples, error variables are $\{\xi_1, \xi_2, \xi_3, \xi_4\}$. We next show the soft and hard clauses in the MaxSAT encoding

$$\begin{aligned}
&E_1 := (\neg\xi_1); \quad E_2 := (\neg\xi_2); \quad E_3 := (\neg\xi_3); \quad E_4 := (\neg\xi_4) \\
&S^1_1 := (\neg B^1_1); \quad S^1_2 := (\neg B^1_2); \quad S^1_3 := (\neg B^1_3); \quad S^1_4 := (\neg B^1_4); \\
&S^2_1 := (\neg B^2_1); \quad S^2_2 := (\neg B^2_2); \quad S^2_3 := (\neg B^2_3); \quad S^2_4 := (\neg B^2_4); \\
&H_1 := (\neg\xi_1 \rightarrow ((B^1_2 \vee B^1_4) \wedge (B^2_2 \vee B^2_4))); \\
&H_2 := (\neg\xi_2 \rightarrow (\neg(B^1_2 \vee B^1_3) \vee \neg(B^2_2 \vee B^2_3))); \\
&H_3 := (\neg\xi_3 \rightarrow (\neg(B^1_1 \vee B^1_4) \vee \neg(B^2_1 \vee B^2_4))); \\
&H_4 := (\neg\xi_4 \rightarrow ((B^1_1 \vee B^1_3) \wedge (B^2_1 \vee B^2_3)));
\end{aligned}$$

In this example, we consider regularizer $\lambda = 0.1$, thereby setting the weight on accuracy as $W(E_i) = 1$ and rule-sparsity weight as $W\big(S^i_j\big) = 0.1$. Intuitively, the penalty for misclassifying a sample is 10 times than the penalty for adding a feature in the classifier.

For this MaxSAT query, the optimal solution classifies all samples correctly by assigning four feature variables $\{B_1^1, B_4^1, B_2^2, B_3^2\}$ to true. Therefore, by applying Construction 2, the classifier is $(X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2)$.[4]

3.3.3 Learning with Non-binary Features

The presented MaxSAT encoding requires input samples to have binary features. Therefore, we discretize datasets with categorical and continuous features into binary features. For each continuous feature, we apply equal-width discretization that splits the feature into a fixed number of bins. For example, consider a continuous feature $X_c \in [a, b]$. In discretization, we split the domain $[a, b]$ into three bins with two split points $\{a', b'\}$ such that $a < a' < b' < b$. Therefore, the resulting three discretized features are $a \leq X_c < a'$, $a' \leq X_c < b'$, and $b' \leq X_c \leq b$. An alternate to this close-interval discretization is open-interval discretization, where we consider six discretized features with each feature being compared with one threshold. In that case, the discretized features are: $X_c \geq a, X_c \geq a', X_c \geq b', X_c < a', X_c < b', X_c \leq b$. Both open-interval and close-interval discretization techniques have their use-cases where one or the other is appropriate. For simplicity, we experiment with close-interval discretization in this chapter.

After applying discretization on continuous features, the dataset contains categorical features only, which we convert to binary features using one-hot encoding [33]. In one-hot encoding, a Boolean vector of features is introduced with cardinality equal to the number of distinct categories. For example, consider a categorical feature having three categories 'red', 'green', and 'yellow'. In one hot encoding, samples with category-value 'red', 'green', and 'yellow' would be converted into binary features by taking values 100, 010, and 001, respectively.

3.3.4 Flexible Interpretability Objectives

IMLI can be extended to consider more flexible interpretability objectives than the simplified one in Eq. (3.1). In Eq. (3.1), we prioritize all features equally by providing the same weight to the clause S_j^i for all values of i and j. In practice, users may prefer rules containing certain features. In IMLI, such an extension can be achieved by modifying the weight function and/or the definition of S_j^i. For example, to constrain the classifier to never include a feature, the weight of the clause $S_j^i := \neg B_j^i$ can be set to ∞. In contrast, to always include a feature, we can define $S_j^i := B_j^i$ with weight ∞. In both cases, we treat S_j^i as a hard clause.

Another use case may be to learn rules where clauses have disjoint set of features. To this end, we consider a pseudo-Boolean constraint $\sum_{i=1}^{k} B_j^i \leq 1$, which specifies that

[4]The presented MaxSAT-based formulation does not learn a CNF classifier with both X_i and $\neg X_i$ in the same clause. The reason is that a clause with both X_i and $\neg X_i$ connected by OR ($\vee$) does not increase accuracy but increases rule-size and hence, this is not an optimal classifier.

the feature X_j appears in at-most one of the k clauses. This constraint may be soft or hard depending on the priority in the application domain. In either case, we convert this constraint to CNF using pseudo-Boolean to CNF translation [152]. Thus, IMLI allows us to consider varied interpretability constraints by only modifying the MaxSAT query without requiring changes in the MaxSAT solving. Therefore, the *separation between modeling and solving* turns out to be the key strength of IMLI.

3.4 Incremental Learning with MaxSAT-based formulation

The complexity of the MaxSAT query in Section 3.3.2 increases with the number of samples in the training dataset and the number of clauses to be learned in a CNF formula. In this section, we discuss an incremental learning technique that along with the MaxSAT formulation can generate classification rules efficiently. Our incremental learning is built on two concepts: (i) mini-batch learning and (ii) iterative learning. We discuss both in detail in the following.

3.4.1 Mini-batch Learning

Our first improvement is to implement a mini-batch learning technique tailored for rule-based classifiers. Mini-batch learning has two-fold advantages. Firstly, instead of solving a large MaxSAT query for the whole training data, this approach solves a sequence of smaller MaxSAT queries derived from mini-batches of smaller size. Secondly, this approach extends to online learning, where the classifier can be updated incrementally with new samples while also generalizing to previously observed samples. In our context of rule-based classifiers, we consider the following heuristic in mini-batch learning.

A Heuristic for Mini-Batch Learning. Let $\mathcal{R}'$ be a classifier learned on the previous batch. In mini-batch learning, we aim to learn a new classifier $\mathcal{R}$ that can generalize to both the current batch and previously seen samples. For that, we consider a soft constraint such that $\mathcal{R}$ does not *differ much* from $\mathcal{R}'$ while training on the current batch. Thus, we hypothesize that by constraining $\mathcal{R}$ to be syntactically similar to $\mathcal{R}'$, it is possible to generalize well; because samples in all batches originate from the same distribution[5]. Since our study focuses on rule-based classifiers, we consider the Hamming distance between two classifiers as a notion of their syntactic dissimilarity. In the following, we define the

[5]We propose Hamming distance heuristics in IMLI with the assumption that distribution remains same across batches. One way to account for distribution shifts is to consider last p (> 1) batches instead of the (single) previous batch in the objective function in mini-batch learning. For a feature variable B^i_j, we consider its majority assignment in last p classification rules and encode as a soft clause to retain the majority assignment in the current batch. Moreover, we can reweigh the soft clause by prioritizing assignments of B^i_j in recent batches.

Hamming distance between two CNF classifiers $\mathcal{R}$, $\mathcal{R}'$ with the same number of clauses as follows.

$$d_H(\mathcal{R}, \mathcal{R}') = \sum_{i=1}^{k} \Big(\sum_{v \in C_i} \mathbb{1}(v \notin C'_i) + \sum_{v \in C'_i} \mathbb{1}(v \notin C_i) \Big),$$

where C_i and C'_i are the i^{th} clause in $\mathcal{R}$ and $\mathcal{R}'$, respectively and $\mathbb{1}$ is an indicator function that returns 1 if the input is true and returns 0 otherwise. Intuitively, $d_H(\mathcal{R}', \mathcal{R})$ calculates the total number of different literals in each (ordered) pair of clauses in $\mathcal{R}$ and $\mathcal{R}'$. For example, consider $\mathcal{R} = (X_1 \vee X_2) \wedge (\neg X_1)$ and $\mathcal{R}' = (\neg X_1 \vee X_2) \wedge (\neg X_1)$. Then $d_H(\mathcal{R}, \mathcal{R}') = 2 + 0 = 2$, because in the first clause, the literals in $\{X_1, \neg X_1\}$ are absent in either formulas, and the second clause is identical for both $\mathcal{R}$ and $\mathcal{R}'$. In the following, we discuss a modified objective function for mini-batch learning.

Objective Function. In mini-batch learning, we design an objective function to penalize both classification errors on the current batch and the Hamming distance between new and previous classifiers. Let $\mathbf{D}_b \subset \mathbf{D}$ be the current mini-batch, where $|\mathbf{D}_b| \ll |\mathbf{D}|$. The objective function in mini-batch learning is

$$\min_{\mathcal{R}} |\mathcal{E}_{\mathbf{D}_b}| + \lambda d_H(\mathcal{R}, \mathcal{R}'). \tag{3.5}$$

In the objective function, $\mathcal{E}_{\mathbf{D}_b}$ is the misclassified subset of samples in the current batch $\mathbf{D}_b$. Unlike controlling the rule-sparsity in the non-incremental approach in the earlier section, in Eq. (3.5), λ *controls the trade-off between classification errors and the syntactic differences between consecutive classifiers*. Next, we discuss how to encode the modified objective function as a MaxSAT query.

MaxSAT Encoding of Mini-batch Learning. In order to account for the modified objective function, we only modify the soft clause S^i_j in the MaxSAT query Q. In particular, we define S^i_j to penalize for the complemented assignment of the feature variable B^i_j in $\mathcal{R}$ compared to $\mathcal{R}'$.

$$S^i_j := \begin{cases} B^i_j & \text{if } X_j \in \mathsf{clause}(\mathcal{R}', i) \\ \neg B^i_j & \text{otherwise} \end{cases}; \qquad W\big(S^i_j\big) = \lambda$$

S^i_j is either a unit clause B^i_j or $\neg B^i_j$ depending on whether the associated feature X_j appears in the previous classifier $\mathcal{R}'$ or not. In this encoding, the Hamming distance of $\mathcal{R}$ and $\mathcal{R}'$ is minimized while attaining minimum classification errors on the current batch (using soft clause E_l in Eq. (3.2)) To this end, mini-batch learning starts with an empty CNF formula as $\mathcal{R}'$ without any feature, and thus $S^i_j := \neg B^i_j$ for the first batch. We next analyze the complexity of the MaxSAT encoding for mini-batch learning.

Proposition 5. *Let $n' \triangleq |\mathbf{D}_b| \ll |\mathbf{D}|$ be the size of a mini-batch, $n'_{neg} \leq n'$ be the number of negative samples in the batch, and m be the number of Boolean features. According to Proposition 3, to learn a k-clause CNF classifier in mini-batch learning, the MaxSAT encoding for each batch has $km + n'$ Boolean variables and kn'_{neg} auxiliary variables. Let $n'_{pos} = n' - n'_{neg}$ be the number of positive samples in the batch. Then, according to Proposition 4, the number of clauses in the MaxSAT query for each batch is $km + n' + kn'_{pos} + (km + 1)n'_{neg} \approx \mathcal{O}(kmn')$ when $n'_{pos} = n'_{neg} = \frac{n'}{2}$.*

Assessing the Performance of $\mathcal{R}$. Since we apply a heuristic objective in mini-batch learning, $\mathcal{R}$ may be optimized for the current batch while generalizing poorly on the full training data. To tackle this, after learning on each batch, we decide whether to keep $\mathcal{R}$ or not by assessing the performance of $\mathcal{R}$ on the training data and keep $\mathcal{R}$ if it achieves higher performance. We measure the performance of $\mathcal{R}$ on the full training data $\mathbf{D}$ using a weighted combination of classification errors and rule-size. In particular, we compute a combined loss function $\mathsf{loss}(\mathcal{R}) \triangleq |\mathcal{E}_{\mathbf{D}}| + \lambda|\mathcal{R}|$ on the training data $\mathbf{D}$, which is indeed the value of the objective function that we minimize in the non-incremental learning in Section 3.2. Additionally, we discard the current classifier $\mathcal{R}$ when the loss does not decrease ($\mathsf{loss}(\mathcal{R}) > \mathsf{loss}(\mathcal{R}')$).

We present the algorithm for mini-batch learning in Algorithm 1.

Algorithm 1 MaxSAT-based Mini-batch Learning

```
1: function MiniBatchLearning(D, λ, k)
2:     R =← ⋀_{i=1}^{k} false                          ▷ Empty CNF formula
3:     loss_max ← ∞
4:     for i ← 1, ..., N do                            ▷ N is the total batch-count
5:         D_b ← GetBatch(D)
6:         Q, W(·) ← MaxSATEncoding(R, D_b, λ, k)      ▷ Returns a weighted-partial CNF
7:         σ* ← MaxSAT(Q, W(·))
8:         R_new ← ConstructClassifier(σ*)
9:         loss ← |E_D| + λ|R_new|                     ▷ Compute loss
10:        if loss < loss_max then
11:            loss_max ← loss
12:            R ← R_new
13:    return R
```

3.4.2 Iterative Learning

We now discuss an iterative learning algorithm for rule-based classifiers. The major advantage of iterative learning is that we solve a smaller MaxSAT query because of

learning a *partial* classifier in each iteration. Our iterative approach is motivated by the set-covering algorithm—also known as separate-and-conquer algorithm—in symbolic rule learning [153]. In this approach, the core idea is to define the *coverage of a partial classifier* (for example, a clause in a CNF classifier). For a specific definition of coverage, this algorithm separates samples covered by the partial classifier and recursively conquers remaining samples in the training data by learning another partial classifier until no sample remains. The final classifier is an aggregation of all partial classifiers—the conjunction of clauses in a CNF formula, for example.

Iterative learning is different from mini-batch learning in several aspects. In mini-batch learning, we learn all clauses in a CNF formula together, while in iterative learning, we learn a single clause of a CNF in each iteration. Additionally, in mini-batch learning, we improve scalability by reducing the number of samples in the training data using mini-batches, while in iterative learning, we improve scalability by reducing the number of clauses to learn at once. Therefore, an efficient integration of iterative learning and mini-batch learning would benefit scalability from both worlds. In the following, we discuss this integration by first stating iterative learning for CNF classifiers.

In iterative learning, we learn one clause of a CNF classifier in each iteration, where the clause refers to a partial classifier. The coverage of a clause in a CNF formula is the set of samples that *do not satisfy* the clause. The reason is that if a sample does not satisfy at least one clause in a CNF formula, the prediction of the sample by the full formula is class 0, because CNF is a conjunction of clauses. As a result, considering covered samples in the next iteration does not change their prediction regardless of whatever clause we learn in later iterations. To this end, a single clause learning can be performed efficiently by applying mini-batch learning discussed before. In Algorithm 2, we provide an algorithm for learning a CNF classifier iteratively by leveraging mini-batch learning. This algorithm is a double-loop algorithm, where in the outer loop we apply iterative learning and in the inner loop, we apply mini-batch learning.

Algorithm 2 Iterative CNF Classifier Learning

procedure IterativeCNFLearning($\mathbf{X}, \mathbf{y}, \lambda, k$)
  $\mathcal{R} \leftarrow$ true ▷ Initial formula
  for $i \leftarrow 1, \ldots, k$ and $\mathbf{D} \neq \emptyset$ **do**
    $C_i \leftarrow$ MiniBatchLearning($\mathbf{D}, \lambda, 1$) ▷ Single clause learning, $k = 1$
    $\mathbf{D}' \leftarrow$ Coverage($\mathbf{D}, C_i$)
    if $\mathbf{D}' = \emptyset$ **then** ▷ Terminating conditions
      break
    $\mathcal{R} \leftarrow \mathcal{R} \wedge C_i$
    $\mathbf{D} \leftarrow \mathbf{D} \setminus \mathbf{D}'$ ▷ Removing covered samples
  return $\mathcal{R}$

Terminating conditions. In Algorithm 2, we terminate iterative learning based on three conditions: (i) when $\mathcal{R}$ contains all k clauses , (ii) the training data $\mathbf{D}$ is empty (that is, no sample remains uncovered), and (iii) no new sample is covered by the current partial classifier. Since the first two conditions are trivial, we elaborate on the third condition. When clause C_i cannot cover any new sample from the training dataset $\mathbf{D}$, the next iteration will result in the same clause C_i because the training data remains the same. In this case, we do not include clause C_i to classifier $\mathcal{R}$ because of zero coverage.

3.5 Applying IMLI on Learning Other Interpretable Classifiers.

In earlier sections, we discuss the learning of CNF classifiers using IMLI. IMLI can also be applied to learning other interpretable rule-based representations. In this section, we discuss how IMLI can be applied in learning DNF classifiers, decision lists, and decision sets.

3.5.1 Learning DNF classifiers

For learning DNF classifiers, we leverage De Morgan's law where complementing a CNF formula results in a DNF formula. To learn a DNF classifier, say $\mathcal{R}'(\mathbf{X})$, we can trivially show that $Y = \mathcal{R}'(\mathbf{X}) \leftrightarrow \neg(Y = \neg\mathcal{R}'(\mathbf{X}))$ for the feature vector $\mathbf{X}$. Here $\neg\mathcal{R}'(\mathbf{X})$ is a CNF formula, by definition. Thus, we learn a DNF classifier by first complementing the class-label $y^{(i)}$ to $\neg y^{(i)}$ for each sample in the training dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, learning a CNF classifier on $\{(\mathbf{x}^{(i)}, \neg y^{(i)})\}_{i=1}^{n}$, and finally complementing the learned classifier to DNF. For example, the CNF classifier "(Male $\vee$ Age $<$ 50) $\wedge$ (Education = Graduate $\vee$ Income $\geq$ 1500)" is complemented to a DNF classifier as "(not Male $\wedge$ Age $\geq$ 50) $\vee$ (Education $\neq$ Graduate $\wedge$ Income $\leq$ 1500)".

To learn a DNF classifier incrementally, such as through mini-batch and iterative learning, we adopt the following procedure. For learning a DNF classifier using mini-batch learning, we first learn a CNF classifier on dataset $\{(\mathbf{x}^{(i)}, \neg y^{(i)})\}_{i=1}^{n}$ and complement the classifier to a DNF classifier at the end of mini-batch learning. To learn a DNF classifier in the iterative approach, we learn a single clause of the DNF classifier in each iteration, remove covered samples, and continue till no training sample remains. In this context, the coverage of a clause in a DNF formula is the set of samples satisfying the clause.

3.5.2 Learning Decision Lists

In IMLI, we apply an iterative learning approach for efficiently learning a decision list. A decision list $\mathcal{R}_L$ is a list of pairs $(C_1, V_1), \ldots, (C_k, V_k)$, where we propose to learn one pair in each iteration. We note that the clause C_i is a conjunction of literals—equivalently, a

single clause DNF formula. Hence, our task is to deploy IMLI to efficiently learn a single clause DNF formula C_i. In particular, we opt to learn this clause for the majority class, say V_i, in the training dataset by setting the majority samples as class 1 and all other samples as class 0. As a result, even if the MaxSAT-based learning, presented in this chapter, is targeted for binary classification, we can learn a multi-class decision list in IMLI.

In Algorithm 3, we present the iterative algorithm for learning decision lists. In each iteration, the algorithm learns a pair (C_i, V_i), separates the training set based on the coverage of (C_i, V_i) and conquers the remaining samples recursively. The coverage of (C_i, V_i) is the set of samples that satisfies clause C_i. Finally, we add a default rule $(C_k, V_{\mathsf{default}})$ to $\mathcal{R}_L$ where $C_k \triangleq$ true denoting that the clause is satisfied by all samples. We select the default class V_{default} in the following order: (i) if any class(s) is not in the predicted classes $\{V_i\}_{i=1}^{k-1}$ of the decision list, V_{default} is the majority class among missing classes, and (ii) V_{default} is the majority class of the original training set $\mathbf{D}$, otherwise.

Algorithm 3 Iterative Learning of Decision Lists

procedure DecisionListLearning($\mathbf{D}, \lambda, k$)
  $\mathcal{R}_L \leftarrow \{\}$
  for $i \leftarrow 1, \ldots, k-1$ and $\mathbf{D} \neq \emptyset$ **do**
    $V_i \leftarrow$ MajorityClass($\{y^{(i)}\}_{i=1}^{|\mathbf{D}|}$) ▷ V_i specifies the target class given class label
    $C_i \leftarrow$ MiniBatchDNFLearning($\mathbf{D}, \lambda, 1, V_i$) ▷ Ref. Section 3.5.1
    $\mathbf{D}' \leftarrow$ Coverage($\mathbf{D}, C_i$)
    if $\mathbf{D}' = \emptyset$ **then**
      break
    $\mathcal{R}_L \leftarrow \mathcal{R}_L \cup \{(C_i, V_i)\}$
    $\mathbf{D} \leftarrow \mathbf{D} \setminus \mathbf{D}'$
  $\mathcal{R}_L \leftarrow \mathcal{R}_L \cup \{(\text{true}, V_{\mathsf{default}})\}$ ▷ Default rule
  return $\mathcal{R}_L$

3.5.3 Learning Decision Sets

We now describe an iterative procedure for learning decision sets. A decision set comprises of an individual clause-class pair (C_i, V_i) where C_i denotes a single clause DNF formula similar to decision lists. In a decision set, a sample can satisfy multiple clauses simultaneously, which is attributed as an *overlapping between clauses* [26]. Concretely, the overlap between two clauses C_i and C_j with $i \neq j$ is the set of samples $\{\mathbf{x} | \mathbf{x} \models C_i \wedge \mathbf{x} \models C_j\}$ satisfying both clauses. One additional objective in learning a decision set is to minimize the overlap between clauses, as studied in [26]. Therefore, along with optimizing accuracy and rule-sparsity, we propose an iterative procedure for decision sets that additionally minimizes

the overlap between clauses.

Algorithm 4 Iterative Learning of Decision Sets

procedure DECISIONSETSLEARNING($\mathbf{D}, \lambda, k$)
  $\mathcal{R}_S = \{\}$
  $\mathbf{D}_{cc} = \{\}$ ▷ Contains correctly covered samples
  for $i \leftarrow 1, \ldots, k-1$ and $\mathbf{D} \neq \emptyset$ **do**
    $V_i \leftarrow$ MAJORITYCLASS($\{y^{(i)}\}_{i=1}^{|\mathbf{D}|}$)
    $\mathbf{D}_w = \mathbf{D} \cup \{(\mathbf{x}, \neg V_i) | (\mathbf{x}, y) \in \mathbf{D}_{cc}\}$ ▷ Appending covered samples with complemented class
    $C_i \leftarrow$ MINIBATCHDNFLEARNING($\mathbf{D}_w, \lambda, 1, V_i$)
    $\mathbf{D}' \leftarrow$ CORRECTCOVERAGE($\mathbf{D}, C_i, V_i$)
    if ($\mathbf{D}' = \emptyset$ **then**
      break
    $\mathcal{R}_S \leftarrow \mathcal{R}_S \cup \{(C_i, V_i)\}$
    $\mathbf{D} \leftarrow \mathbf{D} \setminus \mathbf{D}'$
    $\mathbf{D}_{cc} = \mathbf{D}_{cc} \cup \mathbf{D}'$
  $\mathcal{R}_S \leftarrow \mathcal{R}_S \cup \{(\text{true}, V_{\mathsf{default}})\}$
  return $\mathcal{R}_S$

In Algorithm 4, we present an iterative algorithm for learning a decision set. The iterative algorithm is a modification of separate-and-conquer algorithm by additionally focusing on minimizing overlaps in a decision set. Given a training data $\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, a regularization parameter λ, and the number of clauses k, the core idea of Algorithm 4 is to learn a pair (C_i, V_i) in each iteration, separate covered samples from $\mathbf{D}$, and conquer remaining samples recursively. In contrast to learning decision lists, we have following modifications in Algorithm 4.

- The first modification is with respect to the definition of coverage for decision sets. Unlike decision lists, we separate samples that are *correctly covered* by (C_i, V_i) in each iteration. Given a dataset $\mathbf{D} = \{(\mathbf{x}^{(l)}, y^{(l)})\}$, the correctly covered samples of (C_i, V_i) is a dataset $\mathbf{D}_{cc} = \{(\mathbf{x}^{(l)}, y^{(l)}) | \mathbf{x}^{(l)} \models C_i \wedge y^{(l)} = V_i\}_{l=1}^{|\mathbf{D}|} \subseteq \mathbf{D}$ of samples that satisfy C_i and have matching class-label as V_i.

- The second modification is related to the training dataset considered in each iteration. Let $\mathbf{D}$ denote *the remaining training dataset* in the current iteration and V_i be the majority class in $\mathbf{D}$. Hence, V_i is the target class in the current iteration. Also, let $\mathbf{D}_{cc}$ denotes the set of samples *correctly covered in all previous iterations*. In Algorithm 4, we learn the current rule C_i on the working dataset $\mathbf{D}_w = \mathbf{D} \cup \{(\mathbf{x}, \neg V_i) | (\mathbf{x}, y) \in \mathbf{D}_{cc}\}$. Thus, $\mathbf{D}_w$ is constructed by joining the remaining training samples with correctly covered samples in $\mathbf{D}_{cc}$ with complemented class label to the target class V_i. Thus, by explicitly labeling covered samples as class $\neg V_i$, the new clause C_i learns to falsify

already covered samples. This heuristic allows us to minimize the overlap of C_i compared to previously learned clauses $\{C_j\}_{j=1}^{i-1}$.

Finally, the default clause for decision sets is learned similarly as in decision lists.

3.6 Empirical Performance Analysis

In this section, we empirically evaluate the performance of IMLI. We first present the experimental setup and the objective of the experiments, followed by experimental results.

3.6.1 Experimental Setup

We implement a prototype of IMLI in Python to evaluate the performance of the proposed MaxSAT-based formulation for learning classification rules. To implement IMLI, we deploy a state-of-the-art MaxSAT solver Open-WBO [154], which returns the current best solution upon reaching a timeout.

We compare IMLI with state-of-the-art interpretable and non-interpretable classifiers. Among interpretable classifiers, we compare with RIPPER [113], BRL [27], CORELS [116], and BRS [115]. Among non-interpretable classifiers, we compare with Random Forest (RF), Support Vector Machine with linear kernels (SVM), Logistic Regression classifier (LR), and k-Nearest Neighbors classifier (kNN). We deploy the Scikit-learn library in Python for implementing non-interpretable classifiers.

We experiment with real-world binary classification datasets from UCI [155], OpenML [156], and Kaggle repository (https://www.kaggle.com/datasets), as listed in Table 3.1. In these datasets, the number of samples vary from about 200 to $1,000,000$. The datasets contain both real-valued and categorical features. We process them to binary features by setting the maximum number of bins as 10 during discretization. For non-interpretable classifiers such as RF, SVM, LR, and kNN that take real-valued features as inputs, we only convert categorical features to one-hot encoded binary features.

We perform ten-fold cross-validation on each dataset and evaluate the performance of different classifiers based on the median prediction accuracy on the test data. Additionally, we compare the median size of generated rules among rule-based interpretable classifiers. We consider a comparable combination (100) of hyper-parameters choices for all classifiers, that we fine-tune during cross-validation. For IMLI, we vary the number of clauses $k \in \{1, 2, \ldots, 5\}$ and the regularization parameter λ in a logarithmic grid by choosing 5 values between 10^{-4} and 10^{1}. For mini-batch learning in IMLI, we set the number of samples in each mini-batch, $n' \in \{50, 100, 200, 400\}$. Thus, we consider $\lceil n/n' \rceil$ mini-batches, where n denotes the size of training data. To construct mini-batches from a training dataset, we sequentially split the data into $\lceil n/n' \rceil$ batches with each batch having n' samples. Furthermore, to ignore the effect of batch-ordering, we perform mini-batch learning in two rounds such that each batch participates twice in the training.

For BRL algorithm, we vary four hyper-parameters: the maximum cardinality of rules in $\{2, 3, 4\}$, the minimum support of rules in $\{0.05, 0.175, 0.3\}$, and the prior on the expected length and width of rules in $\{2, 4, 6, 8\}$ and $\{2, 5, 8\}$, respectively. For CORELS algorithm, we vary three hyper-parameters: the maximum cardinality of rules in $\{2, , 5\}$, the minimum support of rules in $\{0.01, 0.17, 0.33, 0.5\}$, and the regularization parameter in $\{0.005, 0.01, 0.015, 0.02, 0.025, 0.03\}$. For BRS algorithm, we vary three hyper-parameters: the number of initial rules in $\{500, 1000, 1500, 2000, 2500, 3000\}$, the maximum length of rules in $\{1, 2, 3, 4\}$, and the minimum support of rules in $\{1, 4, 7, 10\}$. For RF and RIPPER classifiers, we vary the cut-off on the number of samples in the leaf node using a linear grid between 3 to 500 and 1 to 300, respectively. For SVM and LR classifiers, we discretize the regularization parameter on a logarithmic grid between 10^{-3} and 10^{3}. For kNN, we vary the number of neighbors in a linear grid between 1 and 500. We conduct each experiment on an Intel Xeon $E7 - 8857$ v2 CPU using a single core with 16 GB of RAM running on a 64bit Linux distribution based on Debian. For all classifiers, we set the training timeout to 1000 seconds.

Objectives of Experiments. In the following, we present the objectives of our experimental study.

1. How are the accuracy and size of classification rules generated by IMLI compared to existing interpretable classifiers?

2. How is the scalability of IMLI in solving large-scale classification problems compared to existing interpretable classifiers?

3. How does IMLI perform in terms of accuracy and scalability compared to classifiers that are non-interpretable?

4. How does the incremental learning in IMLI perform compared to non-incremental MaxSAT-based learning in terms of accuracy, rule-sparsity, and scalability?

5. How do different interpretable classification rules learned using IMLI perform in terms of accuracy and rule-size?

6. What are the effects of different hyper-parameters in IMLI?

Summary of Experimental Results. To summarize our experimental results, IMLI achieves the best balance among prediction accuracy, interpretability, and scalability compared to existing interpretable rule-based classifiers. Particularly, compared to the most accurate classifier RIPPER, IMLI demonstrates on average 1% lower prediction accuracy, wherein the accuracy of IMLI is higher than BRL, CORELS, and BRS in almost all datasets. In contrast, IMLI generates significantly smaller rules than RIPPER, specifically in large datasets. Moreover, BRL, CORELS, and BRS report comparatively

smaller rule-size than IMLI on average, but with a significant decrease in accuracy. In terms of scalability, IMLI achieves the best performance compared to other interpretable and non-interpretable classifiers by classifying datasets with one million samples. While CORELS also scales to such large datasets, its accuracy is lower than that of IMLI by at least 2% on average. Therefore, IMLI is not only scalable but also accurate in practical classification problems while also being interpretable by design. We additionally analyze the comparative performance of different formulations presented in this chapter, where the incremental approach empirically proves its efficiency than the naïve MaxSAT formulation. Furthermore, we learn and compare the performance of different interpretable representations: decision lists, decision sets, CNF, and DNF formulas using IMLI and present the efficacy of IMLI in learning varied interpretable classifiers. Finally, we study the effect of different hyper-parameters in IMLI, where each hyper-parameter provides a precise control among training time, prediction accuracy, and rule-sparsity. In the following, we discuss our experimental results in detail.

3.6.2 Experimental Results

Comparing IMLI with interpretable classifiers. We compare IMLI with existing interpretable classifiers in three aspects: test accuracy, rule-size, and scalability.

Test accuracy and rule-size. We present the experimental results of test accuracy and rule-size among interpretable classifiers in Table 3.1, where the first, second, and third columns represent the name of the dataset, the number of samples, and the number of features in the dataset, respectively. In each cell from the fourth to the eighth column in the table, the top value represents the median test accuracy and the bottom value represents the median size of rules measured through ten-fold cross-validation.

In Table 3.1, IMLI and CORELS generate interpretable classification rules in all 15 datasets in our experiments. In contrast, within a timeout of 1000 seconds, RIPPER, BRL, and BRS fail to generate any classification rule in three datasets, specifically in large datasets ($\geq 200,000$ samples).

We now compare IMLI with each interpretable classifier in detail. Compared to RIPPER, IMLI has lower accuracy in 9 out of 12 datasets. More specifically, the accuracy of IMLI is 1% lower on average than RIPPER. The improved accuracy of RIPPER, however, results in the generation of higher size classification rules than IMLI in most datasets. In particular, IMLI generates sparser rules than RIPPER in 9 out of 12 datasets, wherein RIPPER times out in 3 datasets. Interestingly, the difference in rule-size is more significant in larger datasets, such as in 'Vote' dataset, where RIPPER learns a classifier with 132 Boolean literals compared to 15 Boolean literals by IMLI. Therefore, IMLI is better than RIPPER in terms of rule-sparsity, but lags slightly in accuracy.

IMLI performs better than BRL both in terms of accuracy and rule-sparsity. In particular, IMLI has higher accuracy and lower rule-size than BRL in 12 and 8 datasets, respectively, in a total of 12 datasets, wherein BRL times out in 3 datasets. While

Table 3.1: Comparison of accuracy and rule-size among interpretable classifiers. Each cell from the fourth to the eighth column contains test accuracy (top) and rule-size (bottom). '—' represents a timeout. Numbers in bold represent the best performing results among different classifiers.

Dataset	Size	Features	RIPPER	BRL	CORELS	BRS	IMLI
Parkinsons	195	202	94.44 7.0	**94.74** 11.5	89.74 **2.0**	84.61 5.0	**94.74** 7.5
WDBC	569	278	**98.08** 13.0	93.81 22.0	92.04 **2.0**	92.98 7.0	94.74 11.5
Pima	768	83	77.14 6.0	68.18 13.5	75.32 **2.0**	75.32 3.0	**78.43** 23.0
Titanic	1,043	38	78.72 6.0	62.98 15.0	**81.9** **4.0**	80.86 **4.0**	81.82 5.5
MAGIC	19,020	100	**82.68** 102.0	76.95 81.0	78.05 4.0	77.5 **3.0**	78.26 8.5
Tom's HW	28,179	946	**85.91** 30.0	— —	83.27 **4.0**	83.13 18.5	85.24 44.5
Credit	30,000	199	**82.39** 32.5	46.12 26.5	81.18 **2.0**	80.45 7.0	82.12 17.5
Adult	32,561	94	**84.37** 115.5	72.08 46.5	79.78 **4.0**	70.75 **4.0**	81.2 30.0
Bank Marketing	45,211	82	**90.01** 36.5	84.66 13.0	89.62 **2.0**	86.75 **2.0**	89.84 24.5
Connect-4	67,557	126	**76.72** 118.0	65.83 18.5	68.68 **4.0**	70.49 11.0	75.36 50.5
Weather AUS	107,696	169	**84.22** 26.0	43.26 22.0	83.67 **2.0**	— —	83.78 22.0
Vote	131,072	16	**97.12** 132.0	94.78 41.5	95.86 3.5	95.14 **1.0**	96.69 15.0
Skin Seg	245,057	30	— —	79.25 6.0	91.62 9.0	68.48 **5.0**	**94.71** 30.0
BNG(labor)	1,000,000	89	— —	— —	88.56 **2.0**	— —	**90.91** 24.0
BNG(credit-g)	1,000,000	97	— —	— —	72.08 **2.0**	— —	**75.48** 27.5

comparing with CORELS, IMLI achieves higher accuracy in almost all datasets (14 out of 15 datasets). A similar trend is observed in comparison with BRS, where IMLI achieves higher accuracy in all of 12 datasets and BRS times out in 3 datasets. CORELS and BRS, however, generates sparser rules than IMLI in most datasets, but by costing a significant decrease in accuracy. For example, in the largest dataset 'BNG(credit-g)' with 1 Million samples, BRS times out and CORELS generates a classifier with 72.08% accuracy with rule-size 2. IMLI, in contrast, learns a classifier with 27.5 Boolean literals achieving 75.48% accuracy, which is 3% higher than CORELS. Therefore, IMLI makes a good balance between accuracy and rule-size compared to existing interpretable classifiers while also being highly scalable. In the following, we discuss the results on the scalability of all interpretable classifiers in detail.

Scalability. We analyze the scalability among interpretable classifiers by comparing their training time. In Figure 3.1, we use cactus plots[6] to represent the training time (in seconds) of all classifiers in 1000 instances (10 folds $\times$ 100 choices of hyper-parameters) derived for each dataset. In the cactus plot, the number of solved instances (within 1000 seconds) is on the X-axis, whereas the training time is on the Y-axis. A point (x, y) on the plot implies that a classifier yields lower than or equal to y seconds of training in x many instances.

In Figure 3.1, we present results in an increasing number of samples in a dataset (from left to right). In WDBC and Adult datasets presented on the first two plots in Figure 3.1, CORELS solves lower than 600 instances within a timeout of 1000 seconds. The scalability performance of BRS is even worse, where it solves around 700 and 200 instances in WDBC and Adult datasets, respectively. The other three classifiers: IMLI, BRL, and RIPPER solve all 1000 instances, where BRL takes comparatively higher training time than the other two. The performance of IMLI and RIPPER is similar, with RIPPER being comparatively better in the two datasets. However, the efficiency of IMLI compared to other classifiers becomes significant as the number of samples in a dataset increases. In particular, in 'Weather AUS' dataset, BRS cannot solve a single instance, BRL and CORELS solves 400 instances, and RIPPER solves around 600 instances. IMLI, however, solves all 1000 instances in this dataset. Similarly, in 'BNG(labor)' dataset, all other classifiers except IMLI and CORELS cannot solve any instance. While IMLI mostly takes the maximum allowable time (1000 seconds) in solving all instances in this dataset, CORELS can solve lower than 400 instances. The improved performance of IMLI is due to incremental learning based on the novel integration of iterative learning and mini-batch learning. Contrary to our incremental learning, earlier approaches are based on heuristic rule-pruning (e.g., RIPPER) and rule-mining followed by Bayesian optimization (e.g., BRL, BRS) and branch and bounds algorithms (e.g., CORELS). Thus, owing to incremental learning, IMLI establishes itself as the most scalable classifier compared to other state-of-the-art interpretable classifiers.

[6]Cactus plots are often used in (Max)SAT community to present the scalability of different solvers/methods [157, 158].

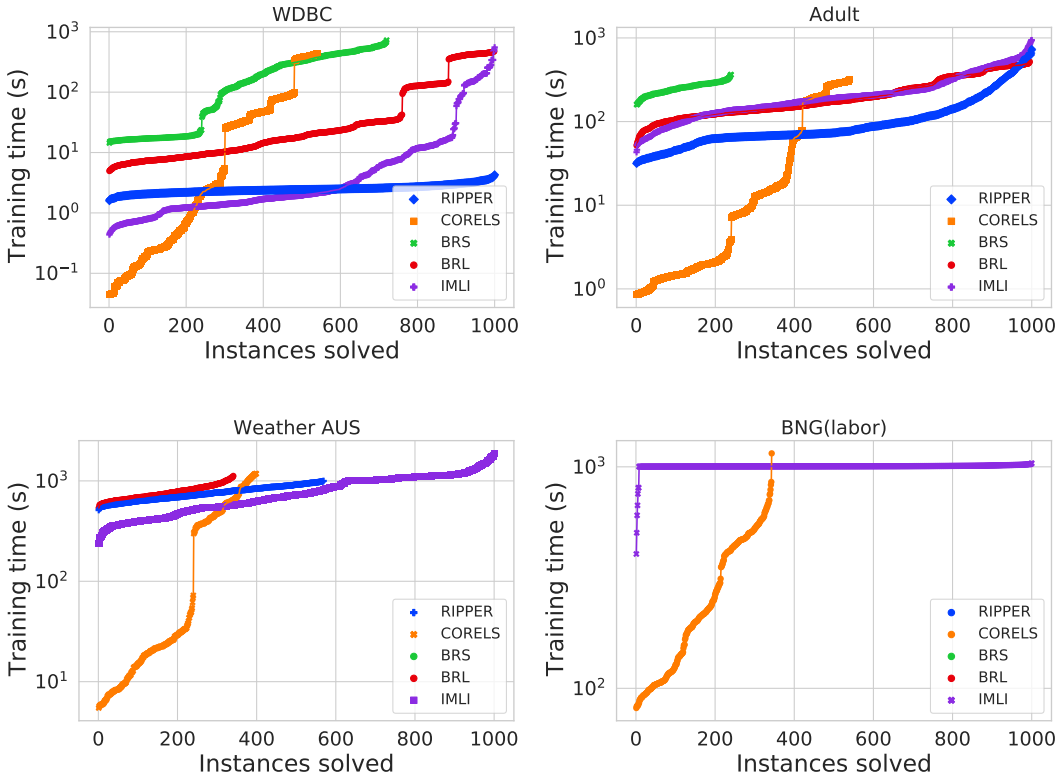


Figure 3.1: Comparison of scalability among interpretable classifiers. The plots are arranged in increasing sizes of datasets (from left to right). In each cactus plot, IMLI solves all 1000 instances for each dataset, while competitive classifiers often fail to scale, specially in larger datasets.

Comparison with non-interpretable classifiers. We compare IMLI with state of the art non-interpretable classifiers such as LR, SVM, kNN, and RF in terms of their median test accuracy in Table 3.2. In the majority of the datasets, IMLI achieves comparatively lower test accuracy than the best performing non-interpretable classifier. The decrease in the test accuracy of IMLI is attributed to two factors. Firstly, while we train IMLI on discretized data, non-interpretable classifiers are trained on non-discretized data and thus IMLI incurs additional classification errors due to discretization. Secondly, IMLI learns a rule-based classifier, whereas non-interpretable classifiers can learn more flexible decision boundaries and thus fit data well. In Table 3.2, we also observe that IMLI achieves impressive scalability than competing classifiers by solving datasets with $1,000,000$ samples where most of the non-interpretable classifiers fail to learn any decision boundary on such large datasets. Thus, IMLI, being an interpretable classifier, demonstrates lower accuracy than competing non-interpretable classifiers, but higher scalability in practice.

Table 3.2: Comparison of IMLI with non-interpretable classifiers in terms of test accuracy. In the table, '—' represents a timeout. Numbers in bold represent the best performing results among different classifiers.

Dataset	Size	LR	SVM	kNN	RF	IMLI
Parkinsons	195	89.74	89.74	**97.5**	90.0	94.74
WDBC	569	**98.25**	**98.25**	98.23	96.49	94.74
Pima	768	78.43	79.08	74.5	**79.22**	78.43
Titanic	1,043	80.86	80.38	81.34	**82.69**	81.82
MAGIC	19,020	79.18	79.34	84.6	**88.2**	78.26
Tom's HW	28,179	96.2	97.13	88.15	**97.78**	85.24
Credit	30,000	**82.2**	81.9	81.83	82.15	82.12
Adult	32,561	85.26	85.05	83.8	**86.69**	81.2
Bank Marketing	45,211	90.09	89.28	89.43	**90.27**	89.84
Connect-4	67,557	79.39	—	85.51	**88.11**	75.36
Weather AUS	107,696	85.64	—	78.59	**86.26**	83.78
Vote	131,072	96.43	96.37	97.05	**97.38**	96.69
Skin Seg	245,057	91.86	—	**99.96**	**99.96**	94.71
BNG(labor)	1,000,000	—	—	—	—	**90.91**
BNG(credit-g)	1,000,000	—	—	—	**80.58**	75.48

Comparison among different formulations in IMLI. We compare the performance of different formulations for learning classification rules as presented in this chapter. In Figure 3.2, we show cactus plots for assessing training time (in seconds), test error (in percentage), and rule-size among different formulations. In the cactus plot, a point (x, y) denotes that the formulation yields lower than or equal to y training time (similarly, test error and rule-size) in x many instances in each dataset.

In Figure 3.2, we denote the baseline non-incremental MaxSAT-based formulation as IMLI-B[7], incremental MaxSAT-based formulation with only mini-batch learning as IMLI-M, and incremental MaxSAT-based formulation with both mini-batch and iterative learning as IMLI. We first observe the training time of different formulations in the left-most column in Figure 3.2, where IMLI-B soon times out and solves lower than 300 instances out of 1000 instances in each dataset. This result suggests that the non-incremental formulation cannot scale in practical classification task. Comparing between IMLI and IMLI-M, both formulations solve all 1000 instances in each dataset with IMLI-M undertaking significantly higher training time than IMLI. Therefore, IMLI achieves better scalability than IMLI-M indicating that an integration of mini-batch and iterative learning achieves a significant progress in terms of scalability than mini-batch learning alone.

[7]Since IMLI-B is a non-incremental formulation and does not involve any mini-batch learning, we consider two hyper-parameters for IMLI-B: the number of clauses in $\{1, 2, \ldots, 10\}$ and the regularization parameter λ as 10 values chosen from a logarithmic grid between 10^{-4} and 10^{1}.

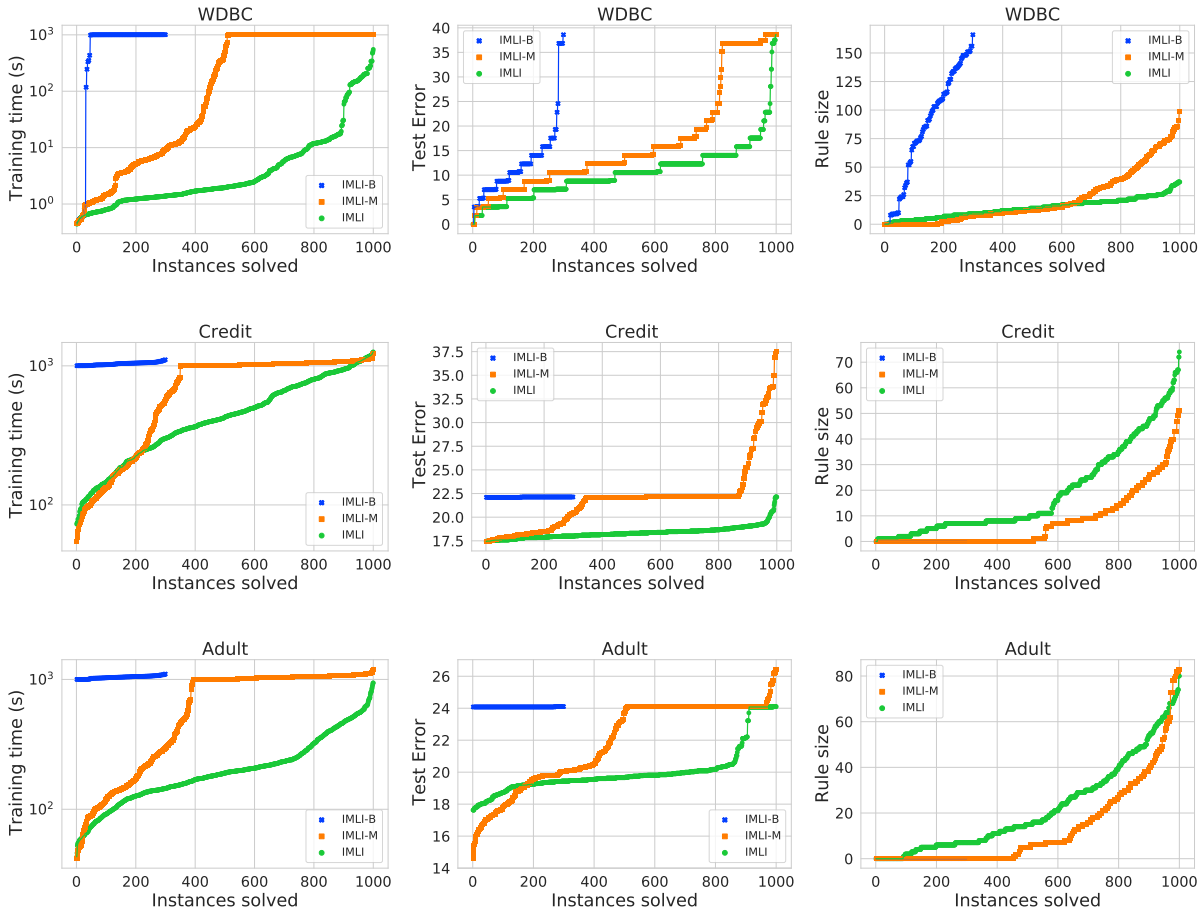


Figure 3.2: Comparison of training time, test error, and rule-size among different formulations presented in the chapter. In each cactus plot, the incremental formulation IMLI with both mini-batch and iterative learning demonstrates the best performance in training time and test error than compared two formulations: non-incremental MaxSAT formulation IMLI-B and incremental formulation with only mini-batch learning IMLI-M. In terms of rule-size, IMLI often generates higher size rules than IMLI-M.

We next focus on the test error of different formulations in the middle column in Figure 3.2. Firstly, IMLI-B has a higher test error than the other two formulations since IMLI-B times out in most instances and learns a sub-optimal classification rule with reduced prediction accuracy. In contrast, IMLI has the lowest test error compared to two formulations in all datasets. This result indicates the effectiveness of integrating both iterative and mini-batch learning with MaxSAT-based formulation in generating more accurate classification rules.

Moving focus on the rule-size in the rightmost column in Figure 3.2, IMLI-B achieves the highest rule-size in WDBC dataset. In contrast, the rule-size of IMLI-B is lowest (zero) in Credit and Adult datasets. In the last two datasets, IMLI-B times out during training

Table 3.3: Comparison of test accuracy (top value) and rule-size (bottom value) among different rule-based representations learned using IMLI. Numbers in bold denote the best performing results among different representations.

Dataset	CNF	DNF	Decision Sets	Decision Lists
Parkinsons	94.74	89.47	**94.87**	89.74
	7.5	**6.0**	15.0	6.5
WDBC	94.74	**96.49**	95.61	95.61
	11.5	15.0	15.5	**10.0**
Pima	**78.43**	77.13	76.97	76.97
	23.0	**9.0**	15.0	13.5
Titanic	81.82	82.29	81.82	**82.3**
	5.5	10.5	8.5	8.0
MAGIC	**78.26**	77.44	75.87	77.79
	8.5	41.5	10.0	14.0
Tom's HW	85.24	85.15	85.72	**85.95**
	44.5	**26.5**	45.0	59.5
Credit	82.12	82.15	82.03	**82.22**
	17.5	14.0	**9.5**	21.5
Adult	81.2	**84.28**	80.07	80.96
	30.0	34.5	**7.0**	24.5
Bank Marketing	**89.84**	89.77	89.67	89.79
	24.5	7.5	**6.0**	10.5
Connect-4	**75.36**	70.63	68.09	69.83
	50.5	42.0	**4.5**	24.0
Weather AUS	83.78	**84.23**	83.69	83.85
	22.0	14.0	**4.0**	26.0
Skin Seg	**94.71**	93.68	87.92	91.17
	30.0	15.0	**3.0**	7.0

and returns the default rule "true" by predicting all samples as class 1. The other two formulations IMLI and IMLI-M demonstrate a similar trend in rule-size in all datasets with IMLI-M generating comparatively smaller size rules in Credit and Adult datasets. In this context, the improvement of rule-sparsity of IMLI-M is due to a comparatively higher test error (or lower accuracy) than IMLI as observed in all three datasets. Therefore, IMLI appears to be the best performing formulation w.r.t. training time, test error, and rule-size by balancing between accuracy and rule-size while being more scalable.

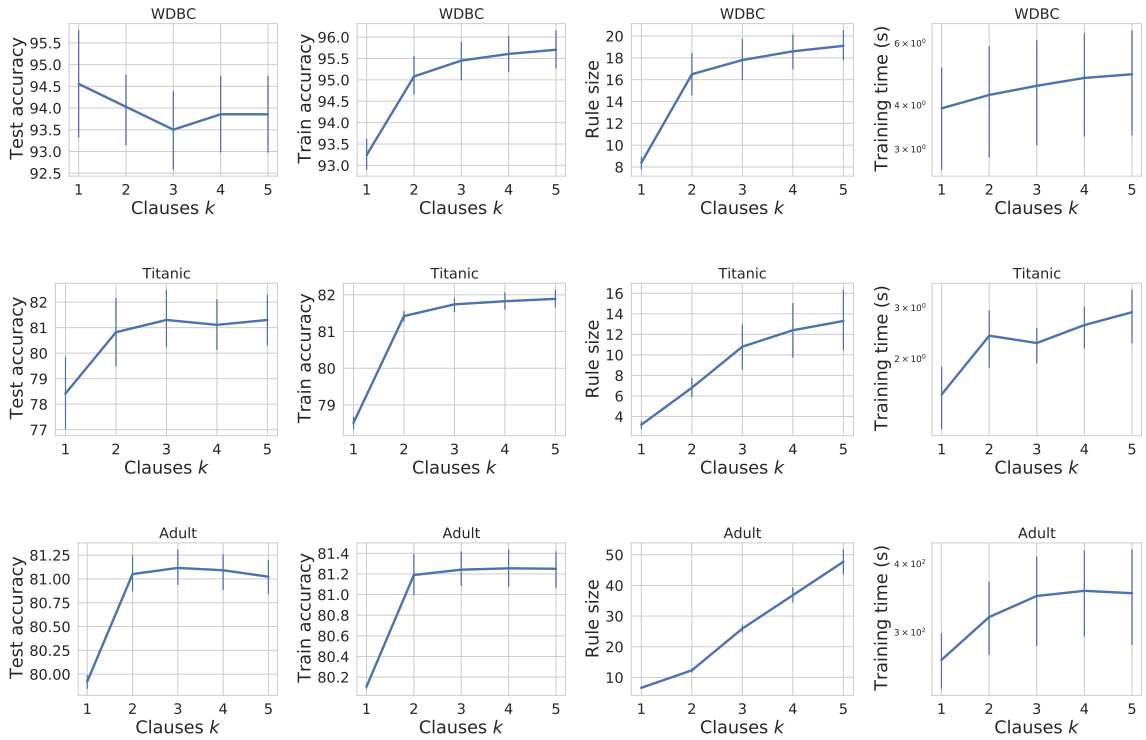


Figure 3.3: Effect of the number of clauses k on accuracy (test and train), rule-size, and training time. As k increases, both train and test accuracy of IMLI increase while generating rules with higher size by incurring higher training time.

Performance evaluation of different interpretable representations in IMLI. We deploy IMLI to learn different interpretable rule-based representations: CNF and DNF classifiers, decision lists, and decision sets and present their comparative performance w.r.t. test accuracy and rule-size in Table 3.3. In each cell in this table, the top value represents the test accuracy and the bottom value represents the size of generated rules.

We learn all four interpretable representations on twelve datasets, where the CNF classifier appears to be the most accurate representation by achieving the highest accuracy in five datasets. In contrast, both DNF and decision lists achieve the highest accuracy in three datasets each; decision sets demonstrate the least performance in test accuracy by being more accurate in one dataset. To this end, the poor accuracy of decision sets is traded off by its rule-size as decision sets generate the sparsest rules compared to other representations. More precisely, decision sets have the smallest rule-size in six datasets, while CNF, DNF, and decision lists have the smallest rule-size in two, three, and one dataset, respectively. These results suggest that CNF classifiers are more favored in applications where higher accuracy is preferred, while decision sets are preferred in applications where higher interpretability is desired. In both cases, one could deploy IMLI for learning varied representations of classification rules.

Ablation study. We experiment the effect of different hyper-parameters in IMLI on prediction accuracy, rule-size, and training time in different datasets. In the following, we discuss the impact of the number of clauses, regularization parameter, and size of mini-batches in IMLI.

Effect of the number of clauses k. In Figure 3.3, we vary k while learning CNF classifiers in IMLI. As k increases, both training and test accuracy generally increase in different datasets (plots in the first and second columns). Similarly, the size of rules increases with k by incurring higher training time (plots in the third and fourth columns). The reason is that a higher value of k allows more flexibility in fitting the data well by incurring more training time and generating higher size classification rules. Therefore, the number of clauses in IMLI provides control on training-time vs accuracy and also on accuracy vs rule-sparsity.

Effect of regularizer λ. In Figure 3.4, we vary λ in a logarithmic grid between 10^{-4} and 10^{1}. As stated in Eq. (3.5), a higher value of λ puts more priority on the minimal changes in rules between consecutive mini-batches in incremental learning while allowing higher mini-batch errors. Thus, in the first and second columns in Figure 3.4, as λ increases, both training and test accuracy gradually decrease. In addition, the size of rules (plots in the third column) also decreases. Finally, we observe that the training time generally decreases with λ. This observation indicates that higher λ puts lower computational load to the MaxSAT solver as a fraction of training examples is allowed to be misclassified. Thus, similar to the number of clauses, regularization parameter λ in IMLI allows to trade-off between accuracy and rule-size in a precise manner.

Effect of the size of mini-batch. In Figure 3.5, we present the effect of mini-batch size in IMLI. As we consider more samples in a batch, both test and training accuracy increase in general as presented in the first and second columns in Figure 3.5. Similarly, the size of generated rules also increases with the number of samples. Due to solving higher size MaxSAT queries, the training time also increases in general with an increase in mini-batch size. Therefore, by varying the size of mini-batches, IMLI allows controlling on training time vs the prediction accuracy (and rule-size) of generated rules.

3.7 Chapter Summary

Interpretable machine learning is gaining more focus with applications in many safety-critical domains. Considering the growing demand for interpretable models, it is challenging to design learning frameworks that satisfy all aspects: being accurate, interpretable, and scalable in practical classification tasks. In this chapter, we have proposed a MaxSAT-based framework IMLI for learning interpretable rule-based classifiers expressible in CNF formulas. IMLI is built on efficient integration of incremental learning, specifically mini-batch and iterative learning, with MaxSAT-based formulation. In our empirical evaluation, IMLI achieves the best balance among prediction accuracy, interpretability, and scalability. In

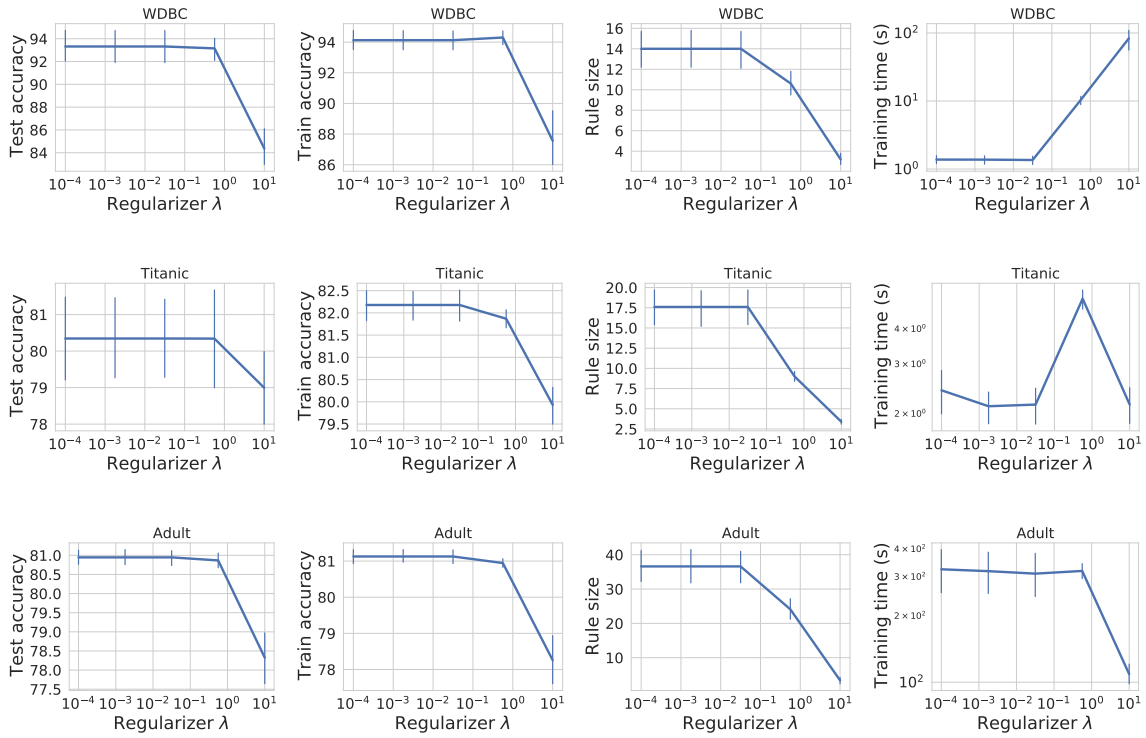


Figure 3.4: Effect of regularization λ on accuracy (test and train), rule-size, and training time. As λ increases, lower priority is given to accuracy. As a result, both training and test accuracy decrease with λ by generating smaller rules.

particular, IMLI demonstrates competitive prediction accuracy and rule-size compared to existing interpretable rule-based classifiers. In addition, IMLI achieves impressive scalability than both interpretable and non-interpretable classifiers by learning interpretable rules on million-size datasets with higher accuracy. Finally, IMLI generates other popular interpretable classifiers such as decision lists and decision sets using the same framework.

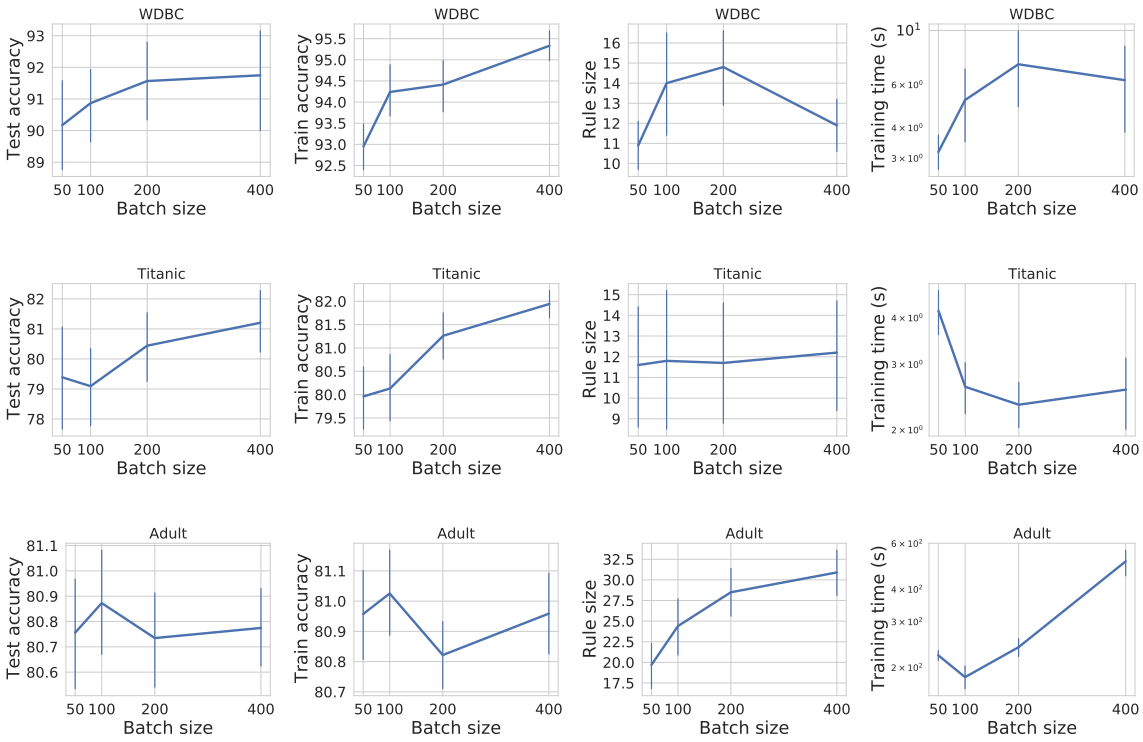


Figure 3.5: Effect of bath size on accuracy (test and train), rule-size, and training time. As we consider more samples in a mini-batch, IMLI generates more accurate and larger size classification rules.

Chapter 4

Classification Rules in Relaxed Logical Form

The widespread adoption of prediction systems in various safety-critical domains such as medical diagnosis, law, education, and many others has led to the increased importance of presenting the decision functions in interpretable representations [159–163]. To enable safe, robust, and trustworthy integration of such systems, the end users require them to support interpretability, privacy, and fairness in decision-making [164–167]. In this context, rule-based representations are considered interpretable in presenting the decision functions to users [168–170]. A recent body of work has studied sparsity-inducing objectives for classification rules in CNF or DNF and demonstrated that they often achieve high interpretability—defined in terms of rule-sparsity—with a minimal sacrifice in classification accuracy [42, 168, 171]. Although CNF/DNF rules are considered interpretable, they are less expressive compared to Boolean cardinality constraints in propositional logic. A Boolean cardinality constraint allows one to express numerical bounds on Boolean variables [36]. In this work, we introduce a novel formulation of interpretable classification rules, namely *relaxed-CNF*, which achieve benefits of both worlds: it is interpretable similar to CNF/DNF but more expressible by allowing cardinality constraints in the representation.

We find the motivation of relaxed-CNF rules from checklists. A checklist is a list of conditions that one needs to check, e.g., a list of items to take on a travel trip. Checklists have several applications in interpretable decision making [32, 172], particularly in the medical domain. For example, the CHADS2 score in medicine is a clinical prediction rule for estimating the risk of stroke [32]. Another example of checklists is a psychometric test, known as Myers–Briggs Type Indicator (MBTI) [172], which indicates differing psychological preferences in how people perceive the world around them and make decisions. An influential study on the importance of checklists [173] finds that highly complex and specialized problems can be handled smoothly by the development and consistent usage of checklists, which we formally call as relaxed-CNF rules.

We now provide an introduction to relaxed-CNF rules. In interpretable rule-based

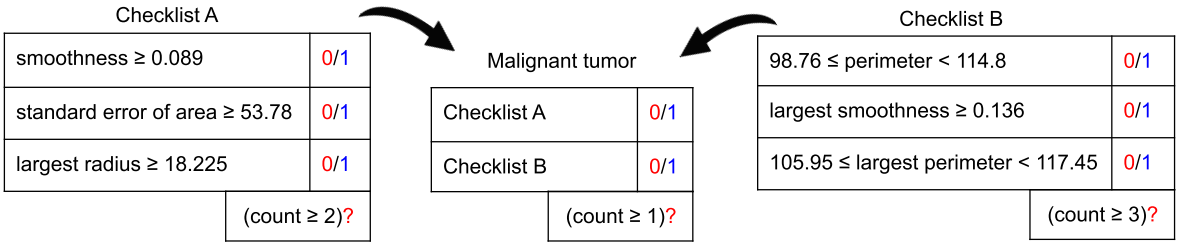


Figure 4.1: An illustrative example of a relaxed CNF classification rule, which describes the decision function in the form of a two-level checklist. This classifier is learned on the WDBC (Wisconsin diagnostic breast cancer) dataset and it predicts whether a tumor cell is malignant or not based on the characteristics of the tumor cell. The first column in each checklist contains Boolean literals. An entry in the second column is 1 if the corresponding literal is true by an observed tumor cell, and 0 otherwise. In the first level, checklist A (resp. B) is true if the count of true literals is at least 2 (resp. 3). In the second level, a tumor cell is predicted as malignant if the count of true checklists is at least 1.

classification, the simplest logical rules are single level-rules: ORs or ANDs of Boolean *literals*, where each literal denotes either a Boolean input feature or its negation. A *clause* is a collection of N literals connected by OR/AND. To satisfy a clause, an OR operator requires 1 out of N literals to be assigned to 1 in the clause, while an AND operator requires all N out of N literals to be assigned to 1. A CNF formula is a conjunction (AND) of clauses where each clause is a disjunction (OR) of literals, and a DNF formula is a disjunction of clauses where each clause is a conjunction of literals. Therefore, CNF and DNF formulas can be viewed as two-level rules with several applications in interpretable decision-making. For example, a decision set is a DNF rule referring to a set of "if-else" conditions [26, 121]. In this chapter, we consider a richer set of logical formulas that capture the structure of checklists. To this end, we consider *hard*-OR clauses, where at least $M > 1$ out of N literals are assigned to 1, and we similarly define *soft*-AND clauses which allow some of the literals (at most $N - M$) to be 0. To be precise, the definitions of hard-OR and soft-AND overlap. Consequently, we use hard-OR when $M \leq N/2$ and soft-AND otherwise. To extend the standard definition of CNF (which is ANDs of ORs), we define *relaxed*-CNF to denote soft-ANDs of hard-ORs. Similarly, *relaxed*-DNF is hard-ORs of soft-ANDs. Since hard-OR and soft-AND are differentiated based on the value of M and N, relaxed-CNF and relaxed-DNF have the same structural representation. In early work, Craven and Shavlik [174] considered single level M-of-N rules to explain black-box neural-network classifiers. Recently, Emad et al. [175] have developed a semi-quantitative group testing approach for learning sparse single level M-of-N rules, which are quite restrictive in their ability to fit the data. In contrast, in this work, we study a much richer family of two-level relaxed-CNF rules.

Relaxed-CNF rules are more flexible than pure CNF rules, and they can accurately

fit more complex classification boundaries. For example, relaxed-CNF clauses allow a compact encoding of the majority function[1] in Boolean logic, which would require exponentially many clauses in CNF, showing the exponential gap in the succinctness of the two representations. In addition, relaxed-CNF and CNF rules have the same functional form where a user has to compute the sum of true literals/clauses and then compare the sum to different thresholds (as in the example in Figure 4.1). From the computational perspective, the structural flexibility of relaxed-CNF compared to CNF/DNF makes it harder to learn. Therefore, in this chapter, we ask the following research question: *can we design a combinatorial framework to efficiently learn relaxed-CNF rules?*

The primary contribution of this chapter is an affirmative answer to the above question by proposing an efficient combinatorial learning framework for relaxed-CNF rules, namely CRR (**C**lassification **R**ules in **R**elaxed form). In CRR, we construct a learning objective to maximize both the prediction accuracy and the rule-sparsity of the generated classification rule. To this end, we design a Mixed-Integer Linear Programming (MILP) formulation for learning the optimal relaxed-CNF rule from data. To learn a k-clause relaxed-CNF rule (say $\mathcal{R}$) using the naïve MILP formulation, the size of the MILP query expressed as the number of constraints is $\mathcal{O}(nk)$, where n is the number of training samples in the dataset. Consequently, this formulation fails to handle large datasets. To address the scalability of CRR, we propose an efficient mini-batch training methodology, where we incrementally learn $\mathcal{R}$ from data by iteratively solving smaller MILP queries corresponding to batches.

Through a comprehensive experimental evaluation over datasets from the UCI and Kaggle repository, we observe that CRR with relaxed-CNF rules achieves an improved trade-off between accuracy and rule sparsity and scales to datasets with more than 10^5 samples. More significantly, CRR generates relaxed-CNF rules with higher accuracy than CNF rules generated by [42]. Furthermore, compared to decision lists generated by [176], relaxed-CNF rules are sparser in large datasets.

4.1 Problem Formulation

Given

1. a dataset $\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ of n samples, where feature vector $\mathbf{x}^{(i)} \in \{0,1\}^m$ contains m features and class label $y^{(i)} \in \{0,1\}$,

2. a positive integer $k \geq 1$ denoting the number of clauses to be learned in the classification rule,

3. an integer threshold on literals $\eta_l \in \{0, \ldots, m\}$ indicating the minimum number of literals required to be true to satisfy a clause,

[1]The majority function is a Boolean function that evaluates to 0 when half or more arguments are false, and 1 otherwise (https://en.wikipedia.org/wiki/Majority_function).

4. an integer threshold on clauses $\eta_c \in \{0, \ldots, k\}$ indicating the minimum number of clauses required to be true to satisfy a formula, and

5. a data-fidelity parameter $\lambda \in \mathbb{R}^+$,

we learn a classification rule $\mathcal{R}$ expressed as a k clause relaxed-CNF formula separating samples of class 1 from class 0.

Our goal is to find rules that balance two goals: being accurate while also sparse to avoid over-fitting. To this end, we seek to minimize the total number of literals in all clauses, which motivates us to find $\mathcal{R}$ with minimum $|\mathcal{R}|$. In particular, suppose $\mathcal{R}$ classifies all samples correctly, i.e., $\forall i, y^{(i)} = \mathcal{R}(\mathbf{x}^{(i)})$. Among all the rules that classify all samples correctly, we choose the sparsest such $\mathcal{R}$:

$$\min_{\mathcal{R}} |\mathcal{R}| \text{ such that } \forall i, y^{(i)} = \mathcal{R}(\mathbf{x}^{(i)})$$

In practical classification tasks, perfect classification is very unusual. Hence, we need to balance rule-sparsity with prediction error. Let $\mathcal{E}_{\mathbf{D}}$ be the set of samples which are misclassified by $\mathcal{R}$ on the dataset $\mathbf{D}$, i.e., $\mathcal{E}_{\mathbf{D}} = \{(\mathbf{x}^{(i)}, y^{(i)}) | y^{(i)} \neq \mathcal{R}(\mathbf{x}^{(i)})\}$. Hence we aim to find $\mathcal{R}$ as follows:[2]

$$\min_{\mathcal{R}} \ \frac{\lambda}{n}|\mathcal{E}_{\mathbf{D}}| + \frac{1-\lambda}{k \cdot m}|\mathcal{R}|$$

Each term in the objective function is normalized in $[0, 1]$. The data-fidelity parameter $\lambda \in [0, 1]$ serves to balance the trade-off between prediction accuracy and sparsity. Higher values of λ produce lower prediction errors by sacrificing the sparsity of $\mathcal{R}$, and vice versa. It can be viewed as an inverse of the regularization parameter.

4.2 CRR: Classification Rules in Relaxed Logical Form

In this section, we describe the main contribution of our work, CRR, a framework for learning relaxed-CNF rules. CRR converts the learning problem into an ILP-based formulation, learns the optimal assignment of variables and constructs rule $\mathcal{R}$ based on the assignment. We organize the rest of this section as follows. We discuss the decision variables in Section 4.2.1, the constraints and the linear programming relaxation in Section 4.2.2, the incremental learning in Section 4.2.3, feature discretization in Section 4.2.4, and adaptation of CRR for learning other classification rules in Section 4.2.5.

4.2.1 Description of Variables

CRR considers two types of *decision* variables: (i) feature variables and (ii) noise or classification error variables. Since feature X_j can be present or not present in each of the

[2]In our formulation, it is straightforward to add class-conditional weights (e.g. to penalize false-alarms more than mis-detects), and to allow instance weights (per sample).

k clauses, CRR considers k variables, each denoted by B_j^i corresponding to feature X_j to denote its participation in the i^{th} clause, i.e, $B_j^i = \mathbb{1}[j^{\text{th}}$ feature is selected in i^{th} clause$]$. The q^{th} sample in the training dataset, however, can be misclassified by $\mathcal{R}$. Therefore, CRR introduces a noise variable $\xi_q \in \{0, 1\}$ corresponding to the q^{th} sample, so that the assignment of ξ_q can be interpreted whether $(\mathbf{x}^{(q)}, y^{(q)})$ is misclassified by $\mathcal{R}$ or not, i.e., $\xi_q = \mathbb{1}[q^{\text{th}}$ sample is misclassified$]$. Hence, the key idea of CRR for learning $\mathcal{R}$ is to define an ILP (Integer Linear Program) query over $k \cdot m + n$ decision variables, denoted by $\{B_1^1, B_2^1, \ldots, B_m^1, \ldots, B_m^k, \xi_1, \ldots, \xi_n\}$. In this context, we define $\mathbf{B}_i = \{B_j^i \mid j = 1, \ldots, m\}$ as a *vector of feature variables* corresponding to the i^{th} clause.

4.2.2 Construction of the ILP Query

In Eq. (4.1), we discuss the ILP query Q for learning a k-clause relaxed-CNF rule $\mathcal{R}$. The objective function in Eq. (4.1a) takes care of both the rule-sparsity and the prediction accuracy of $\mathcal{R}$. Since CRR prefers a sparser rule with as few literals as possible, we construct the objective function by preferring B_j^i to be 0. Moreover, to encourage $\mathcal{R}$ to predict the training samples accurately, we penalize the number of variables ξ_q that are different from 0. In this context, we utilize the parameter λ to trade off between sparsity and accuracy. Therefore, the objective function of the ILP query Q is to minimize the normalized sum of all noise variables ξ_q weighed by the data-fidelity parameter λ and feature variables B_j^i weighed by $1 - \lambda$.

We formulate the constraints of the ILP query Q as follows. Initially, we define the range of the decision variables and add constraints accordingly (Eq. 4.1b and 4.1c). For each sample, at first, we add constraints to mimic the behavior of hard-OR of literals in a clause, and then we add constraints to apply soft-AND of clauses in a formula (refer to Preliminaries in Chapter 2).

We first consider the case when the q^{th} sample has positive class label (Eq. 4.1d). $\mathbf{x}^{(q)} \circ \mathbf{B}_i \geq \eta_l$ resembles the hard-OR operation of literals in a clause. We introduce k *auxiliary* $\{0, 1\}$ variables $\{\xi_{q,1}, \ldots, \xi_{q,k}\}$ to check whether at least η_c clauses are satisfied, i.e., $\xi_{q,i} = \mathbb{1}[i^{\text{th}}$ clause is dissatisfied for q^{th} sample$]$, which let us impose the operation of soft-AND over clauses. We then add a constraint to make sure that at most $k - \eta_c$ clauses are allowed to be dissatisfied, otherwise the noise variable ξ_q is assigned to 1, i.e., the q^{th} sample is detected as aw noise.

A negative labeled sample has to dissatisfy more than $k - \eta_c$ clauses in $\mathcal{R}$ so that the sample is predicted as 0, which is equivalent to satisfying more than $k - \eta_c$ constraints $\mathbf{x}^{(q)} \circ \mathbf{B}_i < \eta_l$. As mentioned earlier, we introduce $\{0, 1\}$ variable $\xi_{q,i}$ to specify if the constraint $\mathbf{x}^{(q)} \circ \mathbf{B}_i < \eta_l$ is dissatisfied or not and restrict the count of dissatisfied clauses $\sum_{i=1}^{k} \xi_{q,i}$ to be less than η_c.

$$\min \frac{\lambda}{n} \sum_{q=1}^{n} \xi_q + \frac{1-\lambda}{k \cdot m} \sum_{i=1}^{k} \sum_{j=1}^{m} B_j^i \tag{4.1a}$$

such that,

$$B_j^i \in \{0,1\}, i = 1, \ldots, k, j = 1, \ldots, m \tag{4.1b}$$

$$\xi_q \in \{0,1\}, q = 1, \ldots, n \tag{4.1c}$$

$$\text{if } \forall q \in \{1, \ldots, n\}, \text{ if } y^{(q)} = 1, \tag{4.1d}$$

$$\mathbf{x}^{(q)} \circ \mathbf{B}_i + m\xi_{q,i} \geq \eta_l, i = 1, \ldots, k \tag{4.1e}$$

$$k\xi_q + k - \eta_c \geq \sum_{i=1}^{k} \xi_{q,i}$$

$$\xi_{q,i} \in \{0,1\}, i = 1, \ldots, k$$

$$\text{if } \forall q \in \{1, \ldots, n\}, y^{(q)} = 0, \tag{4.1f}$$

$$\mathbf{x}^{(q)} \circ \mathbf{B}_i < \eta_l + m\xi_{q,i}, i = 1, \ldots, k \tag{4.1g}$$

$$k\xi_q + \eta_c > \sum_{i=1}^{k} \xi_{q,i}$$

$$\xi_{q,i} \in \{0,1\}, i = 1, \ldots, k$$

In the following, we show the complexity of the ILP query in terms of the number of variables and constraints.

Proposition 6. *Given a training dataset with n samples and m binary features, the ILP query Q for learning a binary classification rule in relaxed-CNF has $k \cdot m + n$ decision variables, $k \cdot n$ auxiliary variables, and $k \cdot m + n \cdot (k + 3)$ integer constraints.*

An ILP solver takes query Q as input and returns the optimal assignment $\sigma^*$of the variables. We extract relaxed-CNF rule $\mathcal{R}$ from the solution as follows.

Construction 7. *Let $\sigma^* = \text{ILP}(Q)$, then $X_j \in \mathsf{clause}(\mathcal{R}, i)$ if and only if $\sigma^*(B_j^i) = 1$.*

Learning thresholds η_l and η_c: Given the training dataset $\mathbf{D}$ and data-fidelity parameter λ, one could learn the optimum value of the thresholds η_c and η_l of the desired rule $\mathcal{R}$ by specifying their range as constraints in the ILP query Q in Eq. 4.1. More precisely, we need to add two integer constraints $\eta_c \in \{0, \ldots, k\}$ and $\eta_l \in \{0, \ldots, m\}$ in the above query and then learn their values from the solution.

A more generalized version to CNF rules would be to learn different thresholds on literals for different clauses, i.e., $\eta_{l,i}$ for the i^{th} clause. This facilitates to capture the complex decision boundaries, while still producing rule-based decisions. In our ILP formulation, it is straight-forward to consider such generalization where we put constraints $\eta_{l,i} \in \{0, \ldots, m\}, i = 1, \ldots, k$ and replace η_l with $\eta_{l,i}$ in Eq. 4.1e and 4.1g.

Relaxing the ILP problem: The ILP query Q has binary integer constraints and the solution of this integer program is computationally intractable. A common approach that efficiently finds an approximate solution to such a problem extends to relax the integer constraints, solves the LP-relaxed (linear programming relaxation) problem, and then rounds the non-integer variables to get an integer solution as in [177]. In our case, we cannot relax all integer constraints because it may violate the structure of relaxed-CNF rules. Specifically, η_c and η_l (or $\eta_{l,i}$) must be integers in the construction of relaxed-CNF rules, and $\xi_{q,i}$ needs to be an integer to precisely calculate the noise variable ξ_q. However, we can relax feature variable B^i_j and noise variable ξ_q and solve the corresponding MILP problem. To construct the MILP problem, we replace Eq. 4.1b with $0 \leq B^i_j \leq 1$ and Eq. 4.1c with $0 \leq \xi_q \leq 1$, and the rest of the constraints in Q remain the same. If non-zero B^i_j is found in the solution, we set it to 1 and then construct the rule according to Construction 7.

4.2.3 Incremental Mini-batch Learning

In Section 4.2.2, we present an ILP formulation for learning relaxed-CNF rules and then discussed the relaxation to the integer constraints in the MILP formulation to make the approach computationally tractable. However, each integer constraints in the formulation cannot be relaxed, as discussed in Section 4.2.2. Thus we require an improved learning technique to achieve scalability. We now describe a mini-batch learning approach to CRR, that learns relaxed-CNF rule $\mathcal{R}$ incrementally from a set of mini-batches while solving a modified MILP query for each mini-batch.

In incremental mini-batch learning, the learning process repeats for a fixed number of iterations. In each iteration, we randomly select an equal number of samples from the full training set and generate a mini-batch with samples. We then construct an MILP query on the current mini-batch with a modified objective function. This objective function simultaneously penalizes the prediction errors on the current mini-batch and the change in the rules learned in consecutive batches. In the following, we discuss the modified objective function and the MILP formulation.

Let $\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ be a training dataset with n samples and m binary features and τ be the number of iterations in the learning process. In the p^{th} iteration, we randomly construct a mini-batch from $\mathbf{D}$ with equal number n_p of samples and $n_p \ll n$, for $p = 1, \ldots, \tau$. Note that all mini-batches have the same binary features $\mathbf{X} = \{X_1, \ldots, X_m\}$ as in the training set and thus share the same feature variables B^i_j in the MILP query. Therefore, we devise an objective function that prefers to keep the assignment of B^i_j learned in the $(p-1)^{\text{th}}$ iteration while minimizing the prediction error on the current mini-batch. To this end, each B^i_j is assigned 0 initially in the learning process. This technique enables us to update the learned rule in terms of the update in the assignment of feature variables B^i_j over mini-batches.

Let Q_p be the MILP query for the p^{th} mini-batch for learning a k-clause relaxed-CNF

rule $\mathcal{R}_p$. Thus, $\mathcal{R}_0$ is an empty rule. We consider an indicator function $I(\cdot) : B_j^i \rightarrow \{1, -1\}$, that takes a feature variable B_j^i as input and outputs -1 if B_j^i is assigned 1 in the solution of Q_{p-1} (i.e., feature X_j is selected in the i^{th} clause of $\mathcal{R}_{p-1}$), otherwise outputs 1.

$$I(B_j^i) = \begin{cases} -1 & \text{if } X_j \in \mathsf{clause}(\mathcal{R}_{p-1}, i) \\ 1 & \text{otherwise} \end{cases}$$

We now discuss the modified objective function where we multiply $I(B_j^i)$ with B_j^i differently from the objective function in Eq. 4.1a.

$$\min \frac{\lambda}{n} \sum_{q=1}^{n_p} \xi_q + \frac{1-\lambda}{k \cdot m} \sum_{i=1}^{k} \sum_{j=1}^{m} B_j^i \cdot I(B_j^i) \tag{4.2}$$

In the objective function, the first term penalizes the prediction error of samples in the current mini-batch and the second term penalizes when B_j^i is assigned differently than its previous assignment. Note that the total prediction error is normalized by dividing by n, which is the size of the full training dataset. This normalization is useful as it assists in updating $\mathcal{R}_p$ while also considering the relative size of the mini-batch. Intuitively, if the size n_p of the mini-batch is close to the size n of the training set, more priority is given to the prediction accuracy on the current batch and vice versa. The constraints in the query Q_p are similar to the constraints in Eq. 4.1. Finally, the prediction rule $\mathcal{R}$ is $\mathcal{R}_\tau$ that is learned for the last mini-batch.

Proposition 8. *At each iteration, the MILP query in the incremental mini-batch learning approach has $k \cdot m + n_p \cdot (k + 3)$ constraints, where n_p is the size of the mini-batch.*

Learning thresholds η_l and η_c: In the incremental learning, the objective function in Eq 4.2 does not impose any constraint on the thresholds. In fact, the thresholds are learned in each iteration by solving the corresponding MILP query and we consider their final values in the last iteration.

4.2.4 Learning with Non-binary Features

Since our problem formulation requires input instances to have binary features, datasets with categorical and continuous features require a preprocessing stage. Initially, for all continuous features, we apply entropy-based discretization [178] to infer the most appropriate number of categories/intervals by recursively splitting the domain of each continuous feature to minimize the class-entropy of the given dataset.[3] For example, let $X_c \in [a, b]$ be a continuous feature, and entropy-based discretization splits the domain

[3] A simple quantile-based discretization also works, but it requires an extra parameter (i.e., the number of quantiles).

$[a, b]$ into three intervals with two split points $\{a', b'\}$, where $a < a' < b' < b$. Therefore, the result intervals are $X_c < a'$, $a' \leq X_c < b'$, and $X_c \geq b'$.

After applying entropy-based discretization on continuous features, the dataset contains only categorical features, that can be converted to binary features using one-hot encoding as in [168]. In this encoding, a Boolean vector is introduced with cardinality equal to the number of distinct categories. Let a categorical feature have three categories 'red','green', and 'yellow'. In one-hot encoding, samples with category-value 'red','green', and 'yellow' would be converted into binary features while taking values 100, 010, and 001, respectively.

4.2.5 Learning Rules in Other Logical Forms

While CRR learns classification rules in relaxed-CNF form, we can leverage this framework for learning classification rules in other logical forms, for example, CNF and DNF. To learn a CNF rule, we set $\eta_l = 1$ and $\eta_c = k$, which converts a relaxed-CNF to a CNF formula. Moreover, to learn a DNF rule, we first complement the class label of all samples, learn a CNF rule by setting the parameters as described and finally negate the learned rule, which we have discussed elaborately in Chapter 3.

4.3 Experiments

We implement a prototype of CRR[4] based on the Python API for CPLEX and conduct an extensive empirical analysis to understand the behavior of CRR on real-world instances. The objective of our experimental evaluation is to answer the following questions:

1. How do the accuracy and training time for CRR behave vis-a-vis state-of-the-art classifiers on large datasets arising in machine learning problems in practice?
2. Can CRR generate sparse rules compared to that of other rule-based models?
3. How do the training time, accuracy, and rule size vary with model hyper-parameters?

In summary, relaxed-CNF rules generated by CRR achieves higher accuracy and more concise representation than CNF rules in most of the datasets. Moreover, relaxed-CNF rules are shown to be sparser than decision lists with competitive accuracy in large datasets. Finally, we show how to control the trade-off between rule-sparsity and accuracy using the hyper-parameter λ; and between accuracy and training time using the hyper-parameter k and size n_p of each mini-batch. In the following, we give a detailed description of the experiments.

[4]The source code is available at https://github.com/meelgroup/mlic

4.3.1 Experiment Methodology

We perform experiments on a high-performance computer cluster, where each node consists of E5-2690 v3 CPU with 24 cores, 96 GB of RAM. Each experiment is run on four cores of a node with 16 GB memory. We compare the performance of CRR with state-of-the-art classifiers, e.g. IMLI [42], RIPPER [176], BRS [170], random forest (RF), support vector classifier (SVC), nearest neighbors classifiers (k-NN), and l_1 penalized logistic regression (LR). Among them, IMLI, BRS, and RIPPER are rule-based classifiers. In particular, IMLI generates classification rules in CNF using a MaxSAT-based formulation and we use Open-WBO [179] as the MaxSAT solver for IMLI. We compare with propositional rule learning algorithm RIPPER, which is implemented in WEKA [180] and generates classification rules in the form of decision lists. BRS is a Bayesian framework for generating rule sets expressed as DNF. For other classifiers, we use the Scikit-learn module of Python [181]. For all classifiers, we set the training cut-off time to 1500 seconds.

We consider a comparable number of hyper-parameter choices for each classifier. Specifically for CRR, we choose the data-fidelity parameter $\lambda \in \{0.5, 0.67, 0.84, 0.99\}$, the number of clauses $k \in \{1, 2, 3\}$, the relative size of mini-batch $\frac{n_p}{n} \in \{0.25, 0.50, 0.75\}$, and the number of iterations $\tau \in \{2, 4, 8, 16\}$. We learn the value of η_c and η_l from the dataset as described in Chapter. 4.2.2. In CPLEX, we set the maximum solving time of the LP solver to 1000 seconds ($\frac{1000}{\tau}$ seconds for each iteration) and the remaining 500 seconds is allotted to construct the MLIP instances, parse the solutions and execute other auxiliary tasks of the learning algorithm. We present the current best solution of CPLEX when the solver times out while finding the optimal solution.

We control the cut-off of the number of examples in the leaf node in the case of RF and RIPPER. For SVC, k-NN, and LR we discretize the regularization parameter on a logarithmic grid. For BRS, we vary the max clause-length $\in \{3, 4, 5\}$, support $\in \{5, 10, 15\}$, and two other parameters $s \in \{100, 1000, 10000\}$ and $\rho \in \{0.9, 0.95, 0.99\}$. For IMLI, we consider $\lambda \in \{1, 5, 10\}$ and $k \in \{1, 2, 3\}$ and vary the number of batches τ such that each batch has at least 32 samples and at most 512 samples.

4.3.2 Results

In the following, we first discuss empirical results of rule-based classifiers, then extend analysis to non-rule-based classifiers, and finally discuss the effect of different choices of hyper-parameters.

Performance Evaluation of CRR with Rule-based Classifiers:

We conduct an assessment of performance using five-fold nested cross-validation as in [182] and report the median of test accuracy, rule-size and training time of all rule-based classifiers in Table 4.1. Specifically, we show the dataset, the number of samples and the number of discretized features in the first three columns in Table 4.1. Inside each cell of

Table 4.1: Comparisons of test accuracy, rule-size and training time among different rule-based classifiers. Every cell in the last four columns contains the test accuracy in percentage (top value), rule size (middle value), and training time in seconds (bottom value). In the experiments, CRR shows higher test accuracy than IMLI and generates sparser rules than RIPPER. Number in bold denotes the best result, such as maximum test accuracy, minimum rule-size, and minimum training time among competitive classifiers.

Dataset	Size	Features	RIPPER	BRS	IMLI	CRR
Heart	303	31	**78.69**	72.13	72.13	77.69
			7	19	13	**4.5**
			5.27s	25.07s	**1.8**s	122.5s
Ionosphere	351	144	88.65	**91.43**	89.29	**91.43**
			8	**4**	8.5	20
			5.87s	75.53s	**2.09**s	5.59s
WDBC	569	88	95.22	**95.65**	93.91	94.69
			7.5	12	**7**	34.5
			5.7s	630.23s	**1.38**s	316.32s
Magic	19020	79	**84.04**	74.15	71.97	81.31
			115	**3**	24	31
			15.86s	56.46s	141.2s	1012.6s
Tom's HW	28179	910	**97.4**		95.88	97.34
			36	—	30	**4**
			42.73s		92.65s	1071.58s
Credit	30000	110	81.68		81.42	**82.04**
			38.5	—	**10**	32
			14.52s		17.66s	1021.35s
Adult	32561	144	84.31		82.08	**84.86**
			94	—	23	**18**
			27.61s		**11.91**s	1016.36s
Twitter	49999	1511	**95.74**		94.24	95.16
			179.5	—	57	**12**
			170.87s		238.29s	1144.66s
Weather-AUS	107696	141	**84.57**		82.83	83.34
			195	—	7	**2**
			121.02s		366.12s	1115.27s

column four to 11, we present the test accuracy (top value), rule-size (middle value) and training time (bottom value) of each classifier for each dataset.

We first compare relaxed-CNF rules generated by CRR with CNF rules generated by IMLI. In Table 4.1, relaxed-CNF rules exhibit higher prediction accuracy than CNF rules in the majority of the datasets, showing the effectiveness of using a more expressive representation of classification rules in capturing the decision boundary. In addition, the generated relaxed-CNF rules are comparatively smaller than CNF rules in terms of rule-size in most of the datasets. Therefore, relaxed-CNF rules improve upon CNF rules in terms of both prediction accuracy and rule size in the majority of the datasets. In this context, CRR provides a trade-off between accuracy and rule-size depending on the choice of hyper-parameters and the experimental results are discussed later in Section 4.3.2. We then compare relaxed-CNF rules with DNF rules generated by BRS and find that relaxed-CNF rules outperform DNF rules with respect to prediction accuracy in several datasets. At this point, BRS fails to scale on larger datasets as shown in Table 4.1. We finally compare relaxed-CNF rules with decision lists generated by RIPPER. In the experiments, relaxed-CNF rules achieve comparable prediction accuracy with decision lists in most of the datasets. In contrast, RIPPER generates very large decision lists compared to relaxed-CNF rules in the majority of the datasets, more precisely in large datasets. To summarize the performance of CRR among different rule-based classifiers, CRR can generate smaller relaxed-CNF rules with better accuracy in numbers of the cases with a couple of exceptions.

Moving focus on the training time, the non-incremental version of CRR times out on larger instances in the experiments, potentially producing sub-optimal rules with reduced accuracy, thereby highlighting the need for the incremental approach. On the other hand, the incremental version of CRR can handle most of the datasets within the allotted amount of time. In Table 4.1, CRR takes a comparatively longer time to generate relaxed-CNF rules in comparison with other rule-based classifiers, e.g., RIPPER, and IMLI because of the flexible combinatorial structure of relaxed-CNF rules. However, the testing time of CRR is insignificant (< 0.01 seconds) and thus can be deployed in practice.

Performance Evaluation of CRR with Non-rule-based Classifiers:

We compare the test accuracy of CRR with non-rule-based classifiers: LR, SVC, RF, and k-NN and report the results in Table 4.2. In the experiments, we find that CRR, in spite of being a rule-based classifier, is able to achieve competitive prediction accuracy with non-rule-based classifiers. In this context, SVC, and k-NN can not complete training within the allotted time particularly in the datasets with more than 10^5 samples, while CRR can still generate relaxed-CNF rules with competitive accuracy. Therefore, CRR shows the promise of applying rule-based classifiers in practice with an added benefit of interpretability along with competitive accuracy.

Table 4.2: Comparisons of test accuracy among CRR and non-rule-based classifiers. In the experiments, CRR achieves competitive prediction accuracy in spite of being a rule-based classifier.

Dataset	LR	SVC	RF	k-NN	CRR
Heart	**84.29**	83.33	81.97	78.69	77.69
Ionosphere	**94.29**	91.43	92.96	91.43	91.43
WDBC	**98.26**	96.46	96.90	95.61	94.69
Magic	85.15	84.45	**85.30**	77.9	81.31
Tom's HW	97.62	**97.66**	97.52	94.59	97.34
Credit	82.04	**82.12**	81.97	80.5	82.04
Adult	**87.24**	86.82	86.84	84.68	84.86
Twitter	96.28	96.34	**96.37**	—	95.16
Weather-AUS	**85.71**	—	85.63	—	83.34
Skin	97.21	—	**99.81**	—	95.08

Varying Model Parameters:

In Figure 4.2, 4.3, , 4.4, and 4.5, we demonstrate the effect of varying the hyper-parameters of CRR. To understand the effect of a single hyper-parameter, we fix the values of other hyper-parameters to a default choice where the default choice results in the most accurate rule.

Varying data-fidelity parameter (λ): In Figure 4.2, as we increase data-fidelity parameter λ in the objective function in Eq. 4.1a and Eq. 4.2, more priority is given to the prediction accuracy than the sparsity of the rules. In most of the datasets, we similarly observe an increase in accuracy and also an increase in the size of the rules when λ is higher. This suggests that improved interpretability can often come at a minor cost in accuracy. In addition, we find an increase in training time for most of the datasets indicating that the MILP query usually takes a longer time to find the solution when more priority is given on the prediction accuracy.

Varying the number of clauses (k): In Figure 4.3, as we increase k, CRR allows the generated rules to capture the variance in the given dataset more effectively, that results in higher accuracy in most of the datasets. The rule-size also increases as we learn more clauses. In addition, the training time increases, that can be reasoned by the fact that the number of constraints in the MILP formulation is linear with k. Thereby, the number of clauses k provides a control over the accuracy of the generated rule and training time.

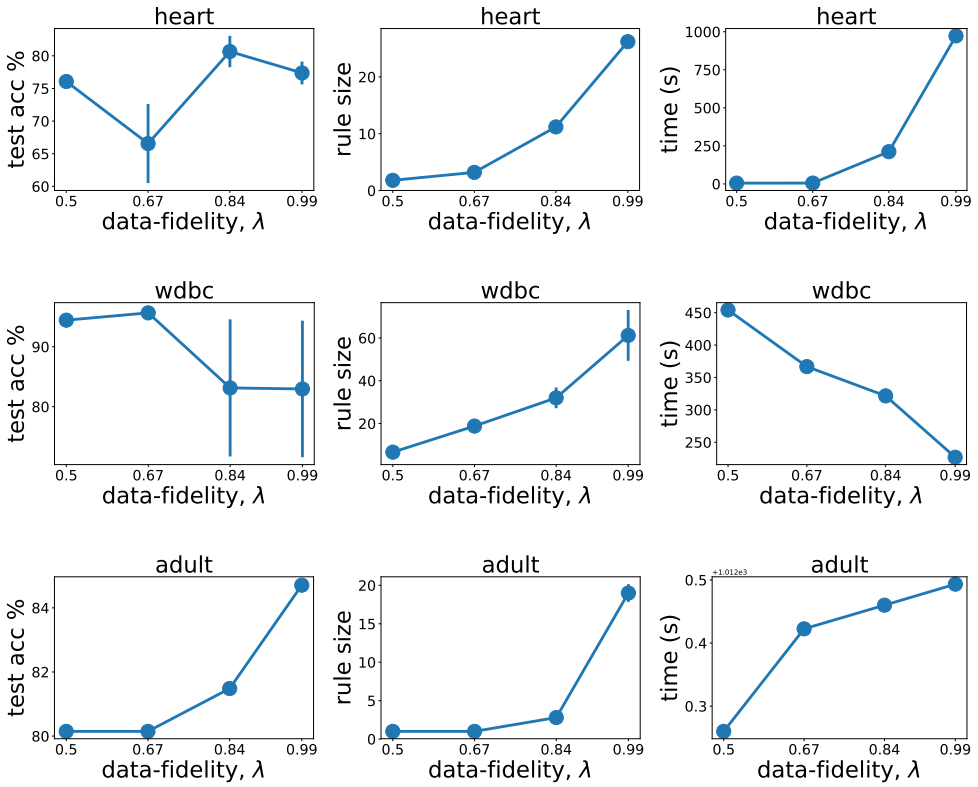


Figure 4.2: Effect of data-fidelity λ on test accuracy, rule-size, and training time in CRR.

Relative size of the mini-batch: In Figure 4.4, we vary the relative size of the mini-batch $\frac{n_p}{n}$ to observe its effect on the accuracy and the size of the rules. In most datasets, the accuracy increases when more samples are considered in the mini-batch, costing higher training time. Moreover, the size of the generated rule increases as $\frac{n_p}{n}$ increases, that can be supported by the increase in the variance of the samples in the mini-batch.

Varying the number of iterations (τ): In Figure 4.5, as we allow more iterations in the learning process, we find an increase in accuracy in most datasets. The training time also increases with τ because CRR is required to solve in total τ queries. We also observe an increase in rule-size in most datasets. The reason is that the objective function in the incremental mini-batch approach in Eq. 4.2 does not put a restriction on the size of the rules, rather on the change of rules in consecutive iterations. In addition, the learned values of the thresholds η_c and η_l in one iteration are not carried to the MILP query in the next iteration, that may cause an increase of rule-size.

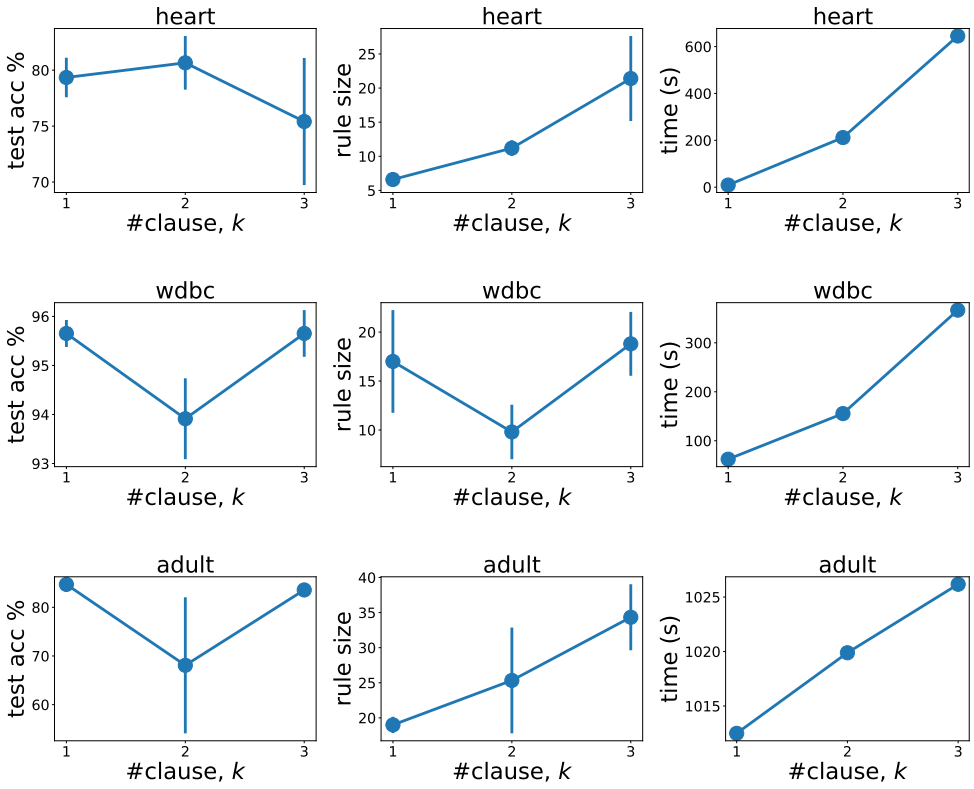


Figure 4.3: Effect of the number of clause k on test accuracy, rule-size, and training time in CRR.

4.4 Chapter Summary

In this chapter, we proposed an efficient combinatorial framework, called CRR, for learning relaxed-CNF classification rules. Relaxed-CNF rules are more expressive than CNF/DNF rules. CRR uses a novel integration of mini-batch learning procedure with the MILP framework to learn sparse relaxed-CNF rules. Our experimental results demonstrate that CRR is able to learn relaxed-CNF rules with higher accuracy and more concise representation than CNF rules. Moreover, the generated rules are sparser than decision lists in large datasets.

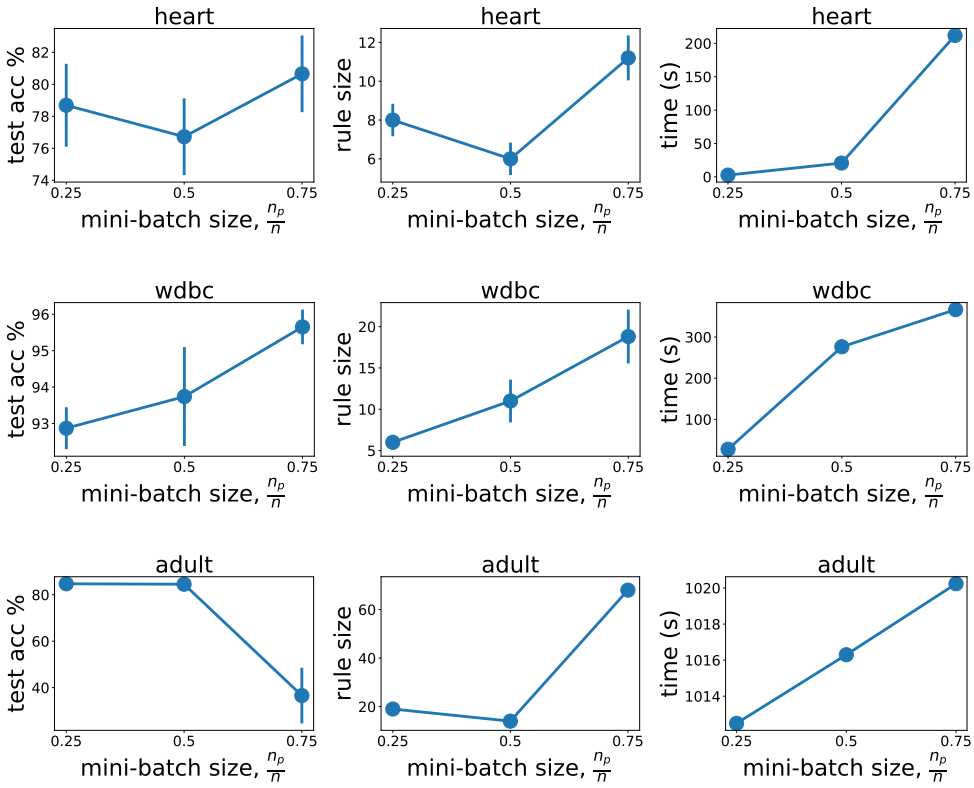


Figure 4.4: Effect of mini-batch size on test accuracy, rule-size, and training time in CRR.

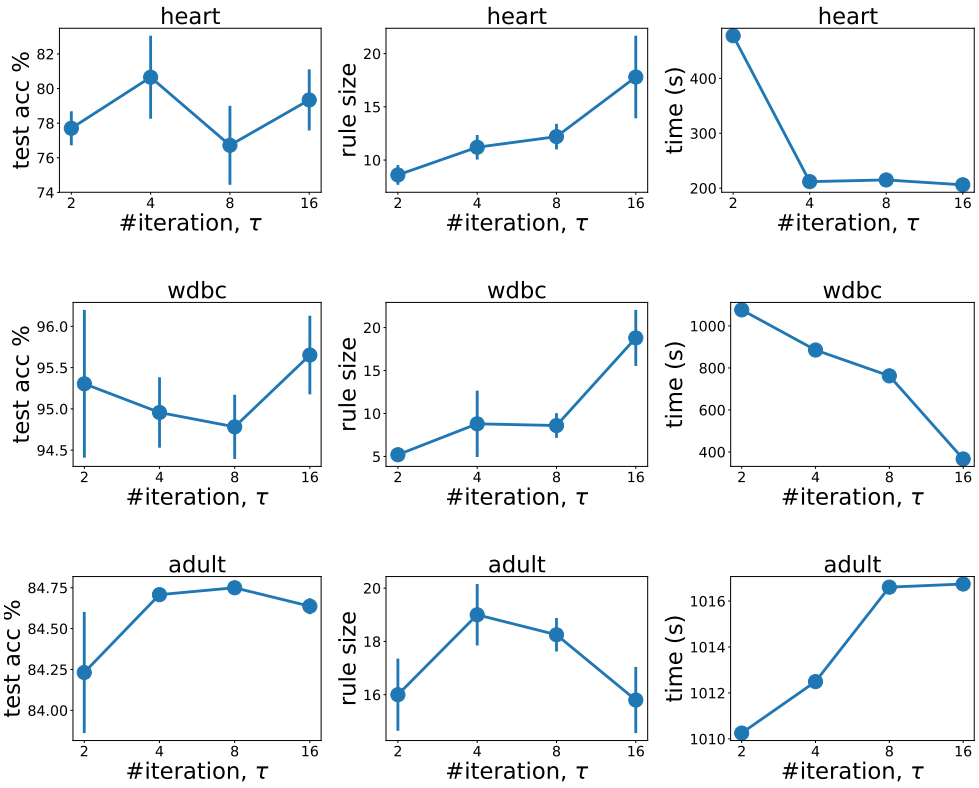


Figure 4.5: Effect of the number of iterations on test accuracy, rule-size, and training time in CRR.

Part III

Fairness in Machine Learning

In this part of the thesis, we discuss fairness verification for classifiers represented as Boolean formulas in Chapter 5 and linear classifiers in Chapter 6. Finally, we discuss the identification of the sources of unfairness in Chapter 7.

Chapter 5

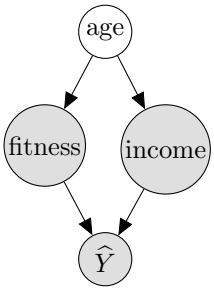
A Stochastic SAT Approach to Formally Verify Fairness

Machine learning is becoming the omnipresent technology of our time. Machine learning classifiers are being used for high-stake decisions such as college admission [44], recidivism prediction [45], job applications [46] etc. Thus, human lives are now pervasively influenced by data, classifiers, and their inherent bias.

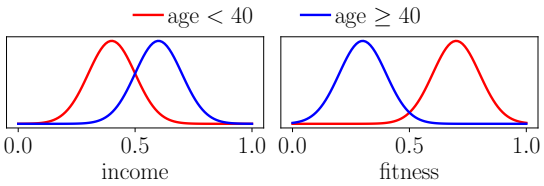
Example 5.0.1. Let us consider an example (Figure 5.1) of deciding the eligibility for health insurance depending on the fitness and income of individuals of different age groups (20-40 and 40-60). Typically, incomes of individuals increase as their ages increase while their fitness deteriorates (Figure 5.1a). We assume that the relation of income and fitness depends on ages as per the Normal distributions in Figure 5.1b. Now, if we train a decision tree [28] to decide the eligibility of an individual to get a health insurance given three features: fitness, income and age, we observe that the 'optimal' decision tree (ref. Figure 5.1c) does not predict based on the sensitive feature age. However, a fairness verifier would verify that the decision tree outputs positive prediction to an individual above and below 40 years with probabilities 0.18 and 0.72 respectively (Figure 5.1d). This simple example demonstrates that even if a classifier does not explicitly learn to differentiate on the basis of a sensitive feature, it discriminates different age groups due to the utilitarian sense of accuracy that it tries to optimize.

Fair Machine Learning

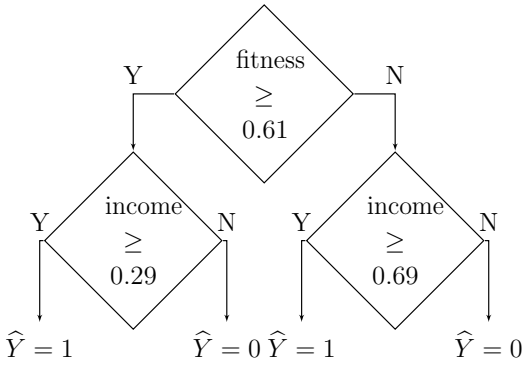
Statistical discrimination caused by classifiers has motivated researchers to formulate several definitions of fairness and develop associated algorithms to mitigate bias. In this chapter, we focus on a popular concept of fairness, known as group fairness. Existing group fairness metrics mostly belong to three categories: *independence*, *separation*, and *sufficiency* [183]. Independence metrics, such as demographic parity, statistical parity, and group parity, try and ensure the outcomes of a classifier to be independent of the groups



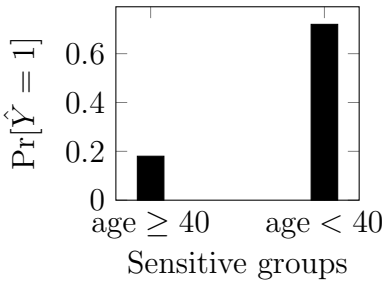
(a) Dependency among features and prediction



(b) Age-dependent distributions of non-sensitive features



(c) Trained decision tree



(d) Group specific conditional probability of the classifier's positive prediction

Figure 5.1: A trained decision tree to learn the eligibility for health insurance using age-dependent fitness and income indicators. This classifier makes unfair prediction to individuals with age above 40.

that the individuals belong to [47, 52]. Separation metrics, such as equalized odds, define a classifer to be fair if the probability of getting the same outcomes for different groups are same, separately for data from each possible ground-truth class label [59]. Sufficiency metrics, such as counterfactual fairness, constrain the probability of outcomes to be independent of individual's sensitive data given their identical non-sensitive data [184].

In Figure 5.1, independence is satisfied if the probability of getting insurance is same for both the age groups. Separation is satisfied if the number of 'actually' (ground-truth) ineligible and eligible people getting the insurance are same. Sufficiency is satisfied if the eligibility is independent of their age given their features are the same. Thus, we see that the metrics of fairness can be contradictory and complementary depending on the application and the data [185]. To this end, different algorithms have also been devised to ensure one or multiple of the fairness definitions. These algorithms try to rectify and mitigate the bias in the data and thus in the classifier in three ways: *pre-processing* the data [54–56], *in-processing* the classifier [57], and *post-processing* the outcomes of a classifier [58, 59].

Fairness Verifiers

Due to the abundance of fairness metrics and difference in algorithms to achieve them, it has become necessary to verify different fairness metrics over datasets and algorithms.

In order to verify fairness as a model property on a dataset, verifiers like *FairSquare* [62] and *VeriFair* [63] have been proposed. These verifiers are referred to as *probabilistic verifiers* owing to the fact that their inputs are a probability distribution of the features in the dataset and a classifier of a suitable form, and their objective is to verify fairness with respect to the distribution and the classifier. Though FairSquare and VeriFair are robust and have asymptotic convergence guarantees, we observe that they scale up poorly with the size of inputs and also do not generalize to non-Boolean and compound sensitive features. In contrast to the probabilistic verifiers, another line of work, referred to as sample-based verifiers, has focused on the design of testing methodologies on a given fixed data sample [61, 186]. Since sample-based verifiers are dataset-specific, they generally do not provide robustness over the distribution.

Thus, a *unified formal framework* to verify *different fairness metrics* of a classifer, which is *scalable*, capable of *handling compound protected groups*, *robust* with respect to the test data, and *operational on real-life* datasets and fairness-enhancing algorithms, is missing in the literature.

Our Contribution.

From this vantage point, *we propose to model verifying different fairness metrics as a Stochastic Boolean Satisfiability (SSAT) problem* [64]. SSAT was originally introduced by [187] to model *games against nature*. In this work, we primarily focus on reductions to the exist-random quantified fragment of SSAT, which is also known as E-MAJSAT [64]. SSAT is a conceptual framework that has been employed to capture several fundamental problems in artificial intelligence such as the computation of maximum a posteriori (MAP) hypothesis [76], propositional probabilistic planning [188], and circuit verification [189]. Furthermore, our choice of SSAT as a target formulation is motivated by the recent algorithmic progress that has yielded efficient SSAT tools [78, 79].

Our contributions are summarised below:

- We propose a unified SSAT-based approach, Justicia, to verify independence and separation metrics of group and causal fairness metrics for different datasets and classifiers.
- Unlike previously proposed formal probabilistic verifiers, namely FairSquare and VeriFair, Justicia verifies fairness for compound and non-Boolean sensitive features.
- Our experiments validate that our method is more accurate and scalable than the probabilistic verifiers, such as FairSquare and VeriFair, and more robust than the sample-based empirical verifiers, such as AIF360.

It is worth remarking that significant advances in artificial intelligence bear testimony to the right choice of formulation, for example, formulation of planning as satisfiability (SAT) [190]. In this context, we view that formulation of fairness as SSAT has potential to spur future work from both the modeling and encoding perspective as well as core algorithmic improvements in the underlying SSAT solvers.

5.1 Justicia: An SSAT-based Fairness Verifier

In this section, we present the primary contribution of this chapter, Justicia, which is an SSAT-based framework for verifying group and causal fairness metrics. Given a binary classifier $\mathcal{M}$, a probability distribution over features $(\mathbf{X}, \mathbf{A}, Y) \sim \mathcal{D}$, and a target fairness metric $f(\mathcal{M}, \mathcal{D})$, our goal is to estimate the fairness $f(\mathcal{M}, \mathcal{D})$ of the classifier $\mathcal{M}$ given the distribution $\mathcal{D}$ according to the fairness definition. Additionally, if a fairness threshold $\epsilon \in [0, 1]$ is provided, Justicia verifies whether the classifier is ϵ-fair by comparing f with ϵ (refer to Chapter 2). In Justicia, we focus on classifiers that can be represented as a CNF formula defined over a set of Boolean variables. Additionally, for each variable, we query the distribution $\mathcal{D}$ to derive the marginal probability of the variable to be assigned to 1. In this section, we discuss two equivalent approaches for fairness verification based on SSAT-based encodings: *enumeration* approach and *inference* approach. In both approaches, we verify fairness with the presence of compound sensitive groups. We then provide a theoretical analysis for a high-probability error bound on the fairness metric and conclude with an extension of Justicia in practical settings.

5.1.1 Enumeration Approach using RE-SSAT encoding

In order to estimate $f(\mathcal{M}, \mathcal{D})$ in the enumeration approach, the key idea is to compute the conditional probability of positive prediction of the classifier, $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$, for the compound sensitive group $\mathbf{A} = \mathbf{a}$ by solving an appropriately designed SSAT formula. For simplicity, we initially make assumptions on the classifier $\mathcal{M}$ and discuss practical relaxations later in this section. We first assume $\mathcal{M}$ to be represented as a CNF formula, denoted by $\phi_{\widehat{Y}}$, such that the prediction $\widehat{Y} = 1$ when $\phi_{\widehat{Y}}$ is satisfied and $\widehat{Y} = 0$ otherwise. Additionally, all features $\mathbf{X} \cup \mathbf{A}$ are assumed to be Boolean variables. Finally, we consider independence probability assumption of non-sensitive features $\mathbf{X}$, where $p_i \triangleq \Pr[X_i = 1]$ is the marginal probability of X_i.

Now, we define an RE-SSAT formula $\Phi_{\mathbf{a}}$ to compute the conditional probability $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ using the probability of satisfaction of $\Phi_{\mathbf{a}}$. In the prefix of $\Phi_{\mathbf{a}}$, all non-sensitive features $\mathbf{X}$ are assigned randomized quantifiers and they are followed by sensitive features $\mathbf{A}$ with existential quantifiers. In addition, the CNF formula ϕ in the SSAT formula $\Phi_{\mathbf{a}}$ is constructed such that ϕ encodes the event inside the target probability $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$. In order to specify the sensitive group $\mathbf{A} = \mathbf{a}$, we take the conjunction of the Boolean variables in $\mathbf{A}$ that symbolically specifies the compound sensitive group $\mathbf{A} = \mathbf{a}$.

For example, let us consider two sensitive features: race $\in$ {White, Colour} and sex $\in$ {male, female} by Boolean variables R and S, respectively. Hence, the compound groups [White, male] and [Colour, female] are represented by $R \wedge S$ and $\neg R \wedge \neg S$, respectively. Thus, the RE-SSAT formula for computing the probability $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ is

$$\Phi_{\mathbf{a}} := \underbrace{\mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k}_{\text{non-sensitive features}}, \underbrace{\exists A_1, \ldots, \exists A_m}_{\text{sensitive features}}, \phi_{\widehat{Y}} \wedge (\mathbf{A} = \mathbf{a}).$$

In the RE-SSAT formula $\Phi_{\mathbf{a}}$, existentially quantified variables $\{A_1, \ldots, A_m\}$ are assigned Boolean values according to the constraint $\mathbf{A} = \mathbf{a}$.[1] Next, an SSAT solver computes the probability $\Pr[\Phi_{\mathbf{a}}]$ by considering the random values of $\{X_1, \ldots, X_k\}$ while fixing the assignment of $\{A_1, \ldots, A_m\}$. Therefore, $\Pr[\Phi_{\mathbf{a}}]$ equals the conditional probability of positive prediction of the classifier, $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$, for the sensitive group $\mathbf{A} = \mathbf{a}$.

For simplicity, we have described the computation of conditional probability $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ without considering the correlation among sensitive and non-sensitive features. In reality, correlation exists among these features (for a detailed study on feature correlations in fairness verification, we refer to Chapter 6). As a result, non-sensitive features may have different conditional distributions for different sensitive groups. For a non-sensitive feature X_i, we incorporate its conditional probability in the RE-SSAT encoding by setting $p_i = \Pr[X_i = 1 | \mathbf{A} = \mathbf{a}]$ instead of the independent probability $\Pr[X_i = 1]$. Next, we illustrate this enumeration approach in Example 5.1.1.

Example 5.1.1 (RE-SSAT encoding)**.** We illustrate the RE-SSAT encoding for calculating the probability of positive prediction for the sensitive group age ≥ 40 in the decision tree of Figure 5.1. We assign three Boolean variables F, I, J for three nodes in the decision tree, where literal F, I, J denote fitness ≥ 0.61, income ≥ 0.29, and income ≥ 0.69, respectively. We consider another Boolean variable A where the literal A represents the sensitive group age ≥ 40 and $\neg A$ denotes age < 40. Thus, the CNF formula for the decision tree is $(\neg F \vee I) \wedge (F \vee J)$. From the distribution in Figure 5.1, we get $\Pr[F] = 0.41, \Pr[I] = 0.93$, and $\Pr[J] = 0.09$. Given this information, we calculate the probability of positive prediction for the sensitive group age ≥ 40 by solving the following RE-SSAT formula:

$$\Phi_A := \mathfrak{U}^{0.41} F, \mathfrak{U}^{0.93} I, \mathfrak{U}^{0.09} J, \exists A,\ (\neg F \vee I) \wedge (F \vee J) \wedge A.$$

From the solution to this SSAT formula, we get $\Pr[\Phi_A] = 0.43$. Similarly, to calculate the probability of positive prediction for the group age < 40, we replace the unit clause[2] A with $\neg A$ in the CNF formula in Φ_A and construct another SSAT formula $\Phi_{\neg A}$.

$$\Phi_{\neg A} := \mathfrak{U}^{0.41} F, \mathfrak{U}^{0.93} I, \mathfrak{U}^{0.09} J, \exists A,\ (\neg F \vee I) \wedge (F \vee J) \wedge \neg A$$

[1] An RE-SSAT formula becomes an R-SSAT formula when the assignment to the existential variables are fixed.

[2] A unit clause is a clause with a single literal.

For $\Phi_{\neg A}$, the solution $\Pr[\Phi_{\neg A}] = 0.43$ is similarly derived from an SSAT solver. Therefore, if $\Pr[F], \Pr[I], \Pr[J]$ are computed independently of the sensitive feature A, both age groups achieve an equal probability of positive prediction as the sensitive feature is not explicitly present in the classifier.

However, there is an implicit bias in the data distribution for different sensitive groups and the classifier unintentionally learns it. To capture this implicit bias, we calculate conditional probabilities $\Pr[F|A] = 0.01, \Pr[I|A] = 0.99$, and $\Pr[J|A] = 0.18$ from the distribution for group age ≥ 40 in Figure 5.1. Providing the conditional probabilities, we construct a modified SSAT formula Φ'_A and compute $\Pr[\Phi'_A] = 0.18$ for age ≥ 40.

$$\Phi'_A := \mathfrak{U}^{0.01} F, \mathfrak{U}^{0.99} I, \mathfrak{U}^{0.18} J, \exists A,\ (\neg F \vee I) \wedge (F \vee J) \wedge A$$

For the sensitive group age < 40, we similarly obtain $\Pr[F|\neg A] = 0.82, \Pr[I|\neg A] = 0.88$, $\Pr[J|\neg A] = 0.01$, construct the modified formula $\Phi'_{\neg A}$ and get $\Pr[\Phi'_{\neg A}] = 0.72$.

$$\Phi'_{\neg A} := \mathfrak{U}^{0.82} F, \mathfrak{U}^{0.88} I, \mathfrak{U}^{0.01} J, \exists A,\ (\neg F \vee I) \wedge (F \vee J) \wedge \neg A$$

In the later case with correlations among sensitive and non-sensitive features, the RE-SSAT encoding detects the discrimination of the classifier among different sensitive groups, where the classifier is more biased towards the younger group with age < 40 than the elderly group with age ≥ 40.

Measuring Fairness Metrics

As we compute $\Pr[\Phi_{\mathbf{a}}] = \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$ by solving the SSAT formula $\Phi_{\mathbf{a}}$, we use $\Pr[\Phi_{\mathbf{a}}]$ to measure different fairness metrics. To this end, we compute $\Pr[\Phi_{\mathbf{a}}]$ for all compound groups $\mathbf{a} \in \mathbf{A}$ by solving an exponential number (with m) of SSAT formulas. We elaborate this enumeration approach, namely Justicia_enum, in Algorithm 5 (Line 1–9).

To measure the disparate impact of a classifier, we calculate the ratio between the minimum and the maximum conditional probability of positive prediction of the classifier, which are $\min_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}]$ and $\max_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}]$, respectively. We compute statistical parity by taking the difference between $\max_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}]$ and $\min_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}]$.

Moreover, to compute equalized odds, we call Justicia twice, one for the distribution $\mathcal{D}$ conditioned on $Y = 1$ and another for $Y = 0$. In both calls, we compute $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}, Y = y] - \min_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}, Y = y]$ for $y \in \{0, 1\}$ and take the maximum difference as the value of equalized odds. For measuring path-specific causal fairness, we compute $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}, \mathbf{Z}]$ and $\min_{\mathbf{a}} \Pr[\widehat{Y} = 1, \mathbf{Z}|\mathbf{A} = \mathbf{a}, \mathbf{Z}]$ by conditioning the distribution $\mathcal{D}$ by mediator features $\mathbf{Z}$ and take their difference. Thus, Justicia_enum allows us to compute different group and causal fairness metrics using a unified algorithmic framework.

Algorithm 5 Justicia: An SSAT-based Fairness Verifier

function Justicia_enum($\mathbf{X}, \mathbf{A}, \widehat{Y}$)
  $\phi_{\widehat{Y}} := \text{CNF}(\widehat{Y} = 1)$
  for all $\mathbf{a} \in \mathbf{A}$ **do**
    $p_i \leftarrow \Pr[X_i = 1 | \mathbf{A} = \mathbf{a}], \forall X_i \in \mathbf{X}$
    $\phi := \phi_{\widehat{Y}} \wedge (\mathbf{A} = \mathbf{a})$
    $\Phi_{\mathbf{a}} := \mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k, \exists A_1, \ldots, \exists A_m, \phi$
    $\Pr[\Phi_{\mathbf{a}}] \leftarrow \mathsf{SSAT}(\Phi_{\mathbf{a}})$ ▷ returns a probability

  return $\max_{\mathbf{a} \in \mathbf{A}} \Pr[\Phi_{\mathbf{a}}], \min_{\mathbf{a} \in \mathbf{A}} \Pr[\Phi_{\mathbf{a}}]$

function Justicia_infer($\mathbf{X}, \mathbf{A}, \widehat{Y}$)
  $\phi_{\widehat{Y}} := \text{CNF}(\widehat{Y} = 1)$
  $p_i \leftarrow \Pr[X_i = 1], \forall X_i \in \mathbf{X}$
  $\Phi_{\mathsf{ER}} := \exists A_1, \ldots, \exists A_m, \mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k, \phi_{\widehat{Y}}$
  $\Phi'_{\mathsf{ER}} := \exists A_1, \ldots, \exists A_m, \mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k, \neg\phi_{\widehat{Y}}$

  return $\mathsf{SSAT}(\Phi_{\mathsf{ER}}), 1 - \mathsf{SSAT}(\Phi'_{\mathsf{ER}})$

5.1.2 Inference Approach using ER-SSAT Encoding

In most practical problems, there can be exponentially many compound sensitive groups due to the combination of categorical sensitive features. As a result, the enumeration approach may suffer from scalability issues due to the exponential number of calls to the SSAT solver. To this end, we propose an efficient SSAT encoding, where we make two SSAT calls, one for inferring the *the most favored sensitive group* with the maximum conditional probability of positive prediction of the classifier and another for inferring *the least favored sensitive group* with the minimum conditional probability of positive prediction of the same classifier. As discussed above, these two probabilities are sufficient to measure different group and causal fairness metrics.

Inferring the Most Favored Sensitive Group

In the prefix of an SSAT formula Φ, the order of quantified variables carries distinct interpretation of $\Pr[\Phi]$. In an ER-SSAT formula, $\Pr[\Phi]$ is the *maximum* satisfying probability of Φ over the optimal assignment of existentially quantified variables given the randomized quantified variables (by Semantic 2, Sec. 2.1.5). In this chapter, we leverage this property to compute the most favored sensitive group with the highest probability of positive prediction of the classifier. In particular, we consider the following ER-SSAT formula:

$$\Phi_{\mathsf{ER}} := \exists A_1, \ldots, \exists A_m, \mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k,\ \phi_{\widehat{Y}}. \tag{5.1}$$

The CNF formula $\phi_{\widehat{Y}}$ in Φ_{ER} is the CNF translation of the classifier $\widehat{Y} = 1$ without any specification of the compound sensitive group. Therefore, as we solve Φ_{ER}, we find the optimal assignment to the existentially quantified variables $A_1 = a_1^{\max}, \ldots, A_m = a_m^{\max}$ for which the probability of satisfaction of the ER-SSAT formula $\Pr[\Phi_{\mathsf{ER}}]$ becomes maximum. Thus, we compute the most favored group $\mathbf{a}_{\max} \triangleq [a_1^{\max}, \ldots, a_m^{\max}]$ achieving the highest probability of positive prediction of the classifier.

Inferring the Least Favored Sensitive Group

In order to infer the least favored sensitive group of the classifier, we compute the *minimum* conditional probability of positive prediction of the classifier with respect to all sensitive groups given the random values of the non-sensitive features. To this end, we solve a 'universal-random' (UR) SSAT formula with universal quantifiers over sensitive features and randomized quantification over non-sensitive features (by Semantic 3, Sec. 2.1.5).

$$\Phi_{\mathsf{UR}} := \forall A_1, \ldots, \forall A_m, \mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k,\ \phi_{\widehat{Y}} \tag{5.2}$$

Solving an UR-SSAT formula raises several practical issues and thus, there is an unavailability of an UR-SSAT solver. To resolve this problem, we leverage the *duality* between UR-SSAT and ER-SSAT formulas, where we solve an UR-SSAT formula on the CNF ϕ using the solution of an ER-SSAT formula on the complemented CNF $\neg\phi$ [64]. More specifically, we solve the following ER-SSAT formula for finding the least favored sensitive group.

$$\Phi'_{\mathsf{ER}} := \exists A_1, \ldots, \exists A_m, \mathfrak{U}^{p_1} X_1, \ldots, \mathfrak{U}^{p_k} X_k,\ \neg\phi_{\widehat{Y}} \tag{5.3}$$

In Lemma 9, we discuss the duality between UR-SSAT and ER-SSAT formulas.

Lemma 9. *Given Eq.* (5.2) *and* (5.3), $\Pr[\Phi_{\mathsf{UR}}] = 1 - \Pr[\Phi'_{\mathsf{ER}}]$.

Proof of Lemma 9. Both Φ_{UR} and Φ'_{ER} have random quantified variables in the identical order in the prefix. According to the definition of SSAT formulas,

$$\Pr[\Phi_{\mathsf{UR}}] = \min_{a_1,\ldots,a_m} \Pr[\phi_{\widehat{Y}}] \text{ and } \Pr[\Phi'_{\mathsf{ER}}] = \max_{a_1,\ldots,a_m} \Pr[\neg\phi_{\widehat{Y}}],$$

where $\Pr[\phi_{\widehat{Y}}]$ and $\Pr[\neg\phi_{\widehat{Y}}]$ are both computed for the random values of non-sensitive features $\mathbf{X}$.

Therefore, we derive the following duality between ER-SSAT and UR-SSAT,

$$\begin{aligned}
\Pr[\Phi'_{\mathsf{ER}}] &= \max_{a_1,\ldots,a_m} \Pr[\neg\phi_{\widehat{Y}}] \\
&= \min_{a_1,\ldots,a_m} (1 - \Pr[\phi_{\widehat{Y}}]) \\
&= 1 - \min_{a_1,\ldots,a_m} \Pr[\phi_{\widehat{Y}}] \\
&= 1 - \Pr[\Phi_{\mathsf{UR}}].
\end{aligned}$$

□

As we solve Φ'_{ER}, we obtain the optimal assignment to the sensitive features $\mathbf{a}_{\min} \triangleq [a_1^{min}, \ldots, a_m^{min}]$ that maximizes Φ'_{ER}. If p is the maximum satisfying probability of Φ'_{ER}, then according to Lemma 9, $1-p$ is the minimum satisfying probability of Φ_{UR}; which is also the minimum probability of positive prediction of the classifier. We present the algorithm for the inference approach, namely Justicia_infer in Algorithm 5 (Line 10–16).

In the ER-SSAT formula in Eq. (5.3), we need to negate the classifier $\phi_{\widehat{Y}}$ to another CNF formula $\neg\phi_{\widehat{Y}}$. The naïve approach of negating one CNF to another CNF generates an exponential number of new clauses. Here, we apply Tseitin transformation for the negation, which increases the number of clauses linearly while introducing a linear number of new variables [191]. As an alternative approach, we directly encode the binary classifier $\mathcal{M}$ for the negative class label $\widehat{Y}=0$ as a CNF formula and pass it to Φ'_{ER}, whenever it is possible. The latter approach is generally more efficient than the former approach as the resulting CNF is often smaller.

Example 5.1.2 (ER-SSAT encoding)**.** Here, we illustrate the ER-SSAT encoding for inferring the most favored and the least favored sensitive group of a classifier in the presence of compound sensitive groups. As the example in Figure 5.1 is degenerate for this purpose, we introduce another Boolean sensitive feature 'sex' $\in$ {male, female}. We consider a Boolean variable S for sex where the literal S denotes sex = male. With this new sensitive feature, let the classifier be $\mathcal{M} \triangleq (\neg F \vee I \vee S) \wedge (F \vee J)$, where variables corresponding to sensitive features F, I, and J have same distributions as discussed in Example 5.1.1. Hence, we obtain the following ER-SSAT formula of $\mathcal{M}$ to infer the most favored sensitive group:

$$\Phi_{\mathsf{ER}} = \exists S, \exists A, \mathfrak{U}^{0.41} F, \mathfrak{U}^{0.93} I, \mathfrak{U}^{0.09} J,\ (\neg F \vee I \vee S) \wedge (F \vee J).$$

As we solve Φ_{ER}, we infer that the optimal assignment to the existential variables $\sigma(S)=1, \sigma(A)=0$, which implies that 'male individuals with age <40' is the most favored group with probability of positive prediction computed as $\Pr[\Phi_{\mathsf{ER}}]=0.46$. Similarly, to infer the least favored group, we negate the CNF translation of the classifier $\mathcal{M}$ to obtain the following ER-SSAT formula:

$$\Phi'_{\mathsf{ER}} = \exists S, \exists A, \mathfrak{U}^{0.41} F, \mathfrak{U}^{0.93} I, \mathfrak{U}^{0.09} J,\ \neg((\neg F \vee I \vee S) \wedge (F \vee J)).$$

Solving Φ'_{ER}, we learn the optimal assignment $\sigma(S)=0, \sigma(A)=0$ and $\Pr[\Phi'_{\mathsf{ER}}]=0.57$. Thus, 'female individuals with age <40' constitute the least favored group with probability of positive prediction as $1-0.57=0.43$. Thus, Justicia_infer allows us to infer the most and least favored sensitive groups and the corresponding discrimination.

We next prove the equivalence of Justicia_enum and Justicia_infer in Lemma 10.

Lemma 10. *Let $\Phi_{\mathbf{a}}$ be an RE-SSAT formula for computing the probability of positive prediction of a classifier corresponding to the sensitive group $\mathbf{A} = \mathbf{a}$. Additionally, for the same classifer, let Φ_{ER} be an ER-SSAT formula for inferring the most favored sensitive group and Φ_{UR} be a UR-SSAT formula for inferring the least favored sensitive group. Therefore,* $\max_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}] = \Pr[\Phi_{\mathsf{ER}}]$ *and* $\min_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}] = \Pr[\Phi_{\mathsf{UR}}]$.

Proof of Lemma 10. It is trivial that the probability of positive prediction of the classifier for the most favored group $\mathbf{a}_{\max}$ is the maximum computed probability of all compound groups $\mathbf{a} \in \mathbf{A}$. Similar argument holds for the least favored group $\mathbf{a}_{\min}$, which obtains the minimum probability of positive prediction of the classifier among all compound groups $\mathbf{a} \in A$.

By construction of the SSAT formulas, $\Pr[\Phi_{\mathsf{ER}}]$ and $\Pr[\Phi_{\mathsf{UR}}]$ are the probabilities corresponding to the groups $\mathbf{A} = \mathbf{a}_{\max}$ and $\mathbf{A} = \mathbf{a}_{\min}$, respectively. Now, since $\Pr[\Phi_{\mathbf{a}}]$ is the probability for the group $\mathbf{A} = \mathbf{a}$, we derive the following.

$$\max_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}] = \Pr[\Phi_{\mathsf{ER}}] \text{ and } \min_{\mathbf{a}\in\mathbf{A}} \Pr[\Phi_{\mathbf{a}}] = \Pr[\Phi_{\mathsf{UR}}]$$

□

5.1.3 Practical Settings

We now relax the assumptions of Justicia on an access to Boolean classifiers and Boolean features, and extend Justicia to verify fairness metrics for more practical settings of decision trees, linear classifiers, and continuous features.

Extending Justicia to Decision Trees and Linear Classifiers.

In the SSAT approach, we assume that the classifier $\mathcal{M}$ is represented as a CNF formula. We extend Justicia beyond CNF classifiers to decision trees and linear classifiers, which are widely used in the fairness studies [55, 71, 192].

Binary decision trees are trivially encoded as CNF formulas. In the binary decision tree, each node in the tree is considered as a literal. A *path from the root to the leaf* is a conjunction of literals and thus, a *clause*. The *tree* itself is a disjunction of all paths and thus, a *DNF (Disjunctive Normal Form)*. In order to derive a CNF of a decision tree, we first construct a DNF by including all paths terminating at leaves with negative class label ($\widehat{Y} = 0$) and then complement the DNF to CNF using De Morgan's rule.

Linear classifiers on Boolean features are encoded into CNF formulas using pseudo-Boolean encoding [152]. We consider a linear classifier $\mathbf{W}\mathbf{X} + b \geq 0$ on Boolean features $\mathbf{X}$ with weights $\mathbf{W} \in \mathbb{R}^{|\mathbf{X}|}$ and bias $b \in \mathbb{R}$. We first normalize $\mathbf{W}$ and b in $[-1, 1]$ and then round to integers so that the decision boundary becomes a pseudo-Boolean constraint [193]. Then we apply pseudo-Boolean constraints to CNF translation [152] to encode the decision boundary to CNF. This encoding usually introduces additional

Boolean variables and results in a large CNF. In order to generate a smaller CNF, we can trivially apply thresholding on the weights to consider features with higher weights only. For instance, if the weight $|W_i| \leq \lambda$ for a threshold $\lambda \in \mathbb{R}^+$ and $W_i \in \mathbf{W}$, we can set $W_i = 0$. Thus, features with lower weights (less important) do not appear in the encoded CNF. Moreover, all introduced variables in this CNF translation are given existential ($\exists$) quantification and they appear in the inner-most position in the prefix of the SSAT formula. Thus, the presented ER-SSAT formulas become effectively ERE-SSAT formulas.

Extending to Continuous Features.

In practical problems, features are generally real-valued or categorical but classifiers, which are naturally expressed as CNF such as [194], are generally trained on a Boolean abstraction of input features. In order to perform the Boolean abstraction, each categorical feature is one-hot encoded and each real-valued feature is discretized into a set of Boolean features [168, 194].

For a binary decision tree, each feature, including the continuous ones, is compared against a constant at each node (except leaves) of the tree. We assign a Boolean variable to each internal node of the tree (except leaves), where the $\{0, 1\}$ assignment to the variable decides one of the two branches to choose from the current node.

Linear classifiers are generally trained on continuous features, where we apply discretization in the following way. Let us consider a continuous feature X_c, where W is its weight during training. We discretize X_c to a set $\mathbf{B}$ of Boolean features and recalculate the weight of each variable in $\mathbf{B}$ based on W. We consider the an interval-based approach for discretizing X_c. For each interval in the continuous space of X_c, we consider a Boolean variable $B_i \in \mathbf{B}$, such that B_i is assigned 1 when the feature-value of X_c lies within the i^{th} interval and B_i is assigned 0 otherwise. Following that, we assign the weight of B_i to be $\mu_i W$, where μ_i is the mean of feature values in the i^{th} interval. We can show that if we consider infinite number of intervals, $X_c \approx \sum_i \mu_i B_i$.

5.2 Empirical Performance Analysis

In this section, we discuss the empirical studies to evaluate the performance of Justicia in verifying different fairness metrics and algorithms. We first discuss the experimental setup and the objective of the experiments and then evaluate the experimental results.

5.2.1 Experimental Setup

We have implemented a prototype of Justicia in Python (version 3.7.3). The core computation of Justicia relies on solving SSAT formulas using an off-the-shelf SSAT solver. To this end, we employ the state of the art RE-SSAT solver of [78] and the ER-SSAT solver of [79]. Both solvers output the exact satisfying probability of the SSAT formula.

For comparative evaluation of Justicia, we have experimented with two state-of-the-art probabilistic fairness verifiers FairSquare and VeriFair, and a sample-based fairness measuring tool: AIF360. In the experiments, we have studied three type of classifiers: decision tree, logistic regression classifier, and CNF learner. Decision tree and logistic regression are implemented using scikit-learn module of Python [181] and we use the MaxSAT-based CNF learner, namely IMLI [42]. We have used the PySAT library [195] for encoding the decision function of the logistic regression classifier into a CNF formula. In our experiments, we have verified two fairness-enhancing algorithms: reweighing algorithm [54] and optimized pre-processing algorithm [56]. We have experimented on multiple datasets containing multiple sensitive features: the UCI[3] Adult and German-credit dataset, ProPublica's COMPAS recidivism dataset [196], Ricci dataset [197], and Titanic dataset[4].

Our empirical studies have following objectives:

1. How accurate and scalable Justicia is with respect to existing fairness verifiers: FairSquare and VeriFair?
2. Can Justicia verify the effectiveness of different fairness-enhancing algorithms on different datasets?
3. Can Justicia verify fairness in the presence of compound sensitive groups?
4. How robust is Justicia in comparison to sample-based tools such as AIF360 for varying sample sizes?
5. How do the computational efficiencies of Justicia_infer and Justicia_enum compare?

Our experimental studies validate that Justicia is more accurate and scalable than the state-of-the-art verifiers: FairSquare and VeriFair. Justicia is able to verify the effectiveness of different fairness-enhancing algorithms for multiple fairness metrics and datasets. Justicia achieves scalable performance in the presence of compound sensitive groups that the existing verifiers cannot handle. Justicia is also more robust than the sample-based tools such as AIF360. Finally, Justicia_infer is significantly efficient in terms of runtime than Justicia_enum.

5.2.2 Experimental Analysis

Accuracy: Less Than 1%-error

In order to assess the accuracy of different verifiers, we have considered the decision tree in Figure 5.1 for which the fairness metrics are analytically computable. In Table 5.1, we show the computed fairness metrics by Justicia, FairSquare, VeriFair, and AIF360. We observe that Justicia and AIF360 yield more accurate estimates of disparate impact and

[3] http://archive.ics.uci.edu/ml
[4] https://www.kaggle.com/c/titanic

Table 5.1: Results on synthetic benchmark. '—' refers that the verifier cannot compute the metric. The number in bold denotes the best result, where the estimated fairness metric is closest to the exact value.

Metric	Exact	Justicia	FairSquare	VeriFair	AIF360
Disparate impact	0.26	**0.25**	0.99	0.99	**0.25**
Statistical parity	0.53	**0.54**	—	—	**0.54**

Table 5.2: Scalability of different verifiers in terms of execution time (in seconds). The number in bold refers to the best result incurring minimum execution time among competitive verifiers. '—' refers to timeout.

Classifier	Dataset	FairSquare	VeriFair	Justicia
Decision Tree	Ricci	4.8	5.3	**0.1**
	Titanic	16	1.2	**0.1**
	COMPAS	36.9	15.9	**0.1**
	Adult	—	295.6	**0.2**
Logistic Regression	Ricci	—	2.2	**0.2**
	Titanic	—	**0.8**	0.9
	COMPAS	—	11.3	**0.2**
	Adult	—	61.1	**1.0**

statistical compared against the ground truth values of fairness metrics with less than 1% error. In contrast, FairSquare and VeriFair estimate disparate impact to be 0.99 and thus, being unable to verify the fairness violation. Therefore, Justicia is significantly more accurate than the existing formal verifiers: FairSquare and VeriFair.

Scalability: 1 to 3 Orders of Magnitude Speed-up

We have tested the scalability of Justicia, FairSquare, and VeriFair on practical benchmarks with a timeout of 900 seconds and reported the execution time of these verifiers on decision tree and logistic regression in Table 5.2. We observe that Justicia shows impressive scalability than the competing verifiers. Particularly, among benchmarks where all three verifiers output results, Justicia is 1 to 2 orders of magnitude faster than FairSquare and 1 to 3 orders of magnitude faster than VeriFair. Additionally, FairSquare times out in most benchmarks. Thus, Justicia is not only accurate but also scalable than the existing verifiers.

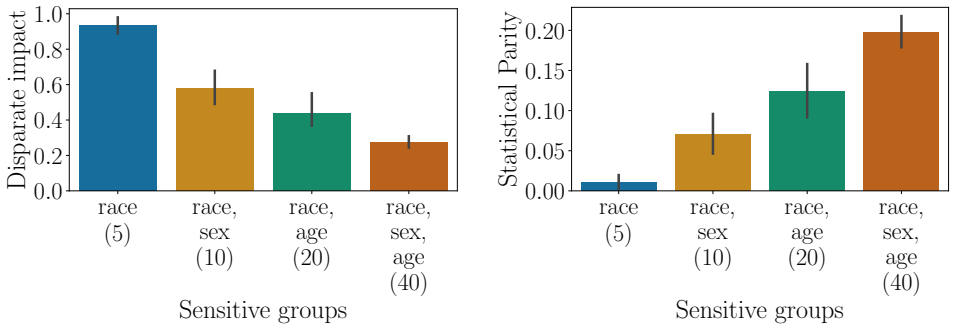


Figure 5.2: Fairness metrics measured by Justicia for different sensitive groups in the Adult dataset. The number within parenthesis in the xticks denotes total compound sensitive groups, which increases due to the increasing combination of sensitive features. For higher sensitive groups, fairness becomes worse.

Verification: Detecting Compounded Discrimination in Sensitive Groups.

We have tested Justicia for datasets consisting of multiple sensitive features and reported results in Figure 5.2. Justicia operates on datasets with even 40 compound sensitive groups and can potentially scale more than that while the state-of-the-art fairness verifiers (e.g., FairSquare and VeriFair) consider a single sensitive feature with two sensitive groups. Thus, Justicia removes an important limitation in practical fairness verification, which was previously restricted to Boolean sensitive groups. Additionally, in most datasets, we observe that disparate impact decreases and thus, discrimination increases as more compound sensitive groups are considered. For instance, when we increase the total groups from 5 to 40 in the Adult dataset, disparate impact decreases from around 0.9 to 0.3, thereby detecting higher discrimination. Thus, Justicia detects that the marginalized individuals of a specific type (e.g., 'race') are even more discriminated and marginalized when they also belong to a marginalized group of another type (e.g., 'sex').

Verification: Fairness of Algorithms on Datasets

We have experimented with two fairness-enhancing algorithms: reweighing (RW) algorithm and optimized-preprocessing (OP) algorithm. Both of them pre-process to remove statistical bias from the dataset. We study the effectiveness of these algorithms using Justicia on different5 datasets each with two different sensitive features. In Table 5.3, we report different fairness metrics on logistic regression and decision tree. We observe that Justicia verifies fairness improvement as the bias mitigating algorithms are applied. For example, for the Adult dataset with 'race' as the sensitive feature, disparate impact increases from 0.23 to 0.85 for applying the reweighing algorithm on logistic regression classifier. In addition, statistical parity decreases from 0.09 to 0.01, and equalized odds decreases from 0.13 to 0.03, thereby showing the effectiveness of reweighing algorithm in all three fairness

Table 5.3: Verification of different fairness enhancing algorithms for multiple datasets and classifiers using Justicia. Numbers in bold refer to fairness improvement compared against the unprocessed (orig.) dataset. RW and OP refer to reweighing and optimized-preprocessing algorithm respectively.

Classifier	Dataset →	Adult						COMPAS					
	Sensitive →	Race			Sex			Race			Sex		
	Algorithm →	orig.	RW	OP	orig.	RW	OP	orig.	RW	OP	orig.	RW	OP
Logistic regression	Disparte impact	0.23	**0.85**	**0.59**	0.03	**0.61**	**0.62**	0.34	**0.36**	**0.47**	0.48	**0.80**	**0.74**
	Statistical parity	0.09	**0.01**	**0.05**	0.16	**0.04**	**0.03**	0.39	**0.33**	**0.21**	0.23	**0.09**	**0.10**
	Equalized odds	0.13	**0.03**	**0.10**	0.30	**0.02**	**0.06**	0.38	**0.33**	**0.18**	0.17	0.19	**0.07**
Decision tree	Disparte impact	0.82	0.60	0.67	0.00	**0.73**	**0.95**	0.61	0.58	0.57	0.94	0.78	0.63
	Statistical parity	0.02	0.05	0.04	0.14	**0.05**	**0.01**	0.18	**0.17**	**0.17**	0.02	0.09	0.18
	Equalized odds	0.07	**0.05**	**0.03**	0.47	**0.03**	**0.04**	0.17	**0.16**	**0.16**	0.07	**0.05**	0.16

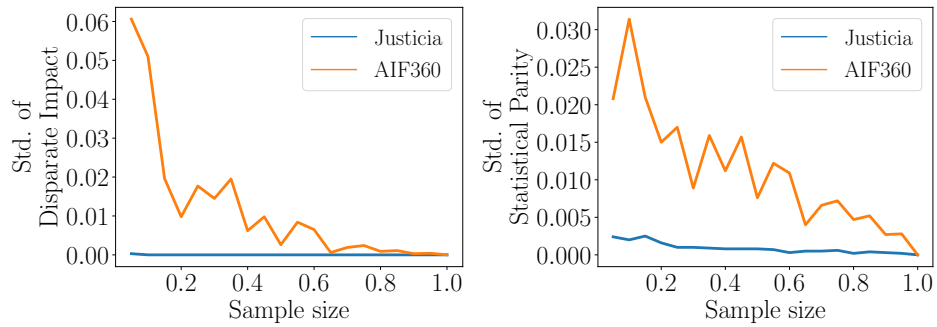


Figure 5.3: Standard deviation in estimation of disparate impact (DI) and stat. parity (SP) for different sample sizes (sample size = 1 refers to the entire dataset). Justicia is more robust with variation of sample size than AIF360. Sample size = 1 denotes the full dataset considering all samples.

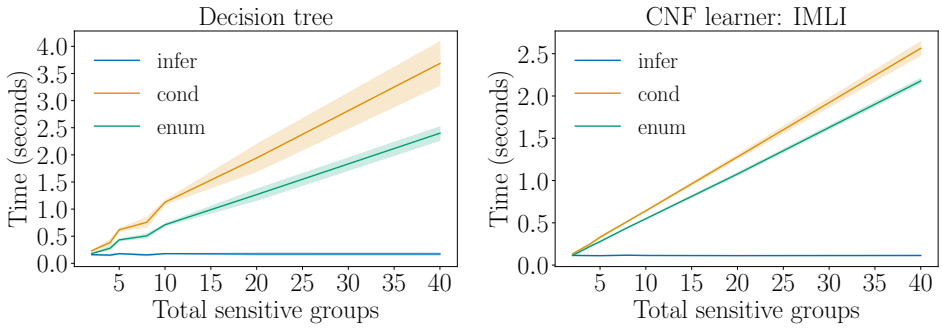


Figure 5.4: Runtime comparison of different encodings while varying total sensitive groups in the Adult dataset.

metrics. Justicia also finds instances where the fairness algorithms fail, specially when considering the decision tree classifier. Thus, Justicia verifies the effectiveness of different fairness enhancing algorithms.

Robustness: Stability to Sample Size.

We have compared the robustness of our probabilistic fairness verifier Justicia with dataset-centric verifier AIF360 by varying the sample-size and reporting the standard deviation of different fairness metrics. In Figure 5.3, AIF360 shows higher standard deviation for lower sample-size and the value decreases as the sample-size increases. In contrast, Justicia shows significantly lower ($\sim 10\times$ to $100\times$) standard deviation for different sample-sizes. The reason is that AIF360 empirically measures on a fixed dataset whereas Justicia provides estimates over the distribution. Thus, Justicia is more robust than the sample-based verifier AIF360.

Comparative Evaluation of Different Encodings.

Both Justicia_enum and Justicia_infer have the same output according to Lemma 10. However, Justicia_infer improves exponentially in runtime than Justicia_enum on both decision tree and Boolean CNF classifiers as we vary the total compound groups in Figure 5.4. Justicia_cond (Justicia_enum encoding where we consider conditional probabilities of non-sensitive features w.r.t. sensitive groups) also has an exponential trend in runtime similar to Justicia_enum. This analysis justifies that the naïve enumeration-based approach cannot verify large-scale fairness problems containing multiple sensitive features, and Justicia_infer is a more efficient approach for practical use.

5.3 Chapter Summary

Formal verification of different fairness metrics of machine learning for different datasets is an important question. Existing fairness verifiers, however, are not scalable, accurate, and extendable to non-Boolean sensitive features. We propose a stochastic SAT-based approach, Justicia, that formally verifies multiple group and causal fairness metrics for different classifiers and distributions for compound sensitive groups. Experimental evaluations demonstrate that Justicia achieves *higher accuracy* and *scalability* in comparison to the state-of-the-art verifiers, FairSquare and VeriFair, while yielding *higher robustness* than the sample-based tools, such as AIF360.

Our work opens up several new directions of research. One direction is to develop SSAT models and verifiers for popular classifiers like Deep networks and SVMs. Other direction is to develop SSAT solvers that can accommodate continuous variables and conditional probabilities by design.

Chapter 6

Algorithmic Fairness Verification with Graphical Models

The significant improvement of machine learning over the decades has led to a host of applications of machine learning in high-stake decision-making such as college admission [44], hiring of employees [46], and recidivism prediction [45, 67]. Machine learning algorithms often have an accuracy-centric learning objective, which may cause them to be biased towards certain part of the dataset belonging to a certain economically or socially sensitive groups [48–50]. The following example illustrates a plausible case of unfairness of machine learning.

Example 6.0.1. Following Example 5.0.1, let us consider a machine learning classification where the classifier decides the eligibility of an individual for health insurance given their income and fitness (Figure 6.1). Here, the sensitive feature 'age' (A) follows a Bernoulli distribution, and income (I) and fitness (F) follow Gaussian distributions. We generate 1000 samples from these distributions and use them to train a Support Vector Machine (SVM) classifier. The decision boundary of this classifier is $9.37I + 9.75F - 0.34A \geq 9.4$, where $A = 1$ denotes the sensitive group 'age ≥ 40'. This classifier selects an individual above and below 40 years of age with probabilities 0.24 and 0.86, respectively. This illustrates a disparate treatment of individuals of two age groups by the SVM classifier.

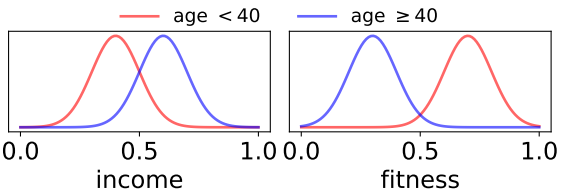


Figure 6.1: Age-dependent distributions of income and fitness in Example 6.0.1.

In order to identify and mitigate the bias of classifiers, different fairness definitions and fairness algorithms have been proposed [59, 183, 184]. In this chapter, we focus on two

families of fairness definitions: *group* and *causal* fairness. Group fairness metrics, such as disparate impact and equalized odds constrain the probability of the positive prediction of the classifier to be (almost) equal among different sensitive groups [47, 52]. On the other hand, causal fairness metrics assess the difference in positive predictions if every feature in the causal relation remains identical except the sensitive feature [70, 84]. The early works on *fairness verification* focused on measuring fairness metrics of a classifier for a given dataset [61]. Naturally, such techniques were limited in enhancing confidence of users for wide deployment. Consequently, recent verifiers seek to achieve verification beyond finite dataset and in turn focus on the probability distribution of features [62, 63, 198]. More specifically, the input to the verifier is a classifier and the probability distribution of features, and the output is an estimate of fairness metrics that the classifier obtains given the distribution. For Example 6.0.1, a fairness verifier takes the SVM classifier and the distribution of features I, F, A as an input and outputs the probability of positive prediction of the classifier for different sensitive groups.

In order to solve the fairness-verification problem, existing works have proposed two principled approaches. Firstly, [198] and [62] propose formal methods that reduce the problem into a solution of an SSAT or an SMT formula respectively. Secondly, [63] propose a sampling approach that relies on extensively enumerating the conditional probabilities of prediction given different sensitive features and thus, incurs high computational cost. Additionally, existing works assume feature independence of non-sensitive features and consider correlated features within a limited scope, such as conditional probabilities of non-sensitive features w.r.t. sensitive features and ignore correlations among non-sensitive features. As a result, the *scalability* and *accuracy* of existing verifiers remains a major challenge.

In this work, we seek to remedy the aforementioned situation. As a first step, we focus on *linear classifiers*, which has attracted significant attention from researchers in the context of fair algorithms [65–68]. At this point, it is worth highlighting that our empirical evaluation demonstrates that the existing techniques fail to scale beyond small examples or provide highly inaccurate estimates for comparatively *small* linear classifiers.

Our Contributions. The contributions of this chapter are summarized below.

- *Framework:* In this chapter, we propose a fairness verification framework, namely FVGM (**F**airness **V**erification with **G**raphical **M**odels), for accurately and efficiently verifying linear classifiers.
- *Scalability:* FVGM proposes a novel *stochastic subset-sum* encoding for linear classifiers with an efficient pseudo-polynomial solution using dynamic programming.
- *Accuracy:* To address feature-correlations, FVGM considers a graphical model, particularly a Bayesian Network that represents conditional dependence (and independence) among features in the form of a Directed Acyclic Graph (DAG).

- *Experimental Results:* Experimentally, FVGM is more accurate and scalable than existing fairness verifiers; FVGM can verify group and causal fairness metrics for multiple fairness algorithms.

- *Applications:* We also demonstrate two novel applications of FVGM as a fairness verifier: (a) detecting fairness attacks, and (b) computing Fairness Influence Functions (FIF) of features as a mean of identifying (un)fairness contribution of a subset of features.

6.1 FVGM: Fairness Verification with Graphical Models

In this section, we present FVGM, a fairness verification framework for linear classifiers that accounts for correlated features represented as a graphical model. The core idea of verifying fairness of a classifier is to compute the probability of positive prediction of the classifier with respect to all compound sensitive groups. To this end, FVGM solves a stochastic subset sum problem, S3P, that is equivalent to computing the probability of positive prediction of the classifier for the most and the least favored sensitive group[1]. In this section, we first define S3P and present an efficient dynamic programming solution for S3P. We then extend S3P to consider correlated features as input. Finally, we conclude by discussing fairness verification based on the solution of S3P.

Problem Formulation. Given a linear classifier $\mathcal{M} : (\mathbf{X}, \mathbf{A}) \rightarrow \widehat{Y}$ and a probability distribution $\mathcal{D}$ of $\mathbf{X} \cup \mathbf{A}$, our objective is to compute $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ and $\min_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ with respect to $\mathcal{D}$. In this study, we express a linear classifier $\mathcal{M}$ as

$$\widehat{Y} = \mathbb{1}\Big[\sum_i W_{X_i} X_i + \sum_j W_{A_j} A_j \geq \tau\Big].$$

Here, W denotes the weight (or coefficients) of a feature, τ denotes the bias or the offset parameter of the classifier, and $\mathbb{1}$ is an indicator function. Hence, the prediction $\widehat{Y} = 1$ if and only if the inner inequality holds. Thus, computing the maximum (resp. minimum) probability of positive prediction is equivalent to finding out the assignment of A_j's for which the probability of satisfying the inner inequality is highest (resp. lowest). We reduce this computation into an instance of S3P. To perform this reduction, we assume weight W and bias τ as integers, and features $\mathbf{X} \cup \mathbf{A}$ as Boolean. In Sec. 6.1.5, we relax these assumptions and extend to the practical settings.

[1]The most (resp. least) favored sensitive group obtains the maximum (resp. minimum) probability of positive prediction of the classifier.

6.1.1 S3P: Stochastic Subset Sum Problem

Now, we formally describe the specification and semantics of S3P. S3P operates on a set of Boolean variables $\mathbf{B} = \{B_i\}_{i=1}^n \in \{0,1\}^n$, where $W_i \in \mathbb{Z}$ is the weight of B_i, and $n \triangleq |\mathbf{B}|$. Given a constant threshold $\tau \in \mathbb{Z}$, S3P computes the *probability* of a subset of $\mathbf{B}$ with sum of weights of non-zero variables to be at least τ. Formally,

$$S(\mathbf{B}, \tau) \triangleq \Pr\Big[\sum_i W_i B_i \geq \tau\Big].$$

Aligning with terminologies in stochastic satisfiability (SSAT) [64], we categorize the variables $\mathbf{B}$ into two types: (i) *chance variables* that are stochastic and have an associated probability of being assigned to 1 and (ii) *choice variables* that we optimize while computing $S(\mathbf{B}, \tau)$. To specify the category of variables, we consider a *quantifier* $Q_i \in \{\text{Ɐ}^{p_i}, \exists, \forall\}$ for each B_i. Elaborately, Ɐ^p is a *random quantifier* corresponding to a chance variable $B \in \mathbf{B}$, where $p \triangleq \Pr[B = 1]$. In contrast, $\exists$ is an *existential quantifier* corresponding to a choice variable B such that a Boolean assignment of B *maximizes* $S(\mathbf{B}, \tau)$. Finally, $\forall$ is an *universal quantifier* for a choice variable B that fixes an assignment to B that *minimizes* $S(\mathbf{B}, \tau)$.

Now, we formally present the semantics of $S(\mathbf{B}, \tau)$ provided that each variable B_i has weight W_i and quantifier Q_i. Let $\mathbf{B}[2:n] \triangleq \{B_j\}_{j=2}^n$ be the subset of $\mathbf{B}$ without the first variable B_1. Then $S(\mathbf{B}, \tau)$ is recursively defined as:

$$S(\mathbf{B}, \tau) = \begin{cases} \mathbb{1}[\tau \leq 0], \text{ if } \mathbf{B} = \emptyset \\ S(\mathbf{B}[2:n], \tau - \max\{W_1, 0\}), \text{ if } Q_1 = \exists \\ S(\mathbf{B}[2:n], \tau - \min\{W_1, 0\}), \text{ if } Q_1 = \forall \\ p_1 \times S(\mathbf{B}[2:n], \tau - W_1) + \\ (1 - p_1) \times S(\mathbf{B}[2:n], \tau), \text{ if } Q_1 = \text{Ɐ}^{p_1} \end{cases} \tag{6.1}$$

Observe that when $\mathbf{B}$ is empty, S is computed as 1 if $\tau \leq 0$, and $S = 0$ otherwise. For existential and universal quantifiers, we compute S based on the weight. Specifically, if $Q_1 = \exists$, we decrement the threshold τ by the maximum between W_1 and 0. For example, if $W_1 > 0$, B_1 is assigned 1, and assigned 0 otherwise. Therefore, by solving for an existential variable, we maximize S. In contrast, when if $Q_1 = \forall$, we fix an assignment of B_1 that minimizes S by choosing between the minimum of W_1 and 0. Finally, for random quantifiers, we decompose the computation of S into two sub-problems: one sub-problem where $B_1 = 1$ and the updated threshold becomes $\tau - W_1$ and another sub-problem where $B_1 = 0$ and the updated threshold remains the same. Herein, we compute S as the expected output of both sub-problems.

Remark. *$S(\mathbf{B}, \tau)$ does not depend on the order of $\mathbf{B}$.*

Computing Minimum and Maximum probability of positive prediction of Linear Classifiers Using S3P. For computing $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$ of a linear classifier, we set existential quantifiers $\exists$ to sensitive features A_j, randomized quantifiers $\mathfrak{R}$ to non-sensitive features X_i and construct a set $\mathbf{B} = \mathbf{A} \cup \mathbf{X}$. The coefficients W_{A_j} and W_{X_i} of the classifier become weights of $\mathbf{B}$. Also, we get $n = m_1 + m_2$. For non-sensitive variables X_i, which are chance variables, we derive their marginal probability $p_i = \Pr[X_i = 1]$ from the distribution $\mathcal{D}$. According to semantic of S3P, setting $\exists$ quantifiers on $\mathbf{A}$ computes the maximum value of $S(\mathbf{B}, \tau)$ that equalizes the maximum probability of positive prediction of the classifier. In this case, the *inferred* assignment of $\mathbf{A}$ implies the most favored group $\mathbf{a}_{\max} = \arg\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$. In contrast, to compute the minimum probability of positive prediction, we instead assign each variable A_j a universal quantifier while keeping random quantifiers over X_i, and infer the least favored group $\mathbf{a}_{\min} = \arg\min_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$.

6.1.2 A Dynamic Programming Solution for S3P

We propose a dynamic programming approach [199, 200] to solve S3P as the problem has overlapping sub-problem properties. For example, $S(\mathbf{B}, \tau)$ can be solved by solving $S(\mathbf{B}[2:n], \tau')$, where the updated threshold τ', called the *residual threshold*, depends on the original threshold τ and the assignment of B_1 as shown in Eq. (6.1). Building on this observation, we propose the recursion and terminating condition leading to our dynamic programming algorithm.

Recursion. We consider a function $\mathsf{dp}(i, \tau)$ that solves the sub-problem $S(\mathbf{B}[i:n], \tau)$, for $i \in \{1, \ldots, n\}$. The semantics of $S(\mathbf{B}, \tau)$ in Eq. (6.1) induces the recursive definition of $\mathsf{dp}(i, \tau)$ as:

$$\mathsf{dp}(i,\tau) = \begin{cases} \mathsf{dp}(i+1, \tau - \max\{W_i, 0\}), \text{ if } Q_i = \exists \\ \mathsf{dp}(i+1, \tau - \min\{W_i, 0\}), \text{ if } Q_i = \forall \\ p_i \times \mathsf{dp}(i+1, \tau - W_i) + \\ (1-p_i) \times \mathsf{dp}(i+1, \tau), \text{ if } Q_i = \mathfrak{R}^{p_i} \end{cases} \tag{6.2}$$

Eq. (6.2) shows that $S(\mathbf{B}, \tau)$ can be solved by instantiating $\mathbf{dp}(1, \tau)$, which includes all the variables in $\mathbf{B}$.

Terminating Condition. Let W_{neg}, W_{pos}, and W_{all} be the sum of negative, positive, and all weights of $\mathbf{B}$, respectively. We observe that $W_{neg} \leq W_{all} \leq W_{pos}$. Thus, for any i, if the residual threshold $\tau \leq W_{neg}$, there is always a subset of $\mathbf{B}[i:n]$ with sum of weights at least τ. Conversely, when $\tau > W_{pos}$, there is no subset of $\mathbf{B}[i:n]$ with sum of weights at least τ. We leverage this bound and tighten the terminating conditions of $\mathsf{dp}(i, \tau)$ in Eq. (6.3).

$$\mathsf{dp}(i,\tau) = \begin{cases} 1 \text{ if } \tau \leq W_{neg} \\ 0 \text{ if } \tau > W_{pos} \\ \mathbb{1}[\tau \leq 0] \text{ if } i = n+1 \end{cases} \tag{6.3}$$

Eq. (6.2) and (6.3) together define our dynamic programming algorithm. While deploying the algorithm, we store $\mathsf{dp}(i,\tau)$ in memory to avoid repetitive computations. This allows us to achieve a pseudo-polynomial algorithm (Lemma 11) instead of a naïve exponential algorithm enumerating all possible assignments. In particular, the time complexity is pseudo-polynomial for chance (random) variables and linear for choice (existential and universal) variables.

Lemma 11. *Let n' be the number of existential and universal variables in* $\mathbf{B}$*. Let*

$$W_\exists = \sum_{B_i \in \mathbf{B} | Q_i = \exists} \max\{W_i, 0\} \text{ and } W_\forall = \sum_{B_i \in \mathbf{B} | Q_i = \forall} \min\{W_i, 0\}$$

be the considered sum of weights of existential and universal variables, respectively. We can exactly solve S3P *using dynamic programming with time complexity $\mathcal{O}((n-n')(\tau + |W_{neg}| - W_\exists - W_\forall) + n')$. The space complexity is $\mathcal{O}((n-n')(\tau + |W_{neg}| - W_\exists - W_\forall))$.*

Proof. **Case 1: All n variables in B have randomized quantifiers.**

At step i of the dynamic programming (Eq. (6.2)), we modify the residual threshold of that step, namely τ_i, either by subtracting W_i or by retaining it. Now, we observe that the residual threshold τ_i for any $i \in \{1, \ldots, n\}$ will be in $[0, \tau + |W_{neg}|]$. This holds because if τ_i crosses these bounds, the dynamic programming is terminated as shown in Equation (6.3). Since all weights of $\{W_i\}_{i=1}^n$ are integers, the maximum number of values that the residual threshold can take, is $\tau + |W_{neg}|$. Thus, we need to store at most $n(\tau + |W_{neg}|)$ values in the memory for performing dynamic programming with n variables and $(\tau + |W_{neg}|)$ number of possible weights. Thus, the space complexity is $\mathcal{O}(n(\tau + |W_{neg}|))$.

In order to construct the dynamic programming table, we have to call the dp function $\mathcal{O}(n(\tau + |W_{neg}|))$ times, in the worst-case. Thus, the time complexity of our method is $\mathcal{O}(n(\tau + |W_{neg}|)$.

Case 2: n' variables have existential and universal quantifiers and $n - n'$ variables have randomized quantifiers in B.

According to Eq. (6.2), $W_\exists$ and $W_\forall$ are the fixed weights of all existential and universal variables, respectively. Therefore, we need to consider at most $\tau + |W_{neg}| - W_\exists - W_\forall$ values of residual weights for random variables. By applying analysis in Case 1, the space and time complexity is derived as $\mathcal{O}((n-n')(\tau + |W_{neg}| - W_\exists - W_\forall))$.

We note that there is an additional time complexity of $\mathcal{O}(n')$ for existential and universal variables in Eq. (6.2). Thus the time complexity becomes $\mathcal{O}((n-n')(\tau + |W_{neg}| - W_\exists - W_\forall) + n')$. We, however, do not require to store any entry for existential and universal variables in dp function and thus, the space complexity remains the same as $\mathcal{O}((n-n')(\tau + |W_{neg}| - W_\exists - W_\forall))$. □

A Heuristic for Faster Computation. We propose two improvements for a faster computation of the dynamic programming solution. Firstly, we observe that in Eq. (6.2), existential/universal variables are deterministically assigned based on their weights. Hence, we reorder $\mathbf{B}$ such that existential/universal variables appear earlier in $\mathbf{B}$ than random variables. This allows us to avoid unnecessary repeated exploration of existential/universal variables in dp. Moreover, according to the remark in Section 6.1.1, reordering $\mathbf{B}$ still produces the same exact solution of $\mathsf{S3P}$. Secondly, to reach the terminating condition of $\mathsf{dp}(i, \tau)$ more frequently, we sort $\mathbf{B}$ based on their weights—more specifically, within each cluster of random, existential, and universal variables. In particular, if $\tau \leq 0.5(W_{pos} - W_{neg})$, τ is closer to W_{pos} than W_{neg}. Hence, we sort each cluster in descending order of weights. Otherwise, we sort in ascending order. We illustrate our dynamic programming approach in Example 6.1.1.

Example 6.1.1. We consider a linear classifier $P + Q + R - S \geq 2$. Herein, P is a Boolean sensitive feature, and Q, R, S are Boolean non-sensitive features with $\Pr[Q] = 0.4, \Pr[R] = 0.5$, and $\Pr[S] = 0.3$. To compute the maximum probability of positive prediction of the classifier, we impose an existential quantifier on P and randomized quantifiers on others. This leads us to the observation that $P = 1$ is the optimal assignment as $W_P = 1 > 0$. We now require to compute $\Pr[Q + R - S \geq 1]$, which by dynamic programming, is computed as 0.55. The solution is visualized as a search tree in Figure 6.2a, where we observe that storing the solution of sub-problems in the memory avoids repetitive computation, such as exploring the node $(4, 0)$. Similarly, the minimum probability of positive prediction of the classifier is 0.14 (not shown in Figure 6.2a) where we impose a universal quantifier on P to obtain $P = 0$ as the optimal assignment.

6.1.3 S3P with Correlated Variables

In $\mathsf{S3P}$ presented in Section 6.1.1, we consider all Boolean variables to be probabilistically independent. This independence assumption often leads to an *inaccurate estimate* of the probability of positive prediction of the classifier because both sensitive and non-sensitive features can be correlated in practical fairness problems. Therefore, we extend $\mathsf{S3P}$ to include correlations among variables.

We consider a Bayesian network $\mathsf{BN} = (G, \theta)$ to represent correlated variables, where $G \triangleq (\mathbf{V}, \mathbf{E})$, $\mathbf{V} \subseteq \mathbf{B}$, $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$, and θ is the parameter of the network. In BN, we constrain that there is no conditional probability of choice (i.e., existential and universal) variables as we optimize their assignment in $\mathsf{S3P}$. Choice variables, however, can impose conditions on chance (i.e., random) variables. In practice, we achieve this by allowing no incoming edge on choice variables while learning BN (ref. Section 6.2).

For a chance variable $B_i \in \mathbf{V}$, let $\mathrm{Pa}(B_i)$ denote its parents. According to Eq. (2.2), for an assignment $\mathbf{u}$ of $\mathrm{Pa}(B_i)$, BN ensures B_i to be independent of other non-descendant variables in $\mathbf{V}$. Hence, in the recursion of Eq. (6.2), we substitute p_i with $\Pr[B_i =$

$1|\mathrm{Pa}(B_i) = \mathbf{u}]$. In order to explicate the dependence on $\mathbf{u}$, we denote the expected solution of $S(\mathbf{B}[i:n], \tau)$ as $\mathsf{dp}(i, \tau, \mathbf{u})$, which for $B_i \in \mathbf{V}$ is modified as follows:

$$\begin{aligned}\mathsf{dp}(i, \tau, \mathbf{u}) = &\Pr[B_i = 0|\mathrm{Pa}(B_i) = \mathbf{u}]\mathsf{dp}(i+1, \tau, \mathbf{u} \cup \{0\}) \\ &+ \Pr[B_i = 1|\mathrm{Pa}(B_i) = \mathbf{u}]\mathsf{dp}(i+1, \tau - W_i, \mathbf{u} \cup \{1\}).\end{aligned}$$

Since $\mathsf{dp}(i, \tau, \mathbf{u})$ involves $\mathbf{u}$, we initially perform a topological sort of $\mathbf{V}$ to enumerate the assignment of parents before computing dp on the child. Moreover, there are $2^{|\mathrm{Pa}(B_i)|}$ assignments of $\mathrm{Pa}(B_i)$, and we compute $\mathsf{dp}(i, \tau, \mathbf{u})$ for $\mathbf{u} \in \{0,1\}^{|\mathrm{Pa}(B_i)|}$ to incorporate all conditional probabilities into $\mathsf{S3P}$. For this enumeration, we do not store $\mathsf{dp}(i, \tau, \mathbf{u})$ in memory. However, for $B_i \notin \mathbf{V}$ that does not appear in the network, we instead compute $\mathsf{dp}(i, \tau)$ and store it in memory as in Section 6.1.2, because B_i is not correlated with other variables. Lemma 12 presents the complexity of solving $\mathsf{S3P}$ with correlated variables, wherein unlike Lemma 11, the complexity differentiates based on variables in $\mathbf{V}$ (exponential) and $\mathbf{B} \setminus \mathbf{V}$ (pseudo-polynomial).

Lemma 12. *Let $\mathbf{V} \subseteq \mathbf{B}$ be the set of vertices in the Bayesian network and n'' be the number of existential and universal variables in $\mathbf{B} \setminus \mathbf{V}$. Let*

$$w'_\exists = \sum_{B_i \in \mathbf{B}\setminus\mathbf{V} | Q_i = \exists} \max\{W_i, 0\} \text{ and } w'_\forall = \sum_{B_i \in \mathbf{B}\setminus\mathbf{V} | Q_i = \forall} \min\{W_i, 0\}$$

be the sum of considered weights of existential and universal variables, respectively that only appear in $\mathbf{B} \setminus \mathbf{V}$. To exactly compute $\mathsf{S3P}$ with correlated variables in the dynamic programming approach, time complexity is $\mathcal{O}(2^{|\mathbf{V}|} + (n - n'' - |\mathbf{V}|)(\tau + |W_{neg}| - w'_\exists - w'_\forall) + n'')$ and space complexity is $\mathcal{O}((n - n'' - |\mathbf{V}|)(\tau + |W_{neg}| - w'_\exists - w'_\forall))$.

Proof. We first separate analysis of space and time complexity for variables in $\mathbf{V}$ and in $\mathbf{B} \setminus \mathbf{V}$. For each Boolean variable in $\mathbf{V}$, we enumerate all assignments, which has time complexity of $2^{|\mathbf{V}|}$ and there is no space complexity as discussed in Section 6.1.3.

For variables in $\mathbf{B}\setminus\mathbf{V}$, we apply analysis from Lemma 11, where we consider $(n - n'' - |\mathbf{V}|)$ random variables, n'' existential/universal variables, and residual weights can take at most $(\tau + |W_{neg}| - w'_\exists - w'_\forall)$ values. Hence, time complexity is $\mathcal{O}((n - n'' - |\mathbf{V}|)(\tau + |W_{neg}| - w'_\exists - w'_\forall) + n'')$, and space complexity is $\mathcal{O}((n - n'' - |\mathbf{V}|)(\tau + |W_{neg}| - w'_\exists - w'_\forall))$

Combining two cases, overall time complexity is $\mathcal{O}(2^{|\mathbf{V}|} + (n - n'' - |\mathbf{V}|)(\tau + |W_{neg}| - w'_\exists - w'_\forall) + n'')$ and space complexity is $\mathcal{O}((n - n'' - |\mathbf{V}|)(\tau + |W_{neg}| - w'_\exists - w'_\forall))$. □

A Heuristic for Faster Computation. We observe that to encode conditional probabilities, we enumerate all assignments of variables in $\mathbf{V}$ that are in the Bayesian network. For computing the probability of positive prediction of a linear classifier with correlated features, we consider a heuristic to sort variables in $\mathbf{B} = \mathbf{A} \cup \mathbf{X}$. Let $\mathbf{V} \subseteq \mathbf{B}$ be the set of vertices in the network and $\mathbf{V}^c = \mathbf{B} \setminus \mathbf{V}$. In this heuristic, we sort sensitive variables $\mathbf{A}$

by positioning $\mathbf{A} \cap \mathbf{V}$ in the beginning followed by $\mathbf{A} \cap \mathbf{V}^c$. Then we order the variables $\mathbf{B}$ such that variables in $\mathbf{X}$ precedes those in $\mathbf{X} \cap \mathbf{V}$, and the variables in $\mathbf{X} \cap \mathbf{V}^c$ follows the ones in $\mathbf{X} \cap \mathbf{V}$. This sorting allows us to avoid repetitive enumeration of variables in $\mathbf{V} \subseteq \mathbf{B}$ as they are placed earlier in $\mathbf{B}$.

Example 6.1.2. We extend Example 6.1.1 with a Bayesian Network (G, θ) with $\mathbf{V} = \{P, Q\}$ and $\mathbf{E} = \{(P, Q)\}$. Parameters θ imply conditional probabilities $\Pr[Q|P] = 0.6$ and $\Pr[Q|\neg P] = 0.3$. In Figure 6.2b, we enumerate all assignment of P and Q to incorporate all conditional probabilities of Q given P. We, however, observe that the dynamic programming solution in Section 6.1.2 still prunes search space for variables that do not appear in $\mathbf{V}$, such as $\{R, S\}$. Hence following the calculation in Figure 6.2b, we obtain the maximum probability of positive prediction of the classifier as 0.65 for $P = 1$. The minimum probability of positive prediction (not shown) is similarly calculated as 0.11 for $P = 0$.

6.1.4 Fairness Verification with Computed Probability of Positive Prediction

Given a classifier $\mathcal{M}$, a distribution $\mathcal{D}$, and a fairness metric f, verifying whether a classifier is ϵ-fair for $\epsilon \in [0, 1]$ is equivalent to computing $\mathbb{1}[f(\mathcal{M}|\mathcal{D}) \leq \epsilon]$. We now compute $f(\mathcal{M}|\mathcal{D})$ based on the maximum probability of positive prediction $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$ and the minimum probability of positive prediction $\min_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$ of a classifier.

For measuring fairness metric SP, we compute the difference $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}] - \min_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$. We, however, deploy FVGM twice while measuring EO, one for the distribution $\mathcal{D}$ conditioned on $Y = 1$ and another for $Y = 0$. In each case, we compute $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}, Y = y] - \min_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}, Y = y]$ for $y \in \{0, 1\}$ and take the maximum difference as the value of EO. For measuring causal metric PCF, we compute $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}, \mathbf{Z}]$ and $\min_{\mathbf{a}} \Pr[\widehat{Y} = 1, \mathbf{Z}|\mathbf{A} = \mathbf{a}, \mathbf{Z}]$ conditioned on mediator features $\mathbf{Z}$ and take their difference. To measure disparate impact DI, we compute the ratio $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}] / \min_{\mathbf{a}} \Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}]$. In contrast to other fairness metrics, DI closer to 1 indicates higher fairness level. Thus, we verify whether a classifier achieves $(1 - \epsilon)$-DI by checking $\mathbb{1}[f_{\mathsf{DI}}(\mathcal{M}|\mathcal{D}) \geq 1 - \epsilon]$.

6.1.5 Extension to Practical Settings

For verifying linear classifiers with real-valued features and coefficients, we preprocess them so that FVGM can be invoked. Let $X \in \mathbb{R}$ be a continuous real-valued feature with coefficient $W \in \mathbb{R}$ in the classifier. We discretize X to a set $\mathbf{B}$ of k Boolean variables using binning-based discretization and assign a Boolean variable to each bin. Hence, $B_i \in \mathbf{B}$ becomes 1, when X belongs to the i^{th} bin. Let μ_i denote the mean of feature-values within i^{th} bin. We then set the coefficient of B_i as $\mu_i W$. By the law of large numbers, $X \approx \sum_i \mu_i B_i$ for infinitely many bins [201]. Finally, we multiply the coefficients of discretized variables

by $l \in \mathbb{N} \setminus \{0\}$ and round to an integer. The accuracy of the preprocessing step relies on the number of bins k and the multiplier l. Therefore, we empirically fine-tune both k and l by comparing the processed classifier with the initial classifier on a validation dataset.

6.2 Empirical Performance Analysis

In this section, we empirically evaluate the performance of FVGM. We first present the experimental setup followed by experimental results.

Experimental Setup. We implement a prototype of FVGM in Python (version 3.8). We deploy the Scikit-learn library for learning linear classifiers such as Logistic Regression (LR) and Support Vector Machine (SVM) with linear kernels. We perform five-fold cross-validation on a dataset. While the classifier is trained on continuous features, we discretize them to Boolean features to be invoked by FVGM. During discretization, we apply a gird-search to estimate the best bin-size within a maximum bin of 10. To convert the coefficients of features into integers, we employ another grid-search to choose the best multiplier within $\{1, 2, \ldots, 100\}$. For learning a Bayesian network on the converted Boolean data, we deploy the PGMPY library [202]. For network learning, we apply a Hill-climbing search algorithm that learns a DAG structure by optimizing K2 score [87]. For estimating parameters of the network, we use Maximum Likelihood Estimation (MLE) algorithm.

We compare FVGM with three existing fairness verifiers: Justicia [198], FairSquare [62], and VeriFair [63].In the following, we discuss a comparative analysis among all verifiers based on scalability and accuracy, where FVGM yields superior performance than others.

6.2.1 Scalability Analysis

Benchmarks. We perform the scalability analysis on five real-world datasets studied in the literature of fairness in machine learning: UCI Adult, German-credit [203], COMPAS [196], Ricci [197], and Titanic (`https://www.kaggle.com/c/titanic`). We consider 100 benchmarks generated from 5 real-world datasets and report the computation times for disparate impact and statistical parity metrics of different verifiers.

Results. In Figure 6.3, we present the scalability results of different verifiers. First, we observe that FairSquare often times out (= 900 seconds) and can solve ≤ 5 benchmarks. This indicates that SMT-based reduction for linear classifiers cannot scale. Similarly, SSAT-based verifier Justicia that performs pseudo-Boolean to CNF translation for linear classifiers, times out for around 20 out of 100 benchmarks. Sampling-based framework, VeriFair, has comparatively better scalability than SMT/SSAT-based frameworks and can solve more than 90 benchmarks. Finally, FVGM achieves impressive scalability by solving all 100 benchmarks with 1 to 2 orders of magnitude runtime improvements than existing

verifiers. Therefore, S3P-*based framework* FVGM *proves to be highly scalable in verifying fairness properties of linear classifiers than the state-of-the-art.*

6.2.2 Accuracy Analysis

Benchmark Generation. To perform accuracy analysis, we require the ground truth, which is not available for real-world instances. Therefore, we focus on generating synthetic benchmarks for analytically computing the ground truth of different fairness metrics, such as disparate impact, from the known distribution of features. In each benchmark, we consider $n \in \{2, 3, 4, 5\}$ features including one Boolean sensitive feature, say A, generated from a Bernoulli distribution with mean 0.5. We generate non-sensitive features X_i from Gaussian distributions such that $\Pr[X_i|A = 1] \sim \mathcal{N}(\mu_i, \sigma^2)$ and $\Pr[X_i|A = 0] \sim \mathcal{N}(\mu'_i, \sigma^2)$, where $\mu_i, \mu'_i \in [0, 1]$, $\sigma = 0.1$, and μ_i, μ'_i are chosen from a uniform distribution in $[0, 1]$. Finally, we create label $Y = \mathbb{1}[\sum_{i=1}^{n-1} X_i \geq 0.5 \sum_{i=1}^{n-1}(\mu_i + \mu'_i)]$ such that Y does not directly depend on the sensitive feature. For each n, we generate 100 random benchmarks, learn LR and SVM classifiers on them, and compute disparate impact using different verifiers.

Analytic Computation of Disparate Impact. Let the coefficients of the classifier be w_i for X_i and w_A for A, and bias be τ. Since all non-sensitive features are from Gaussian distributions, we compute the probability of the predicted class $\Pr[\widehat{Y}|A = 1] \sim \mathcal{N}(\sum_{i=1}^{n-1} w_i\mu_i, \sigma^2_{\widehat{Y}})$ and $\Pr[\widehat{Y}|A = 0] \sim \mathcal{N}(\sum_{i=1}^{n-1} w_i\mu'_i, \sigma^2_{\widehat{Y}})$ with $\sigma^2_{\widehat{Y}} = (\sum_{i=1}^{n-1} w_i^2)\sigma^2$. Hence, the probability of positive prediction of the classifier is $1 - \mathsf{CDF}_{\widehat{Y}|A=1}(\tau - w_A)$ for $A = 1$ and $1 - \mathsf{CDF}_{\widehat{Y}|A=0}(\tau)$ for $A = 0$, where CDF is the cumulative distribution function. Finally, we compute disparate impact by taking the ratio of the minimum and the maximum of the probability of positive prediction of the classifier for $A = 1$ and $A = 0$.

Results. We assess the accuracy of the competing verifiers in estimating fairness metrics, specifically disparate impact with LR and SVM classifiers. In Figure 6.4, FVGM computes disparate impact closest to the *Exact* value for different number of features and both type of classifiers. In contrast, Justicia, FairSquare, and VeriFair measure disparate impact far from the *Exact* because of ignoring correlations among the features. For example, for SVM classifier with $n = 5$ (right plot in Figure 6.4), *Exact* disparate impact is 0.089 (average over 100 random benchmarks). Here, FVGM computes disparate impact as 0.094, while all other verifiers compute disparate impact as at least 0.233. Therefore, FVGM *is more accurate than existing verifiers as it explicitly considers correlations among features.*

6.3 Applications of FVGM

In this section, we apply FVGM for verifying fairness-enhancing algorithms and depreciation attacks. We also demonstrate that FVGM facilitates the computation of fairness influence functions by enabling the shift of bias from the original value due to individual features.

Table 6.1: Verification of fairness algorithms using FVGM. **A** denotes sensitive features. RW and OP refer to reweighing and optimized-preprocessing algorithms. Numbers in bold refer to fairness improvement.

Dataset	**A**	Algo.	ΔDI	ΔPCF	Δ SP	ΔEO
Adult	race	RW	**0.53**	**-0.06**	**-0.06**	**-0.02**
		OP	**0.57**	**-0.07**	**-0.07**	**-0.02**
	sex	RW	**0.96**	**-0.16**	**-0.15**	**-0.08**
		OP	**0.43**	**-0.08**	**-0.08**	0.03
COMPAS	race	RW	**0.13**	**-0.07**	**-0.07**	**-0.06**
		OP	**0.15**	**-0.08**	**-0.08**	**-0.05**
	sex	RW	**0.1**	**-0.04**	**-0.04**	0.04
		OP	**0.09**	**-0.04**	**-0.04**	**-0.03**
German	age	RW	**0.52**	**-0.53**	**-0.52**	**-0.47**
		OP	**0.53**	**-0.53**	**-0.53**	**-0.51**
	sex	RW	−0.06	0.06	0.06	0.02
		OP	−0.12	0.12	0.12	0.07

Verifying Fairness-enhancing Algorithms. We deploy FVGM in verifying the effectiveness of fairness-enhancing algorithms designed to ameliorate bias. For example, fairness pre-processing algorithms can be validated by applying FVGM on the unprocessed and the processed data separately and comparing different fairness metrics. In Table 6.1, we report the effect of fairness algorithms w.r.t. four fairness metrics: disparate impact (DI), path-specific causal fairness (PCF), statistical parity (SP), and equalized odds (EO). Note that, fairness is improved if DI increases and the rest of the metrics decrease. For instance, in most instances for Adult dataset, reweighing (RW) [54] and optimized pre-processing (OP) [56] algorithms are successful in improving fairness. The exceptional case is the unfairness regarding the sensitive feature 'sex', where OP algorithm fails in improving fairness metric EO. Thus, FVGM *verifies the enhancement and decrement in fairness by fairness-enhancing algorithms.*

Verifying Fairness Attacks. We apply FVGM in verifying a fairness poisoning-attack algorithm. This algorithm injects a small fraction of poisoned samples into the training data such that the classifier becomes relatively unfair [60]. We apply this attack to add $\{1, 5, \ldots, 160\}$ poisoned samples and measure the corresponding disparate impact and statistical parity. In Figure 6.5, FVGM verifies that the disparate impact of the classifier decreases and statistical parity increases, i.e. *the classifier becomes more unfair, as the number of poisoned samples increases.* Therefore, FVGM shows the potential of being

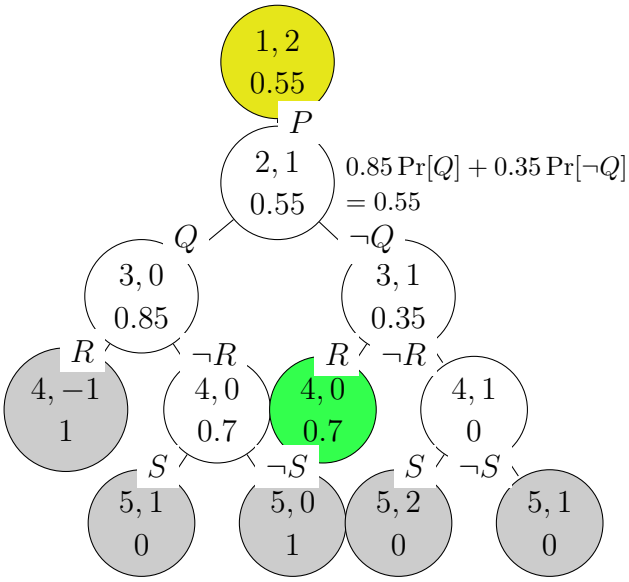
deployed in safety-critical applications to detect fairness attacks. For example, if we set 0.9 as an acceptable threshold of disparate impact, FVGM can raise an alarm once 160 poisoned samples are added.

Fairness Influence Function (FIF): Tracing Sources of Unfairness. Another application of FVGM as a fairness verifier is to quantify the effect of a subset of features on fairness. Thus, we define fairness influence function (FIF) that computes the effect of a subset of non-sensitive features $\mathbf{S} \subseteq \mathbf{X}$ on the probability of positive prediction of a classifier given a specific sensitive group $\mathbf{A} = \mathbf{a}$, $\mathsf{FIF}(\mathbf{S}) \triangleq \Pr[\hat{Y} = 1|\mathbf{A} = \mathbf{a}, \mathcal{D}] - \Pr[\hat{Y} = 1|\mathbf{A} = \mathbf{a}, \mathcal{D}_{-\mathbf{S}}]$. FIF allows us to explain the sources of unfairness in the classifier. In practice, we compute FIF of $\mathbf{S}$ by replacing the probability distribution of $\mathbf{S}$ with a uniformly random distribution, referred to as $\mathcal{D}_{-\mathbf{S}}$, and reporting differences in the conditional probability of positive prediction of the classifier.

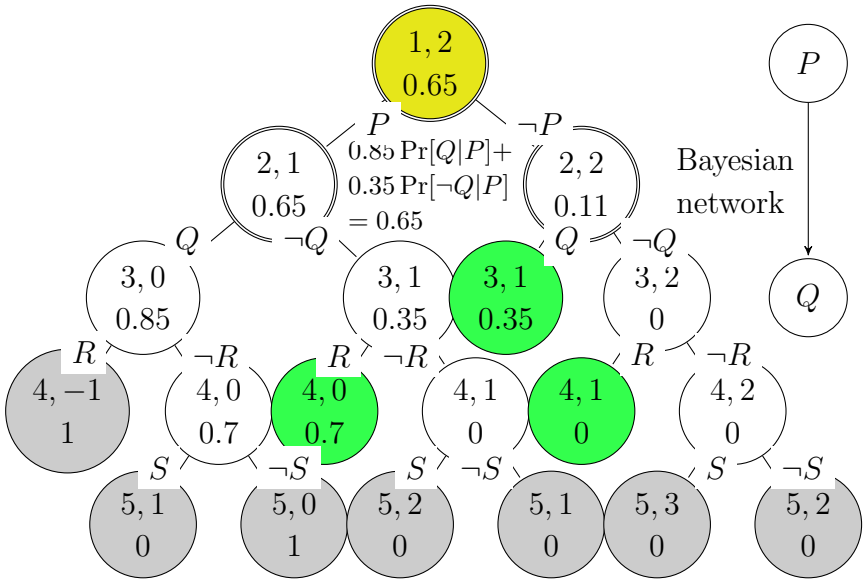
In Figure 6.6, we compute FIF for all features in COMPAS dataset, separately for two sensitive groups: Female ('sex' = 1) and Male. This dataset concerns the likelihood of a person of re-offending crimes within the next two years. We first observe that the base values of the probability of positive prediction are different for the two groups (0.46 vs 0.61 for Female and Male), thereby showing Male as more probable to re-offend crimes than Female. Moreover, FIF of the feature 'age' is comparatively higher in magnitude for Male than Female. This implies that while deciding recidivism the algorithm assumes that Female individuals across different ages re-offend crimes with almost the same probability and the probability of re-offending for Male individuals highly depends on age. Thus, applying FVGM to compute FIF provides us insights about sources of bias and thus, indicators to improve fairness.

6.4 Chapter Summary

In this chapter, we propose FVGM, an efficient fairness verification framework for linear classifiers based on a novel stochastic subset-sum problem. FVGM encodes a graphical model of feature-correlations, represented as a Bayesian Network, and computes multiple group and causal fairness metrics accurately. We experimentally demonstrate that FVGM is not only more accurate and scalable than the existing verifiers but also applicable in practical fairness tasks, such as verifying fairness attacks and enhancing algorithms, and computing the fairness influence functions. As a future work, we aim to design fairness-enhancing algorithms certified by fairness verifiers, such as FVGM. Since FVGM serves as an accurate and scalable fairness verifier for linear classifiers, it will be interesting to design such verifiers for other machine learning models.



(a) Known marginal probabilities.



(b) Probabilities computed with a Bayesian network.

Figure 6.2: Search tree representation of S3P for computing the maximum probability of positive prediction of the classifier on variables $\mathbf{B} = \{P, Q, R, S\}$ with weights $\{1, 1, 1, -1\}$ and threshold $\tau = 2$. Each node is labeled by (i, τ'), where i is the index of $\mathbf{B}$ and τ' is the residual threshold. The tree is explored using Depth-First Search (DFS) starting with left child. Within a node, the value in the bottom denotes $\mathsf{dp}(i, \tau')$ that is solved recursively based on sub-problems $\mathsf{dp}(i+1, \cdot)$ in child nodes. Yellow nodes denote *existential* variables and all other nodes are *random* variables. Additionally, a green node denotes a collision, in which case a previously computed dp solution is returned. Leaf nodes (gray) are computed based on terminating conditions in Eq. (6.3). In Figure 6.2b, nodes with double circles, such as $\{(1, 2), (2, 1), (2, 2)\}$, are enumerated exponentially to compute conditional probabilities from the Bayesian network.

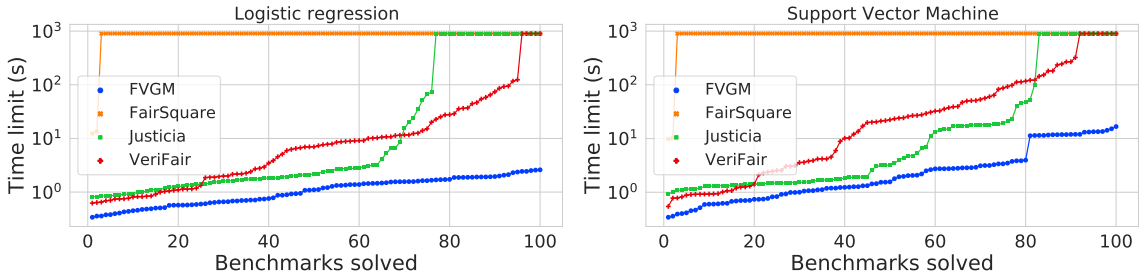


Figure 6.3: A cactus plot to present the scalability of different fairness verifiers. The number of solved benchmarks are on the X-axis and the required time is on the Y-axis; a point (x, y) implies that a verifier takes less than or equal to y seconds to compute fairness metrics of x many benchmarks. We consider 100 benchmarks generated from 5 real-world datasets using 5-fold cross-validation. In each fold, we consider $\{25, 50, 75, 100\}$ percent of non-sensitive features.

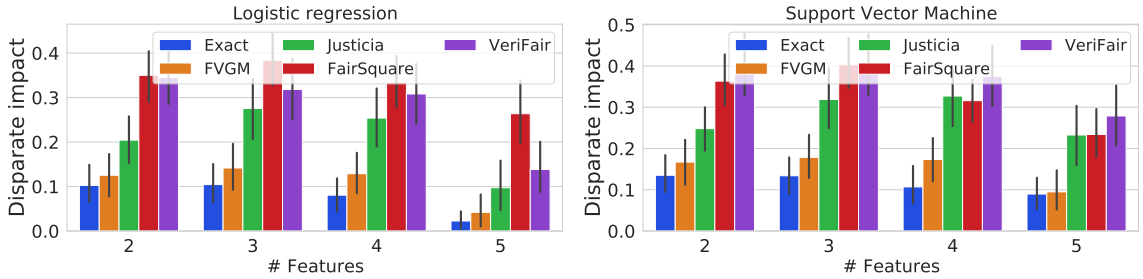


Figure 6.4: Comparing the average accuracy of different verifiers over 100 synthetic benchmarks while varying the number of features. FVGM yields the closest estimation of the analytically calculated *Exact* values of disparate impact for LR and SVM classifiers.

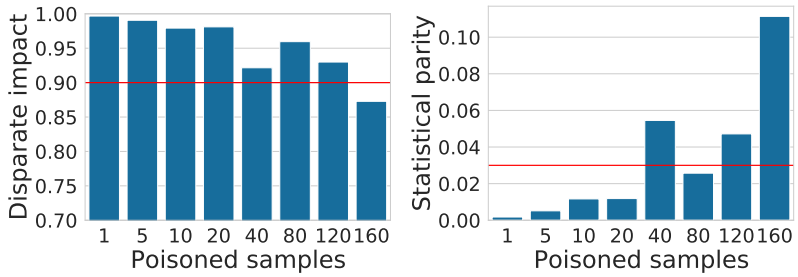


Figure 6.5: Verifying poisoning attack against fairness using FVGM. The red line denotes the safety margin of the machine learning model against the attack.

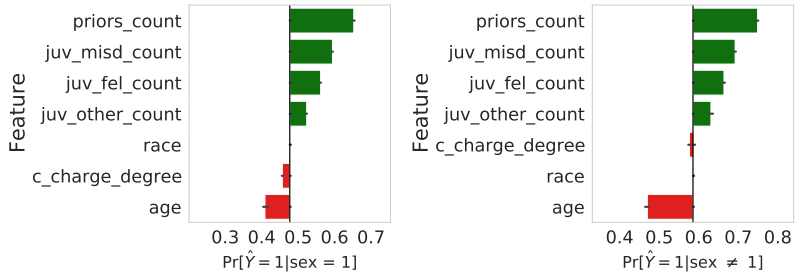


Figure 6.6: FIF computation for COMPAS dataset. For Female (left plot) and Male (right plot) individuals, the influence of 'age' decreases and the probability of positive prediction of the classifier increases by different magnitudes.

Chapter 7

Computing Fairness Influence Functions with Global Sensitivity Analysis

The last decades have witnessed a significant progress in machine learning with applications in high-stake decision making, such as college admission [44], recidivism prediction [45], job applications [46] etc. In such applications, the deployed machine learning classifiers often demonstrate bias towards certain demographic groups involved in the data [47]. For example, a classifier deciding the eligibility of college admission may offer more admission to White-male candidates than to Black-female candidates—possibly because of the historical bias in the admission data, or the accuracy-centric learning objective of the classifier, or a combination of both [48–50]. Following such phenomena, multiple fairness metrics, such as *statistical parity*, *equalized odds*, *predictive parity* etc, have been proposed to quantify the bias of the classifier on a dataset. For example, if the classifier in college admission demonstrates a statistical parity of 0.6, it means that White-male candidates are offered admission 60% more than Black-female candidates [51–53].

Although fairness metrics globally quantify bias, they cannot detect or explain the sources of bias [1, 2, 69]. In order to identify the sources of bias and also the effect of affirmative/punitive actions to alleviate/deteriorate bias, it is important to understand *which factors contribute how much to the bias of a classifier on a dataset*. To this end, we follow a feature-attribution approach to understand the sources of bias [1, 2], where we relate the *influences* of input features towards the resulting bias of the classifier. Particularly, we define and compute *Fairness Influence Function* (FIF) that quantifies the contribution of individual and subset of features to the resulting bias. FIFs do not only allow practitioners to identify the features to act up on but also to quantify the effect of various affirmative [55–59, 70, 71] or punitive actions [60, 72, 73] on the resulting bias.

Our Contributions. The contribution of this chapter is three-fold.

1. *Formalism:* We propose to compute individual and intersectional **F**airness **I**nfluence

Functions (FIFs) of features as a measure of contribution towards the bias of the classifier (Sec 7.2). The *intersectionality* [204] allows us to detect the higher order interactions among the features. In the formalism, we first axiomatize that to be a proper attribution of bias, the sum of FIFs of all subsets of the features is equal to the bias of the classifier (Axiom 1). Following that, we show that computing FIFs over subsets of features is equivalent to computing a scaled difference in the *conditional variances* (Theorem 13) of the classifier between the sensitive groups of interest. This allows us to connect global sensitivity analysis, a standard technique recommended by regulators to asses numerical models [92, 93], with computing feature influences leading to bias in the classifier's output.

2. Algorithmic: We instantiate an algorithm, FairXplainer, for computing FIFs for subsets of features for a given classifier, a dataset, and any group fairness metrics, namely statistical parity, equalized odds, and predictive parity (Sec. 7.3). The key ideas are to import techniques from Global Sensitivity Analysis (GSA) [91] to decompose the variance of the prediction of the classifier among the subset of features and to use a local regression method [205] to compute FIFs.

3. Experimental: We experimentally validate that FairXplainer is significantly more accurate and can capture the intersectional or joint effect of the features on the bias induced by a classifier (Sec. 7.4). Improvement in accuracy and extension to intersectional FIFs are both absent in the existing works aimed to this problem [1, 2]. Also, FairXplainer enables us to detect the change in FIFs due to different fairness enhancing algorithms and fairness reducing attacks, which opens up new avenues to analyze the effects of these algorithms.

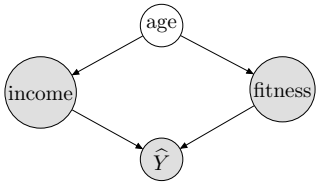
We illustrate the usefulness of our contributions via an example scenario in Example 5.0.1.

Example 7.0.1. Following [198], we consider a classifier that decides an individual's eligibility for health insurance based on non-sensitive features 'fitness' and 'income'. 'fitness' and 'income' depend on a sensitive feature 'age' leading to two sensitive groups "young" (age < 40) and "elderly" (age ≥ 40), as highlighted in (Figure 7.1a–7.1b).

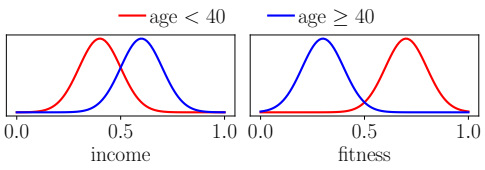
Case study 1: For each sensitive group, we generate 500 samples of (income, fitness) and train a decision tree (DT1), which predicts without explicitly using the sensitive feature 'age' (Figure 7.1c). Using standard off-the-shelf techniques, we can compute statistical parity as $\Pr[\widehat{Y} = 1|\text{age } < 40] - \Pr[\widehat{Y} = 1|\text{age } \geq 40] = 0.7 - 0.17 = 0.53$, and therefore DT1 is unfair towards "elderly".

Applying techniques developed in this chapter, we investigate the source of unfairness of DT1 by computing FIFs of features, where *positive numbers denote a reduction in fairness and negative numbers denotes fairness improvement*. In Figure 7.1e, fitness (FIF $= 0.74$), and the joint effect of fitness and income (FIF $= 0.05$) cause higher statistical parity, i.e. higher bias. This observation is invoked as DT1 predicts positively for the higher values of fitness (root node in DT1) and elderly individuals often possess lower fitness (Figure 7.1b). In contrast, the income of individuals (FIF $= -0.33$) decreases statistical parity, as elderly individuals have a higher income but lower fitness.

Data:

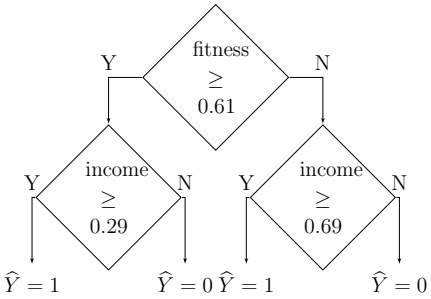


(a) Dependency among features and prediction

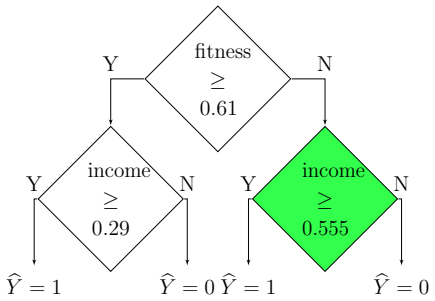


(b) Age-dependent distributions of non-sensitive features

Classifier:

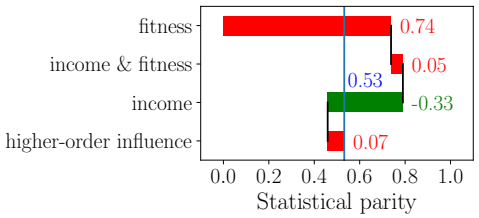


(c) Decision tree (DT1)

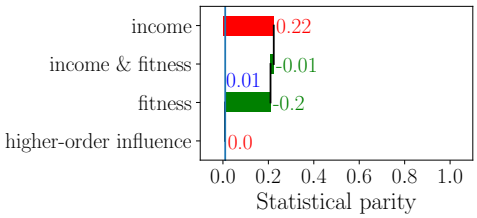


(d) Decision tree with affirmative action (DT2)

FIF:



(e) Fairness influence functions (FIF) for DT1



(f) Modified FIFs for DT2

Figure 7.1: FIFs of input features to investigate the bias (i.e., statistical parity) of a decision tree classifier predicting the eligibility for health insurance using age-dependent features 'fitness' and 'income'. An affirmative action reduces bias, and corresponding FIFs present the evidence.

Case study 2: Since DT1 is unfair to the "elderly", we learn another decision tree (DT2) by applying an affirmative action, where we decrease the threshold on income from 0.69 to 0.555 when the fitness is lower (green node in Figure 7.1d). This action allows more elderly individuals to receive insurance by including more people with lower fitness, and thus the statistical parity becomes $\Pr[\widehat{Y} = 1|\text{age } < 40] - \Pr[\widehat{Y} = 1|\text{age } \geq 40] = 0.71 - 0.7 = 0.01$. This is significantly less than the earlier statistical parity of 0.53. Again, applying techniques developed in this chapter, we compute FIF of features in Figure 7.1f. Here, the FIF of income and fitness reflects the reduction in statistical parity as their influences almost nullify each other. Thus, FIF depicts the effects of different features and their combinations on the resultant bias incurred by the classifier.

7.1 Related Work

Recently, several studies apply *local explanation methods* of black-box prediction to explain sources of bias by feature-attribution [1, 2] and causal path decomposition [69]. Since our work adopts feature-attribution approach, it reveals two-fold limitations of existing methods: (i) *inaccuracy* in computing FIFs and (ii) *failing to compute intersectional* FIFs. Elaborately, FIF computation in [1, 2] is inaccurate because of applying local explainers such as SHAP [17] to compute global properties of the classifier such as group fairness. In addition, features are often correlated in practical fairness tasks, and computing only individual FIFs ignores the joint contribution of multiple features on the unfairness of the classifer. Also, these works provide empirical evaluations to justify the choices of SHAP-based tools for explaining fairness and does not consider the global nature of group fairness metrics. In this chapter, *we develop a formal framework to explain sources of unfairness in a classifier and also a novel methodology to address it. To the best of our knowledge, this is the first work to do the both.* Among other related works, [206] link GSA measures such as Sobol and Cramér-von-Mises indices to different fairness metrics. While their approach relates the GSA of sensitive features on the resulting bias, we focus on applying GSA to all features to compute FIFs. *Their approach only detects the presence or absence of bias, while we focus on decomposing bias as the sum of FIFs of all features.* In another line of work, [207] and [208] compute feature-influence as the shift of bias from its original value by randomly intervening features. Their work is different from our axiomatic approach, where the sum of FIFs equals the bias.

7.2 Fairness Influence Functions: Definition and Computation

In this section, we discuss our contribution of formalizing and computing Fairness Influence Functions (FIF) as a quantifier of the contribution of a subset of features to the resultant bias of a classifier on a dataset. The key idea in the formalization of FIFs is their additive formulation. The key idea in the computation of FIFs is to relate bias with the scaled difference of the conditional variance of positive prediction of the classifier over specific sensitive groups, and to apply variance decomposition to compute individual and intersectional FIFs.

FIF: An Axiomatic Formulation. We are given a binary classifier $\mathcal{M} : (\mathbf{X}, \mathbf{A}) \to \widehat{Y}$, a dataset $\mathbf{D} = \{(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)})\}_{i=1}^{n}$, and a fairness metric $f(\mathcal{M}, \mathbf{D}) \in \mathbb{R}^{\geq 0}$. Our objective is to compute the influences of non-sensitive features on f. We compute influence of each subset of non-sensitive features $\mathbf{X_S}$, where $\mathbf{S} = \{S_i | 1 \leq S_i \leq k\} \subseteq [k] \setminus \emptyset$ is a non-empty subset of indices.

Definition 1 (Fairness Influence Function)**.** *Fairness Influence Function (FIF)* $w_{\mathbf{S}} : \mathbf{X_S} \to$

$\mathbb{R}$ *measures the quantitative contribution of the subset of features* $\mathbf{X_S} \subseteq \mathbf{X}_{[k]}$ *on the incurred bias* $f(\mathcal{M}, \mathbf{D})$ *of the classifier* $\mathcal{M}$ *for dataset* $\mathbf{D}$. *We refer* $w_\mathbf{S}$ *as an* individual influence *when* $|\mathbf{S}| = 1$ *and an* inter-sectional influence *when* $|\mathbf{S}| > 1$—*particularly, a second-order influence when* $|\mathbf{S}| = 2$ *and a higher-order influence when* $3 \leq |\mathbf{S}| \leq k$.

Since FIF is meant to attribute the bias of the classifier to the subset of features, it is natural to choose a function such that the FIFs of all the subsets of features exactly adds up to the total bias. This leads to the following additivity axiom.

Axiom 1 (Additivity of influence)**.** *Let* $f(\mathcal{M}, \mathbf{D}) \in \mathbb{R}^{\geq 0}$ *be the bias of the classifer and* $w_\mathbf{S} : \mathbf{X_S} \to \mathbb{R}$ *be the FIF of features* $\mathbf{X_S}$ *on* f. *The additivity axiom states that the total influences of all possible subset of non-sensitive features is equal to the bias of the classifier.*

$$f(\mathcal{M}, \mathbf{D}) = \sum\nolimits_{\mathbf{S} \subseteq [k] \setminus \emptyset} w_\mathbf{S} \tag{7.1}$$

Computing FIF: Reduction to Variance Decomposition. Henceforth, our objective is to design an estimator of FIF $w_\mathbf{S}$ that satisfies Axiom 1. In the following, we discuss the closed form representation of $w_\mathbf{S}$ for statistical parity fairness metric and extend to other metrics in Section 7.3.

Connecting Statistical Parity and Conditional Variance. Let $\mathbf{a}_{\max} = \arg\max_\mathbf{a} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ and $\mathbf{a}_{\min} = \arg\min_\mathbf{a} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ be the most favored and the least favored sensitive group of the classifier, respectively, according to their conditional probability of positive prediction. Next, we consider a Bernoulli random variable $B_\mathbf{a}$ to denote the positive prediction of the classifier for the sensitive group $\mathbf{A} = \mathbf{a}$. Then, the probability of the random variable is $p_\mathbf{a} \triangleq \Pr[B_\mathbf{a} = 1] = \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$, and variance is $V_\mathbf{a} = p_\mathbf{a}(1 - p_\mathbf{a})$. Therefore, considering two random variables $B_{\mathbf{a}_{\max}}$ and $B_{\mathbf{a}_{\min}}$, we state the statistical parity of the classifier as $p_{a_{\max}} - p_{a_{\min}} = (V_{a_{\max}} - V_{a_{\min}})/(1 - (p_{a_{\max}} + p_{a_{\min}}))$, where $p_{a_{\max}}$ and $p_{a_{\min}}$ is the probability of positive prediction of the classifier for the group $\mathbf{A} = \mathbf{a}_{\max}$ and $\mathbf{A} = \mathbf{a}_{\min}$, respectively. Intuitively, the statistical parity of the classifier is the scaled (by $1/(1 - (p_{a_{\max}} + p_{a_{\min}}))$) difference of the conditional variance of the positive prediction of the classifier over the most and the least favored sensitive groups.

Example 7.2.1. In Example 7.0.1 (Figure 7.1c), the conditional probabilities of positive prediction of the decision tree for the most and least favored groups are $p_{a_{\max}} = \Pr[\widehat{Y} = 1 | \text{age } < 40] = 0.704$ and $p_{a_{\min}} = \Pr[\widehat{Y} = 1 | \text{age } \geq 40] = 0.172$, respectively. Thus, the statistical parity of the classifier is $p_{a_{\max}} - p_{a_{\min}} = 0.704 - 0.172 = 0.532$. Now, we compute conditional variances of positive prediction as $V_{a_{\max}} = \mathsf{Var}[\widehat{Y} = 1 | \text{age } < 40] = 0.208$ and $V_{a_{\min}} = \mathsf{Var}[\widehat{Y} = 1 | \text{age } \geq 40] = 0.142$. Thus, following our argument, we compute the scaled difference in conditional variances as $(0.208 - 0.142)/(1 - (0.702 + 0.172)) = 0.532$, which coincides with the statistical parity.

Now, we apply variance decomposition (Eq. (2.5)) for both the conditional variances $V_{\mathbf{a}_{\max}}$ and $V_{\mathbf{a}_{\min}}$ to compute the FIF of features, as stated in Theorem 13.

Theorem 13 (FIF as Difference of Conditional Variances)**.** *Let* $V_{\mathbf{a},\mathbf{S}} \triangleq \mathsf{Var}_{\mathbf{X_S}}[\widehat{Y}=1|\mathbf{A}=\mathbf{a}]$ *be the decomposed conditional variance of the classifier's positive prediction for the sensitive group* $\mathbf{A}=\mathbf{a}$*, where features* $\mathbf{X_S}$ *are jointly varied. Then, we can express FIF of* $\mathbf{X_S}$ *as*

$$w_{\mathbf{S}} = \frac{V_{\mathbf{a}_{\max},\mathbf{S}} - V_{\mathbf{a}_{\min},\mathbf{S}}}{1 - (\Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a}_{\max}] + \Pr[\widehat{Y}=1|\mathbf{A}=\mathbf{a}_{\min}])} \tag{7.2}$$

and FIFs defined by Equation (7.2) *satisfy Axiom 1.*

Proof. Let $B_{\mathbf{a}} \in \{0,1\}$ be a Bernoulli random variable such that $B_{\mathbf{a}}=1$ denotes the classifier predicting positive class $\hat{Y}=1$ for an individual belonging to the sensitive group $\mathbf{A}=\mathbf{a}$, and $B_{\mathbf{a}}=0$ denotes $\hat{Y}=0$. Hence, if $p_{\mathbf{a}} \triangleq \Pr[B_{\mathbf{a}}=1]$, then $p_{\mathbf{a}}$ is also the conditional probability of positive prediction of the classifier, $p_{\mathbf{a}} = \Pr[\hat{Y}=1|\mathbf{A}=\mathbf{a}]$. The variance of $B_{\mathbf{a}}$ is computed as

$$\mathsf{Var}[B_{\mathbf{a}}] = p_{\mathbf{a}}(1-p_{\mathbf{a}}).$$

Now, we consider two Bernoulli random variables $B_{\mathbf{a}}$ and $B_{\mathbf{a}'}$ associated with two different sensitive groups $\mathbf{A}=\mathbf{a}$ and $\mathbf{A}=\mathbf{a}'$, respectively. The difference in variances between $B_{\mathbf{a}}$ and $B_{\mathbf{a}'}$ implies that

$$\begin{aligned}\mathsf{Var}[B_{\mathbf{a}}] - \mathsf{Var}[B_{\mathbf{a}'}] &= p_{\mathbf{a}}(1-p_{\mathbf{a}}) - p_{\mathbf{a}'}(1-p_{\mathbf{a}'}) \\ &= p_{\mathbf{a}} - p_{\mathbf{a}}^2 - p_{\mathbf{a}'} + p_{\mathbf{a}'}^2 \\ &= (p_{\mathbf{a}} - p_{\mathbf{a}'})(1-(p_{\mathbf{a}}+p_{\mathbf{a}'}))\end{aligned}$$

After reorganization, we express the difference in probabilities $p_{\mathbf{a}} - p_{\mathbf{a}'}$ as

$$p_{\mathbf{a}} - p_{\mathbf{a}'} = \frac{\mathsf{Var}[B_{\mathbf{a}}] - \mathsf{Var}[B_{\mathbf{a}'}]}{1-(p_{\mathbf{a}}+p_{\mathbf{a}'})} \tag{7.3}$$

By setting $\mathbf{a}=\mathbf{a}_{\max}$ and $\mathbf{a}'=\mathbf{a}_{\min}$ in Equation (7.3), we express the statistical parity of the classifier in terms of the scaled difference in the conditional variance of the positive prediction of the classifier.

$$\begin{aligned}f_{\mathsf{SP}}(\mathcal{M},\mathbf{D}) \triangleq p_{\mathbf{a}_{\max}} - \quad p_{\mathbf{a}_{\min}} &= \frac{\mathsf{Var}[B_{\mathbf{a}_{\max}}] - \mathsf{Var}[B_{\mathbf{a}_{\min}}]}{1-(p_{\mathbf{a}_{\max}}+p_{\mathbf{a}_{\min}})} \\ &= \frac{\mathsf{Var}[\hat{Y}=1|\mathbf{A}=\mathbf{a}_{\max}] - \mathsf{Var}[\hat{Y}=1|\mathbf{A}=\mathbf{a}_{\min}]}{1-(p_{\mathbf{a}_{\max}}+p_{\mathbf{a}_{\min}})}.\end{aligned} \tag{7.4}$$

Next, we apply variance decomposition (Equation (2.5)) to estimate the influence of each $\mathbf{X_S}$. From Equation (2.5), we obtain that the variance of $B_{\mathbf{a}}$ can be decomposed as

$$\mathsf{Var}[B_{\mathbf{a}}] = \sum_{\mathbf{S}\subseteq[k]\setminus\emptyset} V_{\mathbf{a},\mathbf{S}},$$

where $V_{\mathbf{a},\mathbf{S}}$ denotes the decomposed variance of $B_{\mathbf{a}}$ w.r.t. the features $\mathbf{X_S}$ conditioned on the sensitive group $\mathbf{A}=\mathbf{a}$.

Now, we apply variance decomposition on both $\mathsf{Var}[B_{\mathbf{a}_{\max}}]$ and $\mathsf{Var}[B_{\mathbf{a}_{\min}}]$ in Equation 7.4 to obtain

$$f_{\mathsf{SP}}(\mathcal{M}, \mathbf{D}) = \frac{\mathsf{Var}[B_{\mathbf{a}_{\max}}] - \mathsf{Var}[B_{\mathbf{a}_{\min}}]}{1 - (p_{\mathbf{a}_{\max}} + p_{\mathbf{a}_{\max}})} = \sum_{\mathbf{S} \subseteq [k] \setminus \emptyset} \frac{V_{\mathbf{a}_{\max},\mathbf{S}} - V_{\mathbf{a}_{\min},\mathbf{S}}}{1 - (p_{\mathbf{a}_{\max}} + p_{\mathbf{a}_{\min}})} \tag{7.5}$$

Finally, we estimate the FIF of the subset of features $\mathbf{X_S}$ as

$$w_{\mathbf{S}} = \frac{V_{\mathbf{a}_{\max},\mathbf{S}} - V_{\mathbf{a}_{\min},\mathbf{S}}}{1 - (p_{\mathbf{a}_{\max}} + p_{\mathbf{a}_{\min}})} \tag{7.6}$$

by separating the terms corresponding to $\mathbf{X_S}$.

Following Equation (7.5), $w_{\mathbf{S}}$ satisfies Axiom 1 as $f_{\mathsf{SP}}(\mathcal{M}, \mathbf{D}) = \sum_{\mathbf{S} \subseteq [k] \setminus \emptyset} w_{\mathbf{S}}$. □

Consequences of Theorem 13. Here, we discuss the algorithmic consequences of Theorem 13 and probable issues that might appear in computation.

1. *Algorithm Design.* Expressing FIF in terms of the variance decomposition over subset of features allows us to import and extend well-studied techniques of global sensitivity analysis to perform FIF estimation accurately and at scale.

2. *Bias Amplifying and Eliminating Features.* The sign of a FIF $w_{\mathbf{S}}$ denotes whether the features $\mathbf{X_S}$ are amplifying the bias of the classifier or eliminating it. When $w_{\mathbf{S}} > 0$, $\mathbf{X_S}$ increases bias. When $w_{\mathbf{S}} < 0$, $\mathbf{X_S}$ eliminates bias, and thus improves fairness. The features $\mathbf{X_S}$ are neutral when $w_{\mathbf{S}} = 0$.

3. *Degenerate Cases:* If the classifier always makes positive prediction or never makes positive prediction, i.e. $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}] \in \{1, 0\}$ for any sensitive group $\mathbf{A} = \mathbf{a}$, the conditional variance $\mathsf{Var}[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}] = 0$. When total variance is zero, the decomposed variances $V_{\mathbf{a},\mathbf{S}}$ may become zero for each feature combination $\mathbf{X_S}$, and Equation (7.2) cannot trivially compute fairness influence functions. Though such degenerate cases rarely occur for real-life data, we can avoid them by randomly perturbing at least one sample in the dataset to yield the opposite predicted class.

4. *Numerical Instability.* In Equation (7.2), the conditional probabilities of positive prediction of the classifier $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}_{\max}]$ and $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}_{\min}]$ are constant. When $\Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}_{\max}] + \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}_{\min}] = 1$, the denominator in $w_{\mathbf{S}}$ becomes 0 and hence, $w_{\mathbf{S}}$ is undefined. In this case, we add/subtract a small number in the denominator to avoid numerical instability of division though we have not encountered any such case in our experiments.

7.3 FairXplainer: An Algorithm to Compute Fairness Influence Functions

We propose an algorithm, FairXplainer, that leverages the variance decomposition of Eq. (7.2) to compute the Fairness Influence Functions (FIFs) of all the subset of features.

FairXplainer has two algorithmic blocks: (i) local regression to decompose the classifier into component functions and (ii) computing the variance (or covariance) of each component. We describe the schematic of FairXplainer in Algorithm 6.

A Set-additive Representation of the Classifier. To apply variance decomposition (Eq. (2.5)), we learn a set-additive representation of the classifier (Eq. (2.4)). Let us denote the classifier $\mathcal{M}$ conditioned on a sensitive group $\mathbf{a}$ as $g_{\mathbf{a}}(\mathbf{X}) \triangleq \mathcal{M}(\mathbf{X}, \mathbf{A} = \mathbf{a})$. We express $g_{\mathbf{a}}$ as a set-additive model:

$$g_{\mathbf{a}}(\mathbf{X}) = g_{\mathbf{a},0} + \sum_{\mathbf{S} \subseteq [k] \setminus \emptyset, |\mathbf{S}| \leq \lambda} g_{\mathbf{a},\mathbf{S}}(\mathbf{X}_{\mathbf{S}}) + \delta \tag{7.7}$$

Algorithm 6 FairXplainer: A Framework for Computing Fairness Influence Function (FIF)

Input: $\mathcal{M} : (\mathbf{X}, \mathbf{A}) \to \widehat{Y}, \mathbf{D} = \{(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)})\}_{i=1}^{n}, f(\mathcal{M}, \mathbf{D}) \in \mathbb{R}^{\geq 0}, \lambda$
Output: $w_{\mathbf{S}}$

1: $\mathbf{a}_{\max} = \arg\max_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}], \mathbf{a}_{\min} = \arg\min_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}], k \leftarrow |\mathbf{X}|$
2: **for** $\mathbf{a} \in \{\mathbf{a}_{\max}, \mathbf{a}_{\min}\}$ **do** ▷ Enumerate for specific sensitive groups
3: $\quad g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a},0} \leftarrow \textsc{LocalRegression}(\mathcal{M}(\mathbf{X}, \mathbf{A} = \mathbf{a}), \{\mathbf{x}^{(i)}\}_{i=1}^{n}, \lambda, k)$
4: $\quad V_{\mathbf{a},\mathbf{S}} \leftarrow \textsc{Covariance}(g_{\mathbf{a}}, \{\mathbf{x}^{(i)}\}_{i=1}^{n}, g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a},0})$
5: Compute $w_{\mathbf{S}}$ using $V_{\mathbf{a}_{\max},\mathbf{S}}$ and $V_{\mathbf{a}_{\min},\mathbf{S}}$ as in Equation (7.2) ▷ Theorem 13
6: **function** $\textsc{LocalRegression}(g_{\mathbf{a}}, \{\mathbf{x}^{(i)}\}_{i=1}^{n}, \lambda, k)$
7: $\quad$ **Initialize**: $g_{\mathbf{a},0} \leftarrow \textsc{Mean}(\{g(\mathbf{x}^{(i)})\}_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n}), \widehat{g}_{\mathbf{a},\mathbf{S}} \leftarrow 0, \forall \mathbf{S} \in [k] \setminus \emptyset, |\mathbf{S}| \leq \lambda$
8: $\quad$ **while** each $\widehat{g}_{\mathbf{a},\mathbf{S}}$ does not converge **do**
9: $\quad\quad$ **for** each $\mathbf{S}$ **do**
10: $\quad\quad\quad \widehat{g}_{\mathbf{a},\mathbf{S}} \leftarrow \textsc{Smooth}[\{g_{\mathbf{a}}(\mathbf{x}^{(i)}) - g_{\mathbf{a},0} - \sum_{\mathbf{S} \neq \mathbf{S}'} \widehat{g}_{\mathbf{a},\mathbf{S}'}(\mathbf{x}_{\mathbf{S}}^{(i)})\}_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n}]$ ▷ Backfitting
11: $\quad\quad\quad \widehat{g}_{\mathbf{a},\mathbf{S}} \leftarrow \widehat{g}_{\mathbf{a},\mathbf{S}} - \textsc{Mean}(\{\widehat{g}_{\mathbf{a},\mathbf{S}}(\mathbf{x}_{\mathbf{S}}^{(i)})\}_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n})$ ▷ Mean centering
12:
13: $\quad$ **return** $g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a},0}$
14: **function** $\textsc{Covariance}(g_{\mathbf{a}}, \{\mathbf{x}^{(i)}\}_{i=1}^{n}, g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a},0})$
15:
16: $\quad$ **return** $\sum_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n} g_{\mathbf{a},\mathbf{S}}(\mathbf{x}_{\mathbf{S}}^{(i)})(g_{\mathbf{a}}(\mathbf{x}^{(i)}) - g_{\mathbf{a},0})$

Here, $g_{\mathbf{a},0}$ is a constant and $g_{\mathbf{a},\mathbf{S}}$ is a *component function* of $g_{\mathbf{a}}$ taking $\mathbf{X}_{\mathbf{S}}$ as input, and δ is the approximation error. For computational tractability, we consider only components of *maximum order* λ. FairXplainer deploys backfitting algorithm for learning component functions in Eq. (7.7).

Local Regression with Backfitting. We perform local regression with backfitting algorithm to learn the component functions up to order λ (Line 6–13). Backfitting

algorithm is an iterative algorithm, where in each iteration one component function, say $g_{\mathbf{a},\mathbf{S}}$, is learned while keeping other component functions fixed. Specifically, $g_{\mathbf{a},\mathbf{S}}$ is learned as a smoothed function of g and rest of the components $g_{\mathbf{a},\mathbf{S}'}$, where $\mathbf{S}' \neq \mathbf{S}$ is a non-empty subset of $[k] \setminus \emptyset$. To keep every component function mean centered, backfitting requires to impose the constraints $g_{\mathbf{a},0} = \textsc{Mean}(\{g(\mathbf{x}^{(i)})\}_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n})$ (Line 7), which is the mean of $g_{\mathbf{a}}$ evaluated on samples belonging to the sensitive group $\mathbf{A} = \mathbf{a}$; and $\sum_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n} g_{\mathbf{a},\mathbf{S}}(\mathbf{x}_{\mathbf{S}}^{(i)}) = 0$ (Line 11), where $\mathbf{x}_{\mathbf{S}}^{(i)}$ be the subset of feature values associated with feature indices $\mathbf{S}$ for the i-th sample $\mathbf{x}^{(i)}$. These constraints assign the expectation of $g_{\mathbf{a}}$ on the constant term $g_{\mathbf{a},0}$ and the variance of $g_{\mathbf{a}}$ to the component functions.

While performing local regression, backfitting uses a smoothing operator [209] over the set of samples (Line 10). A smoothing operator, referred as Smooth, allows us to learn a global representation of a component function by smoothly interpolating the local curves obtained by local regression [209]. In this chapter, we apply cubic spline smoothing [210] to learn each component function. Cubic spline is a piecewise polynomial of degree 3 with C^2 continuity. Hence, up to the second derivatives of each piecewise term are zero at the endpoints of intervals.

Variance and Covariance Computation. Once each component function $g_{\mathbf{a},\mathbf{S}}$ is learned with LocalRegression (Line 6–13), we compute variances of the component functions and their covariances with $g_{\mathbf{a}}$. Since each component function is mean centered (Line 11), we compute the variance of $g_{\mathbf{a},\mathbf{S}}$ on the dataset as $\mathsf{Var}[g_{\mathbf{a},\mathbf{S}}] = \sum_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n}(g_{\mathbf{a},\mathbf{S}}(\mathbf{x}_{\mathbf{S}}^{(i)}))^2$. Hence, variance captures the independent effect of $g_{\mathbf{a},\mathbf{S}}$. Covariance is computed to account for the correlation among features $\mathbf{X}$. We compute the covariance of $g_{\mathbf{a},\mathbf{S}}$ with $g_{\mathbf{a}}$ on the dataset as $\mathsf{Cov}[g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a}}] = \sum_{i=1,\mathbf{a}^{(i)}=\mathbf{a}}^{n} g_{\mathbf{a},\mathbf{S}}(\mathbf{x}_{\mathbf{S}}^{(i)})(g_{\mathbf{a}}(\mathbf{x}^{(i)}) - g_{\mathbf{a},0})$. Here, $g_{\mathbf{a}}(\cdot) - g_{\mathbf{a},0}$ is the mean centered form of $g_{\mathbf{a}}$. Covariance of $g_{\mathbf{a},\mathbf{S}}$ can be both positive and negative depending on whether the features $\mathbf{X}_{\mathbf{S}}$ are positively or negatively correlated with $g_{\mathbf{a}}$. Specifically, under the set additive model, we obtain $\mathsf{Cov}[g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a}}] = \mathsf{Var}[g_{\mathbf{a},\mathbf{S}}] + \mathsf{Cov}[g_{\mathbf{a},\mathbf{S}}, \sum_{\mathbf{S}\neq\mathbf{S}'} g_{\mathbf{a},\mathbf{S}'}]$. Now, we use $V_{\mathbf{a},\mathbf{S}} = \mathsf{Cov}[g_{\mathbf{a},\mathbf{S}}, g_{\mathbf{a}}]$ as the effective variance of $\mathbf{X}_{\mathbf{S}}$ for a given sensitive group $\mathbf{A} = \mathbf{a}$ (Line 14–16). In Line 1–5, we compute $V_{\mathbf{a},\mathbf{S}}$ for the most and the least favored groups, and plug them in Theorem 13 to compute FIF of $\mathbf{X}_{\mathbf{S}}$.

Extension of FairXplainer to Equalized Odds and Predictive Parity. We extend FairXplainer in computing FIF of group fairness metrics beyond statistical parity, namely equalized odds and predictive parity. For equalized odds, we deploy FairXplainer twice, one for computing FIFs on a subset of samples in the dataset where $Y = 1$ and another on samples with $Y = 0$. Then, the maximum of the sum of FIFs between $Y = 1$ and $Y = 0$ quantifies the equalized odds of the classifer.

To compute FIFs for predictive parity, we condition the dataset by the predicted class $\widehat{Y}$ and separate into two sub-datasets: $\widehat{Y} = 1$ and $\widehat{Y} = 0$. For each sub-dataset, we deploy

FairXplainer by setting the ground-truth class Y as label. This contrasts the computation for statistical parity and equalized odds, where the predicted class $\widehat{Y}$ is considered as label. Finally, the maximum of the sum of FIFs between two sub-datasets for $\widehat{Y} = 1$ and $\widehat{Y} = 0$ quantifies the predictive parity of the classifier.

7.4 Empirical Performance Analysis

In this section, we perform an empirical evaluation of FairXplainer. In the following, we discuss the experimental setup, the objectives of experiments, and experimental results.

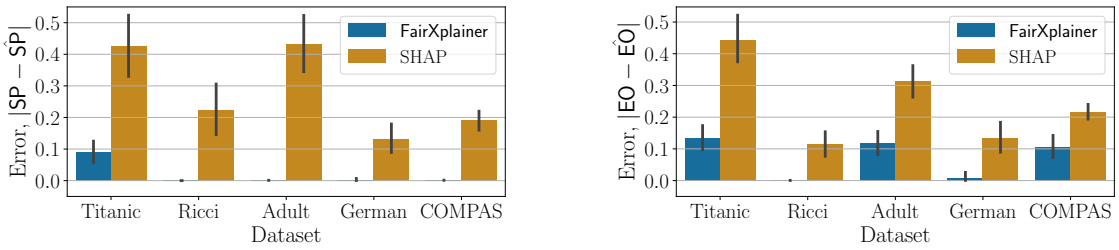
Experimental Setup. We implement a prototype of FairXplainer in Python (version 3.8). In FairXplainer, we deploy SAlib library [211] to compute FIFs leveraging techniques of global sensitivity analysis. In our experiments, we consider five widely studied datasets from fairness literature, namely COMPAS [196], Adult [212], German-credit [203], Ricci [197], and Titanic (https://www.kaggle.com/c/titanic). We deploy Scikit-learn [213] to learn different classifiers: Logistic Regression, Support Vector Machine, Decision Tree, and Neural Networks. In experiments, we specify FairXplainer to compute intersectional influences up to second order ($\lambda = 2$). We compare FairXplainer with Shapley-valued based FIF computational framework, referred as SHAP [2]. In addition, we deploy FairXplainer along with different fairness-enhancing [54] and fairness attack [60] algorithms, and analyze the effect of these algorithms on the FIFs and the resultant fairness metric. In the following, we discuss the objective of our empirical study.

1. How do **accuracies** of FairXplainer and SHAP compare for estimating FIFs and resulting bias?

2. What is the impact of **intersectional vs. individual FIFs** in tracing the sources of bias?

3. How do FIFs change while **applying** different fairness enhancing algorithms, i.e. **affirmative actions**, and fairness attacks, i.e. **punitive actions**?

In summary, FairXplainer *achieves significantly less estimation error than SHAP in all the datasets.* This shows the importance of adopting global sensitivity analysis to compute FIFs for group fairness metrics than local explanation approaches. Moreover, FairXplainer *effectively traces the sources of bias by computing intersectional FIFs, which earlier method like SHAP did not capture.* FairXplainer also detects the effects of the affirmative and punitive actions on the bias of a classifier and the corresponding tensions between different subsets of features.



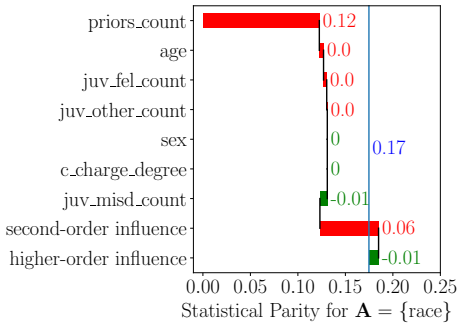
(a) Statistical parity

(b) Equalized odds

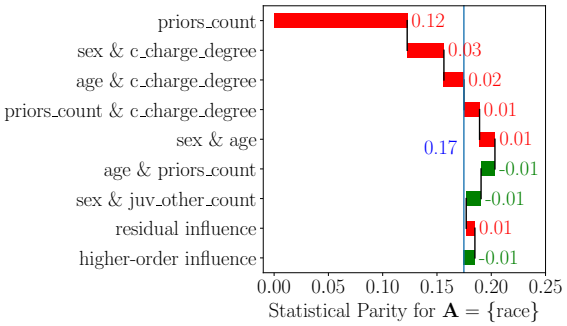
Figure 7.2: Comparing FairXplainer and SHAP on the estimation error of fairness metrics. Lower values on the Y-axis denote a better result. FairXplainer has significantly less error than SHAP.

Accurate Estimation of Bias. We compare FairXplainer with SHAP in estimating Statistical Parity (SP) and Equalized Odds (EO), where each metric is calculated by summing all FIFs (Axiom 1). We consider five datasets, each trained on different classifiers by applying five-fold cross-validation. We compute estimation error by taking the absolute difference between the exact and estimated value of a metric, and present results in Figure 7.2. In both metrics, FairXplainer demonstrates significantly less error than SHAP. For example, the mean error of FairXplainer in estimating SP on Titanic dataset is 0.1 vs. 0.42 for SHAP. In this context, the error of FairXplainer is often zero in most datasets and only insignificant (≤ 0.13) due to the degenerate cases (Section 7.2) with $\Pr[\widehat{Y} = 1|\mathbf{A} = \mathbf{a}] \in \{0, 1\}$. Therefore, *global sensitivity analysis based approach* FairXplainer *is significantly more accurate in computing FIFs and corresponding fairness metric than local explanation based approach SHAP.*

Individual vs. Intersectional FIFs. Now, we aim to understand the importance of intersectional FIFs over individual FIFs. We consider COMPAS dataset with race = {Caucasian, non-Caucasian} as the sensitive feature, and a logistic regression classifier to predict whether a person will re-offend crimes within the next two years. Since the classifier only optimizes training error, it demonstrates statistical parity as 0.17, which means a non-Caucasian has 0.17 higher probability of re-offending crimes than a Caucasian. Next, we investigate the source of bias and present individual FIFs in Figure 7.3a, and both individual and intersectional FIFs in Figure 7.3b. In both figures, we show top seven influential FIFs sorted by absolute values, followed by residual higher-order FIFs. In Figure 7.3a, 'priors count' of an individual is dominating in increasing statistical parity (FIF = 0.12). Other non-sensitive features appear to have almost zero FIFs. However, the sum of second-order influences of all features increases statistical parity by 0.06, denoting that the data is highly correlated and presenting only individual FIFs does not trace the true sources of bias. For example, while both 'sex' and 'age' of persons have zero influence on bias (Figure 7.3a), these features together with 'c charge degree', 'priors count', and 'juvenile other count' contribute highly on statistical parity (sum of FIFs

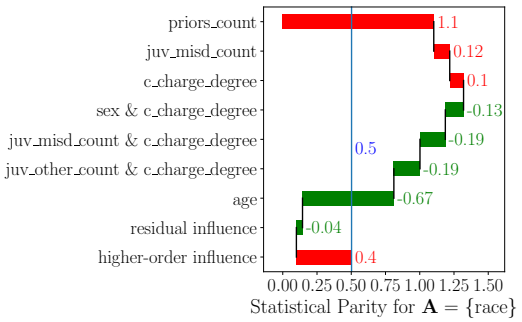


(a) Individual FIFs

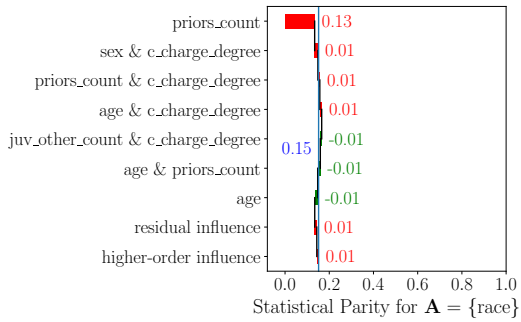


(b) Individual and intersectional FIFs

Figure 7.3: FIFs for COMPAS dataset on explaining statistical parity. Intersectional FIFs depict sources of bias in detail than individual FIFs.



(a) Fairness attack (punitive actions)



(b) Fairness enchancing algorithm (affirmative actions)

Figure 7.4: Effects of a fairness attack [60] and a fairness enhancing algorithm [54] on FIFs.

$= 0.03 + 0.02 + 0.01 + 0.01 - 0.01 - 0.01 = 0.05$). *Therefore, intersectional influences together with individual influences lead to a clearer understanding on the source of bias of a classifier. Note that, unlike SHAP,* FairXplainer *is the only framework that computes beyond individual FIFs.*

Fairness affirmative/punitive actions. Continuing on the experiment in Figure 7.3, we evaluate the effect of fairness attack and enhancing algorithms on FIFs for COMPAS dataset in Figure 7.4. In Figure 7.3, statistical parity of the classifier is 0.17. Applying a data poisoning fairness attack [60] increases statistical parity to 0.5 and the data reweighing-based fairness-enhancing algorithm [54] decreases it to 0.15. We observe that in both cases, there are a subset of features decreasing bias while another subset of features increasing it, e.g. 'age' vs. 'priors count'. Figure 7.4a suggests that the attack algorithm would be more successful if it could hide the influence of 'age' of a person in receiving discriminating prediction for re-offending crimes. Figure 7.4b suggests that the fairness enhancing algorithm can improve by ameliorating the effect of 'priors count' further. Thus, FairXplainer *provides a dissecting tool to undertake necessary steps to improve or worsen*

fairness of the classifier.

7.5 Chapter Summary

We propose the Fairness Influence Function (FIF) to quantify the contribution of input features on the resulting bias of a classifier for a given dataset. Relying on an additive axiom, we express group fairness metrics computed for sensitive groups as the sum of FIFs of all the subsets of non-sensitive features. To compute FIFs, we first prove existing group fairness metrics as the scaled difference of the conditional variances in the predictions of the classifier and then apply variance decomposition based on global sensitivity analysis. Finally, we propose FairXplainer, an algorithm for efficiently and accurately computing FIFs by deploying a local regression to learn a set-additive decomposition of the classifier. The experimental results show that FairXplainer estimates bias with significantly higher accuracy than the local explanation based approach SHAP. Also, FairXplainer computes both individual and intersectional FIFs unlike SHAP to yield a better understanding of the sources of bias. In future, we aim to develop algorithms that leverage FIFs to yield unbiased decisions.

Part IV

Epilogue

Chapter 8

Conclusion And Future Work

In the past decade, the advancement of machine learning has witnessed a host of applications in different safety-critical domains. In such domains, the interpretability, fairness, robustness, and privacy etc. of classifiers are considered important in trustworthy and responsible AI (artificial intelligence). In this thesis, we focus on the interpretability and fairness aspects of machine learning, where we prioritize on improving the scalability and accuracy of the underlying problems. Based on formal methods, the contributions of our thesis are summarized as follows. (i) In interpretable machine learning, we design incremental learning technique for interpretable rule-based classifiers. (ii) In fairness in machine learning, we develop a formal fairness verification framework to verify multiple fairness definitions, algorithms, and classifiers; and we develop techniques to identify the sources of unfairness of classifiers by computing fairness influence functions.

To validate the efficiency of our methods, we develop open-source tools and conduct experiments on real-world interpretability and fairness datasets in machine learning. In interpretable machine learning, our framework IMLI scales classification to million-size datasets while achieving competitive prediction accuracy and rule-size compared to existing rule-based classifiers. In our quest for learning more expressible interpretable classifiers, we develop another learning framework CRR for logical relaxed classification rules based on incremental learning. In experiments, CRR learns more concise and accurate rule-based classifiers while demonstrating scalability owing to the incremental learning.

In fairness in machine learning, our probabilistic fairness verifier Justicia and FVGM demonstrates a superior performance in accuracy and scalability than the state of the art verifiers. Being a stochastic-SAT-based verifier, Justicia verifies fairness of compound sensitive groups of Boolean classifiers such as decision trees with higher scalability. FVGM, in addition, considers correlated features and verifies the fairness of linear classifiers with better scalability and accuracy than existing verifiers. Finally, given a classifier and a dataset, our framework FairXplainer computes fairness influences of each subset of non-sensitive features as their contribution to the incurred bias of the classifier. In experiments, FairXplainer estimates bias with significantly higher accuracy than the existing local explanation approach, while enabling computation of intersectional influences as a

mean of better understanding the sources fo bias.

As a future work, we continue our research on fair and interpretable machine learning. An immediate research question is to design a fairness improvement or repair algorithm of machine learning classifiers, which benefits from fairness verifier such as Justicia and bias identification framework such as FairXplainer. Another research question is to extend fairness verification to more complex classifiers, for example, neural networks. In addition, in future, we plan to deploy FairXplainer on image classifiers, language models, and computer vision tasks. In a separate line of work, we aim to apply incremental solving, as presented in the context of interpretable classification learning, on other optimization problems, even beyond machine learning. Therefore, the long term vision of this thesis is to improve machine learning in interpretability, fairness and beyond, with the assistance of formal methods and automated reasoning.

Bibliography

[1] T. Begley, T. Schwedes, C. Frye and I. Feige, *Explainability for fair machine learning*, arXiv preprint arXiv:2010.07389 (2020).

[2] S. M. Lundberg, *Explaining quantitative measures of fairness*, in Fair & Responsible AI Workshop@ CHI2020, 2020.

[3] B. J. Erickson, P. Korfiatis, Z. Akkus and T. L. Kline, *Machine learning for medical imaging*, vol. 37, Radiological Society of North America, 2017, pp. 505–515.

[4] G. A. Kaissis, M. R. Makowski, D. Rückert and R. F. Braren, *Secure, privacy-preserving and federated machine learning in medical imaging*, vol. 2, Nature Publishing Group, 2020, pp. 305–311.

[5] I. Kononenko, *Machine learning for medical diagnosis: history, state of the art and perspective*, vol. 23, Elsevier, 2001, pp. 89–109.

[6] R. S. S. Kumar, D. R. O'Brien, K. Albert and S. Vilojen, *Law and adversarial machine learning*, 2018.

[7] H. Surden, *Machine learning and law*, vol. 89, HeinOnline, 2014, p. 87.

[8] R. Luckin, *Machine Learning and Human Intelligence: The future of education for the 21st century.*, ERIC, 2018.

[9] I. Peled, F. Rodrigues and F. C. Pereira, *Model-based machine learning for transportation*, in Mobility patterns, big data and transport analytics, Elsevier, 2019, pp. 145–171.

[10] F. Zantalis, G. Koulouras, S. Karabetsos and D. Kandris, *A review of machine learning and IoT in smart transportation*, vol. 11, Multidisciplinary Digital Publishing Institute, 2019, p. 94.

[11] B. Eshete, *Making machine learning trustworthy*, Science **373** (2021), pp. 743–744.

[12] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*, vol. 1, Nature Publishing Group, 2019, pp. 206–215.

[13] S. Barocas, M. Hardt and A. Narayanan, *Fairness in machine learning*, NIPS Tutorial **1** (2017).

[14] J. Rauber, W. Brendel and M. Bethge, *Foolbox: A python toolbox to benchmark the robustness of machine learning models*, arXiv preprint arXiv:1707.04131 (2017).

[15] N. Papernot, P. McDaniel, A. Sinha and M. Wellman, *Towards the science of security and privacy in machine learning*, arXiv preprint arXiv:1611.03814 (2016).

[16] N. Gill, P. Hall, K. Montgomery and N. Schmidt, *A responsible machine learning workflow with focus on interpretable models, post-hoc explanation, and discrimination testing*, vol. 11, Multidisciplinary Digital Publishing Institute, 2020, p. 137.

[17] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions*, in Proceedings of the 31st international conference on neural information processing systems, 2017, pp. 4768–4777.

[18] M. Moradi and M. Samwald, *Post-hoc explanation of black-box classifiers using confident itemsets*, vol. 165, Elsevier, 2021, p. 113941.

[19] M. T. Ribeiro, S. Singh and C. Guestrin, *" why should i trust you?" explaining the predictions of any classifier*, in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1135–1144.

[20] D. Slack, S. Hilgard, E. Jia, S. Singh and H. Lakkaraju, *Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods*, in Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, 2020, pp. 180–186.

[21] C. Bessiere, E. Hebrard and B. O'Sullivan, *Minimising decision tree size as combinatorial optimisation*, in International Conference on Principles and Practice of Constraint Programming, Springer, 2009, pp. 173–187.

[22] S. Dash and J. Goncalves, *LPRules: Rule induction in knowledge graphs using linear programming*, 2021.

[23] A. Ignatiev, J. Marques-Silva, N. Narodytska and P. J. Stuckey, *Reasoning-based learning of interpretable ML models*, in International Joint Conference on Artificial Intelligence (IJCAI), 2021.

[24] Y. Izza, A. Ignatiev and J. Marques-Silva, *On explaining decision trees*, 2020.

[25] H. Lakkaraju, E. Kamar, R. Caruana and J. Leskovec, *Interpretable & explorable approximations of black box models*, 2017.

[26] H. Lakkaraju, S. H. Bach and J. Leskovec, *Interpretable decision sets: A joint framework for description and prediction*, in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1675–1684.

[27] B. Letham, C. Rudin, T. H. McCormick and D. Madigan, *Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model*, vol. 9, Institute of Mathematical Statistics, 2015, pp. 1350–1371.

[28] N. Narodytska, A. Ignatiev, F. Pereira, J. Marques-Silva and I. RAS, *Learning optimal decision trees with sat.*, in IJCAI, 2018, pp. 1362–1368.

[29] R. L. Rivest, *Learning decision lists*, vol. 2, Springer, 1987, pp. 229–246.

[30] F. Wang and C. Rudin, *Falling rule lists*, in Artificial Intelligence and Statistics, PMLR, 2015, pp. 1013–1022.

[31] J. Yu, A. Ignatiev, P. L. Bodic and P. J. Stuckey, *Optimal decision lists using SAT*, 2020.

[32] B. F. Gage, A. D. Waterman, W. Shannon, M. Boechler, M. W. Rich and M. J. Radford, *Validation of clinical classification schemes for predicting stroke: results from the national registry of atrial fibrillation*, Jama **285** (2001), pp. 2864–2870.

[33] H. Lakkaraju, E. Kamar, R. Caruana and J. Leskovec, *Faithful and customizable explanations of black box models*, in Proc. of AIES, 2019.

[34] D. Malioutov and K. Varshney, *Exact rule learning via boolean compressed sensing*, in International Conference on Machine Learning, PMLR, 2013, pp. 765–773.

[35] I. B. Myers, *The Myers-Briggs type indicator: Manual (1962).*, Consulting Psychologists Press, 1962.

[36] C. Sinz, *Towards an optimal cnf encoding of boolean cardinality constraints*, in International conference on principles and practice of constraint programming, Springer, 2005, pp. 827–831.

[37] P. Clark and T. Niblett, *The CN2 induction algorithm*, Mar. 1989.

[38] W. W. Cohen and Y. Singer, *A simple, fast, and effective rule learner*, in Proc. of AAAI, Orlando, FL, Jul. 1999.

[39] J. R. Quinlan, *C4. 5: Programming for machine learning*, 1993.

[40] D. Malioutov and K. S. Meel, *MLIC: A MaxSAT-based framework for learning interpretable classification rules*, in International Conference on Principles and Practice of Constraint Programming, Springer, 2018, pp. 312–327.

[41] B. Ghosh, D. Malioutov and K. S. Meel, *Efficient learning of interpretable classification rules*, in Proc. of JAIR, 2022.

[42] B. Ghosh and K. S. Meel, *IMLI: An incremental framework for MaxSAT-based learning of interpretable classification rules*, in Proceedings of AAAI/ACM Conference on AI, Ethics, and Society(AIES), 2019.

[43] B. Ghosh, D. Malioutov and K. S. Meel, *Classification rules in relaxed logical form*, in Proceedings of ECAI, 2020.

[44] B. Martinez Neda, Y. Zeng and S. Gago-Masague, *Using machine learning in admissions: Reducing human and algorithmic bias in the selection process*, in Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 2021, pp. 1323–1323.

[45] N. Tollenaar and P. Van der Heijden, *Which method predicts recidivism best?: a comparison of statistical, machine learning and data mining predictive models*, Journal of the Royal Statistical Society: Series A (Statistics in Society) **176** (2013), pp. 565–584.

[46] I. Ajunwa, S. Friedler, C. E. Scheidegger and S. Venkatasubramanian, *Hiring by algorithm: predicting and preventing disparate impact*, Available at SSRN (2016). URL: http://sorelle.friedler.net/papers/SSRN-id2746078.pdf.

[47] C. Dwork, M. Hardt, T. Pitassi, O. Reingold and R. Zemel, *Fairness through awareness*, in Proceedings of the 3rd innovations in theoretical computer science conference, 2012, pp. 214–226.

[48] R. Berk, *Accuracy and fairness for juvenile justice risk assessments*, Journal of Empirical Legal Studies **16** (2019), pp. 175–194.

[49] F. J. Landy, J. L. Barnes and K. R. Murphy, *Correlates of perceived fairness and accuracy of performance evaluation.*, Journal of Applied psychology **63** (1978), p. 751.

[50] I. Zliobaite, *On the relation between accuracy and fairness in binary classification*, arXiv preprint arXiv:1505.05723 (2015).

[51] P. Besse, E. del Barrio, P. Gordaliza, J.-M. Loubes and L. Risser, *A survey of bias in machine learning through the prism of statistical parity*, The American Statistician (2021), pp. 1–11.

[52] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger and S. Venkatasubramanian, *Certifying and removing disparate impact*, in proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, 2015, pp. 259–268.

[53] P. Garg, J. Villasenor and V. Foggo, *Fairness metrics: A comparative analysis*, in 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 3662–3666.

[54] F. Kamiran and T. Calders, *Data preprocessing techniques for classification without discrimination*, Knowledge and Information Systems **33** (2012), pp. 1–33.

[55] R. Zemel, Y. Wu, K. Swersky, T. Pitassi and C. Dwork, *Learning fair representations*, in International Conference on Machine Learning, 2013, pp. 325–333.

[56] F. Calmon, D. Wei, B. Vinzamuri, K. N. Ramamurthy and K. R. Varshney, *Optimized pre-processing for discrimination prevention*, in Advances in Neural Information Processing Systems, 2017, pp. 3992–4001.

[57] B. H. Zhang, B. Lemoine and M. Mitchell, *Mitigating unwanted biases with adversarial learning*, in Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, 2018, pp. 335–340.

[58] F. Kamiran, A. Karim and X. Zhang, *Decision theory for discrimination-aware classification*, in 2012 IEEE 12th International Conference on Data Mining, IEEE, 2012, pp. 924–929.

[59] M. Hardt, E. Price and N. Srebro, *Equality of opportunity in supervised learning*, in Advances in neural information processing systems, 2016, pp. 3315–3323.

[60] D. Solans, B. Biggio and C. Castillo, *Poisoning attacks on algorithmic fairness*, arXiv preprint arXiv:2004.07401 (2020).

[61] R. K. E. Bellamy et al., *AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias*, Oct. 2018. URL: https://arxiv.org/abs/1810.01943.

[62] A. Albarghouthi, L. D'Antoni, S. Drews and A. V. Nori, *FairSquare: probabilistic verification of program fairness*, Proceedings of the ACM on Programming Languages **1** (2017), pp. 1–30.

[63] O. Bastani, X. Zhang and A. Solar-Lezama, *Probabilistic verification of fairness properties via concentration*, Proceedings of the ACM on Programming Languages **3** (2019), pp. 1–27.

[64] M. L. Littman, S. M. Majercik and T. Pitassi, *Stochastic boolean satisfiability*, Journal of Automated Reasoning **27** (2001), pp. 251–296.

[65] G. Pleiss, M. Raghavan, F. Wu, J. Kleinberg and K. Q. Weinberger, *On fairness and calibration*, arXiv preprint arXiv:1709.02012 (2017).

[66] M. B. Zafar, I. Valera, M. G. Rogriguez and K. P. Gummadi, *Fairness constraints: Mechanisms for fair classification*, in Artificial Intelligence and Statistics, 2017, pp. 962–970.

[67] J. Dressel and H. Farid, *The accuracy, fairness, and limits of predicting recidivism*, Science advances **4** (2018), p. eaao5580.

[68] P. G. John, D. Vijaykeerthy and D. Saha, *Verifying individual fairness in machine learning models*, arXiv preprint arXiv:2006.11737 (2020).

[69] W. Pan, S. Cui, J. Bian, C. Zhang and F. Wang, *Explaining algorithmic fairness through fairness-aware causal path decomposition*, in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 1287–1297.

[70] J. Zhang and E. Bareinboim, *Fairness in decision-making—the causal explanation formula*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[71] W. Zhang and E. Ntoutsi, *Faht: an adaptive fairness-aware decision tree classifier*, arXiv preprint arXiv:1907.07237 (2019).

[72] X. Hua, H. Xu, J. Blanchet and V. Nguyen, *Human imperceptible attacks and applications to improve fairness*, arXiv preprint arXiv:2111.15603 (2021).

[73] N. Mehrabi, M. Naveed, F. Morstatter and A. Galstyan, *Exacerbating algorithmic bias through fairness attacks*, arXiv preprint arXiv:2012.08723 (2020).

[74] B. Benhamou, L. Sais and P. Siegel, *Two proof procedures for a cardinality based language in propositional calculus*, in Annual Symposium on Theoretical Aspects of Computer Science, Springer, 1994, pp. 71–82.

[75] S. M. Majercik and B. Boots, *Dc-ssat: a divide-and-conquer approach to solving stochastic satisfiability problems efficiently*, in AAAI, 2005, pp. 416–422.

[76] D. J. Fremont, M. N. Rabe and S. A. Seshia, *Maximum model counting.*, in AAAI, 2017, pp. 3885–3892.

[77] J. Huang et al., *Combining knowledge compilation and search for conformant probabilistic planning.*, in ICAPS, 2006, pp. 253–262.

[78] N.-Z. Lee, Y.-S. Wang and J.-H. R. Jiang, *Solving stochastic boolean satisfiability under random-exist quantification.*, in IJCAI, 2017, pp. 688–694.

[79] N.-Z. Lee, Y.-S. Wang and J.-H. R. Jiang, *Solving exist-random quantified stochastic boolean satisfiability via clause selection.*, in IJCAI, 2018, pp. 1339–1345.

[80] T. Sang, F. Bacchus, P. Beame, H. A. Kautz and T. Pitassi, *Combining component caching and clause learning for effective model counting.*, SAT **4** (2004), p. 7th.

[81] S. Chakraborty, K. S. Meel and M. Y. Vardi, *A scalable approximate model counter*, in International Conference on Principles and Practice of Constraint Programming, Springer, 2013, pp. 200–216.

[82] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia and M. Y. Vardi, *Distribution-aware sampling and weighted model counting for sat*, arXiv preprint arXiv:1404.2984 (2014).

[83] J. Kleinberg and E. Tardos, *Algorithm design* (2006).

[84] R. Nabi and I. Shpitser, *Fair inference on outcomes*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[85] A. Chouldechova and A. Roth, *A snapshot of the frontiers of fairness in machine learning*, Communications of the ACM **63** (2020), pp. 82–89.

[86] S. Verma and J. Rubin, *Fairness definitions explained*, in 2018 IEEE/ACM International Workshop on Software Fairness (FairWare), IEEE, 2018, pp. 1–7.

[87] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*, MIT press, 2009.

[88] J. Pearl, *Bayesian netwcrks: A model cf self-activated memory for evidential reasoning*, in Proceedings of the 7th conference of the Cognitive Science Society, University of California, Irvine, CA, USA, 1985, pp. 15–17.

[89] M. Chavira and A. Darwiche, *On probabilistic inference by weighted model counting*, Artificial Intelligence **172** (2008), pp. 772–799.

[90] S. K. Murakonda, R. Shokri and G. Theodorakopoulos, *Quantifying the privacy risks of learning high-dimensional graphical models*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 2287–2295.

[91] A. Saltelli et al., *Global sensitivity analysis: the primer*, John Wiley & Sons, 2008.

[92] European Commission, *Better regulation toolbox*, 2021.

[93] Office of the Science Advisor, Council for Regulatory Environmental Modeling, *Guidance on the development, evaluation, and application of environmental models*, 2009. https://web.archive.org/web/20110426180258/http://www.epa.gov/CREM/library/cred_guidance_0309.pdf.

[94] A. Saltelli et al., *Five ways to ensure that models serve society: a manifesto*, 2020.

[95] I. M. Sobol', *On sensitivity estimation for nonlinear mathematical models*, Matematicheskoe modelirovanie **2** (1990), pp. 112–118.

[96] I. M. Sobol', *Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates*, Mathematics and computers in simulation **55** (2001), pp. 271–280.

[97] A. N. Bhagoji, D. Cullina, C. Sitawarin and P. Mittal, *Enhancing robustness of machine learning systems via data transformations*, in 2018 52nd Annual Conference on Information Sciences and Systems (CISS), IEEE, 2018, pp. 1–5.

[98] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, 2017.

[99] M. Du, N. Liu and X. Hu, *Techniques for interpretable machine learning*, vol. 63, ACM New York, NY, USA, 2019, pp. 68–77.

[100] L. Hancox-Li, *Robustness in machine learning explanations: does it matter?*, in Proceedings of the 2020 conference on fairness, accountability, and transparency, 2020, pp. 640–647.

[101] K. Holstein, J. Wortman Vaughan, H. Daumé III, M. Dudik and H. Wallach, *Improving fairness in machine learning systems: What do industry practitioners need?*, in Proceedings of the 2019 CHI conference on human factors in computing systems, 2019, pp. 1–16.

[102] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman and A. Galstyan, *A survey on bias and fairness in machine learning*, vol. 54, ACM New York, NY, USA, 2021, pp. 1–35.

[103] C. Molnar, *Interpretable machine learning*, Lulu. com, 2020.

[104] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl and B. Yu, *Interpretable machine learning: definitions, methods, and applications*, 2019.

[105] A. F. M. Agarap, *On breast cancer detection: an application of machine learning algorithms on the wisconsin diagnostic dataset*, in Proceedings of the 2nd international conference on machine learning and soft computing, 2018, pp. 5–9.

[106] N. Robinson, C. Gretton, D. N. Pham and A. Sattar, *Partial weighted MaxSAT for optimal planning*, in Pacific rim international conference on artificial intelligence, Springer, 2010, pp. 231–243.

[107] R. Walter, C. Zengler and W. Küchlin, *Applications of MaxSAT in automotive configuration.*, in Configuration Workshop, vol. 1, Citeseer, 2013, p. 21.

[108] L. Ciampiconi, B. Ghosh, J. Scarlett and K. S. Meel, *A MaxSAT-based framework for group testing*, in Proceedings of AAAI, 2020.

[109] O. J. Berg, A. J. Hyttinen and M. J. Järvisalo, *Applications of MaxSAT in data analysis*, EasyChair Publications, 2019.

[110] P.-C. K. Lin and S. P. Khatri, *Application of Max-SAT-based ATPG to optimal cancer therapy design*, vol. 13, Springer, 2012, pp. 1–10.

[111] J. R. Quinlan, *Induction of decision trees*, vol. 1, Springer, 1986, pp. 81–106.

[112] J. R. Quinlan, *Simplifying decision trees*, vol. 27, Elsevier, 1987, pp. 221–234.

[113] W. W. Cohen, *Fast effective rule induction*, in Machine learning proceedings 1995, Elsevier, 1995, pp. 115–123.

[114] G. Su, D. Wei, K. R. Varshney and D. M. Malioutov, *Learning sparse two-level Boolean rules*, in 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2016, pp. 1–6.

[115] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl and P. MacNeille, *A Bayesian framework for learning rule sets for interpretable classification*, vol. 18, JMLR. org, 2017, pp. 2357–2393.

[116] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer and C. Rudin, *Learning certifiably optimal rule lists for categorical data*, The Journal of Machine Learning Research **18** (2017), pp. 8753–8830.

[117] J. P. M. Silva and K. A. Sakallah, *GRASP—a new search algorithm for satisfiability*, in The Best of ICCAD, Springer, 2003, pp. 73–89.

[118] J. Alos, C. Ansotegui and E. Torres, *Learning optimal decision trees using MaxSAT*, 2021.

[119] M. Janota and A. Morgado, *SAT-based encodings for optimal decision trees with explicit paths*, in International Conference on Theory and Applications of Satisfiability Testing, Springer, 2020, pp. 501–518.

[120] P. Shati, E. Cohen and S. McIlraith, *SAT-based approach for learning optimal decision trees with non-binary features*, in 27th International Conference on Principles and Practice of Constraint Programming (CP 2021), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[121] A. Ignatiev, F. Pereira, N. Narodytska and J. Marques-Silva, *A SAT-based approach to learn explainable decision sets*, in International Joint Conference on Automated Reasoning, Springer, 2018, pp. 627–645.

[122] A. Ignatiev, E. Lam, P. J. Stuckey and J. Marques-Silva, *A scalable two stage approach to computing optimal decision sets*, 2021.

[123] A. Schidler and S. Szeider, *SAT-based decision tree learning for large data sets*, in Proceedings of AAAI, vol. 21, 2021.

[124] J. Yu, A. Ignatiev, P. J. Stuckey and P. L. Bodic, *Computing optimal decision sets with sat*, arXiv preprint arXiv:2007.15140 (2020).

[125] N. A. Syed, S. Huan, L. Kah and K. Sung, *Incremental learning with support vector machines*, Citeseer, 1999.

[126] S. Ruping, *Incremental learning with support vector machines*, in Proc. of ICDM, 2001.

[127] G. Cauwenberghs and T. Poggio, *Incremental and decremental support vector machine learning*, in Proc. of NIPS, 2001.

[128] L. Ralaivola and F. d'Alché Buc, *Incremental support vector machine learning: A local approach*, in Proc. of ICANN, 2001.

[129] G. Hinton, N. Srivastava and K. Swersky, *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*, vol. 14, 2012, p. 2.

[130] M. Li, T. Zhang, Y. Chen and A. J. Smola, *Efficient mini-batch training for stochastic optimization*, in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 661–670.

[131] D. Masters and C. Luschi, *Revisiting small batch training for deep neural networks*, 2018.

[132] J. Konečnỳ, B. McMahan and D. Ramage, *Federated optimization: Distributed optimization beyond the datacenter*, 2015.

[133] J. Konečnỳ, H. B. McMahan, D. Ramage and P. Richtárik, *Federated optimization: Distributed machine learning for on-device intelligence*, 2016.

[134] M. L. Fisher, *The lagrangian relaxation method for solving integer programming problems*, vol. 27, INFORMS, 1981, pp. 1–18.

[135] J. K. Johnson, D. M. Malioutov and A. S. Willsky, *Lagrangian relaxation for MAP estimation in graphical models*, 2007.

[136] C. Lemaréchal, *Lagrangian relaxation*, in Computational combinatorial optimization, Springer, 2001, pp. 112–156.

[137] N. Barakat and J. Diederich, *Learning-based rule-extraction from support vector machines*, not found, 2004.

[138] N. Barakat and J. Diederich, *Eclectic rule-extraction from support vector machines*, vol. 2, Citeseer, 2005, pp. 59–62.

[139] J. Diederich, *Rule extraction from support vector machines: An introduction*, in Rule extraction from support vector machines, Springer, 2008, pp. 3–31.

[140] D. Martens, J. Huysmans, R. Setiono, J. Vanthienen and B. Baesens, *Rule extraction from support vector machines: an overview of issues and application in credit scoring*, Springer, 2008, pp. 33–63.

[141] H. Núñez, C. Angulo and A. Català, *Rule extraction from support vector machines.*, in Esann, 2002, pp. 107–112.

[142] M. G. Augasta and T. Kathirvalavakumar, *Rule extraction from neural networks—a comparative study*, in International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012), IEEE, 2012, pp. 404–408.

[143] T. Hailesilassie, *Rule extraction algorithm for deep neural networks: A review*, 2016.

[144] R. Setiono and H. Liu, *Understanding neural networks via rule extraction*, in IJCAI, vol. 1, Citeseer, 1995, pp. 480–485.

[145] M. Sato and H. Tsukimoto, *Rule extraction from neural networks via decision tree induction*, in IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222), vol. 3, IEEE, 2001, pp. 1870–1875.

[146] J. R. Zilke, E. L. Mencía and F. Janssen, *Deepred–rule extraction from deep neural networks*, in International Conference on Discovery Science, Springer, 2016, pp. 457–473.

[147] Z.-H. Zhou, *Rule extraction: Using neural networks or for neural networks?*, vol. 19, Springer, 2004, pp. 249–253.

[148] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini and F. Giannotti, *Local rule-based explanations of black box decision systems*, 2018.

[149] E. Pastor and E. Baralis, *Explaining black box models by means of local rules*, in Proceedings of the 34th ACM/SIGAPP symposium on applied computing, 2019, pp. 510–517.

[150] D. Rajapaksha, C. Bergmeir and W. Buntine, *LoRMIkA: Local rule-based model interpretability with K-optimal associations*, vol. 540, Elsevier, 2020, pp. 221–241.

[151] M. T. Ribeiro, S. Singh and C. Guestrin, *Anchors: High-precision model-agnostic explanations*, in Proceedings of the AAAI conference on artificial intelligence, vol. 32, 2018.

[152] T. Philipp and P. Steinke, *Pblib–a library for encoding pseudo-boolean constraints into cnf*, in International Conference on Theory and Applications of Satisfiability Testing, Springer, 2015, pp. 9–16.

[153] J. Fürnkranz, *Separate-and-conquer rule learning*, vol. 13, Springer, 1999, pp. 3–54.

[154] R. Martins, V. Manquinho and I. Lynce, *Open-WBO: A modular MaxSAT solver*, in International Conference on Theory and Applications of Satisfiability Testing, Springer, 2014, pp. 438–445.

[155] D. Dua and C. Graff, *UCI machine learning repository*, 2017. URL: http://archive.ics.uci.edu/ml.

[156] J. Vanschoren, J. N. van Rijn, B. Bischl and L. Torgo, *OpenML: Networked science in machine learning*, vol. 15, New York, NY, USA, 2013, ACM, pp. 49–60. URL: http://doi.acm.org/10.1145/2641190.2641198, doi: 10.1145/2641190.2641198.

[157] J. Argelich, C.-M. Li, F. Manya and J. Planes, *The first and second Max-SAT evaluations*, vol. 4, IOS Press, 2008, pp. 251–278.

[158] T. Balyo, M. Heule and M. Jarvisalo, *SAT competition 2016: Recent developments*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, 2017.

[159] I. Kononenko, *Machine learning for medical diagnosis: history, state of the art and perspective*, Artificial Intelligence in medicine **23** (2001), pp. 89–109.

[160] M. Možina, J. Žabkar, T. Bench-Capon and I. Bratko, *Argument based machine learning applied to law*, Artificial Intelligence and Law **13** (2005), pp. 53–73.

[161] P. Srikanth, C. Anusha and D. Devarapalli, *A computational intelligence technique for effective medical diagnosis using decision tree algorithm*, i-Manager's Journal on Computer Science **3** (2015), p. 21.

[162] H. Surden, *Machine learning and law*, Wash. L. Rev. **89** (2014), p. 87.

[163] N. Tollenaar and P. Van der Heijden, *Which method predicts recidivism best?: a comparison of statistical, machine learning and data mining predictive models*, Journal of the Royal Statistical Society: Series A (Statistics in Society) **176** (2013), pp. 565–584.

[164] J. Dressel and H. Farid, *The accuracy, fairness, and limits of predicting recidivism*, Science advances **4** (2018), p. eaao5580.

[165] A. Vellido, J. D. Martín-Guerrero and P. J. Lisboa, *Making machine learning models interpretable.*, in ESANN, vol. 12, Citeseer, 2012, pp. 163–172.

[166] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl and P. MacNeille, *Or's of and's for interpretable classification, with application to context-aware recommender systems*, arXiv preprint arXiv:1504.07614 (2015).

[167] J. Zeng, B. Ustun and C. Rudin, *Interpretable classification models for recidivism prediction*, Journal of the Royal Statistical Society: Series A (Statistics in Society) **180** (2017), pp. 689–722.

[168] H. Lakkaraju, E. Kamar, R. Caruana and J. Leskovec, *Faithful and customizable explanations of black box models*, in Proc. of AIES, 2019.

[169] F. Wang and C. Rudin, *Falling rule lists*, in Artificial Intelligence and Statistics, 2015, pp. 1013–1022.

[170] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl and P. MacNeille, *A bayesian framework for learning rule sets for interpretable classification*, The Journal of Machine Learning Research **18** (2017), pp. 2357–2393.

[171] D. Malioutov and K. S. Meel, *MLIC: A maxsat-based framework for learning interpretable classification rules*, in Proceedings of International Conference on Constraint Programming (CP), 08 2018.

[172] K. C. Briggs, *Myers-Briggs type indicator*, Consulting Psychologists Press Palo Alto, CA, 1976.

[173] A. Gawande, *Checklist manifesto, the (HB)*, Penguin Books India, 2010.

[174] M. Craven and J. W. Shavlik, *Extracting tree-structured representations of trained networks*, in Advances in neural information processing systems, 1996, pp. 24–30.

[175] A. Emad, K. R. Varshney and D. M. Malioutov, *A semiquantitative group testing approach for learning interpretable clinical prediction rules*, in Proc. Signal Process. Adapt. Sparse Struct. Repr. Workshop, Cambridge, UK, 2015.

[176] W. W. Cohen, *Fast effective rule induction*, in Machine Learning Proceedings 1995, Elsevier, 1995, pp. 115–123.

[177] D. Malioutov and K. Varshney, *Exact rule learning via boolean compressed sensing*, in International Conference on Machine Learning, 2013, pp. 765–773.

[178] U. Fayyad and K. Irani, *Multi-interval discretization of continuous-valued attributes for classification learning* (1993).

[179] R. Martins, V. Manquinho and I. Lynce, *Open-wbo: A modular maxsat solver*, in International Conference on Theory and Applications of Satisfiability Testing, Springer, 2014, pp. 438–445.

[180] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, *The weka data mining software: an update*, ACM SIGKDD explorations newsletter **11** (2009), pp. 10–18.

[181] F. Pedregosa et al., *Scikit-learn: Machine learning in python*, Journal of machine learning research **12** (2011), pp. 2825–2830.

[182] S. Dash, O. Gunluk and D. Wei, *Boolean decision rules via column generation*, in Advances in Neural Information Processing Systems, 2018, pp. 4655–4665.

[183] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman and A. Galstyan, *A survey on bias and fairness in machine learning*, arXiv preprint arXiv:1908.09635 (2019).

[184] M. J. Kusner, J. Loftus, C. Russell and R. Silva, *Counterfactual fairness*, in Advances in neural information processing systems, 2017, pp. 4066–4076.

[185] S. Corbett-Davies and S. Goel, *The measure and mismeasure of fairness: A critical review of fair machine learning*, arXiv preprint arXiv:1808.00023 (2018).

[186] S. Galhotra, Y. Brun and A. Meliou, *Fairness testing: testing software for discrimination*, in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 498–510.

[187] C. H. Papadimitriou, *Games against nature*, Journal of Computer and System Sciences **31** (1985), pp. 288–301.

[188] S. M. Majercik, *Appssat: Approximate probabilistic planning using stochastic satisfiability*, International Journal of Approximate Reasoning **45** (2007), pp. 402–419.

[189] N.-Z. Lee and J.-H. R. Jiang, *Towards formal evaluation and verification of probabilistic design*, IEEE Transactions on Computers **67** (2018), pp. 1202–1216.

[190] H. A. Kautz, B. Selman et al., *Planning as satisfiability.*, in ECAI, vol. 92, Citeseer, 1992, pp. 359–363.

[191] G. S. Tseitin, *On the complexity of derivation in propositional calculus*, in Automation of reasoning, Springer, 1983, pp. 466–483.

[192] E. Raff, J. Sylvester and S. Mills, *Fair forests: Regularized tree induction to minimize model bias*, in Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, 2018, pp. 243–250.

[193] O. Roussel and V. M. Manquinho, *Pseudo-boolean and cardinality constraints.*, Handbook of satisfiability **185** (2009), pp. 695–733.

[194] B. Ghosh, D. Malioutov and K. S. Meel, *Classification rules in relaxed logical form*, in Proceedings of ECAI, 6 2020.

[195] A. Ignatiev, A. Morgado and J. Marques-Silva, *PySAT: A Python toolkit for prototyping with SAT oracles*, in SAT, 2018, pp. 428–437. URL: https://doi.org/10.1007/978-3-319-94144-8_26, doi:10.1007/978-3-319-94144-8_26.

[196] J. Angwin, J. Larson, S. Mattu and L. Kirchner, *Machine bias risk assessments in criminal sentencing*, ProPublica, May **23** (2016).

[197] A. C. McGinley, *Ricci v. destefano: A masculinities theory analysis*, Harv. JL & Gender **33** (2010), p. 581.

[198] B. Ghosh, D. Basu and K. S. Meel, *Justicia: A stochastic SAT approach to formally verify fairness*, in Proceedings of AAAI, 2 2021.

[199] D. Pisinger, *Linear time algorithms for knapsack problems with bounded weights*, Journal of Algorithms **33** (1999), pp. 1–14.

[200] G. J. Woeginger and Z. Yu, *On the equal-subset-sum problem*, Information Processing Letters **42** (1992), pp. 299–302.

[201] G. Grimmett and D. Stirzaker, *Probability and random processes*, Oxford university press, 2020.

[202] A. Ankan and A. Panda, *pgmpy: Probabilistic graphical models using python*, in Proceedings of the 14th Python in Science Conference (SCIPY 2015), Citeseer, 2015.

[203] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, arXiv preprint arXiv:1702.08608 (2017).

[204] J. Buolamwini and T. Gebru, *Gender shades: Intersectional accuracy disparities in commercial gender classification*, in Conference on fairness, accountability and transparency, PMLR, 2018, pp. 77–91.

[205] C. Loader, *Local regression and likelihood*, Springer Science & Business Media, 2006.

[206] C. Bénesse, F. Gamboa, J.-M. Loubes and T. Boissin, *Fairness seen as global sensitivity analysis*, arXiv preprint arXiv:2103.04613 (2021).

[207] A. Datta, S. Sen and Y. Zick, *Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems*, in 2016 IEEE symposium on security and privacy (SP), IEEE, 2016, pp. 598–617.

[208] B. Ghosh, D. Basu and K. S. Meel, *Algorithmic fairness verification with graphical models*, in Proceedings of AAAI, 2 2022.

[209] C. Loader, *Smoothing: local regression techniques*, in Handbook of computational statistics, Springer, 2012, pp. 571–596.

[210] G. Li et al., *Global sensitivity analysis for systems with independent and/or correlated inputs*, The journal of physical chemistry A **114** (2010), pp. 6022–6032.

[211] J. Herman and W. Usher, *SALib: An open-source python library for sensitivity analysis*, The Journal of Open Source Software **2** (2017). URL: https://doi.org/10.21105/joss.00097, doi:10.21105/joss.00097.

[212] D. Dua and C. Graff, *UCI machine learning repository*, 2017. URL: http://archive.ics.uci.edu/ml.

[213] F. Pedregosa et al., *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research **12** (2011), pp. 2825–2830.

Appendix A

Feature Correlations in SSAT-based Fairness Verifier: Justicia

In Chapter 5, Justicia verifies the fairness of CNF classifiers without considering correlation among features. In this chapter, we address fairness verification of CNF classifiers with correlated features. In particular, we present the encoding of conditional probabilities from a Bayesian network into the SSAT formulation in Justicia.

Methodology

For CNF classifiers, SSAT is a natural choice as it computes the probability of satisfaction of a CNF formula given quantified Boolean variables, where quantifiers distinguish between (random) non-sensitive variables and (existential or universal) sensitive variables. Let $\phi_{\widehat{Y}}$ be a CNF classifier such that a satisfying assignment of $\phi_{\widehat{Y}}$ denotes the positive prediction of the classifier $\widehat{Y} = 1$. We consider another CNF formula ϕ_{BN} to encode the conditional dependencies among variables in the Bayesian Network BN, which is given as the input distribution. ϕ_{BN} contains auxiliary variables to encode the conditional dependencies, which we discuss shortly. The outline of our methodology is to construct a conjoined CNF formula $\phi_{\widehat{Y}} \wedge \phi_{\mathsf{BN}}$, assign appropriate quantifiers to the variables, and solve an SSAT problem on quantified formula $\phi_{\widehat{Y}} \wedge \phi_{\mathsf{BN}}$ to answer queries such as $\max_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$ and $\min_{\mathbf{a}} \Pr[\widehat{Y} = 1 | \mathbf{A} = \mathbf{a}]$—the maximum and minimum conditional positive prediction of the classifiers with correlated features, respectively.

Encoding a Bayesian Network as a CNF Formula. Our goal is to encode the Bayesian network $\mathsf{BN} = (G, \theta)$ into a CNF formula ϕ_{BN} such that *the weighted model count* of ϕ_{BN} exactly computes the joint probability distribution in BN [89]. In this context, an SSAT formula trivially does not allow conditional probabilities of randomized quantified variables. Hence, ϕ_{BN} contains additional *auxiliary variables* to capture the conditional probabilities, as discussed next.

Let $G = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V} \subseteq \mathbf{X} \cup \mathbf{A}$, $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$, and each variable $V_i \in \mathbf{V}$ is Boolean.

For each network variable $V_i \in \mathbf{V}$, we define a Boolean *indicator* variable λ_{V_i} such that $\Pr[\lambda_{V_i}] \triangleq \Pr[V_i]$. We add following constraint in ϕ_{BN} to establish the relation between λ_{V_i} and V_i.

$$\lambda_{V_i} \leftrightarrow V_i, \tag{A.1}$$

Intuitively, both λ_{V_i} and V_i are either true or false. This constraint can be trivially translated to clauses in CNF using the equivalence rule $A \leftrightarrow B \equiv (\neg A \vee B) \wedge (A \vee \neg B)$ for Boolean variables A, B.

We now present the encoding of conditional probabilities induced by parameters in the network θ. Let $V_i \in \mathbf{V}$ be a vertex in G where $\mathrm{Pa}(V_i) \neq \emptyset$ be V_i's parents and $|\mathrm{Pa}(V_i)| = k$. Additionally, let v and $\mathbf{u} \triangleq [u_1, .., u_k]$ be an assignment of V_i and $\mathrm{Pa}(V_i)$, respectively. To encode $\Pr[V_i = v | \mathrm{Pa}(V_i) = \mathbf{u}]$, we introduce auxiliary variable $\lambda_{v,\mathbf{u}}$ and add following constraints in ϕ_{BN}.

$$\lambda_{v,\mathbf{u}} \wedge \bigwedge_{j=1}^{k} \lambda_{u_j} \rightarrow \lambda_v \tag{A.2}$$

$$\neg\lambda_{v,\mathbf{u}} \wedge \bigwedge_{j=1}^{k} \lambda_{u_j} \rightarrow \neg\lambda_v \tag{A.3}$$

where $\lambda_v \equiv \lambda_{V_i}$. Moreover, λ_{u_j} is the indicator variable corresponding to the j^{th} parent in $\mathrm{Pa}(V_i)$. In the above two constraints, for a fixed assignment $\mathbf{u}$ of parents $\mathrm{Pa}(V_i)$, both λ_v and $\lambda_{v,\mathbf{u}}$ are either true or false. Hence, these two constraints encode the conditional probability of $V_i = v$ given $\mathrm{Pa}(V_i) = \mathbf{u}$ using $\Pr[\lambda_{v,\mathbf{u}}] = \Pr[V_i = v | \mathrm{Pa}(V_i) = \mathbf{u}]$. Both constraints can be translated to CNF clauses trivially. For example, Eq. A.2 is translated as $\neg\lambda_{v,\mathbf{u}} \vee \bigvee_{j=1}^{k} \neg\lambda_{u_j} \vee \lambda_v$. We next analyze the complexity of ϕ_{BN} in terms of the number of variables and clauses.

Lemma 14. *For a Bayesian network $\mathsf{BN} = (G, \theta)$ defined over n Boolean variables and $C(G)$ network complexity, the encoded CNF formula ϕ_{BN} has $n + C(G)$ variables and $2(n + C(G))$ clauses.*

Proof. Since the DAG in the Bayesian network has n vertices, we consider n indicator variables. Moreover, for encoding conditional probabilities, we consider $C(G)$ auxiliary variables where $C(G)$ denotes the the number of independent parameters in the network (ref. Chapter 2.3.3). Hence, total variables in ϕ_{BN} is $n + C(G)$.

According to Eq. (A.1), (A.2), (A.3), there are $2(n + C(G))$ clauses in ϕ_{BN}.

□

Quantifiers in $\phi_{\widehat{Y}} \wedge \phi_{\mathsf{BN}}$. We now discuss the quantifiers in $\phi_{\widehat{Y}} \wedge \phi_{\mathsf{BN}}$, the SSAT solution of which constitutes the maximum (minimum) probability of positive prediction of a CNF classifier. $\phi_{\widehat{Y}} \wedge \phi_{\mathsf{BN}}$ contains four categories of variables : (i) sensitive variables $\mathbf{A}$, (ii) non-sensitive variables $\mathbf{X}$, (iii) indicator variables λ_{V_i}, and (iv) auxiliary variables $\lambda_{v,\mathbf{u}}$. Among them, (iii) and (iv) are associated with ϕ_{BN} and the rest for $\phi_{\widehat{Y}}$. For computing the maximum probability of positive prediction of the classifier, we construct an exists-random-exists (ERE) SSAT formula with following quantifiers: we set sensitive features $\mathbf{A}$ with existential quantifiers in the beginning of the prefix of the SSAT formula followed by $\lambda_{V_i}, \lambda_{v,\mathbf{u}}$, and $X_j \in \mathbf{X} \setminus \mathbf{V}$ with randomized quantifiers. The remaining variables $X_i \in \mathbf{V}$ are existentially quantified as their assignment is fixed by indicator variables λ_{V_i}. In contrast, for computing the minimum probability of positive prediction of the classifier, we consider an universal-random-exists (URE) SSAT formula where we set sensitive features $\mathbf{A}$ as universal quantifiers with all other quantifiers remaining same.