Expert Systems with Applications: X 8 (2020) 100040

Contents lists available at ScienceDirect

Expert Systems with Applications: X

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

GIMO: A multi-objective anytime rule mining system to ease iterative feedback from domain experts

Tobias Baum[a,*], Steffen Herbold[b], Kurt Schneider[a]

[a] FG Software Engineering, Leibniz Universität Hannover, Hannover, Germany
[b] SWE, Universität Göttingen, Göttingen, Germany

ARTICLE INFO

Article history:
Received 6 July 2019
Revised 27 July 2020
Accepted 17 August 2020
Available online 19 August 2020

Keywords:
Rule mining
Human-in-the-loop
Multi-objective
Set cover
Interpretable artificial intelligence
Explainable artificial intelligence

ABSTRACT

Data extracted from software repositories is used intensively in Software Engineering research, for example, to predict defects in source code. In our research in this area, with data from open source projects as well as an industrial partner, we noticed several shortcomings of conventional data mining approaches for classification problems: (1) Domain experts' acceptance is of critical importance, and domain experts can provide valuable input, but it is hard to use this feedback. (2) Evaluating the quality of the model is not a matter of calculating AUC or accuracy. Instead, there are multiple objectives of varying importance with hard to quantify trade-offs. Furthermore, the performance of the model cannot be evaluated on a per-instance level in our case, because it shares aspects with the set cover problem. To overcome these problems, we take a holistic approach and develop a rule mining system that simplifies iterative feedback from domain experts and can incorporate the domain-specific evaluation needs. A central part of the system is a novel multi-objective anytime rule mining algorithm. The algorithm is based on the GRASP-PR meta-heuristic but extends it with ideas from several other approaches. We successfully applied the system in the industrial context. In the current article, we focus on the description of the algorithm and the concepts of the system. We make an implementation of the system available.

1. Introduction

Decision support systems are an important tool to support domain experts in their decision making. Understanding the decision made by the system is vital for many practical applications. Firstly, understanding is often important for the acceptance of decision support systems by domain experts (Bose & Mahapatra, 2001). In addition, it is often required due to laws and regulations. For example, the German government recently clarified that black-box systems for decision support at insurance companies would not be compliant with existing regulations (German Government, 2018). For a recent research project, we required a decision support system to help us understand how data mining may be used to make code reviews more efficient (Baum, Herbold, & Schneider, 2018a). Because we wanted to work together with domain experts and provide insights for their decisions, the understandability of models was a key factor for our work. However, we found that current approaches for data mining fall short for such a research project that directly incorporates domain experts:

- Some mining approaches create opaque models that cannot be analyzed by domain experts at all. Some types of models (e.g., support vector machines or neural networks) are nearly always hard to understand, while for others (e.g., decision rules or decision trees) understandability depends on complexity (Barredo Arrieta et al., 2020; Dam, Tran, & Ghose, 2018).
- It is often hard to map feedback from domain experts to the parameters needed by the mining algorithm.
- One type of human input concerns the cost of misclassification. Many approaches are not cost-sensitive at all or need a cost matrix to be specified at the start. In reality, problems are cost-sensitive, but the exact cost matrix is often not known. Moreover, costs are usually assumed to be independent of each other for each input in an algorithm. However, costs often depend on each other, e.g., due to synergy effects.
- Most approaches allow input to be given only at the start of a run and create a single model as the result of such a run. This limits the points in time when domain experts can give feedback.

* Corresponding author.
E-mail addresses: tobias.baum@inf.uni-hannover.de (T. Baum), herbold@cs.uni-goettingen.de (S. Herbold), kurt.schneider@inf.uni-hannover.de (K. Schneider).

https://doi.org/10.1016/j.eswax.2020.100040

- The knowledge discovery process encompasses multiple phases (Mariscal, Marban, & Fernandez, 2010), including cleaning of the data, selection or perhaps even creation of features, and interpretation of the results. Some systems, like Weka (Witten, Frank, & Hall, 2011), combine support for many of these steps, but in a general way not streamlined for specific applications.

A further aspect of our use case is that there is a strong relationship between some of the instances. Our goal was to reduce the effort during code reviews by presenting less code to reviewers, such that they still write the same remarks during the reviews. Since multiple source code fragments could lead to the same remarks, there is a strong relationship between groups of code fragments and the remarks by the reviewers. Such relationships are not supported by current approaches for the mining of interpretable models.

In this article, we propose *GIMO*, a novel data mining approach that creates decision rules and addresses the limitations discussed above. GIMO's design is based on the belief that, currently, the most promising approach to extract knowledge from data is to let human and computer work as a team (Ankerst, 2002; Holzinger, 2016). The computer can sift through vast amounts of data swiftly and without loss of concentration. Human domain experts can provide input that is not readily available from the data, e.g., on preferences or on conditions that hold in the respective context. Like every design process, this teamwork benefits from being iterative: The humans can see preliminary results from the computer and provide focus or new insights based on the results.

The contribution of this article is the GIMO system for mining decision rules that helps to overcome the above-mentioned limitations due to the following features:

- Multi-objective optimization, i.e., the system can find rules with varying trade-offs for objectives like reduction of false positives, reduction of false negatives, but also custom cost functions that take relationships between instances into account or estimate the understandability of a model. This avoids the need to specify a cost matrix at the start.
- Interactive feedback, i.e., the users can interactively explore the data and the results as well as provide feedback to guide the generation of decision rules.
- Iterative design, i.e., the feedback by the user is integrated into the mining process and can be iteratively refined. This also includes the possibility to undo earlier decisions.
- Anytime, i.e., the user can explore the data and the current results and interact with the system at any time. The user rarely needs to wait for the system, and neither does the system need to wait for the user.
- Geared towards domain expert feedback, i.e., the user can provide feedback without the need to translate it into opaque tuning parameters or to understand the data mining algorithms.

Following the taxonomy of Barredo Arrieta et al. (2020), the created models are always "algorithmically transparent" and the system's goal is to find models that are "simulatable" as much as possible. In the two next sections, we provide background information on the application area that motivated our work and on related research. Then, we proceed to formalize the requirements that a decision support system must fulfill to address such data mining problems (Section 4). We go on to describe the GIMO rule mining algorithm (Section 5) and provide results from applying our system in an industrial case study (Section 6). We conclude by discussing future work in Section 7. The used mathematical notations are summarized in Appendix A. This article focuses on GIMO's central algorithm, which is also shown in pseudo-code in Appendix B. Further features to support the complete data mining process are discussed in an extended report (Baum, Herbold, & Schneider, 2018b).

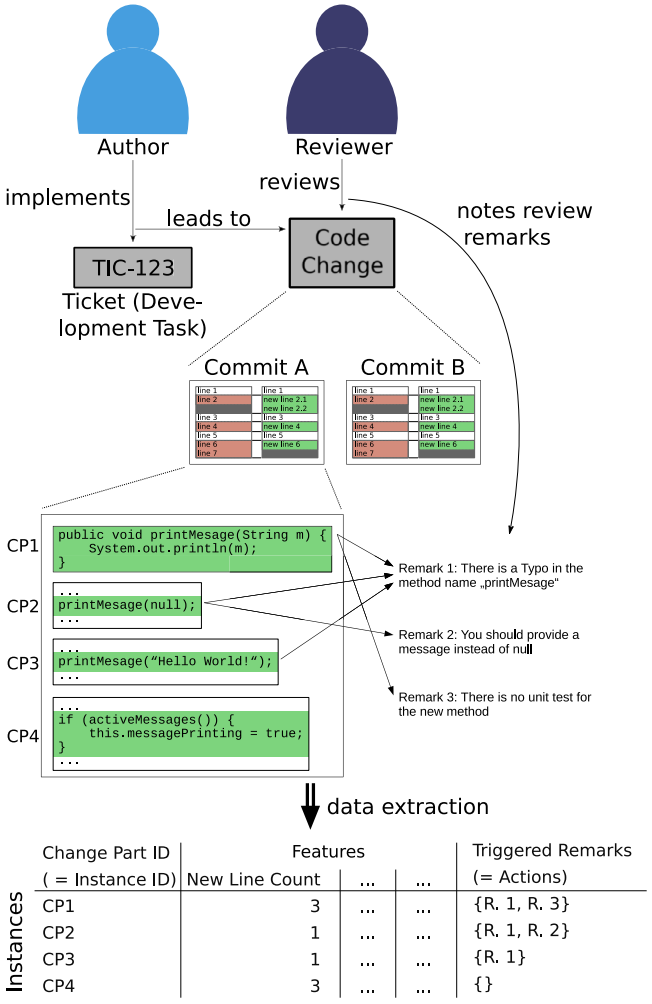

Fig. 1. Example to illustrate the application domain and many of the terms used in this article.

2. Motivation

The motivation for the data mining system we present in this article stems from our software engineering research on code reviews. Code reviews are a software quality assurance technique in which code or code changes are manually checked by one or more developers (Baum, Liskin, Niklas, & Schneider, 2016). In their modern change-based form, they are widely used in industrial practice (Baum, Leßmann, & Schneider, 2017; Rigby, Cleary, Painchaud, Storey, & German, 2012). In our setting, development revolves around 'tickets' that describe programming tasks: One developer writes some code and marks the corresponding ticket as 'ready for review' when done with programming. Another developer then reviews the code changes that belong to this ticket. When a reviewer spots a defect or some other point that needs to be discussed, she usually creates a 'review remark' that will be communicated to the code's author. Fig. 1 illustrates this graphically. In our research on code reviews, we are interested in finding parts of the code change that will not lead to review remarks. These parts can then be left out from the review, saving time and leaving more mental resources for the rest of the review. The underlying problem behind some review remarks can be manifested in several parts of the code, and it suffices to check one of them to note it.

In a case study in a medium-sized software company, we extracted data from software repositories on which code change parts led to which remarks (Baum et al., 2018a). The rule mining system described in the current article was then used to derive rules that characterize change parts that do not need to be reviewed.

3. Related work

In the following, we discuss related work. With our system, we aim to support the whole data mining process. A survey of various data mining and knowledge discovery process models was performed by Mariscal et al. (2010). Most of these models acknowledge that the process is iterative and interactive. In line with this, the panelists at a 2002 SIGKDD panel regarded cooperation between human and computer for data mining as beneficial (Ankerst, 2002). Holzinger (2016) discusses interactive machine learning in health informatics. Interactive data mining has been studied for example by Hellerstein et al. (1999), Zhao, Yao, and Yan (2007), and, with a multi-objective approach, by Mühlbacher, Linhardt, Möller, and Piringer (2018). Mühlbacher's TreePOD system emphasizes the visual exploration of a two-dimensional Pareto front of decision trees. Some approaches load off most of the construction work to the user (Ankerst, Ester, & Kriegel, 2000; Han & Cercone, 2002). More similar to our approach is "constraint-based data mining" (Han, Lakshmanan, & Ng, 1999), in which the user restricts the search space for association rules using "rule constraints" and guides the search with "interestingness constraints".

In the current study, we assume that simpler models and models with fewer features are more comprehensible (Huysmans, Dejaeger, Mues, Vanthienen, & Baesens, 2011). These are not the only factors that influence comprehensibility (Fürnkranz, Kliegr, & Paulheim, 2018; Pazzani, 2000). Dam et al. (2018) state that for software analytics, explainability is as important as accuracy. Freitas (2014) suggests regarding comprehensibility as one objective in a multi-objective approach. Our approach to create transparent machine learning models belongs to the larger field of eXplainable Artificial Intelligence (XAI) (Barredo Arrieta et al., 2020). Rule-based systems form an important part in recent applications of XAI, sometimes with differences in rule representation, like temporal rules based on association rules (Anguita-Ruiz, Segura-Delgado, Alcalá, Aguilera, & Alcalá-Fdez, 2020) or a fuzzy logic belief-rule-base (Sachan, Yang, Xu, Benavides, & Li, 2020). Other researchers in the XAI field use a two-step process: First, a black-box model is learned, and this model is transformed into a comprehensible model (Johansson, Niklasson, & König, 2004; Moeyersoms, de Fortuny, Dejaeger, Baesens, & Martens, 2015; Zhang, Li, & Cui, 2005). Still another approach is to create explanations for the model's decision for a specific instance upon request (Dam et al., 2018; Ribeiro, Singh, & Guestrin, 2016; Tan, Tan, Dara, & Mayeux, 2015). However, these attempts on posthoc explainability are not without criticism, due to the associated risks (Rudin, 2018).

We use a variant of multi-objective GRASP-PR (Martí, Campos, Resende, & Duarte, 2015; Resende & Ribeiro, 2016) for rule mining, and so did Ishida, De Carvalho, Pozo, Goldbarg, and Goldbarg (2008); Ishida, Pozo, Goldbarg, and Goldbarg (2009), Reynolds and De la Iglesia (2009) and Pavanelli, Arns Steiner, Góes, Pavanelli, and Costa (2014), with promising results. These prior studies differ from ours in the specific meta-heuristic operators and various other details and do not take domain expert feedback into account. Many other studies use multi-objective evolutionary algorithms for data mining (Dehuri & Mall, 2006; Dehuri, Patnaik, Ghosh, & Mall, 2008; Kaya, 2010). A comprehensive discussion of the vast amounts of literature on this topic is not possible within this article. Early studies were done by De Jong, Spears, and Gordon (1993) and Janikow (1993), with successors for example by Eggermont, Eiben, and van Hemert (1999), Fidelis, Lopes, and Freitas (2000), Bernardó-Mansilla and Ho (2005), and Baykasoğlu and Özbakir (2007). Kwedlo and Kretowski (2001) explicitly discuss cost-sensitivity in their approach. Freitas (2003) surveys further applications of evolutionary algorithms for data mining.

The rule generation in our system is derived from standard rule mining algorithms (Breiman, 2001; Cohen, 1995; Fürnkranz & Kliegr, 2015; Lavrač, Fürnkranz, & Gamberger, 2010). The generated candidates are optimized with heuristic search. A similar approach was used by Ryan and Rayward-Smith (1998), who hybridized C4.5 and genetic programming to improve scalability. Hybridization of evolutionary algorithms is studied more deeply by Grosan and Abraham (2007). Another influence for our approach was the "shooting" procedure for multi-objective optimization (Benson & Sayin, 1997; Wagner, Beume, & Naujoks, 2007). Further inspiration was ROCCER's approach of constructing a convex hull in ROC space (Prati & Flach, 2005).

We apply our mining approach for prediction in software engineering. Over the last two decades, defect prediction and other applications of data mining in software engineering have become vast research areas. Literature surveys (Hall, Beecham, Bowes, Gray, & Counsell, 2012; Hosseini, Turhan, & Gunarathna, 2018; Radjenović, Heričko, Torkar, & Živkovič, 2013) provide an overview of this area.

4. Problem statement

We will now state the mining problem more abstractly. Consider a problem, like the review remark prediction problem introduced above, in which an *action* (e.g., noting a review remark) may be caused/triggered by several instances. One instance can be the cause of one, several, or no action. Checking or otherwise processing a potential cause costs effort, and the goal is to minimize the total effort while still triggering the actions. We want to solve this problem by predicting which instances to select to spend the effort on. Fig. 2 illustrates such a situation. The instance I2 is a cause for both actions A1 and A2. Other sufficient, but sub-optimal combinations of instances such that causes for both A1 and A2 are identified would be I1 and I3, and I1 and I4. We call identifying an instance as a cause of an action a *positive* prediction and not identifying an instance as a cause a *negative* prediction. As our example shows, there may be multiple combinations of instances where the identification would lead to the same actions. Furthermore, we assume that domain experts who are responsible for the actions want to understand the reasoning behind predictions and do not simply accept the predictions as true. If domain experts understand the reason for a prediction, but disagree, they should be able to modify the prediction model. Thus, the problem we want to solve has the following properties:

- Minimize the number of positive predictions (i.e., predicted instances) while still covering (nearly) all actions.
- Account for the relationship between actions and instances.
- The model is understandable and modifiable by domain experts.

Our problem is a binary classification problem. There are many ways to describe hypotheses for the classification of objects, e.g., through support vector machines, neural networks, regression functions like logistic regression, and rules. Of these approaches, rules have the advantage that they are easy to interpret and modify by humans. Within this article, we focus on rules described as boolean expressions written in the disjunctive normal form (DNF), i.e.,

$$C_1 \vee \ldots \vee C_p \tag{1}$$

where $C_k = c_{k.1} \wedge \ldots \wedge c_{k.q_k}$ with $k = 1, \ldots, p$ are conjunctions over atomic conditions. Every boolean expression can also be described

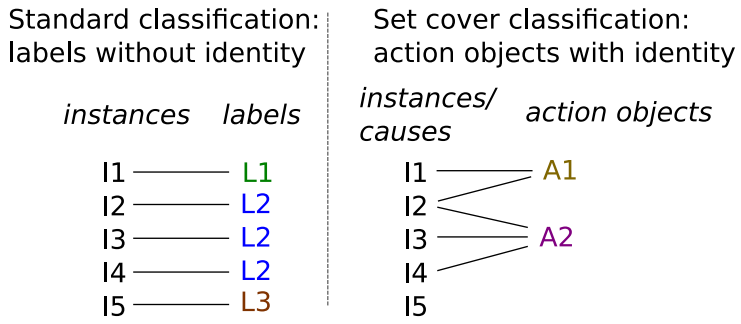

Fig. 2. Standard classification versus set cover classification.

as a DNF. Another popular way to describe decision rules through boolean expressions is decision trees (Witten et al., 2011). Decision trees have equal expressive power to DNFs. Although it is easier for humans to use decision trees when evaluating them for a specific data point (Huysmans et al., 2011), we assume that DNFs are preferable for understanding: In an unordered set of conjunctions, each conjunction can be treated as an independent nugget of knowledge. In contrast, a node in a decision tree has to be interpreted with all previous nodes in mind, and the class is only known when reaching a leaf node. Moreover, there is a high redundancy in the way conjunctions are encoded in decision trees, which leads to overhead in the effort spent to understand them in detail.

For a given training set X over a feature space $\mathcal{F}$ with labels Y such that $X = \{x_1, \ldots, x_m\} \subset \mathcal{F} = \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$ and $Y = \{y_1, \ldots, y_m\} \in \{0, 1\}^m$, rule mining algorithms try to find a DNF that postulates logical rules of the instances $x_i \in X$ of the form $x_{i,j} \leq A$ (only for $j \leq n_1$, i.e., numeric features), $x_{i,j} \geq A$ ($j \leq n_1$), $x_{i,j} = A$, or $x_{i,j} \neq A$ for constants $A \in \mathbb{R}$. Thus, such rule mining algorithms determine a hypothesis

$$h : \mathcal{F} \to \{0, 1\} \tag{2}$$

such that h is a DNF of atomic conditions over the feature space $\mathcal{F}$. The problem with learning DNFs is that the search space is huge. Thus, a strategy to search for DNFs that yield good results is required. One of the most popular algorithms for mining of rules is the RIPPER algorithm (Cohen, 1995). The algorithm first greedily creates rules: new atomic conditions over features are added by using information gain as the criterion for the feature selection until no negative examples are covered. This process leads to overly complex DNFs that overfit the data. To counter overfitting, the DNFs are pruned such that the ratio $\frac{tp-fp}{tp+fp}$ is optimized. This algorithm efficiently searches for good conjunctions given that positive and negative examples are equally important for the use case. Although it is possible to use RIPPER for our problem, several limitations are likely to degrade the performance for the prediction task. We discuss these limitations of RIPPER in the following sections and use this to outline the requirements on an algorithm to solve our problem in greater detail. We note that while we discuss the problems for RIPPER, the problems for other approaches, e.g., based on decision tree learning (C4.5, PART, CART) would be similar.

A drawback of DNFs is that they can be inefficient, i.e., needing complex rules to describe simple concepts. Based on the intuition that it might be easier to express the opposite concept in such a case, we use a combination of two formulae in disjunctive normal form: one for rules that describe elements that should be covered and one for elements that should be excluded.

$$(C_1^{incl} \vee \ldots \vee C_p^{incl}) \wedge \neg(C_1^{excl} \vee \ldots \vee C_{p'}^{excl}) \tag{3}$$

This notation is equally expressive as DNFs but allows that certain rules can be written more concisely as negations. To account for this difference to normal DNFs, we use the term *ruleset* to refer to this in the following. Moreover, we use the term *rule* as a synonym to a single conjunction C in the model.

4.1. Multi-objective rule learning

For domain-specific use cases, there is a cost associated both with false positive predictions and false negative predictions. These costs can be modeled in a cost function $cost(h, X, Y)$ that estimates the costs using labeled data. For the remainder of this paper, we assume without loss of generality that cost functions should be minimized. Thus, a learning algorithm should optimize the cost function, i.e.,

$$\min_h cost(h, X, Y). \tag{4}$$

In general, use cases might have multiple competing cost objectives, e.g., minimizing the time to market and minimizing the costs of the development. This leads to a multi-objective optimization problem of the form

$$\min_h (cost_1(h, X, Y), \ldots, cost_o(h, X, Y)) \tag{5}$$

Usually, there is no single solution that is optimal for all cost functions. Therefore, we consider Pareto optimal solutions instead, i.e., solutions that are not dominated by any other solution. One solution h dominates another solution h' if $cost(h, X, Y) \leq cost(h', X, Y)$ for all $cost \in \{cost_1, \ldots, cost_o\}$ and there exists at least one cost function $cost \in \{cost_1, \ldots, cost_o\}$ such that $cost(h, X, Y) < cost(h', X, Y)$. While it is certainly possible to use RIPPER and afterward calculate cost functions, the algorithm itself ignores the cost functions during the training. It would be possible to modify the growing and pruning criteria to account for a single cost function directly. However, RIPPER cannot easily be modified to determine Pareto optimal solutions in a multi-objective setting.

For our use case, we have three types of cost functions: (1) cost functions for the estimation of the effort spent by domain experts to act on predictions of the decision support system, (2) cost functions for the number of actions not covered, and (3) costs for the cognitive complexity of understanding the rules. The exact definitions of our cost functions will be given in Section 6.3.

4.2. Relationships between instances

Another property of RIPPER is that the label of an instance only depends on the instance itself, not on other instances. When the instances are independent given the class label, this is not a problem. However, this is not the case for our type of problem. Instead, as stated at the start of this section, our labels can be explained by multiple instances at the same time: They form groups $\bar{X}_1 \subset X, \ldots, \bar{X}_{\bar{m}} \subset X$ such that all instances in $\bar{X}_i$ have the same action for the positive labels of the instances (Fig. 2). To identify the actions, it is sufficient to identify one instance of $\bar{X}_i$ through a rule.

Thus, an action $\bar{X}_i$ is correctly identified if $h(x) = 1$ for at least one $x \in \bar{X}_i$. For simplicity, we use the notation $h(\bar{X}_i) = 1$ for identified actions and $h(\bar{X}_i) = 0$ for missed actions. To minimize the effort for downstream analysis of identified instances, it would be optimal to identify only one instance $x \in \bar{X}_i$, instead of all instances with positive labels (i.e., a 'set cover'). Moreover, an instance $x \in X$ can be the cause of multiple actions, i.e., $x \in \bar{X}_{i_1} \cap \bar{X}_{i_2}$. Thus, the effort can be reduced further by selecting instances that are the cause of many actions. There is no way to modify RIPPER to account for this relationship without completely changing the algorithm.

4.3. Feedback

For our problem, domain experts must not only be able to understand the generated hypothesis but also to modify it with feedback. Although they certainly could modify DNFs produced by RIPPER, the implications of these modifications for the performance would be unclear. To solve the problem, the learning should directly interact with domain experts and allow for the following:

- Definition of rules by domain experts that must be part of the hypothesis.
- Definition of atomic conditions or rule patterns that must not be part of the hypothesis.
- Analysis of the impact of manual modifications of the hypothesis on the Pareto front.

This interaction should be iterative and allow users to modify the manually defined conditions and get feedback about the results at any time. Please note that these interactions are different from active learning (Settles, 2009): In active learning, domain experts may also interact with the algorithm, but only by labeling instances on-the-fly. In our case, labeled instances are available and the domain experts can directly modify the generated rules.

5. The GIMO algorithm

As we discussed in Section 4, current state-of-the-art rule mining algorithms cannot be used to solve the interactive set cover problem we encountered. To address all requirements we have on the algorithm, we combined ideas from different approaches to design a "GRASP-based Interactive Multi-Objective" (GIMO) rule mining algorithm.

- We use the metaheuristic search algorithm GRASP-PR (Resende & Ribeiro, 2016) as the foundation for our work.
- We use established concepts from Multi-Objective Evolutionary Algorithms (MOEAs) for our extension of GRASP-PR, especially from MSOPS (Wagner et al., 2007).
- We use separate-and-conquer rule learning (Fürnkranz & Kliegr, 2015; Janssen & Fürnkranz, 2010) to generate new candidates for rules to seed the metaheuristic search.
- We adopt a randomized sampling of instances and features similar to random forests (Breiman, 2001) to generate new rules, to avoid locally optimal solutions for generated rules.
- We enable constraint-based data mining (Han et al., 1999) by allowing users to restrict the search space interactively.

5.1. Overview of GIMO

Fig. 3 shows a general overview of GIMO. Like GRASP-PR, the algorithm consists of three main steps:

1. randomized and greedy generation of candidate rulesets (Section 5.2);
2. optimization of rulesets by local search (Section 5.3); and
3. combination of rulesets through path relinking for further improvement (Section 5.4).

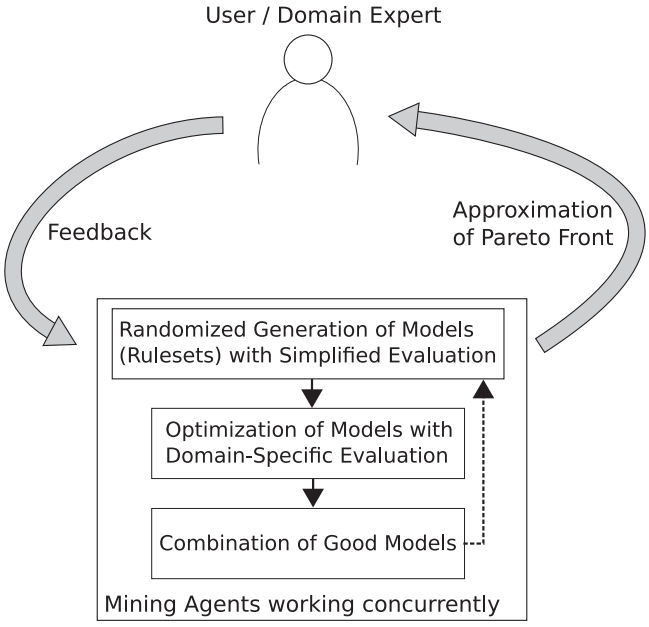

Fig. 3. Schematic overview of GIMO.

However, unlike in standard GRASP-PR, the steps are executed by one or several concurrently working processes (mining threads).[1] Each mining thread runs in an infinite loop in which it processes work packages from a queue of work items. Such a work package is a combination of a ruleset and a task that needs to be done with that ruleset, e.g., a newly created ruleset that now needs to be optimized by local search. When there is no open work, a mining thread generates new items, or it tries to improve existing solutions further. A second difference to standard GRASP-PR is that our algorithm works on a Pareto front of solutions and takes the multi-objective nature of the problem into account. This Pareto front is kept as a shared data structure, which is updated every time a new ruleset is evaluated.

The algorithmic design with parallel mining threads that communicate over a shared data structure allows that the algorithm runs in the background of a web-based graphical user interface. Thereby, it enables interactive feedback by domain experts at any time, instead of having to wait hours for an algorithm to finish. The data structure that keeps the current Pareto front and the queues with open work items will be called "blackboard", because it is conceptually similar to blackboard algorithms (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996). With the mining threads running in the background, the user can explore the best rules found so far without the need to wait for a mining job to complete. This is the foundation for interactive feedback from the user to the mining threads, which will be discussed in Section 5.6.

The appendix of this article contains pseudo-code descriptions for the algorithm's main parts. The pseudo-code for the mining threads' main loop is shown in Fig. B.5 in the appendix. Furthermore, the full source code of the system is available (Baum, Herbold, & Schneider, 2018c).

5.2. Rule set generation

One key component of GRASP heuristics is a randomized and greedy heuristic to generate new solutions that will be optimized by the other parts of the algorithm. We combine the concepts of separate-and-conquer rule learning (Janssen & Fürnkranz, 2010)

[1] called *mining agents* in our implementation.

with the sampling strategy from random forests (Breiman, 2001) to generate candidate rules. GIMO's rule generation has the following random components:

- a random subset of features (random subspace method Ho, 1998);
- a random subset of instances;
- a random integer for the maximum number of rules in the ruleset; and
- a randomly selected search bias function for the greedy top-down rule search.

The random selection of instances includes an undersampling of the majority class to treat a potential class-level imbalance in the data. According to Janssen and Fürnkranz (2010), precision, Laplace, relative costs, and the m-estimate are good biases for rule generation. Therefore, we use these as candidates for the random selection of search biases. In addition, there is a small but non-zero probability of choosing a random atomic condition. Thus, the randomization is biased towards "good" rules according to the separate-and-conquer learning but allows any possible ruleset on the data.

The search bias for the rule generation ignores the relationship between instances, i.e., that multiple instances can be the cause for the same action, so that generated rules may be sub-optimal for cost functions that take this relationship into account. Therefore, we include the set cover relationship indirectly in the input for the separate-and-conquer algorithm. We achieve this by re-calculating the binary class labels required by the separate-and-conquer algorithm. Specifically, we use a randomized greedy heuristic for the NP-complete set cover problem (Skiena, 1998) to determine a set cover $X' \subset X$ for $\bar{X}_1, \ldots, \bar{X}_{\bar{m}}$. We then use labels Y' instead of Y as input for the separate-and-conquer algorithm such that $y'_i = 1$ if $x_i \in X'$ and $y'_i = 0$ otherwise. This way, candidate rules are directly created for set covers. The randomized nature of the set cover heuristic is consistent with the remainder of the design of our algorithm and should prevent local optima.

An additional caveat of the ruleset generation is that the ruleset syntax requires two sets of rules, one for inclusion and one for exclusion of instances. Because the rulesets are optimized by the subsequent parts of the algorithm anyways, we create the two sets of rules independently of each other. The local search and the path relinking are responsible for removing redundancies. The algorithm for the generation of rulesets that implements the above concepts is outlined in Fig. B.6 in the appendix.

5.3. Local search

The basic structure of the local search is that of a steepest descent hill climbing meta-heuristic (Russell & Norvig, 2009): A neighborhood of the current ruleset is searched for the ruleset with the largest improvement. The procedure is iterated with this best choice until no further improvement is possible. The local search is performed for each ruleset that is randomly generated as a candidate, so it can be seen as a random-restart strategy to avoid locally optimal results. We modified the heuristic to be suitable for a multi-objective cost function. The local search keeps a Pareto set of visited rules, which forms the result at the end of the search. This Pareto set is based on the cost vector, i.e., the cost values for the different objectives. The search itself is guided by a *target function* that maps a cost vector into a single numeric target value. Commonly, this 'collapsing' is a linear combination of the costs. Since the target function does not affect the calculation of the cost vector, the Pareto front is independent of the target function. In each iteration, the algorithm moves to the best neighbor, i.e., the neighbor with the smallest target function value. The search can encounter a plateau where the target function value stays the same in the best case. It then moves to one of these neighbors with an equal target value. Such a 'plateau move' is only performed a limited number of times and only to neighbors that could at least be added to the Pareto set. In this case, the Pareto set has a role similar to the tabu list in tabu search (Glover, 1989) and ensures that once visited solutions will not be revisited to avoid redundantly searching the same parts of the solution space multiple times.

To reduce the size of the neighborhood and, therefore, reduce the time needed for the search, the search alternates between two restricted types of neighborhood. In the 'rule adding' neighborhood, the neighbors are determined by adding a candidate rule to the current ruleset. In the 'rule adjusting' neighborhood, the rule that was added last to the ruleset is adjusted: an atomic condition is removed, the value of a numeric comparison is changed to the nearest split point, or a random atomic condition is added. Search in the 'rule adjusting' neighborhood is kept up as long as an improvement is found, then 'rule adding' is tried for one iteration again. This strategy is similar to the iterative greedy strategy of RIPPER, except that we do not modify the rules with the best local change, but with a randomized strategy that is evaluated using a cost function for Pareto optimality. A pseudo-code version of the algorithm can be seen in Fig. B.7 in the appendix.

5.4. Path relinking

Often, the singular rules in a ruleset are mostly independent of each other. Therefore, it is likely that they complement each other so that combining two good rulesets will lead to another good ruleset. In the GRASP literature, this process of combining solutions is called path relinking (Resende & Ribeiro, 2016). Path relinking begins by determining the difference between two rulesets, the start and end ruleset. The difference can be regarded as a collection of independent adjustments 'add rule X' or 'remove rule Y' that move the start ruleset closer to the end ruleset. Beginning at the start ruleset, the algorithm looks for a good adjustment to apply next, applies it, and iterates until the end ruleset is reached. In this process, every visited candidate is compared to the Pareto front and added if possible.

Like for other parts of GIMO, the decision what a 'good adjustment' is depends on a target function that collapses the cost vector into a single target value. To restrict the number of full evaluations, the mining thread does not look for the best adjustment but stops the search as soon as it finds one that improves upon the start value. Ideally, the start ruleset should already have a good target function value. When none of the adjustments leads to an absolute improvement, the algorithm picks the least bad adjustment and uses Pareto dominance when breaking ties for equal target function values. Pseudo code for the algorithm can be seen in Fig. B.8 in the appendix.

5.5. Respecting relationships between instances

Except for the preprocessing step during rule generation, we did not yet mention how to solve our problem that there is a relationship between instances, i.e., that the instances are not independent given the class label (Section 4.2). Due to the meta-heuristic design, the algorithm can be easily enabled to take these relationships into account by adding appropriate cost functions: The value of the cost function needs to decrease if and only if one instance that is the cause of an action is covered and if no other instance that is related to that action is already covered. We achieve this by counting the actions that are covered by the rules

$$cost_{cover}(h, X, Y) = \frac{\left|\bar{X}_{all}\right| - \left|\{\bar{X} \in \bar{X}_{all} : h(\bar{X}) = 1\}\right|}{\left|\bar{X}_{all}\right|} \tag{6}$$

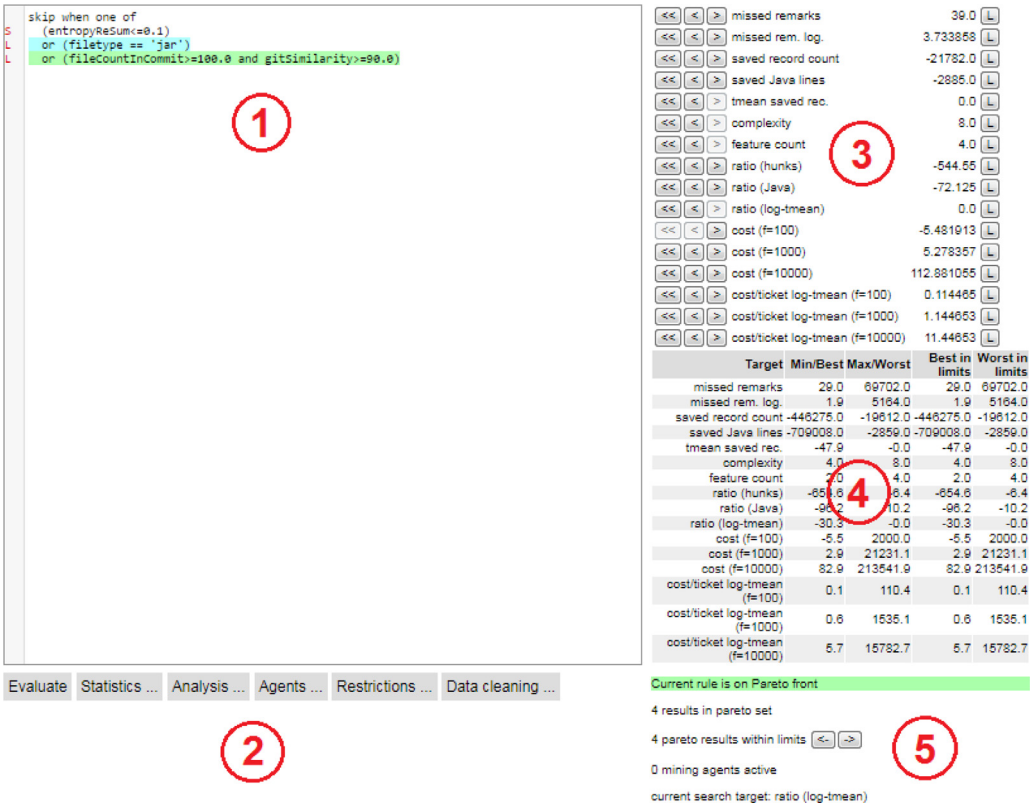

Fig. 4. Screenshot of the web UI's main view. (1) Text field with the ruleset currently navigated to or entered by the user, (2) Action menus for interactive feedback and data analysis, (3) evaluation results of currently shown ruleset, with buttons to navigate the Pareto front along the target functions, (4) summary information about the current approximation of the Pareto front, (5) other status information.

This objective is an adoption of the concept of classification errors to the problem of actions that are sets of related instances. The major drawback of this cost function is the computational effort required to determine which actions are covered by the current rules.

5.6. Interactive feedback for the algorithm

Our algorithm already addresses most requirements we formulated as part of the problem statement in Section 4. However, we have not yet detailed how domain experts can interact with the rule mining algorithm, i.e., we have not yet addressed the problem described in Section 4.3. The data mining system allows users to give interactive feedback to the algorithm through a web-based graphical user interface (GUI, Fig. 4) in four different ways: by changing the target function for the search; by providing rulesets to consider; by restricting the rule search space; and by reducing the size of the Pareto front. This feedback can be given at any time of the execution of the mining algorithm due to the parallel mining threads that perform the mining tasks.

Changing the target function In several parts of the algorithm, the multi-dimensional cost vector needs to be transformed into a single numeric target value. This single number is used to select the best element when hill-climbing during the local search, as well as during path relinking. It is also used to pick a decent rule as a foundation for further search and generation. Initially, this target function returns the precision of the ruleset. The user can change it to another function to try out other optimization biases at any time. This can be useful to try out different cost factors. Additionally, results from MSOPS (Wagner et al., 2007) indicate that switching between different target functions leads to a better approximation of the Pareto front.

Providing rulesets The user might have an idea for a good rule, e.g., based on her domain knowledge or based on an earlier rule. She can try it out via the user interface. Such a user-provided ruleset is treated similarly to a ruleset generated by a mining thread: If the ruleset is Pareto-optimal, the system adds it to the Pareto front. Moreover, it is added to the mining threads' work items for further optimization by local search and path relinking.[2]

Restricting the rule search space Providing rulesets allows users to suggest solutions to the mining algorithm. However, there are cases where domain experts want to enforce rule constraints, too, e.g., because parts of rules violate domain expertise. Such feedback can be incorporated into the mining process by restricting the search space. One possibility is to reject rules or patterns of rules, for example, all rules with a specific combination of atomic conditions, all rules that compare a numeric feature in a direction deemed invalid, or even all rules using a particular feature. Once the restrictions are in place, the generation algorithm will not create rules that violate the restrictions anymore. Furthermore, all rules that are now forbidden are removed from the Pareto front.

Restricting the search space in the opposite way is also possible: The user can mark a rule as "accepted". This will make the rule appear in every ruleset that is created. Furthermore, the rule is added to all already existing rulesets on the Pareto front, and the combined rulesets are re-evaluated.

It is also possible to undo these restricting actions. To be able to quickly recreate rules that were once found but later declared invalid, invalid rules are cached in the background even after deleting them from the Pareto front.

[2] To improve responsiveness, the treatment of user-generated and thread-generated rulesets differs in the details; see the system's source code (Baum et al., 2018c).

Table 1
Summary of training and test data.

	Training data	Test data
Data collection timeframe	March 2013 to July 2018	August 2018 to November 2018
Number of instances	703,706	26,968
Number of actions	68,960	1,940
MUST_BE instances	14,385 (2.0%)	617 (2.2%)
CAN_BE instances	249,695 (35.5%)	7943 (29.5%)
NOT instances	439,626 (62.5%)	18,408 (68.3%)
Tickets	6005	311

Reduction of Pareto front size When one or several mining threads are running, they keep on creating new rulesets. After some time, this can lead to a large number of rulesets on the Pareto front. This increases the resource consumption of the system. It can also hinder the interactive exploration of the rulesets, especially if many similar rules are found. Many multi-objective evolutionary algorithms restrict the number of solutions in the Pareto front by eagerly removing solutions that are close to each other in objective space (Deb, Pratap, Agarwal, & Meyarivan, 2002). Our system does not clean up eagerly but waits for the user to ask for a clean-up. The user also specifies the number of rulesets that shall be kept. By waiting for a user request, the "knowledge nuggets" in the mined rules are kept as long as possible. Furthermore, the rulesets to remove are not selected by looking at their distance in objective space, because it is easily possible for two semantically different rulesets to have a similar cost vector. Instead, a fingerprint of the rulesets in terms of matched instances is created by selecting a random subset of instances and recording for each ruleset whether it matches or not. The 'distance' between two rulesets is the number of instances for which they differ. This distance is used with hierarchical agglomerative clustering (Duda, Hart, & Stork, 2001) to pick one random rule per cluster that will be kept. Additionally, the optimal rules according to the possible target functions are always kept, too.

6. Evaluation

We evaluated GIMO on a real-world data set in the context of review remark prediction. The details of the review remark prediction study are out of the scope of this article and available separately (Baum et al., 2018a). Within this article, we focus on the comparison between GIMO and existing approaches for the mining of decision rules: the popular machine learning algorithms C4.5 decision trees (Quinlan, 1993), RIPPER (Cohen, 1995), and MetaCost + RIPPER (Domingos, 1999).

6.1. Data

We are not aware of any benchmark data set that is suitable for the use case with a relationship between instances through actions. Therefore, we use the same real-world data we used in our prior study for this comparison. The instances in the data are changes made to a software product, the actions are remarks by reviewers. Statistical evaluation mechanisms like n-fold cross-validation are not possible for interactive data mining because the domain experts cannot repeat the tasks without being influenced by previous trials. Therefore, we use a test set of unseen data that was collected after the rules were created based on the training data. Table 1 shows statistics about the training and test data. The data contains 36 numeric and 15 categorical features. Based on our definition of 'actions' (Section 4), we have three classes of instances as dependent variable:

- instances that `MUST_BE` identified, because they are the only instance causing an action.
- instances that `CAN_BE` identified, because they are part of a set of instances that are causes for an action.
- instances that should `NOT` be identified, because they are not responsible for any action.

Another aspect of our data is that the instances can be partitioned into 'tickets' (development tasks). In our case, a review is performed for all changes that were done in a ticket. Formally, let T be the tickets. Each ticket $t \in T$ is responsible for a subset of the changes made, i.e., $t \subset X$. Moreover, each change part can only belong to exactly one ticket, i.e., $t \cap t' = \emptyset$ for all $t, t' \in X$. Finally, there are no change parts without a ticket, i.e., $\bigcup_{t \in T} t = X$. Moreover, our actions are review remarks. Consequently, we can also define the tickets as the collection of the related remarks, i.e., $\tilde{t} \in \tilde{T}$ such that $\tilde{t} \subset \tilde{X}$ with $\tilde{t} \cap \tilde{t}' = \emptyset$. The tickets are not directly related to the rule generation. However, they are needed for the evaluation of the domain-specific performance metrics in Section 6.3, because the reduction of review effort is also measured on a per-ticket basis.

6.2. Evaluation strategy

We compare GIMO with C4.5, RIPPER, and MetaCost + RIPPER. We use the implementations provided by Weka (Eibe, Hall, & Witten, 2016). We optimize the confidence factor for the pruning of the C4.5 tree with five-fold cross-validation and sampled values between 0.05 and 0.25 in intervals of 0.05. The optimal value is 0.25. We set the minimal number of objects per leaf node to 143, i.e., at least 1% of the instances of the minority class MUST_BE. As stated in the introduction, one of the benefits of GIMO compared to classic cost-sensitive algorithms like MetaCost is that the cost matrix does not need to be specified explicitly. For MetaCost, we chose the cost matrix so that a misclassification that leads to a missed review remark is 100 times as costly as a misclassification that leads to unnecessarily reviewing an instance.

Neither C4.5 nor RIPPER can account for the set cover relationship between actions and instances. This relationship is only relevant for the CAN_BE instances, as they do not lead to a clear binary labeling, but require one object from a grouping to be predicted. We use two options to deal with this: (1) train a binary model in which every instance that is responsible for any action is labeled as positive, which means that the MUST_BE and the CAN_BE instances are merged; and (2) train a multiclass model with all three classes and merge the classes MUST_BE and CAN_BE after the classification. We trained both variants, i.e., binary models RIPPER_2, C4.5_2, and MetaCost_2, as well as multiclass models RIPPER_3, C4.5_3, and MetaCost_3. Additionally, we evaluate the impact of the manual interaction on the rules by the domain experts. To this aim, we use our GIMO approach without any manual interactions and refer to this as GMO in the following.

6.3. Performance metrics

Since GIMO is designed to be directly tailored to the requirements of domain experts, we did not use standard performance measures like accuracy, the F1-Measure or Matthews Correlation Coefficient for the evaluation of our approach. Instead, we used criteria that directly fit the domain requirements: minimize instances, while maximizing the identified actions. Additionally, the complexity of the rules is also relevant in an interactive scenario, as domain experts must be able to interpret the rules on-the-fly. This leads to six relevant performance metrics based on the interaction with domain experts in the use case:

1. Minimize $miss = \sum_{\tilde{t} \in \tilde{T}} \left| \{\tilde{X} \in \tilde{t} : h(\tilde{X}) = 0\} \right| \frac{\log(|\tilde{t}|+1)}{|\tilde{t}|}$.
2. Maximize $gain = \frac{|\{x \in X : h(x) = 0\}|}{|X|}$.

Table 2
Performance metrics on the test data.

	Objectives to minimize			Objectives to maximize		
Algorithm	|C|	FC	miss	gain	wgain	tgain
GIMO	40	17	1.3%	51.9%	23.2%	25.2%
GMO	184	24	0.9%	23.3%	5.0%	3.5%
RIPPER_2	342	25	35.4%	53.0%	41.3%	56.1%
RIPPER_3	200	17	32.5%	52.5%	38.2%	50.8%
C4.5_2	151,084	39	19.2%	52.9%	56.1%	39.5%
C4.5_3	63,872	38	34.1%	48.6%	46.3%	38.3%
MetaCost_2/3	–	–	–	–	–	–

3. Maximize $wgain = \frac{\sum_{x \in X: h(x)=0} weight(x)}{\sum_{x \in X} weight(x)}$.
4. Maximize *tgain*, i.e., the 20% trimmed mean (Wilcox, 2011) over all tickets T of the gain per ticket $\frac{|\{x \in t: h(x)=0\}|}{|t|}$.
5. Minimize $|C|$, i.e., the number of atomic conditions c of the rule C.
6. Minimize *FC*, i.e., the number of features used by the rules.

The first metric measures how well the actions are identified per ticket, i.e., the remarks per ticket. We use the logarithm because otherwise, tickets with many remarks would be overrepresented in the metric. The second and third objectives measure how much the effort is reduced with respect to the predicted instances. The third metric takes into account that the effort per instance is not necessarily the same. In our use case, the effort is directly related to the lines of code that must be reviewed, which are our weights for the instances. The fourth metric measures the reduced effort per ticket as the trimmed mean of the gain per ticket. The trimmed mean ensures that this is not dominated by outlier tickets with many or very few remarks. The fifth measure is the length of rules, as each atomic condition must be processed and understood by a domain expert. Rules with fewer conditions are easier to process and, therefore, preferable. The sixth criterion uses this intuition for the number of features that are used. Every feature that is used by the rules must be understood by the domain expert. If fewer features are used, this reduces the burden on the domain experts.

6.4. Results

Table 2 shows the results of the algorithms on the test data. The results show that the rule set that was created by GIMO that includes the developer feedback is smaller than most others. It has only 40 conjunctions using 17 of the features. Especially for the number of conjunctions, this is far smaller than the others. GMO is the next smallest with 184 conjunctions in the ruleset. The C4.5 decision trees are very verbose and lead to many different rules. The problem is that due to the amount of data, there is strong support for many decisions. Both variants of MetaCost led to the trivial rule “review everything”, which means no misses but also no gains.

For the missed actions, GIMO performs second best with 1.3% missed actions and is only outperformed by GMO which misses 0.9% of the actions. Thus, both the interactive and non-interactive variants of our approach are very good at identifying the actions. RIPPER and C4.5 both perform worse regarding this metric with performance values between 19.2% and 34.1%.

Regarding the *gain* without weighting, all algorithms except for GMO lead to similar results with values between 46.3% and 53.0%. GMO only achieves a gain of 23.3%. If we use the weighting, this changes a bit. RIPPER and C4.5 save most effort if weighting is used, i.e., between 38.2% and 56.1%. GIMO still saves 23.2%, GMO only 5.0%. If we consider the 20% trimmed mean of the gain per ticket *tgain*, we observe by far the largest relative difference to *gain* for GMO, which drops from 23.2% *gain* to only 3.5% *tgain*. This indicates that GMO mostly saves effort by ignoring changes for a few very large tickets, but does not save effort in most cases. This limits the usefulness of GMO in a real-world scenario, as savings would only occur sporadically. A likely cause is overfitting of GMO, which only uses precision as the target function. Overfitting is countered by the manual interactions with GIMO.

Overall, GIMO is among the best algorithms for most performance metrics. We further observe that the interactive feedback provided by domain experts was very helpful for the overall result, i.e., at the cost of very few missed actions, there was a big gain in the saved effort. We also observe that standard rule mining systems like RIPPER and C4.5 are not able to accurately discover actions, i.e., the relationship between instances must be known for the identification of good rules.

6.5. Threats to validity

We identified several threats to the validity of our results. The generalizability of our findings is questionable, because we only used a single data set for our experiments. While we believe that there are other use cases that have a similar relationship between instances, we have no data from such an additional use case. Because the interactive elements of GIMO require interaction with domain experts, we also cannot mitigate this problem using simulated data. Moreover, we only compared GIMO to its non-interactive variant GMO, and to three rule miners from the literature: RIPPER, C4.5 rules, and MetaCost + RIPPER. Other algorithms may perform better in our context. However, the literature suggests that there are no major performance differences between standard rule miners (Rijnbeek & Kors, 2010; Rückert & Raedt, 2008), i.e., we would expect that the result may be slightly different, but without an impact on our conclusions. The results may also change if different performance criteria are used. However, the biggest advantage of GIMO is in the *miss* criterion, which is the false negative rate adopted to our scenario with relations between instances.[3] Thus, it is likely that other use cases with such data have similar performance criteria. Our case study design does not allow us to distinguish the exact causes of GIMO’s success, e.g., which interactions exactly were responsible for GIMO outperforming GMO.

Interpretability of the found rulesets is an important concern in our study. With 17 features and roughly 40 conditions, one might ask whether the GIMO ruleset is still too complex. On the other hand, due to the multi-objective nature of our approach, the domain experts could have easily chosen a simpler ruleset. Because they did not do so, we consider this threat under control. The highly structured form of the rulesets and the interactive features of the UI might have contributed to ease understanding.

7. Future work and conclusions

The current implementation of GIMO (Baum et al., 2018c) is application-specific, but many of the concepts are not. Furthermore, the concepts are separable to some degree: The mining algorithm is independent of the system’s exploration features; the algorithm does not depend on the specifics of the evaluation function (albeit some design choices could have been different if the

[3] We also apply the logarithm to the misses per ticket, but this does not impact the difference. Additional results without the log transformation can be found in (Baum et al., 2018a).

evaluation of a ruleset was less time-consuming); and in the overall approach as depicted in Fig. 3, it would even be possible to replace rulesets with other human-understandable models that allow fine-grained feedback (i.e., "decomposable" models, (Barredo Arrieta et al., 2020)). By adjusting the implementation's architecture to reflect this separability, the approach would be more easily reusable by other researchers. Alternatives for parts of the algorithm could be studied independently, e.g., an alternative rule generator based on association rules, similar to ROCCER (Prati & Flach, 2005). It could be possible to devise adapters to data mining suites like Weka (Witten et al., 2011), which already contains simple user-driven classifiers.

In its current form, GIMO can be used for binary classification that respects set covers. We are working on a general-purpose version for multi-class problems. It is available online[4], but has not yet been tested in a real-life setting. Besides, the GIMO algorithm contains multiple options for tweaking, like the probability of generation versus perfection in the mining thread's main loop, or the strategy for choosing rulesets to optimize and combine. So far, we only studied these options for our application area. Future work could systematically determine which variants are best under which conditions.

GMO used precision as a fixed target function. The bad results for GMO on the test data indicate overfitting. Future work could analyze whether using a different target function or automatically rotating target functions leads to more robust results.

The explainability of opaque models, like deep neural networks, is heavily researched at the moment. One approach is to build a simplified human-understandable model that approximates the original model. As the right balance between understandability and approximation quality is of prime importance here, our multi-objective user-informed approach could be useful.

Declaration of Competing Interest

None.

Appendix A. Notations

Mathematical notations used in this article, for reference:

- m: number of instances
- n_1: number of numeric features
- n_2: number of nominal features
- $n = n_1 + n_2$: number of features
- $\mathcal{F} = \mathbb{R}^{n_1} \times \mathbb{Z}_+^{n_2}$ feature space with $\mathbb{R}$ the real valued numbers and $\mathbb{Z}_+$ the positive integers.
- $X = \{x_1, \ldots, x_m\} \subset \mathcal{F}$: instances described by features
- $Y = \{y_1, \ldots, y_m\} \in \{0, 1\}^m$: labels
- $\bar{X}_i$: actions as a subset of X such that the instances in $\bar{X}_i$ are all causes for an action.
- $\bar{m}$: number of actions
- $\bar{X}^{all}$: all actions; $\bar{X}^{all} = \{\bar{X}_1, \ldots, \bar{X}_{\bar{m}}\}$
- T: tickets as a partition of the set X, i.e., for all $t, t' \in T : t \cap t' = \emptyset$ and $\bigcup_{t \in T} t = X$.
- Upper case C: conjunction
- Lower case c: atomic boolean condition
- h: hypothesis/classifier calculated by a learning algorithm
- *cost*(*h, X, Y*): cost function for the evaluation of the quality of a hypothesis h on labeled data X with labels Y
- o: number of cost functions in multi-objective problem
- p: number of conjunctions in a DNF (disjunctive normal form)
- q: number of atomic conditions on a conjunction
- Ruleset: a disjunctive normal form
- Rule: a single disjunction C
- *tp, tn, fp, fn*: true/false positive/negative

Appendix B. Pseudo-code descriptions of the algorithm

This appendix contains pseudo-code descriptions for the main parts of the algorithm: Its main loop (Fig. B.5), the greedy generation of rulesets (Fig. B.6), the local search for improved rulesets (Fig. B.7), and the combination of rulesets with path relinking (Fig. B.8).

[4] https://github.com/tobiasbaum/GIMO-m/

```
1  repeat
2    if there is work on the local search queue
3      ruleToWorkOn := blackboard.takeFromLocalSearchQueue()
4      localSearch(
5        combineRuleSets(ruleToWorkOn, blackboard.getBestResult()),
6        blackboard.currentTargetFunction)
7      resultParetoSet := localSearch(
8        ruleToWorkOn,
9        blackboard.currentTargetFunction)
10     blackboard.addToPathRelinkingQueue(best item from
         resultParetoSet)
11     blackboard.addToPathRelinkingQueue(random item from
         resultParetoSet)
12   else if there is work on the path relinking queue
13     ruleToWorkOn := blackboard.takeFromPathRelinkingQueue()
14     pathRelink(
15         ruleToWorkOn,
16         blackboard.getBestResult(),
17         blackboard.currentTargetFunction)
18     pathRelink(
19         ruleToWorkOn,
20         blackboard.getRandomResult(),
21         blackboard.currentTargetFunction)
22   else
23     pick one at random:
24       //perfection by path relinking
25       pathRelink(
26           blackboard.getBestResult(),
27           blackboard.getRandomResult(),
28           blackboard.currentTargetFunction)
29     or
30       //perfection by local search
31       localSearch(
32           blackboard.getRandomResult(),
33           blackboard.currentTargetFunction)
34     or
35       //generation of new rule material
36       newRuleSet := generateNewRuleSet(iteration count)
37       blackboard.addToParetoFrontIfPossible(newRuleSet)
38       blackboard.addToLocalSearchQueue(newRuleSet)
39     end-pick
40   end-if
41 until the user stops the mining thread
```

Fig. B5. Pseudo code for the mining thread's main loop (simplified; see Baum et al., 2018c for full source code).

```
1  function generateNewRuleSet(countLimit)
2    data := sampleSubsetOfFeatures(blackboard.allData)
3    data := sampleSubsetOfInstances(data)
4
5    ruleSet := empty ruleset
6
7    maxExclusionRuleCount := random number between 0 .. countLimit
8    uncovered := data
9    while there are less than maxExclusionRuleCount exclusion
         rules
10     searchBias := select random search bias
11     rule := findExclusionRuleByGreedyTopDown(uncovered,
           searchBias)
12     if no rule found
13       break
14     end-if
15     ruleSet := ruleSet.addExclusion(rule)
16     uncovered := removeCoveredInstances(uncovered, rule)
17   end-while
18
19   maxInclusionRuleCount := random number between 0 .. countLimit
20   uncovered := data
21   while there are less than maxInclusionRuleCount inclusion
         rules
22     searchBias := select random search bias
23     rule := findInclusionRuleByGreedyTopDown(uncovered,
           searchBias)
24     if no rule found
25       break
26     end-if
27     ruleSet := ruleSet.addInclusion(rule)
28     uncovered := removeCoveredInstances(uncovered, rule)
29   end-while
30
31   return ruleSet
32 end-function
```

Fig. B6. Pseudo code for the generation of new rulesets (simplified and not including the set cover heuristic; see Baum et al., 2018c for full source code).

```
1  function localSearch(initialRuleSet, targetFunction)
2    candidateRules := all rules from initialRuleSet
3    neighborhoodType := RULE_ADDING
4    resultParetoSet := {initialRuleSet}
5    resultParetoSet.add(emptyRuleSet())
6    current := emptyRuleSet()
7
8    repeat
9      possibleMoves := rulesets in neighborhood of current ruleset
           , restricted to neighborhoodType
10     shuffle possibleMoves randomly
11
12     //add neighbors to Pareto set and determine best neighbor
           for targetFunction
13     bestNeighbor := current
14     foreach move in possibleMoves
15       resultParetoSet.add(move)
16       if (targetFunction(move) < targetFunction(bestNeighbor)
17           or (targetFunction(move) == targetFunction(
                 bestNeighbor) and move could be added))
18         bestNeighbor := move
19         isOnPlateau := costVector(current) == costVector(
               bestNeighbor)
20       end-if
21     end-foreach
22
23     if bestNeighbor == current
24         or (isOnPlateau and search has been on plateau for too
               long)
25       //no improvement found, leave the current neighborhood
26       if neighborhoodType == RULE_ADJUSTING
27         neighborhoodType := RULE_ADDING
28       else
29         ruleAddingDidNotImproveAnything := true
30       end-if
31     else
32       //move to best neighbor and try to improve it by adjusting
33       current := bestNeighbor
34       neighborhoodType := RULE_ADJUSTING
35     end-if
36   until ruleAddingDidNotImproveAnything
37   return resultParetoSet
38 end-function
```

Fig. B7. Pseudo code for local search (simplified; see Baum et al., 2018c for full source code).

```
1  procedure pathRelink(startRuleSet, endRuleSet, targetFunction)
2    relinkActions := differences from startRuleSet to endRuleSet
3    current := startRuleSet
4    while relinkActions is not empty
5      goodAction := determineGoodAction(current, relinkActions)
6      current := goodAction.apply(current)
7      relinkActions.remove(goodAction)
8    end-while
9  end-procedure
10
11 function determineGoodAction(current, relinkActions,
       targetFunction)
12   currentCost := targetFunction(current)
13   bestValue := positive infinity
14   bestAction := null
15   foreach action in relinkActions
16     candidate := action.apply(current)
17     candidateValue := targetFunction(candidate)
18     //implicitly add each candidate to Pareto front if possible
19     try to add candidate to Pareto front
20     if candidateValue < currentCost
21       //stop at the first improvement to keep the number of
             evaluations down
22       return action
23     end-if
24
25     if (candidateValue < bestValue or (candidateValue ==
         bestValue and candidate dominates bestAction.apply(
         current)))
26       bestValue := candidateValue
27       bestAction := action
28     end-if
29   end-foreach
30
31   return bestAction
32 end-function
```

Fig. B8. Pseudo code for path relinking (simplified; see Baum et al., 2018c for full source code).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswax.2020.100040.

CRediT authorship contribution statement

Tobias Baum: Conceptualization, Methodology, Software, Investigation, Resources, Data curation, Writing - original draft, Project administration, Funding acquisition. **Steffen Herbold:** Software, Validation, Investigation, Resources, Writing - original draft, Funding acquisition. **Kurt Schneider:** Resources, Writing - review & editing, Supervision, Funding acquisition.

References

Anguita-Ruiz, A., Segura-Delgado, A., Alcalá, R., Aguilera, C. M., & Alcalá-Fdez, J. (2020). Explainable artificial intelligence (XAI) for the identification of biologically relevant gene expression patterns in longitudinal human studies, insights from obesity research. *PLOS Computational Biology, 16*(4), e1007792.

Ankerst, M. (2002). Report on the SIGKDD-2002 panel the perfect data mining tool: Interactive or automated? *ACM SIGKDD Explorations Newsletter, 4*(2), 110–111.

Ankerst, M., Ester, M., & Kriegel, H.-P. (2000). Towards an effective cooperation of the user and the computer for classification. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 179–188). ACM.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., … Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion, 58*, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012.

Baum, T., Herbold, S., & Schneider, K. (2018a). An industrial case study on shrinking code review changesets through remark prediction. arXiv preprint arXiv:1812.09510

Baum, T., Herbold, S., & Schneider, K. (2018b). A multi-objective anytime rule mining system to ease iterative feedback from domain experts. ArXiv preprint arXiv:1812.09746

Baum, T., Herbold, S., & Schneider, K. (2018c). Online appendix for "an industrial case study on shrinking code review changesets through remark prediction". 10.6084/m9.figshare.7438676.

Baum, T., Leßmann, H., & Schneider, K. (2017). The choice of code review process: A survey on the state of the practice. In M. Felderer, D. Méndez Fernández, B. Turhan, M. Kalinowski, F. Sarro, & D. Winkler (Eds.), *Product-focused software process improvement* (pp. 111–127). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-69926-4_9.

Baum, T., Liskin, O., Niklas, K., & Schneider, K. (2016). A faceted classification scheme for change-based industrial code review processes. In *Software quality, reliability and security (QRS), 2016 ieee international conference on* (pp. 74–85). Vienna, Austria: IEEE. https://doi.org/10.1109/QRS.2016.19.

Baykasoğlu, A., & Özbakir, L. (2007). MEPAR-miner: Multi-expression programming for classification rule mining. *European Journal of Operational Research, 183*(2), 767–784.

Benson, H. P., & Sayin, S. (1997). Towards finding global representations of the efficient set in multiple objective mathematical programming. *Naval Research Logistics, 44*(1), 47–67.

Bernadó-Mansilla, E., & Ho, T. K. (2005). Domain of competence of XCS classifier system in complexity measurement space. *IEEE Transactions on Evolutionary Computation, 9*(1), 82–104.

Bose, I., & Mahapatra, R. K. (2001). Business data mining – a machine learning perspective. *Information & Management, 39*(3), 211–225. https://doi.org/10.1016/S0378-7206(01)00091-X.

Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5–32.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture, vol. 1: A system of patterns*. Wiley.

Cohen, W. W. (1995). Fast effective rule induction. In *Twelfth international conference on machine learning* (pp. 115–123). Morgan Kaufmann.

Dam, H. K., Tran, T., & Ghose, A. (2018). Explainable software analytics. In *Proceedings of the 40th international conference on software engineering: New ideas and emerging results* (pp. 53–56). ACM.

De Jong, K. A., Spears, W. M., & Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning, 13*(2–3), 161–188.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

Dehuri, S., & Mall, R. (2006). Predictive and comprehensible rule discovery using a multi-objective genetic algorithm. *Knowledge-Based Systems, 19*(6), 413–421.

Dehuri, S., Patnaik, S., Ghosh, A., & Mall, R. (2008). Application of elitist multi-objective genetic algorithm for classification rule generation. *Applied Soft Computing, 8*(1), 477–487.

Domingos, P. (1999). MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 155–164). ACM.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. Wiley.

Eggermont, J., Eiben, A. E., & van Hemert, J. I. (1999). A comparison of genetic programming variants for data classification. In *International symposium on intelligent data analysis* (pp. 281–290). Springer.

Eibe, F., Hall, M., & Witten, I. (2016). *The WEKA workbench. Online appendix for data mining: practical machine learning tools and techniques*. Morgan Kaufmann.

Fidelis, M. V., Lopes, H. S., & Freitas, A. A. (2000). Discovering comprehensible classification rules with a genetic algorithm. In *Evolutionary computation, 2000. proceedings of the 2000 congress on: 1* (pp. 805–810). IEEE.

Freitas, A. A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing* (pp. 819–845). Springer.

Freitas, A. A. (2014). Comprehensible classification models: A position paper. *ACM SIGKDD Explorations Newsletter, 15*(1), 1–10.

Fürnkranz, J., & Kliegr, T. (2015). A brief overview of rule learning. In *International symposium on rules and rule markup languages for the semantic web* (pp. 54–69). Springer.

Fürnkranz, J., Kliegr, T., & Paulheim, H. (2018). On cognitive preferences and the interpretability of rule-based models. arXiv preprint arXiv:1803.01316

German Government (2018). Big data und künstliche intelligenz im versicherungssektor. http://dipbt.bundestag.de/extrakt/ba/WP19/2418/241800.html.

Glover, F. (1989). Tabu search part I. *ORSA Journal on Computing, 1*(3), 190–206. https://doi.org/10.1287/ijoc.1.3.190.

Grosan, C., & Abraham, A. (2007). Hybrid evolutionary algorithms: Methodologies, architectures, and reviews. In *Hybrid evolutionary algorithms* (pp. 1–17). Springer.

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. *Software Engineering IEEE Transactions, 38*(6), 1276–1304. https://doi.org/10.1109/TSE.2011.103.

Han, J., & Cercone, N. (2002). Interactive construction of classification rules. In *Pacific-asia conference on knowledge discovery and data mining* (pp. 529–534). Springer.

Han, J., Lakshmanan, L. V., & Ng, R. T. (1999). Constraint-based, multidimensional data mining. *Computer, 32*(8), 46–50.

Hellerstein, J. M., Avnur, R., Chou, A., Hidber, C., Olston, C., Raman, V., … Haas, P. J. (1999). Interactive data analysis: The control project. *Computer, 32*(8), 51–59.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(8). https://doi.org/10.1109/34.709601.

Holzinger, A. (2016). Interactive machine learning for health informatics: When do we need the human-in-the-loop? *Brain Informatics, 3*(2), 119–131.

Hosseini, S., Turhan, B., & Gunarathna, D. (2018). A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, 1-1. https://doi.org/10.1109/TSE.2017.2770124.

Huysmans, J., Dejaeger, K., Mues, C., Vanthienen, J., & Baesens, B. (2011). An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems, 51*(1), 141–154.

Ishida, C. Y., De Carvalho, A. B., Pozo, A. T., Goldbarg, E. F., & Goldbarg, M. C. (2008). Exploring multi-objective PSO and grasp-pr for rule induction. In *European conference on evolutionary computation in combinatorial optimization* (pp. 73–84). Springer.

Ishida, C. Y., Pozo, A., Goldbarg, E., & Goldbarg, M. (2009). Multiobjective optimization and rule learning: Subselection algorithm or meta-heuristic algorithm? In *Innovative applications in data mining* (pp. 47–70). Springer.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning, 13*(2–3), 189–228.

Janssen, F., & Fürnkranz, J. (2010). On the quest for optimal rule learning heuristics. *Machine Learning, 78*(3), 343–379.

Johansson, U., Niklasson, L., & König, R. (2004). Accuracy vs. comprehensibility in data mining models. In *Proceedings of the seventh international conference on information fusion: 1* (pp. 295–300).

Kaya, M. (2010). Autonomous classifiers with understandable rule using multi-objective genetic algorithms. *Expert Systems with Applications, 37*(4), 3489–3494.

Kwedlo, W., & Kretowski, M. (2001). An evolutionary algorithm for cost-sensitive decision rule learning. In *European conference on machine learning* (pp. 288–299). Springer.

Lavrač, N., Fürnkranz, J., & Gamberger, D. (2010). Explicit feature construction and manipulation for covering rule learning algorithms. In *Advances in machine learning i* (pp. 121–146). Springer.

Mariscal, G., Marban, O., & Fernandez, C. (2010). A survey of data mining and knowledge discovery process models and methodologies. *The Knowledge Engineering Review, 25*(2), 137–166.

Martí, R., Campos, V., Resende, M. G., & Duarte, A. (2015). Multiobjective grasp with path relinking. *European Journal of Operational Research, 240*(1), 54–71.

Moeyersoms, J., de Fortuny, E. J., Dejaeger, K., Baesens, B., & Martens, D. (2015). Comprehensible software fault and effort prediction: A data mining approach. *Journal of Systems and Software, 100*, 80–90.

Mühlbacher, T., Linhardt, L., Möller, T., & Piringer, H. (2018). TreePOD: Sensitivity-aware selection of pareto-optimal decision trees. *IEEE Transactions on Visualization and Computer Graphics, 24*(1), 174–183.

Pavanelli, G., Arns Steiner, M. T., Góes, A. R. T., Pavanelli, A. M., & Costa, D. M. B. (2014). Extraction of classification rules in databases through metaheuristic procedures based on grasp. In *Advanced materials research: 945* (pp. 3369–3375). Trans Tech Publ.

Pazzani, M. J. (2000). Learning with globally predictive tests. *New Generation Computing, 18*(1), 29–38.

Prati, R. C., & Flach, P. A. (2005). ROCCER: An algorithm for rule learning based on ROC analysis.. In *Ijcai* (pp. 823–828).

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Radjenović, D., Heričko, M., Torkar, R., & Živkovič, A. (2013). Software fault prediction metrics: A systematic literature review. *Information and Software Technology, 55*(8), 1397–1418.

Resende, M. G., & Ribeiro, C. C. (2016). *Optimization by GRASP*. Springer.

Reynolds, A. P., & De la Iglesia, B. (2009). A multi-objective grasp for partial classification. *Soft Computing, 13*(3), 227–243.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). 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* (pp. 1135–1144). ACM.

Rigby, P. C., Cleary, B., Painchaud, F., Storey, M., & German, D. M. (2012). Contemporary peer review in action: Lessons from open source development. *Software IEEE, 29*(6), 56–61.

Rijnbeek, P. R., & Kors, J. A. (2010). Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. *Machine Learning, 80*(1), 33–62. https://doi.org/10.1007/s10994-010-5168-9.

Rückert, U., & Raedt, L. D. (2008). An experimental evaluation of simplicity in rule learning. *Artificial Intelligence, 172*(1), 19–28. https://doi.org/10.1016/j.artint.2007.06.004.

Rudin, C. (2018). Please stop explaining black box models for high stakes decisions. arXiv preprint arXiv:1811.10154

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A Modern approach* (3rd). Upper Saddle River, NJ, USA: Prentice Hall Press.

Ryan, M., & Rayward-Smith, V. (1998). The evolution of decision trees. In *Genetic programming 1998: Proc. 3rd annual conf* (pp. 350–358).

Sachan, S., Yang, J.-B., Xu, D.-L., Benavides, D. E., & Li, Y. (2020). An explainable ai decision-support-system to automate loan underwriting. *Expert Systems with Applications, 144*, 113100.

Settles, B. (2009). Active learning literature survey. *Technical Report*. University of Wisconsin-Madison Department of Computer Sciences.

Skiena, S. S. (1998). *The algorithm design manual*. Springer.

Tan, M., Tan, L., Dara, S., & Mayeux, C. (2015). Online defect prediction for imbalanced data. In *Proceedings of the 37th international conference on software engineering-volume 2* (pp. 99–108). IEEE Press.

Wagner, T., Beume, N., & Naujoks, B. (2007). Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *International conference on evolutionary multi-criterion optimization* (pp. 742–756). Springer.

Wilcox, R. R. (2011). *Introduction to robust estimation and hypothesis testing*. Academic Press.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining–practical machine learning tools and techniques* (3rd). Elsevier.

Zhang, Y., Li, Z., & Cui, K. (2005). DRC-BK: Mining classification rules by using boolean kernels. In *International conference on computational science and its applications* (pp. 214–222). Springer.

Zhao, Y., Yao, Y., & Yan, M. (2007). ICS: An interactive classification system. In *Advances in artificial intelligence* (pp. 134–145). Springer.