Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Explaining Mispredictions of Machine Learning Models with Influential Input Features

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von: Dennis Buchwinkler
geboren am: 05.05.1991
geboren in: Berlin

Gutachter/innen: Prof. Dr. Lars Grunske
Prof. Dr. Jürgen Cito

eingereicht am: .... verteidigt am: ....

Abstract

Explainable machine learning is important to provide us humans with the means to understand the decisions of an otherwise black box like machine learning model. One way to obtain insight into the model is trying to construct a rule set which describes under which conditions a given machine learning model's prediction is most likely not reliable. Machine Learning Model Diagnosis (MMD) [Cit+21] proposed by Cito et al. is an approach to generate such a rule set by linking input feature values to model mispredictions. The rule set is built iteratively by utilizing rule induction to find the best performing combination of atomic predicates. Atomic predicates are derived from the input data set and are a combination of an input feature, a value and a comparison operator. In this thesis, we build on the work of Cito et al., to improve rule set performance by replacing the rule generation algorithm based on rule induction with three different approaches: ISLearn, decision trees and Bayesian optimization. We also tackle the problem of high computational demand with growing input data sets, by proposing and evaluating the idea to first identifying the features most influential in causing a misprediction of the model. Then we utilize only these most influential features to generate the misprediction rule sets. We implement our approaches to generate misprediction explanations and evaluate their effectiveness on machine learning models trained on a set of 11 different real-world data sets. Our evaluation shows that our decision tree approach, in comparison to the baseline, was able to improve performance slightly, while reducing computational demand considerably in almost all cases. Utilizing only the most influential input features in rule construction reduced execution time significantly (for approaches with scalability problems), with only a small decrease in performance.

Contents

1 Introduction 7

2 Related Work 11

3 Background 15
- 3.1 Terminology . . . 15
- 3.2 Machine Learning . . . 16
  - 3.2.1 Decision Tree . . . 17
  - 3.2.2 Random Forest . . . 18
- 3.3 MMD: Machine Learning Model Diagnosis . . . 19
- 3.4 ISLearn . . . 20
- 3.5 Bayesian Optimization . . . 21

4 Generating Misprediction Explanations 23
- 4.1 Training Machine Learning Model . . . 24
- 4.2 Labeling Mispredictions . . . 24
- 4.3 Extracting Influential Input Features for Faster Computation . . . 25
- 4.4 Learning Rule Sets . . . 27
- 4.5 Formatting Outputs . . . 33

5 Experimental Setup 35

6 Evaluation 39
- 6.1 Experimental Results with all Input Features . . . 39
  - 6.1.1 Performance Metrics . . . 39
  - 6.1.2 Computational Demand . . . 44
  - 6.1.3 Explanation Length . . . 47
- 6.2 Experimental Results with most influential Input Features . . . 49
  - 6.2.1 Performance Metrics . . . 49
  - 6.2.2 Computational Demand . . . 55
  - 6.2.3 Explanation Length . . . 59
- 6.3 Threats to Validity . . . 61

7 Discussion and Limitations 63

8 Conclusion and Future Work 67

References 69

Appendix 73

1 Introduction

In today's world machine learning is a promising approach to tackle a great variety of problems. As the complexity of the machine learning systems grows it becomes more and more complicated, if not impossible, for a human to comprehend, how the employed machine learning model arrived at a specific conclusion. When machine learning is utilized in safety-critical systems or systems that might severely alter a person's life, for example in traffic control [Lee+20], in medical diagnosis [CG19], and self-driving vehicles [Ni+20], it is essential to resort to every possible option to understand risk factors. Explainable machine learning is a research field that attempts to extract some information explaining the reasons of decisions made by a machine learning model, thus helping to shed some light into the otherwise magical black box. It is done by utilizing varying methods to produce a human interpretable output, including assignment of importance values to input features [LL17] and constructing more interpretable surrogate models imitating the behavior of the black box. This enables the possibility to debug the system when an error occurred, to actually grasp what caused faulty decisions and hopefully understand how to reduce them. However, most well-known explainable machine learning approaches like LIME [RSG16] or partial dependence plots (PDP) [Fri01] focus on the relationship between input data and output of the trained machine learning model, never considering the possible ground truth. In Machine Learning Model Diagnosis (MMD) [Cit+21] Cito et al. proposed an approach to employ rule induction to construct an easily human interpretable rule set containing the input feature value mappings, under which a machine learning model's output cannot be trusted, because it is likely a misprediction. First an input data set with a known ground truth is used on a trained model to label the data set on which data is mispredicted. Then MMD generates a catalogue of atomic predicates from the input data set, which are a combination of an input feature, a value and a comparison operator. Finally rule induction is utilized to find the best performing combination of atomic predicates and iteratively build a rule set.

In this thesis, we build on the research of Cito et al. [Cit+21] improving the concept in two different ways. On the one hand, we replace rule induction, the rule set generation algorithm in MMD, with three different approaches, specifically ISLearn, decision trees and Bayesian optimization, which could improve performance metrics of the obtained rule set. However, while performance metrics are important, rule set length must also be considered, to keep rule set interpretable for humans and prevent overfitting to the input data set. On the other hand, we seek to reduce computational demand by selecting input features that have a high impact on causing a misprediction of the machine learning model. Similar to MMD the first step of our approach uses a trained machine learning model to label the mispredictions made for an input data set. In the second step we train a random forest classifier on the now misprediction labeled data set. Then we utilize random forest feature importance to extract which input features are the most important

```
Best ruleset with all features and bayesian optimization with disjunctions:

serum_creatinine > 1.02 and diabetes > 0.53 or
smoking <= 0.16 and age > 80.35 or
high_blood_pressure > 0.06 and diabetes > 0.88 or
high_blood_pressure <= 0.02 and age > 79.80

Specificity: 0.833 Precision: 0.706 Recall: 0.8
```

Figure 1: Part of an output for a run creating a rule set by utilizing Bayesian optimization for a heart failure machine learning model

ones to predict mispredictions of our original machine learning model. In the next step we remove uninfluential features from the input data set and then use the drastically reduced data set to iteratively build a rule set with our approaches: ISLearn, decision trees and Bayesian optimization. In the final step the output is formatted to have the same look, no matter which approach was used. In Figure 1 a part of an output example is shown. This output belongs to a run where we used Bayesian optimization to create the rule set for a machine learning model trained on a heart failure data set. It describes under which combination of input feature values the model likely mispredicts. To limit the scope of this thesis we only allowed one conjunction per rule which impairs performance metrics but increases interpretability.

With this thesis, we strive to utilize and evaluate a number of different approaches to generate misprediction explanations, which replace rule induction employed in MMD. We also tackle the problem of high computational demand when constructing misprediction explanations for machine learning models based on big data sets with many features, by identifying and using only the most influential input features for rule set generation. Finally, we consider the length of the rule sets generated, because shorter misprediction explanations are easier to comprehend for humans, therefore they are preferable. We implemented our approaches to evaluate and compare their performance against each other and against MMD. For that we took advantage of 11 machine learning models trained on a set of 11 different real-world data sets. The following questions encapsulate the essence of our research:

RQ1 Do our approaches generate misprediction explanations with better performance metrics?

RQ2 Do our approaches reduce computational demand?

RQ3 Do our approaches generate short misprediction explanations?

In summary, this thesis makes the following contributions:

- We propose three new approaches to generate misprediction explanations for machine learning models to improve performance metrics: ISLearn, decision trees and Bayesian optimization
- We propose a technique to reduce computational demand creating the rule sets by identifying and utilizing only the most influential features
- We implement our notions, building on the concept proposed by MMD
- We evaluate our implemented approaches and present promising results for our decision tree technique and using only influential input features

The rest of the thesis is organized as follows: Section 2 explores related work about the core concepts relevant to this thesis, interpretability and debugging of machine learning models, rule learning approaches and their field of use. Section 3 lays out background knowledge for the different techniques that are employed in this research: machine learning, MMD, ISLearn and Bayesian optimization. In Section 4 we detail our approach of utilizing ISLearn, decision trees and Bayesian optimization to construct misprediction explanations for machine learning models. In addition, we explain how we extract the input features that are most influential for mispredictions. Section 5 contains information concerning the experimental setup while Section 6 provides the results of our sizeable evaluation. Subsequent to the evaluation in Section 7 we discuss our findings and mention limitations. Finally, in Section 8 we draw attention to future research possibilities.

2 Related Work

In this Section, we provide a short overview of the related work established in literature and current research.

Interpretability of Machine Learning Models Interpretable machine learning has grown into an increasingly important research field, as machine learning algorithms are used for life altering decisions or find their way into more safety-critical systems. A general overview for this research field was compiled by Molnar in his book [Mol22] and in his paper about the history, state-of-the-art and challenges of Explainable Machine Learning [MCB20]. Often research in this area can be categorized into two fields: local and global interpretability. While local interpretability focuses on explaining the decision for one prediction, global interpretability tries to capture the behavior of the whole model. Very well-known local interpretability techniques are LIME [RSG16] proposed by Ribeiro et al. and SHAP [LL17] (which can also be used for global interpretability) proposed by Lundberg and Lee, combining LIME with Shapley Values [Sha+53]. However, these techniques in its basic form have some considerable drawbacks. Glaring stability issues of LIME are documented in [AJ18], while [Sla+20] proposes a way to exploit weak point in the approaches of LIME and SHAP to hide biases and craft desired explanations. Much additional research has been done to improve upon the shortcomings or adapt LIME for different use cases. To name a few: [Vis+22] introduces stability indices to better quantify the stability of LIME, ALIME [SR19] employs a better weighting function for improved stability and local fidelity, MeLIME [Bot+20] considers the distribution of the data used to train the black box to improve local explanations, and OptiLIME [VBC20] proposes a version of LIME that automatically chooses the kernel width, to maximize the stability for a user given adherence. K-LIME [Hal+17] and LIME-SUP [Hu+18] partition the entire input space to fit multiple locally interpretable models. For global interpretability besides the already mentioned SHAP, GALE [LHK19] and DENAS [Che+20] have their focal point on discovering globally influential features, while a different approach is generating simpler, easier explainable surrogate models for the complex models [Lak+]. Many studies are focusing on explaining the reasons for predictions made by machine learning models, but only very few consider the ground truth for inputs and try to explain under which conditions a model likely mispredicts. In Machine Learning Model Diagnosis (MMD) [Cit+21], on which our thesis builds on, Cito et al. proposed an approach to employ rule generation through rule induction to construct an easily human interpretable rule set containing the input feature value mappings, under which a machine learning models output cannot be trusted. One of our goals for this thesis was to reduce computational demand for generating misprediction explanations by selecting important input features, from which to build the rule set. In [Ges+23] Gesi et al. focus on the same problem. They leverage feature bias to select important features to narrow down the feature count, while we consider feature importance of machine learning model trained to predict mispredictions.

Debugging of Machine Learning Models More in line with our work to debug machine learning models and identify groups of data causing mispredictions is Errudite [Wu+19] proposed by Wu et al., seeking to conduct error analysis for NLP models. Errudite requires significant user involvement in manually formulating hypotheses and evaluating misprediction explanations. In contrast, our approach is mostly automated and can be easily adapted to create misprediction explanations for any underlying machine learner. In [Kim+20] Kim et al. proposed an approach, tailored to Computer Vision, that automatically generates compact rules for deep neural networks, which can be used to significantly increase the correct detection rate and help with debugging the network's behavior. A technique to automatically repair neural networks MODE [Ma+18] was proposed by Ma et al. They try to fix model bugs by identifying the model's internal features that are the cause for the model bugs and then select tailored training inputs to alleviate these bugs. Their focus lies on generating additional customized training data to directly fix certain bugs in a neural model, while we strive to provide a model agnostic way to produce misprediction explanations, which can also be used for data augmentation, or be employed in various other ways like output suppression.

Rule Learning Approaches Our approach generating misprediction explanations produces decision lists and therefore is associated to [Riv87] and decision sets [LBL16]. The objective of these techniques is exact classification through accuracy. However, maximizing accuracy does not produce good misprediction explanations. We instead utilize weighted F1 score to evaluate the performance of our rules, escalating the importance of precision while not fully neglecting recall. Rule learning approaches can be split into two classes: predictive rule discovery and descriptive rule discovery. Predictive rule discovery generalizes data so that predictions for new examples can be made. Some notable techniques are Ripper [Coh95], CN2 [CN89] and ID3 [Qui87]. In descriptive rule discovery the key focal point lies on defining rules that encapsulate patterns present in a given data set. Furthermore, for descriptive rule discovery two sub classes can be defined: association rule discovery (unsupervised learning) and subgroup discovery (supervised learning). In association rule discovery arbitrary dependencies between attributes are considered. Subgroup discovery, where subgroups for a chosen property of interest are established [Atz15], is the sub class we would assign our research to, since we aim to find the subgroup of data with a high misprediction rate.

Applications of Rule Learning In the following we note some research utilizing rule learning techniques. Song et al. proposed association rule mining-based methods to predict defect associations and defect correction effort [Son+06] to support developers in finding defects in software and help in better managing testing resources. With STUCCO [Cas+17] Castelluccio et al. present an algorithm, inspired by contrast set mining, to automatically determine statistically significant properties in crash groups, to make crash reports more understandable and thus help in discovering the root cause of the crash. Contrast set mining has also been employed by Qian et al. to discern different groups of crashes [Qia+20]. They tackle scalability by proposing a way to apply contrast learning to continuous data, removing the need for discretization.

3 Background

In this thesis we build on the concept of MMD to create misprediction explanations for machine learning models. Instead of utilizing rule induction, to construct the rule set, we employ three different approaches: ISLearn, decision trees and Bayesian optimization. In this Section, we give a quick overview of relevant definitions and introduce the techniques employed and the approaches used to generate the rule sets. This contains brief overviews for machine learning algorithms, ISLearn, Bayesian optimization and the research we build on: MMD.

3.1 Terminology

The following definitions that are closely associated to [Mol22] will lay the foundation to avoid different connotations in literature.

Definition 1: Machine Learning Machine learning is a set of methods, where not all instructions must be explicitly given. Instead, computers learn from data to make and improve predictions.

Definition 2: Machine Learning Algorithm A machine learning algorithm, often referred to as a Learner, is the program used to train a machine learning model from a data set

Definition 3: Machine Learning Model A machine learning model is the model trained by a machine learning algorithm, that is able to link inputs to predictions. Since in this thesis we only utilize machine learning models that provide binary classification as predictions, we also refer to them as classifiers.

Definition 4: Black Box Model A black box model is a system where no details about the inner processes are known. In machine learning it means that the model does not disclose the cause for a given prediction. An example for a black box model is a neural network.

Definition 5: White Box Model The opposite of a black box model is a white box model, also called interpretable model. In such a model the individual decision made can be followed and understood. An example for a white box model is a decision tree.

Definition 6: Features Features are the inputs used for classification. They are the columns in a data set. For example, the age of a persons or the number of commits for git repositories.

Definition 7: Interpretable/Explainable Machine Learning Interpretable or Explainable Machine Learning refers to the techniques which make the behavior and predictions of machine learning systems comprehensible to humans.

Definition 8: Model Explanation A model explanation can be many things, like an interpretable decision tree as a surrogate for a more complex model, or feature importance values to be able to gauge the impact of different features. We often use the term "misprediction explanation", which for us refers to a simple and human interpretable decision list, containing only comparisons of features to constant values separated by conjunctions and disjunctions.

3.2 Machine Learning

MMD and our approaches construct explanations for machine learning models, which describe under which conditions a given model is likely to mispredict, allowing to render predictions unreliable in these cases. We also utilize machine learning techniques, namely random forests and decision trees, to select features with high impact on causing mispredictions and as a procedure to build our rule sets. Machine learning techniques can be categorized into three major classes:

- supervised learning,
- unsupervised learning,
- reinforcement learning.

Supervised learning is based on training data, containing inputs and corresponding outputs, and tries to devise a generalized system to link inputs to outputs. In contrast, the data for unsupervised learning does not contain outputs and the goal of the learner is to independently discover patterns or clusters in the data to sort it into different groups. Finally, reinforcement learning focuses on discerning the optimal behavior in an environment to maximize a reward. The behavior is discovered by interacting with the environment and evaluating how it reacts. Our work in this thesis falls under the area of supervised learning.

Supervised learning techniques work on a set of training data consisting of a number of input features and a target, which contains the desired output (label) for every instance (row in data set). The target can be boolean for a binary classification, more than two classes for a multiclass classification, or a continuous value for a regression problem. Based on this data the algorithm builds a mathematical model to learn a function, by continuously optimizing an objective function, that is able to predict the label of a never-before-seen input. The value of a trained model is measured by how accurate it can predict the output for inputs not contained in the training data. In the following, we will introduce the two supervised learning algorithms relevant to our work.

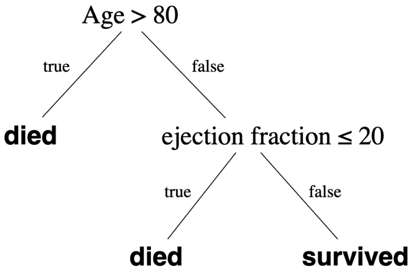


Figure 2: A made up example decision tree (of height h=2) predicting the survival chance after a heart failure. The survival chance is the highest for patients under the age of 80 and with a heart ejection fraction over 20 percent.

3.2.1 Decision Tree

Decision trees can be utilized to handle classification, as well as regression problems. We will focus on decision tree classifiers [SH77], since that is the variation, we use in this thesis. The decision tree classifier operates by deducing a number of decision rule to predict the target for a given training data set. The decision rules are structured in a tree-like composition where every branch in the tree constitutes one of these rules. Every node in the tree contains a comparison of a feature f to a constant value c in the form $f > c$ or $f \leq c$. The leaf node of a branch holds the label for that decision rule while the first node from which all branches originate is called root node. Figure 2 shows a made-up decision tree learned on medical data of heart failure patients. The target of the data is whether a patient survived or died due to a heart failure. The tree predicts that patients, which are either above the age of 80 or have a heart ejection fraction below 20 percent, have the highest risk of death. Based on the training data and training parameters provided the model learned to relate the input features age and ejection fraction with the patient's chance of survival. When utilizing the model to predict the output for a new data instance the evaluation starts at the root node. The comparison contained in the root node is checked against the feature of the new data instance. If the check is successful the branch labeled "true" is traversed down, else if the check fails the "false" branch is chosen. This procedure continues until a leaf node of the tree is reached. The label of the leaf node is the prediction provided by the decision tree model.

The CART algorithm [Bre+84] is one of the most established techniques to construct such decision trees. The most common way to find the best split conditions is to consider the Gini index metric. The Gini index is a metric that ranges from 0 to 1 and measures how pure a node is. A Gini impurity of 0 means that all inputs, used to learn the tree, reaching this node are allied to the same class, while a Gini impurity of 1 signifies that the elements are randomly distributed across the classes. When choosing the split condition, the algorithm tries to minimize Gini impurity for the newly spawned sub nodes. The construction is finished when certain user defined conditions are reached, like a maximum tree depth or a minimum number

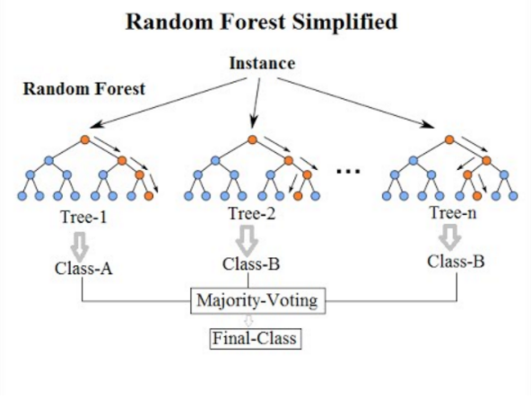


Figure 3: Overview of the Random Forest classifier [Jag17], depicting the approach to combining the predictions of multiple decision trees to derive the final class for an input.

of training instances per leaf node.

Decision trees are white-box machine learning models because their behavior and reasons for predictions can be easily understood by humans. For any arbitrary new input, a human can follow the decision process step by step, starting at the root node, considering each threshold comparison and follow the corresponding branch down the tree, to get a comprehensive grasp on which input features had what impact on the prediction. However, one considerable drawback of decision trees is, that they tend to overfit to the training data when they grow too big. This means, the model might perform well for the training data, but as soon as new unseen inputs are used, performance drops and small changes in feature values can suddenly wildly change the predicted class.

3.2.2 Random Forest

To combat the disadvantages that come with using a single decision tree, ensemble learning can be employed. Ensemble learning is a technique to group up a number of weak base learners, like decision trees, which then work together to present a better final prediction. Every model in the ensemble may struggle with high variance or high bias, but the results can then be combined to reduce the impact and yield increased model performance. One way to build such an ensemble is called bagging [Bre96], where multiple models are trained on different subsets of the training data set. For decision trees specifically this technique was employed to create random forest estimators [Bre01] proposed by Leo Breiman, thus in random forests many CART based decision trees work in tandem to provide a better prediction.

In Figure 3 the concept of the random forest algorithm is shown. A random forest predicts the label for a new input by evaluating the input with every decision tree of the ensemble. After every tree has given its prediction, the final classification is done by either majority voting in case of random forest classifiers, or by calculating the average in case of random forest regressors. As already mentioned for the technique of bagging, random forests in comparison to decision trees, provide better predictions, by reducing variance and bias.

3.3 MMD: Machine Learning Model Diagnosis

Machine Learning Model Diagnosis (MMD) [Cit+21] proposed by Cito et al. is the research which our thesis builds on. It describes an approach to employ rule generation through rule induction to construct an easily human interpretable rule set containing the input feature value mappings, under which a machine learning model's output cannot be trusted. This misprediction explanation can then for example be utilized to gain insight into the limitations of the training data or model itself, to elevate the performance of a predictive model, or for output suppression.

MMD automatically constructs misprediction explanations based on two inputs: a trained machine learning model and a corresponding input data set with known ground truth. When disjunctions are allowed in the rule set then the desired recall value of caught mispredictions also needs to be specified. The technique works fully model agnostic, meaning it can be used for models trained by any kind of learner. The process of generating the rule set works in three steps:

1. Labeling Mispredictions
2. Generating Atomic Predicates
3. Learning Rules

Labeling Mispredictions First, utilizing the provided machine learning model and inputs, a new data set is built by labeling each instance of the inputs on whether they are mispredicted or not. This is done by letting the model predict an output for each input and then comparing that output with the known ground truth.

Generating Atomic Predicates Next, the algorithm determines the building blocks for the misprediction explanation. These are called atomic predicates and are based on the input data set and have the form $x_i op c$ where x_i is a feature, op is a comparison symbol and c is a constant. Every feature of the input data set is considered and if it is a categorical variable all predicates of the form $x_i = c_j$ and $x_i \neq c_j$ are added, where c_j are all the possible categories in the feature. The operators $/leq$, $>$ are used for numerical and ordinal features, while the constants are chosen with equal frequency binning [KK06].

This approach of generating predicates does not scale well for data sets with many features, especially in the next step, when during rule learning all the possible

```
[[Existential]]

name = "Existence Numeric String Larger Than"
constraint = '''
forall <?NONTERMINAL> elem in start:
  (> (str.to.int elem) (str.to.int <?STRING>))
'''
```

Figure 4: An ISLa pattern which covers the correct sample inputs by instantiating an invariant that compares if an input feature is greater than an integer.

predicates need to be evaluated to find the best performing rule. This is why we propose an approach to preselect a number of influential features to reduce computational demand.

Learning Rules Lastly, after generating all the atomic predicates the rule set can be built from them. To reach the desired coverage of mispredictions the rule set is created in iterations. Starting with full input data set, the best performing rule is deduced. Then, if the desired coverage is not reached the cycle starts anew with a subset of the input data, built by removing all instances already covered by the current rule set. In this way every additional rule is generated on a smaller and smaller subset of data, specifically to cover the mispredictions not already caught by other rules. To construct the best performing rule from the atomic predicates beam search is employed. When no further improvements can be made, a single atomic predicate is added to one of the rules in the beam and then compared to the worst rule in the beam. Performance is measured as a linear combination of precision, recall, and rule size. While precision is given a higher weight because it is the primary factor, recall and rule size have to be a part of the evaluation to prevent the misprediction explanation from becoming too long.

With our work we evaluate the outcome of three different approaches (ISLearn, decision trees, Bayesian optimization) of learning the best performing rules and constructing the misprediction explanation.

3.4 ISLearn

ISLearn is a technique for learning ISLa constraints from a set of defined patterns. Both ISLa and ISLearn were proposed by Steinhöfel and Zeller [SZ22].

ISLa, or Input Specification Language, is a string constraint solver that has its own specification language. It allows the user to specify input constraints like “each row of a .csv file has to have at least 10 columns” or “a variable has to be defined before it is used”. This tackles the problem, that grammar-based fuzzers tend to produce an amount of syntactically valid but semantically invalid system inputs. ISLa allows to generate tailored system inputs for testing. By adding or removing input specifications, distinct parts of the program functionality can be checked. The inputs are derived from a language defined by a context-free grammar in Backus–Naur form.

ISLearn, as already mentioned, learns ISLa constraints from inputs and a set of patterns. The patterns are instantiated according to the given inputs and then merged into conjunctive normal form. Inputs can be provided to ISLearn already categorized into positive and negative examples, or a prop function can be defined, which is able to differentiate between valid and invalid inputs. ISLearn can also generate more sample inputs on a given program property. A number of common patterns are already predefined in a catalog, but new patterns can easily be added. In Figure 4 an exemplary pattern is depicted. When running ISLearn that pattern will be instantiated with nonterminal symbols of the grammar defining the language of our inputs and certain integers derived from our sample inputs. Then the performance of the resulting invariants in covering the correct sample inputs are evaluated and ranked based on specificity and recall.

3.5 Bayesian Optimization

Bayesian optimization is a machine learning based optimization method for finding the optimal value of a black box function that is time consuming to evaluate. An extensive overview is given by Frazier [Fra18]. The goal of the algorithm is to maximize a continuous objective function and works best when the input vector does not exceed 20 dimensions. As already mentioned, it is mostly used for functions that are extremely computationally expansive to evaluate, so only a limited amount of evaluation is feasible for reasons like monetary cost or time needed per run. The focus of Bayesian optimization is locating the global optimum of a function and not get stuck in local optima. Since its specialty is optimizing expensive black box functions, Bayesian optimization works very well for tuning hyperparameters in machine learning algorithms, where each new training cycle can be costly.

The two major components of Bayesian optimization are a Bayesian statistical model and an acquisition function. The Bayesian statistical model is a technique to represent the beliefs about the behavior of an unknown function. Bayes' theorem is applied to update the beliefs after new observations. In our context it models the given objective function that has to be optimized. A common choice for the Bayesian statistical model is a gaussian process. A gaussian process is defined by its mean function and covariance function. While the mean function carries the prior assumptions about the behavior of the function, the covariance function holds the expectation regarding the smoothness of the function. As new observations about the objective function are made the mean and covariance functions are updated to form the posterior distribution, which guides the search for the optimum. The second key component of Bayesian optimization is the acquisition function. This function determines the next point to assess when searching for the optimum of the objective function. It tries to balance exploration and exploitation, so new areas of the search space are explored to not get trapped in local optima, while still investigating areas of known high value. The user can choose from different acquisition functions, with each one being more or less effective for different problems.

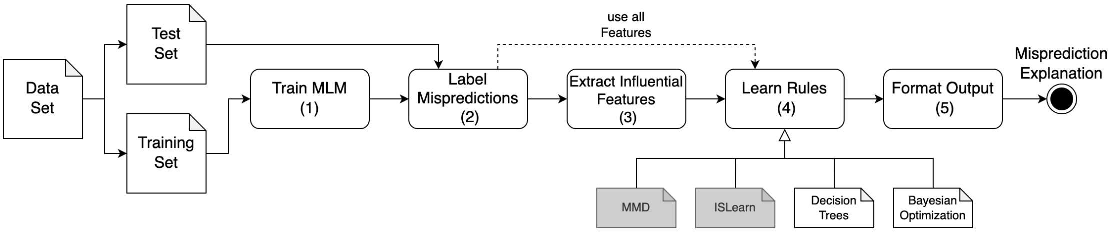


Figure 5: Overview of our approach

4 Generating Misprediction Explanations

In this section, we will introduce our approach of extending the concept of Machine Learning Model Diagnosis (MMD) [Cit+21], to construct better performing misprediction explanations for black box machine learning models and to reduce computational demand of the technique. The basis for our approaches is the following two key issues: On the one hand, utilizing rule induction to construct the rule set might not be the optimal choice, and can be outperformed by a different technique. On the other hand, generating atomic predicates for every input feature and using them all, can be very costly for feature rich input data. That is why we propose a procedure to first identify the features that have a high impact on causing mispredictions.

Figure 5 gives an overview about the sequence and individual steps of our proposed approach. MMD and ISLearn, both part of Step 4 are colored grey to highlight that we utilized the original implementations of the corresponding research papers and only did small adjustments, like constructing the required grammars and adding relevant patterns for ISLearn.

Overview Similar to the approach of MMD, our technique requires a machine learning model and an input data set with known ground truth for said model. For that we use 11 real-world binary classification data sets and split them into training and test sets, with 70 percent of the instances sorted into the training set, while the remaining 30 percent establish the test set. Then we train a machine learning model on the training set (**Step 1**). This is the machine learning model we later generate our misprediction explanation for. In the next step we make use of the trained model to identify the instances of the test set, which are mispredicted (**Step 2**). The following step of our process sequence is optional: We extract the input features that are most influential in causing a misprediction, by considering the feature importance of a newly trained random forest classifier (**Step 3**). After utilizing the information about the most impactful features to reduce the input data considerably, or skipping that step and just using all features, one of the four approaches to generate the rule set can be chosen: MMD, ISLearn, Decision Trees or Bayesian Optimization (**Step 4**). Lastly, the output given by the different rule learning approaches is formatted to always present a uniform output (**Step 5**).

4.1 Training Machine Learning Model

Since we require black box machine learning models for analyzing and building misprediction explanations, the first step in our process is to train the needed models. For the basis of the models, we chose 11 real-world data sets, which are presented in more detail in Section 5. For the scope of this work, we decided to limit the chosen data sets to binary classification problems, but the procedure can be adapted to also work for multi-class classification or regression problems. There are various machine learners to choose from for classification task, we decided to train random forests for two reasons. They are very fast to train and to predict labels for data instances. In addition, since random forests are a tree-based and not distance based models, less data pre-processing, like feature scaling, is required. Some pre-processing is obviously still required to handle problems like multi-class features or incomplete instances, often found in real-world data sets. As we already noted before, our approach is fully model agnostic, which means any other machine learner for binary classifications could replace the random forest classifier. We trained our models on 70 percent of the data set and kept 30 percent as the test set. The test set is essential later, because we still need data with a known ground truth to label mispredictions of the trained model. To prevent an evaluation of our approach based on a "luckily" chosen training set, we trained five models on different training sets derived from the data set. Furthermore, the data is split in a stratified fashion to always assure that both sets contain the right proportions of target labels. The numerous trained random forests and corresponding test sets are stored for later use, so they never need to be retrained.

4.2 Labeling Mispredictions

After preparing the machine learning black boxes, for which we will construct misprediction explanations, and their corresponding test sets, the process of labeling the mispredictions is pretty straightforward. Considering a labeled data set $D : X \to Y$ and a trained machine learning model $M : X \to Y$, where $x \in X$ is a data instance and $y \in Y, y \in \{0, 1\}$ are the matched labels, and noting that D actually contains the ground truth for its data instances, we define the function $L : X \to \{0, 1\}$ as follows:

$$L(x) = \begin{cases} 1 & \text{if } M(x) \neq D(x) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Summarized in words: The function L(x) is 1 if the prediction given by the machine learning model for a data instance is not the label found in the test set, which is the known ground truth. In this way we build a new data set $D_{mis} : X \to \{0, 1\}$ so that the following equation is valid:

$$D_{mis}(x) = 1 \Leftrightarrow (M(x) \neq D(x)) \tag{2}$$

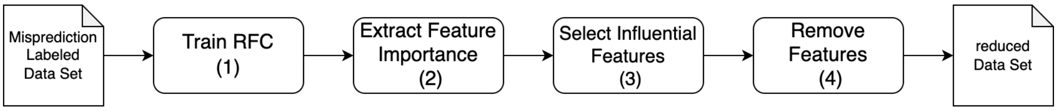

Misprediction Labeled Data Set
Train RFC (1)
Extract Feature Importance (2)
Select Influential Features (3)
Remove Features (4)
reduced Data Set

Figure 6: Overview of how we reduce the data set to the input features with the highest impact on causing mispredictions.

This is done by utilizing the trained random forest to predict the label for every single data instance in the test set and then comparing them to the ground truth. The resulting new misprediction labeled data set D_{mis} contains each input instance in D but now linked to a new boolean label, indicating if the given instance is mispredicted by the model. This information is of course essential for later deducing rules and then assembling our misprediction explanations.

4.3 Extracting Influential Input Features for Faster Computation

Next, after labelling the data instances from the test set, which are mispredicted, follows the process step in which we employ our approach to improve scalability of explanation generation. Computational demand for constructing misprediction explanations with MMD in general does not scale well with growing data sets. Especially the number of input features has a dramatic effect on execution time, because the number of combinations of different atomic predicates in a rule, that need to be checked, grows exponentially. To combat that phenomenon the idea for our approach is to reduce the number of input features that need to be considered when assembling the rule set. So, the main question is: Which input features have the highest impact on causing the machine learning model to mispredict? Figure 6 shows an overview of the process and the steps taken to filter out the important features. The general idea is, since we have access to the data set from the previous step, mapping input features to model mispredictions, we can train a machine learning model that tries to predict, for a given data instance, whether our original black box mispredicts for the same data instance (**Step 1**). Afterwards we can use an explainable machine learning technique, called feature importance, to learn, which input features have the most influence on predicting mispredictions and thus also likely have a high impact on causing the mispredictions in the original black box model (**Step 2**). Next, we can choose a desired number of most influential input features (**Step 3**) and finally we considerably reduce the number of features in the input data set by removing all features not deemed influential enough (**Step 4**).

Feature importance in machine learning refers to a measure of how much an input feature influences the model's prediction. So, this measure can be utilized to quantify the degree of usefulness of each specific feature. A comprehensive overview of various importance techniques is provided by Wei [WLS15]. To predict mispredictions, we decided to train random forests as our models, because they allow for very fast calculation of the needed feature importance values, but in theory the usage of every other machine learning model, that allows determination of importance values is possible. We tried three different approaches to extract feature importance values from our trained misprediction models: Gini importance, permutation importance and SHAP.

Gini importance is a measure that can directly be computed from the random forest structure. It considers the average decrease in impurity caused by each feature over all the trees in the random forest. The impurity of a node in a decision tree describes how well the tree was able to split the training data, so that a node contains mostly instances of one class. This information can easily be accessed, since all needed values were already computed during random forest training, this method is extremely fast. However, a drawback is that Gini importance can be misleading since it has a bias towards high cardinality features.

Permutation importance is an alternative to the impurity-based approach of Gini importance. Permutation importance works by permuting the values of a feature in a test data set and observing the difference in the performance when training a need model. Permutation importance overcomes the limitations of Gini importance, since it does not overvalue high cardinality features, but the technique becomes computationally expensive for large data sets, since it requires refitting the model many times. Additionally, to get the most out of permutation importance a test set for the model is needed, which would reduce the data available to train our misprediction model on.

The third option was SHAP, since Shapley values can be combined to get a global explanation. The intention behind feature importance with SHAP is that large absolute Shapley values indicate important features. To get a global importance value, the absolute Shapley values per feature can be averaged across the whole data. Utilizing SHAP in this way can become computationally expensive.

We realized pretty fast, that using SHAP is not an option for us, because it took more time finding the influential features and constructing the misprediction explanation on a smaller data set, then just using the full data set with all features. When we compared Gini importance to permutation importance the suggested most influential features were certainly different, but when we evaluated some misprediction explanations constructed with these features, the results of permutation importance were in many cases drastically worse, no matter the amount of used data permutations. Thus, at least for our data sets, permutation importance was considerably slower, reduced training data for our misprediction model (works on test set) and provided worse results. These are the reasons why we decided to default to the somewhat error prone, but very performant method of Gini importance. In our implementation of our approach, we make the number of returned most influential input features definable by an input parameter, which allows us to evaluate the effect of the number of features on misprediction explanation performance, computational demand, and rule set length.

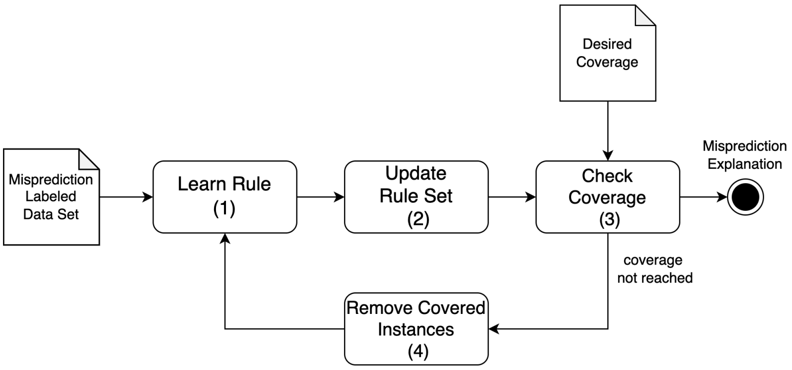
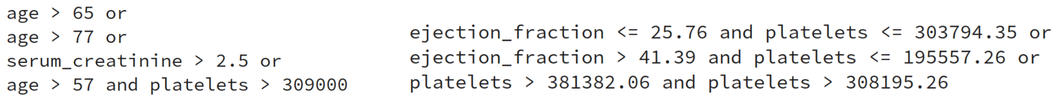


Figure 7: Overview of the iterative process of constructing a misprediction explanation.

4.4 Learning Rule Sets

In this process step we present the rule set construction techniques, which we evaluated. Hence in the following we describe the general idea and the different steps of constructing a misprediction explanation for a black box machine learning model. Then we will take a closer look at the individual approaches and their subtleties in our implementation. The explanation is assembled based on the information contained in the previously misprediction labeled test set. Since the techniques require the ground truth for every data instance, additional inputs cannot simply be generated and we need to work with the fixed amount of information contained in the data sets chosen for our evaluation. Similar to the procedure described by Cito et al. for MMD [Cit+21], our rule set construction approaches, utilizing decision trees and Bayesian optimization, also operate iteratively. For ISLearn we had to give up on allowing disjunctions and could only create single rules, because ISLearn's method to derive the rule sets becomes extremely computationally expensive for data sets with many features. Figure 7 shows the iterative sequence of process steps when generating an explanation with MMD, decision trees or Bayesian optimization. In (**Step 1**) a rule consisting of a series of predicates connected by conjunctions is constructed to cover as many instances in the labeled data set as possible, while minimizing false positives. This is the step that works differently for every of the four approaches we evaluate and will be discussed in detail later. After a rule has been built, it is added to the current rule set. From the second rule onwards, the new rules are connected to the rule set by disjunctions (**Step 2**). Next, we calculate the recall of the current rule set, meaning, which percentage of instances in the labeled data set are covered after adding the new rule and then we compare the calculated recall with a defined desired coverage value (**Step 3**). If the desired coverage of mispredictions is reached by the current rule set, the construction process is finished and we can begin with formatting the final output. For handling, output parsing of MMD and ISLearn, and calculating metrics for misprediction explanations, we implemented a specialized RuleSet class. If the desired coverage is not reached, we have to add another rule to the rule set, so a loop back to Step



```
age > 65 or
age > 77 or
serum_creatinine > 2.5 or
age > 57 and platelets > 309000
```

(a) Redundant rules

```
ejection_fraction <= 25.76 and platelets <= 303794.35 or
ejection_fraction > 41.39 and platelets <= 195557.26 or
platelets > 381382.06 and platelets > 308195.26
```

(b) Redundant predicates

Figure 8: Example for a rule set with redundant rules or predicates.

1 is needed. However, after the first rule subsequent rules are not constructed by considering the whole data set again, but only the subset of instances not already covered by the current rule set is used. In (**Step 4**) these already covered data instances are removed from the data set to create a new one, wholly consisting of uncovered data. In this fashion additional rules are iteratively generated to specifically cover not already covered mispredictions. For this thesis we restricted each rule to be composed of a maximum of two predicates. The approaches and the implementation are however easily adaptable to allow for more predicates. We also have refrained from minimizing rule sets. This makes it possible, that rule sets may contain rules or predicates in a rule, that are redundant. Figure 8 shows an example for both: In 8a the second rule is redundant because of the first rule. In 8b the first predicate of the last rule is redundant. Since MMD also provides the option to only produce and present the performance for a single best rule, our outputs always contain both results: for a single rule with only conjunctions, and for a rule set reaching the desired coverage of mispredictions.

Misprediction Explanations with MMD To be able to assess the performance of our proposed rule generation approaches, and to analyze how MMD fares when it is employed together with our method for identifying the most influential input features, we utilized the implementation of MMD [cit22] provided by Cito et al. We kept all the provided standard parameter settings, especially the weights that are assigned to precision, recall and rule length when MMD evaluates the performance of a rule to find the best one. We only did a small change to the code that allows us to set the parameter "allow_disjunctions" when calling the function, making it easy to switch between single rule and rule set generation mode. For our use case MMD requires following additional inputs: a data frame with the tabular data, a tuple consisting of the name of the target feature and the boolean value that should be explained, a dictionary mapping from relevant attributes in the tabular data to their type (D (discrete), I (integer), C (continuous)), and a value for the desired coverage that is aimed to be reached when using disjunctions. However, since MMD also considers rule set length, the desired coverage is not guaranteed and can be lower if too many rules would be needed to reach the coverage. As a result, MMD provides a number of possible rule sets, ranked by precision first and recall second. It always is a trade-off between precision and recall, more precision means less recall and vice versa. Since all of our proposed approaches guarantee a rule set that fulfills the desired coverage, we first try to filter for results, which also reach the coverage requirement. If no option can provide that, then a rule set

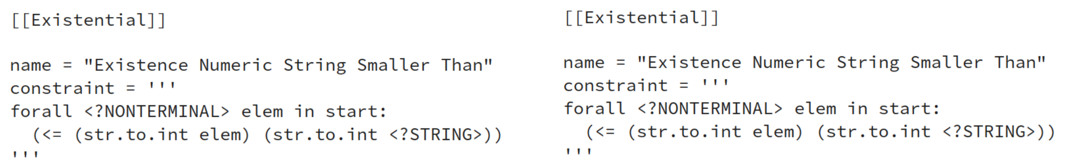
```
HEARTFAILUREWT: Grammar = {
    "<start>": ["<age> <anaemia> <creatinine_phosphokinase> <diabetes> <ejection_fraction> " +
                "<high_blood_pressure> <platelets> <serum_creatinine> <serum_sodium> <sex> <smoking>"],
    "<age>": ["0", "<onenine><maybe_digits>"],
    "<anaemia>": ["0", "<onenine><maybe_digits>"],
    "<creatinine_phosphokinase>": ["0", "<onenine><maybe_digits>"],
    "<diabetes>": ["0", "<onenine><maybe_digits>"],
    "<ejection_fraction>": ["0", "<onenine><maybe_digits>"],
    "<high_blood_pressure>": ["0", "<onenine><maybe_digits>"],
    "<platelets>": ["0", "<onenine><maybe_digits>"],
    "<serum_creatinine>": ["0", "<onenine><maybe_digits>"],
    "<serum_sodium>": ["0", "<onenine><maybe_digits>"],
    "<sex>": ["0", "<onenine><maybe_digits>"],
    "<smoking>": ["0", "<onenine><maybe_digits>"],
    "<onenine>": srange("123456789"),
    "<maybe_digits>": ["", "<digits>"],
    "<digits>": ["<digit>", "<digit><digits>"],
    "<digit>": list(string.digits)
}
```

Figure 9: Grammar for the heart failure data set.

with a lower coverage is selected. Because we try to compare to the “best” rule set provided by MMD and multiple rule sets with trade-off between precision and recall make the choice unclear, we opted to find a good middle ground by calculating the weighted F-score for every option. The F1-score is the harmonic mean of precision and recall, thus symmetrically represents both in one metric. However, instead of weighting precision and recall the same, we put more focus on precision, because a misprediction explanation with a high recall and a low precision is worthless. Finally, after choosing the rule set with the best weighted F-score we use our RuleSet class to parse and format the MMD output and calculate specificity of the misprediction explanation.

Misprediction Explanations with ISLearn The first approach we present utilizes ISLearn [SZ22] to choose the best rules and generate misprediction explanations. We use the implementation of ISLearn version 0.2.13 provided by Dominic Steinhöfel [rin22]. Like with MMD we deliberately keep all the standard parameters and make no code changes. However, in comparison to MMD, ISLearn requires a considerable amount of additional preparation. This stems from the input requirements of ISLearn, because it comes with the ability to independently construct new data instances if needed. That is not relevant for us but still enforces some overhead. First ISLearn requires a grammar for the input language of every data set we use. Exemplary, the grammar constructed for the heart failure data set is shown in Figure 9. As shown by the example, the grammars are very generic and do not consider negative or floating-point values. The reason is, that we preprocess the data sets to remove those. On the one hand, it makes the grammars shorter and on the other hand, it helps with defining the patterns needed for ISLearn. To remove negative and floating-point values we preprocess the data in the following way: We normalize the values for every column, by first finding the min and max value for a feature f and then calculating the normalized value for each data instance with:

$$\text{normalized}_f = \frac{\text{value}_f - \text{min}_f}{\text{max}_f - \text{min}_f} \tag{3}$$

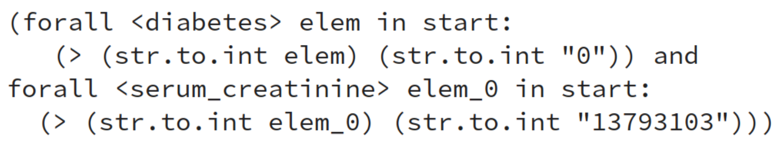


```
[[Existential]]

name = "Existence Numeric String Smaller Than"
constraint = '''
forall <?NONTERMINAL> elem in start:
  (<= (str.to.int elem) (str.to.int <?STRING>))
'''
```

```
[[Existential]]

name = "Existence Numeric String Smaller Than"
constraint = '''
forall <?NONTERMINAL> elem in start:
  (<= (str.to.int elem) (str.to.int <?STRING>))
'''
```

Figure 10: Patterns used for ISLearn.

Now that all values are scaled to between 0 and 1, thus there are no more negative values, we round to the eighth decimal places and then multiply each value by one hundred million to deal with floating point values as well. Later, after ISLearn has provided us results, when we parse and format the output, we reverse the just mentioned steps to display the correct values. After preprocessing the data set, it needs to be split into positive and negative input samples based on the label. Since ISLearn expects the inputs to be in ISla DerivationTree string form, we need to parse each data instance of the positive and negative samples with the aid of the defined input grammar. The last step we have to take before running ISLearn is to define all the irrelevant non-terminals in the grammar that we do not want to be considered when constructing instances from patterns. For our example grammar in Figure 9 that would be the non-terminals “onenine”, “maybe_digit”, “digits” and “digit”. We find these by collecting all non-terminals reachable from the start symbol and then removing all non-terminals produced directly by the start symbol. Additionally, when working with our approach to only use the most influential features, the grammar needs to be re-written to remove irrelevant features from the start symbol and as non-terminals. ISLearn is then executed with following inputs: grammar, positive and negative examples, desired recall value, maximum number for conjunctions and disjunctions, the irrelevant non-terminals and a pattern file. As we already mentioned we kept the number of conjunctions at a maximum of one per rule and we had to disallow disjunctions for ISLearn because of high computational cost, scaling with both number of features and data instances. So, even when reducing the number of features with our approach it was not feasible to allow disjunctions. The two patterns defined by us and used by ISLearn for instantiation are shown in Figure 10. The patterns correspond to the allowed structure of predicates in rules: “feature $\leq$ value” or “feature $>$ value”. ISLearn’s results are presented as instantiated patterns in conjunctive normal form ranked by specificity first and recall second. Figure 11 depicts an example output. In this case the rule “diabetes $>$ 0 and serum_creatinine $>$ 13793103” was identified as the best performing one. The second integer is so high because it is before the



```
(forall <diabetes> elem in start:
   (> (str.to.int elem) (str.to.int "0")) and
forall <serum_creatinine> elem_0 in start:
  (> (str.to.int elem_0) (str.to.int "13793103")))
```

Figure 11: ISLearn output example.

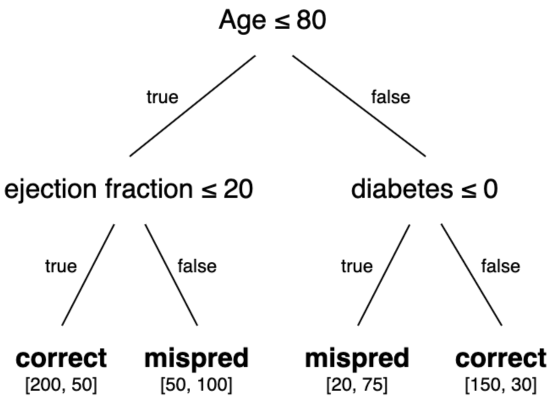


Figure 12: Decision tree example with depth = 2.

data preprocessing is reversed. To make the output easier to comprehend and compare to the other approaches it is parsed and transitioned into our unified rule set format. In contrast to MMD we immediately choose the highest ranked rule as the “best” rule, because all of ISLearn’s proposed rules satisfy the desired recall and under that condition are objectively ranked best to worst based on specificity. As a last step before presenting the misprediction explanation we calculate the precision of the rule set, since it is not provided by ISLearn.

Misprediction Explanations with Decision Trees The third approach we evaluate for rule construction utilizes training decision trees to determine the best possible rule to cover mispredictions. As a first step, before we start training decision trees and extracting rules, we examine the data set to establish the maximum number of decimal places present in a single value. This is necessary to be able to round the later extracted threshold values to a reasonable length. Like with our other approaches the final misprediction rule set is constructed iteratively rule by rule, until the desired coverage of mispredictions is reached. One iterative cycle works as follows: First, we use the currently still uncovered data instances (in the first cycle this amounts to the whole data set) to train a decision tree with the information, if a data instance is mispredicted as the target. We always utilize defined random seeds to keep the results reproducible. To train the trees we employ the python library scikit-learn and keep the default input parameters, but set the maximum allowed tree depth to two. This limits the rules, we extract from the trees, to two predicates, like we aimed to do for all of our approaches. In the next step four possible rules are derived from the trained decision tree. Considering the example tree depicted in Figure 12 we now explain the rule extraction process. We begin assembling the rule strings by traversing the tree structure starting at the root node. The feature name and threshold are extracted and depending on whether the left or right child node are considered next $Age \leq 80$ or $Age > 80$ are concatenated to the rule, followed by an *and* if the child node is not a leaf node. The same procedure is then repeated for every child node, in this case only two more times, but our implementation fully works for deeper trees, to construct rules with more predicates. When a leaf node is reached the corresponding rule string is complete and we save the information about the number of training samples that reach this leaf. For our example tree the four extracted rules would be:

1. Age $\leq$ 80 and ejection_fraction $\leq$ 20
2. Age $\leq$ 80 and ejection_fraction $>$ 20
3. Age $>$ 80 and diabetes $\leq$ 0
4. Age $>$ 80 and diabetes $>$ 0

Since our training target for the decision tree was the misprediction of a data instance, the mentioned saved sample information of the leaf nodes tells us how well a rule performs. Rule 1 for example covers 250 data instances of the data set. Fifty of those are instances that are mispredicted, but we also cover 200 instances for which the model decides correctly, therefore this would not be a good rule to add to the rule set. Hence, the next step is choosing the best performing rule from these options. This is done by calculating precision, recall and then weighted F-score for every rule. We opted to select the rules by weighted F-score to find a good middle ground between long, very precise, probably overfitted rule sets and short rule sets with high coverage but low precision. In this example, even if Rule 2 covers more misprediction Rule 3 with a higher weighted F-score is chosen because of the higher precision. The weight used for the F-score can be used as a hyperparameter to change the balance between precision and recall, thus allows to tailor the misprediction explanation based on requirements for those. After identifying the best performing rule, the chosen one is added to the rule set. Then we calculate the coverage of mispredictions of the current rule set based on the full input data set. If the desired coverage is not reached, the data instances that are already covered are removed from the data set and a new iterative cycle starts by training a decision tree on the reduced data set to find the next best performing rule for not already covered data instances. Otherwise, if the desired coverage is reached, the precision and specificity of the final misprediction explanation are calculated and the results are presented. It is noteworthy, because of the scope of this thesis, we utilize this approach in its simplest form. Many improvements are conceivable to possibly improve performance of the misprediction explanation, like influencing the decision tree composition with training parameters or using a random forest like method to train multiple decision trees on slightly varying data subsets when searching for the best rule.

Misprediction Explanations with Bayesian Optimization The last of the four rule construction approaches we implement utilizes Bayesian optimization, to derive the best misprediction covering rule for given data set. This technique

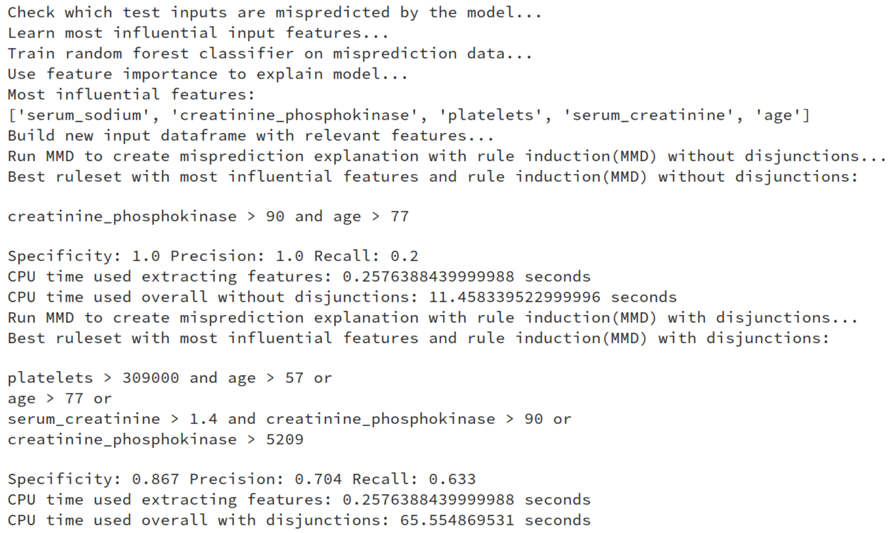
```
option1 = inputs[(inputs.iloc[:, feature1_idx] > x) & (inputs.iloc[:, feature2_idx] > y)]
option2 = inputs[(inputs.iloc[:, feature1_idx] > x) & (inputs.iloc[:, feature2_idx] <= y)]
option3 = inputs[(inputs.iloc[:, feature1_idx] <= x) & (inputs.iloc[:, feature2_idx] > y)]
option4 = inputs[(inputs.iloc[:, feature1_idx] <= x) & (inputs.iloc[:, feature2_idx] <= y)]
option5 = inputs[(inputs.iloc[:, feature1_idx] <= x)]
option6 = inputs[(inputs.iloc[:, feature1_idx] > x)]
option7 = inputs[(inputs.iloc[:, feature2_idx] <= y)]
option8 = inputs[(inputs.iloc[:, feature2_idx] > y)]
```

Figure 13: The eight possible rules checked per optimization cycle.

also assembles the rule set in iterative steps. The objective function expects four parameters for which we do not know the value combination that produces the best result: The two possible features in a rule, represented by an integer corresponding to the column number, and two reasonable threshold values to compare to. Since bounds need to be set for the range of threshold values that are investigated by the Bayesian optimization and the reasonable values for each feature vary greatly, we opted to first normalize the data set, resulting in a consistent value range from 0 to 1. We employed the Bayesian optimization implementation gp_minimize from the scikit-optimize python library and set it to test 25 initial points and to perform 40 optimization steps. We used a defined random state to keep the results reproducible. In our objective function all the covered data instances for each of the eight possible rules are determined (shown in Figure 13). Then for every rule option the precision, recall and finally weighted F-score is calculated. The highest weighted F-score is chosen and returned as a negative, since gp_minimize tries to minimize the result. After 40 cycles Bayesian optimization we receive the index and threshold values that produced the highest weighted F-score in a rule. First, we reverse the normalization of the threshold values and afterwards due to the fact that the only the features and threshold values are known, but not the exact rule structure, we need to check each rule option once more to identify the correct structure. After adding the rule to the current rule set the coverage of the misprediction explanation is calculated. If the desired coverage is not reached all already covered instances are removed from the data set, and the next rule is derived by beginning a new iterative cycle with the reduced data set. Otherwise, if the desired coverage is reached, the precision and specificity of the final misprediction explanation is calculated and the results are presented.

4.5 Formatting Outputs

The last process step is formatting the outputs of MMD and our approaches. This produces a final output that is structured identically no matter of the rule construction method employed and thus allows for a better comparability. Figure 14 shows the structure of a final output based on an example, produced by utilizing MMD to construct misprediction explanations for a model trained on a heart failure data set. As we already mentioned, the outputs contain to separate rule sets: One without disjunctions, consisting of the best single rule, and one rule set with disjunctions aiming to reach a desired coverage. In this case the desired



```
Check which test inputs are mispredicted by the model...
Learn most influential input features...
Train random forest classifier on misprediction data...
Use feature importance to explain model...
Most influential features:
['serum_sodium', 'creatinine_phosphokinase', 'platelets', 'serum_creatinine', 'age']
Build new input dataframe with relevant features...
Run MMD to create misprediction explanation with rule induction(MMD) without disjunctions...
Best ruleset with most influential features and rule induction(MMD) without disjunctions:

creatinine_phosphokinase > 90 and age > 77

Specificity: 1.0 Precision: 1.0 Recall: 0.2
CPU time used extracting features: 0.2576388439999988 seconds
CPU time used overall without disjunctions: 11.458339522999996 seconds
Run MMD to create misprediction explanation with rule induction(MMD) with disjunctions...
Best ruleset with most influential features and rule induction(MMD) with disjunctions:

platelets > 309000 and age > 57 or
age > 77 or
serum_creatinine > 1.4 and creatinine_phosphokinase > 90 or
creatinine_phosphokinase > 5209

Specificity: 0.867 Precision: 0.704 Recall: 0.633
CPU time used extracting features: 0.2576388439999988 seconds
CPU time used overall with disjunctions: 65.554869531 seconds
```

Figure 14: Structure of final outputs no matter the rule construction approach.

coverage was set to 60 percent of the mispredictions. The explanation in this example also uses our technique to reduce the size of the input data, by discovering the five most influential input features, which are then the only ones considered to build the rule sets. The output provides the following metrics for evaluation: Specificity, precision, recall, CPU time used to create an explanation, and the length of the given rule set. Next to precision and recall we opted to also display and evaluate specificity, because focusing at only precision and recall can be misleading. Assuming a precision of 50 percent and recall of 100 percent the explanation can either be pretty good or completely useless. If the model mispredicts only for a low percentage of data instances, a low precision is not as detrimental, because only a small proportion of correct predictions is wrongly classified to cover all mispredictions. However, if the models misprediction rate is higher, for example 35 percent, the same value for precision and recall would mean, that another 35 percent of the correct predictions are misclassified by the rule set, cutting out more than half of them. Specificity helps with that by measuring the number of false positives.

Summary In this Section, we presented our proposed approach to improve on the ideas of Machine Learning Model Diagnosis [Cit+21] by Cito et al. We showed how we employ different techniques to iteratively construct the rules from which the final misprediction explanation is composed of. We also illustrated how we aim to reduce high computational cost when explaining model mispredictions based on feature rich data sets. This is done by identifying the features, which possess the biggest impact on causing mispredictions. In the following sections, we will evaluate our approach, based on 11 real-world data sets and study the effect on performance metrics, computational cost and explanation length.

5 Experimental Setup

In this section we introduce our experimental approach to evaluate our proposed techniques based on 11 real world data sets. We have implemented the different rule construction approaches and the procedure to reduce the input data set by identifying influential input features. As a baseline to compare our results to, we utilize the initial approach of Cito et al. [Cit+21]. In our introduction (Section 1) we already proposed the following research questions:

RQ1 Do our approaches generate misprediction explanations with better relevance metrics?

RQ2 Do our approaches reduce computational demand?

RQ3 Do our approaches generate short misprediction explanations?

To answer these research questions, we compare the misprediction explanations generated by our proposed approaches against the baseline in six different ways. First, we compare the precision, recall and specificity metrics, to assess the performance of the resulting explanations (**RQ1**). Next, we examine computational demand, which we measured by observing CPU time required to construct misprediction explanations with each approach (**RQ2**). Then we analyzed the length of the generated rule sets by summing up the number of used predicates (**RQ3**). In our evaluation section (Section 6) the answer to each of these research questions is split into two partial answers. The reason is, that we evaluate each rule construction approach when considering all input features (Section 6.1), and another time, together with our approach to utilize only influential input features to improve scalability (Section 6.2).

Technical Setup To construct misprediction explanations the black box models to be examined and corresponding test sets are required, so we have access to input data with known ground truth. We keep the model training process consistent to facilitate similar starting conditions with the different data sets. First, we split the whole data set into training set and test set with a 70/30 split. This provides us with a sufficient test set to construct the misprediction explanation from. We also always utilize a fixed random seed when splitting the data set to enable reproducibility. We used the python library scikit-learn to train random forests as our black box models, because they are fast to train and quick to predict labels for new data instances, but our approaches work seamlessly with any other type of model. When training the models, we kept the default settings of 100 trees and no maximum depth, but again used a fixed random seed. The trained models and corresponding test sets are then saved to a storage device, to allow for repeated and fast access later. Lastly, we use a python script to automatically create and save the grammars for each test set, which are needed to generate explanations with ISLearn.

Data Sets We utilize 11 different real world data sets to train our black box models, where one is a relatively small heart failure data set. The remaining ten, which are far bigger in size, we selected to evaluate our approaches, since they were also used by Gesi et al. [Ges+23] when proposing a technique to reduce computational demand when constructing misprediction explanations, which is different but close to our idea. Contained are five software engineering data sets (merge conflict prediction for four programming languages and one bug report close time prediction) and five non software engineering tasks (Job Change, Bank Marketing, Hotel Booking, Water Quality and Spam Email). Some data sets required extensive preprocessing to be used for training a model. In the following we only give a short summary of the changes made. In case of interest the commented implementation can be inspected for all changes. For the heart failure data set the follow-up time feature is removed, since it is related too closely to the target (only patients that survived still need follow-up visits). The non-software engineering data sets required many changes, including: removing instances with empty features or assigning values if possible, removing features that are unknown for the majority of instances, replacing string features with integers, consolidating multiple features with the same information and handling multi class features. The software engineering data sets did not require any preprocessing.

Measures To assess the performance of the approaches, we utilize the following statistical metrics: precision, recall, specificity and the weighted F-score. In the following we use these abbreviations: TP = True positive, TN = True negative, FP = False positive, FN = False negative. In our case precision is the fraction of data instances that are correctly identified as mispredictions among all instances covered by the explanation. Precision is formally defined as:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

While precision measures if the rule set covers the correct data instances recall quantifies the amount of mispredictions the explanation is able to cover and how many are missed. The formal definition of recall is:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

Together precision and recall can already give a good overview of the performance, but without in-depth knowledge of the underlying composition of the data set, results can be deceiving. With only these two metrics it does not become clear, how big the portion of the data instances is, which are correctly predicted by the model, but wrongly covered by the explanation. Therefore, we opted to also include specificity, which reveals the percentage of data instances that are correctly predicted by the model and uncovered by the rule set, thus making clear how many correct predictions are unnecessarily deemed untrustworthy. Specificity is formally defined as:

$$Specificity = \frac{TN}{TN + FP} \tag{6}$$

Another important measure we utilized is the weighted F-score, we used to choose the best rule, out of multiple options. The default F1-score is the harmonic mean of precision and recall, which means the contribution of precision and recall to the F1-score are equal. Since precision is essential for a useful misprediction explanation we doubled the weight of precision. F_β is the formal symbol for the weighted F-score, where recall is considered β times as important as precision, thus in our case $\beta = 0.5$. Formally, weighted F-score has the following definition:

$$F_\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \tag{7}$$

Experimental Process Because of the way the data set is split into training set and test set can have significant impact on the model performance and thus by extension change the resulting misprediction explanation, we cannot depend on the evaluation of a single possible split. Therefore, we trained five models for each data set, based on five different data splits, to allow for more reliable conclusions. In addition, since CPU time required also varies slightly each run, we also execute each experiment five times. Considering, that we utilize random seeds at every step, these extra executions always provide the exact same misprediction explanations and are solely to measure the variance in execution time.

Provided that the five random forest models, corresponding test sets and grammars are already accessible in saved form, an experiment proceeded as follows: (1) Our implementation is provided with following inputs: black box model to be examined, model type to train to predict mispredictions (in our case always a random forest classifier, but can be extended), if our approach to only use influential features should be used and how many, the technique that should be used to generate the rules, and the desired coverage of the resulting misprediction explanation. (2) Labeling each data instance if it is mispredicted by the black box model. (3) Optional step: Train random forest on misprediction labeled data set and utilize feature importance to identify most influential features, then shrink data set. (4) Generate misprediction explanation iteratively rule by rule with the chosen approach, until desired coverage is reached. (5) Measure performance and CPU time used for single best rule and full rule set.

To keep to scope of the evaluation reasonable we opted to use following combinations of values for desired coverage and number of most influential input features:

- all features + desired coverage of 30/40/50/60/70/80%
- 5 most influential features + desired coverage of 30/40/50/60/70/80%
- 2/3/4/6 most influential features + desired coverage of 60%

Computational Setup The experiments were run exclusively on a compute server with an AMD EPYC 7713P processor (64 cores), 1.5 GHz and 256 GB of system memory.

Source Code and Results The complete source code used for the evaluation, and all results are available through a public repository (`github.com/buchwind/misprediction_explanation`).

6 Evaluation

This Section presents the results of our experiments and we assess the performance for each approach, with and without our proposed technique to only utilize influential input features. This makes the answer to our **RQ1**, **RQ2** and **RQ3** multifaceted, since in addition to comparing one rule construction approach to another, we also need to consider the effect on the misprediction explanations when relying only on selected features. For **RQ1** we examine weighted F-score and specificity of the constructed rule set. The answer for **RQ2** is given by measuring the CPU time needed for each experiment. Finally, for **RQ3** we inspect the explanation length by summing-up the number of predicates the rule sets consist of. To present our results we utilize boxplots, which are a standardized way of capturing the distribution of gathered data and indicate eventual outliers. As we mentioned before, when allowing disjunctions and building an explanation with ISLearn the computational demand was too high, especially for feature rich data sets. For this reason, the ISLearn approach is only shown when evaluating the generation of single rules.

6.1 Experimental Results with all Input Features

In this Section we evaluate how our proposed rule creation approaches perform, compared to the baseline of MMD [Cit+21] in the three categories: **Performance Metrics**, **Computational Demand** and **Explanation Length**. In these experiments the misprediction explanations are constructed by considering all input features, hence our technique to prune the less important features is not used.

6.1.1 Performance Metrics

First, we display the obtained statistical measures for the 11 data sets and five repetitions. Figure 15 through 19 show the weighted F-score and specificity for the different rule construction approaches when generating a single rule or a full misprediction explanation satisfying a desired coverage. We opted to utilize the weighted F-score for our evaluation, since the overall performance of a misprediction explanation is a balance between precision and recall. We weighted the F-score such that precision is considered two times as important as recall. The reason is, that recall is important, but an explanation with high recall and low precision is not desirable, because of many false positives, thus leaving only an unnecessarily low number of true negatives to work with. Still, even when weighted, F-score alone might convey a misleading picture for some cases i.e. low precision and high recall for data sets with many mispredictions. That is why we also evaluate specificity to quantify data instances, that are accurately identified as correctly predicted from the black box model.

Single Rule Figure 15 displays the weighted F-score and specificity for single rules constructed by MMD, Bayesian optimization and decision trees for 11 data

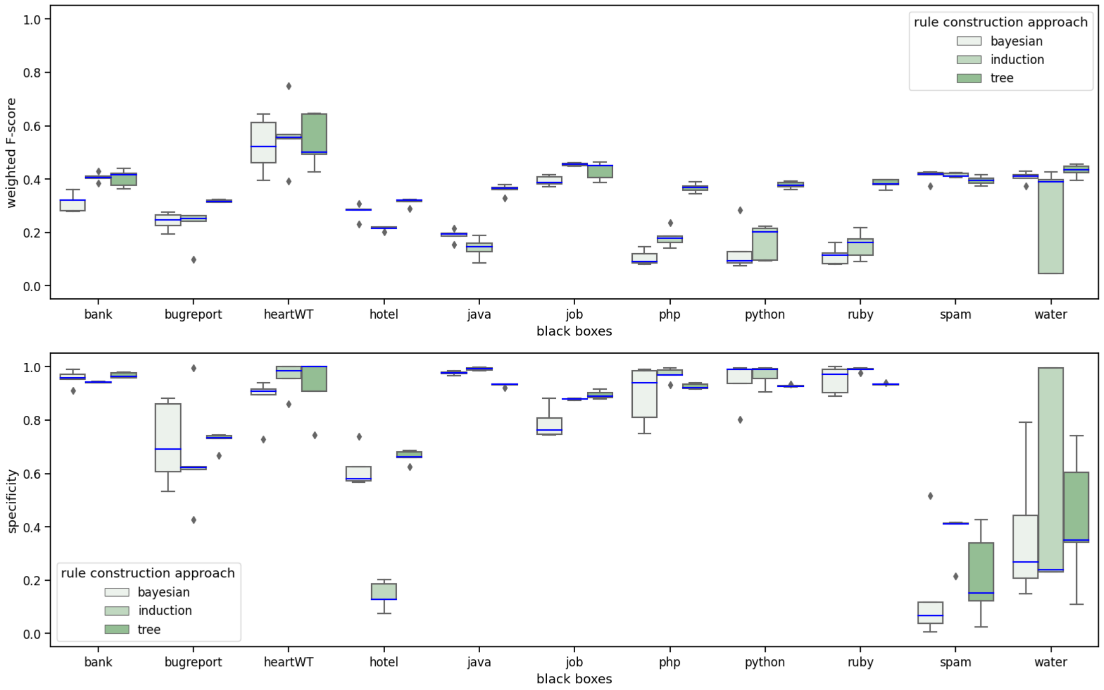


Figure 15: Weighted F-scores and specificity for Bayes, MMD and Trees (single rule).

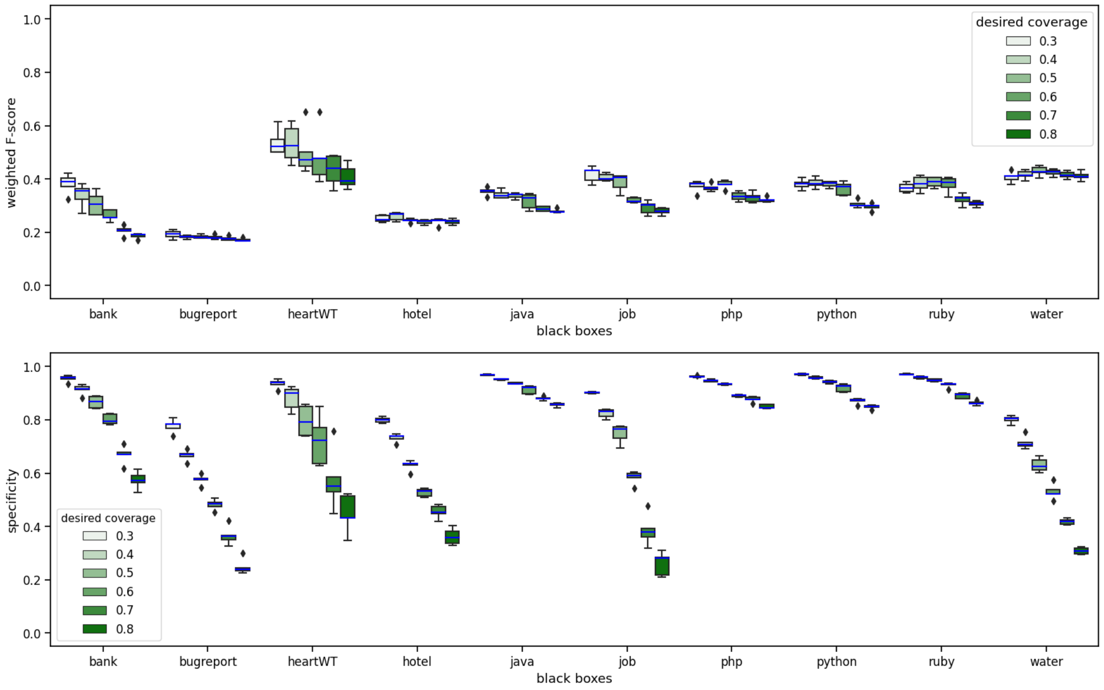


Figure 16: Weighted F-scores and specificity for ISLearn (single rule).

sets and five repetitions. Figure 16 contains the statistical measures for the ISLearn approach. The results for MMD, Bayesian optimization and decision trees can be presented in one Figure because, when constructing only one rule without disjunctions, the given desired coverage has no effect and the outcome is always the same for these approaches. In contrast ISLearn is able to adhere to the desired coverage even when only generating a single rule. Overall, the visual data shows, that MMD had a problem to find a good rule without covering the whole hotel data set. The same is true for MMD, Bayesian optimization and decision trees for the spam and water data sets. The reason is likely an already badly performing black box, causing a high amount of mispredictions for the test sets, thus making it hard to cover many mispredictions without also covering correctly predicted data instances. Bayesian optimization achieves a higher or comparable median F-score for four data sets while keeping a better or similar specificity. For the spam data set the F-score is slightly better but the specificity results show that nearly the whole data set is covered by the rule, making the explanation useless. For the remaining six data sets Bayesian optimization could not reach the baseline performance. The decision tree approach performs better or comparable in nine of the eleven cases. Like with Bayesian optimization, the F-score for the spam data set is close but the specificity is lacking. Overall, the specificity for this approach is better than the baseline in most cases, or only slightly worse but instead a big increase of the F-score can be observed in these cases. For ISlearn the spam data set is missing, because even if only allowing conjunctions, handling around 100 features took too much time. The advantage of ISLearn seems to be, that it can handle the water data set far better than the other approaches, providing rules with reasonable specificity. Since we use F-score weighted towards precision and the F-score in most cases only slightly decreases for a higher desired coverage, this means ISLearn is able to find good rules to increase recall without reducing precision too much. It is not surprising, that the specificity reduces drastically for higher coverage values, since with only one rule and a maximum of two predicates, in most cases it is impossible to only cover mispredictions. In eight cases ISLearn is able to provide rules with F-scores comparable or higher than the baseline. On average, extracting rules from decision trees performed best.

Rule Set Figure 17 displays weighted F-score and specificity for misprediction explanations adhering to a given desired coverage, constructed by MMD for 11 data sets and five repetitions. Figure 18 and Figure 19 depict the same for the Bayesian optimization and decision tree approach respectively. It is important to know, that for MMD, when disjunctions are allowed, a desired coverage can be set, but it is not guaranteed that a rule set option is generated, which reaches this coverage. Since MMD provide multiple rule set options, we first selected the ones which achieved the desired coverage and then chose the explanation with the highest weighted F-score. If no option had a high enough coverage we settled for a lower coverage. The visualized data shows, no matter the approach, that

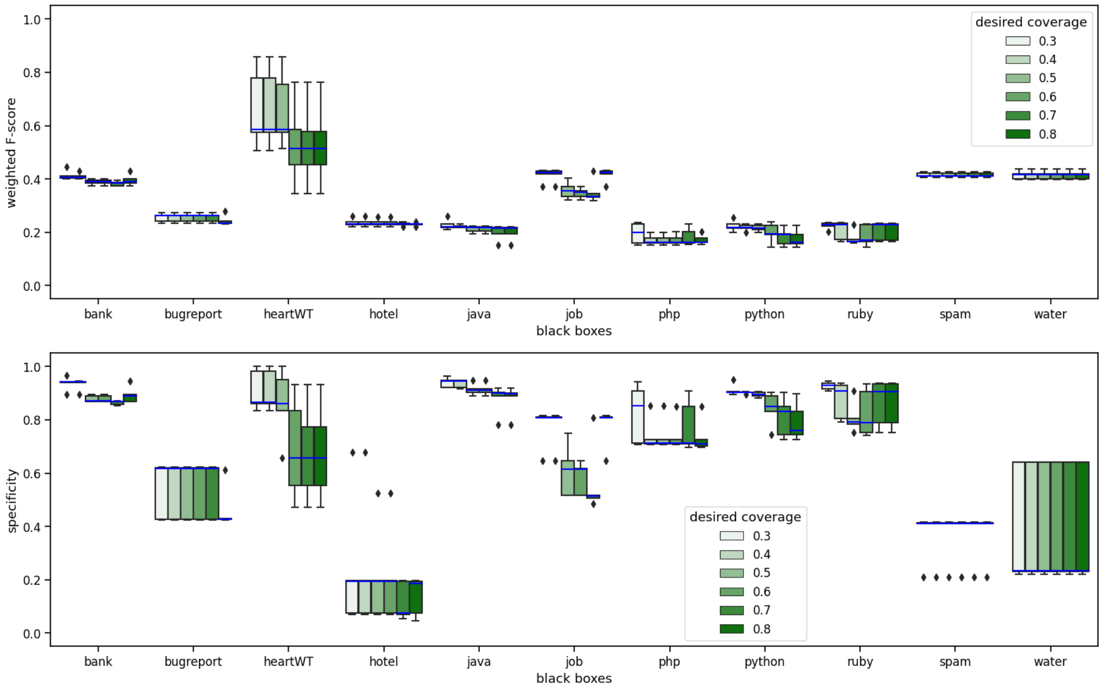


Figure 17: Weighted F-scores and specificity for MMD (rule set).

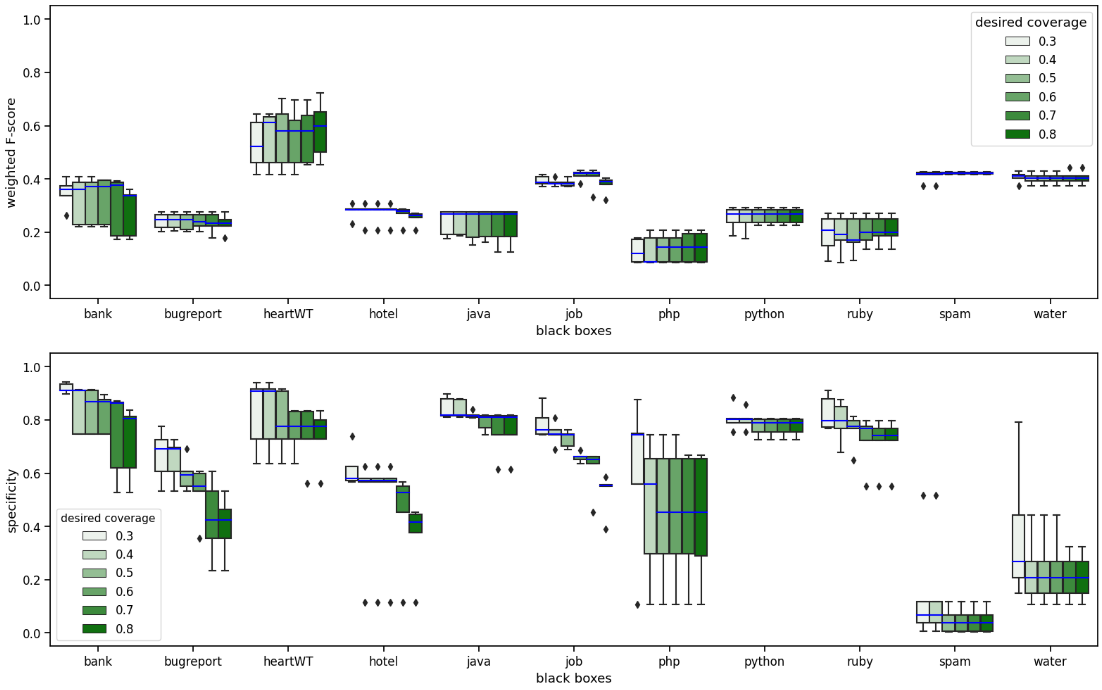


Figure 18: Weighted F-scores and specificity for Bayesian optimization (rule set).

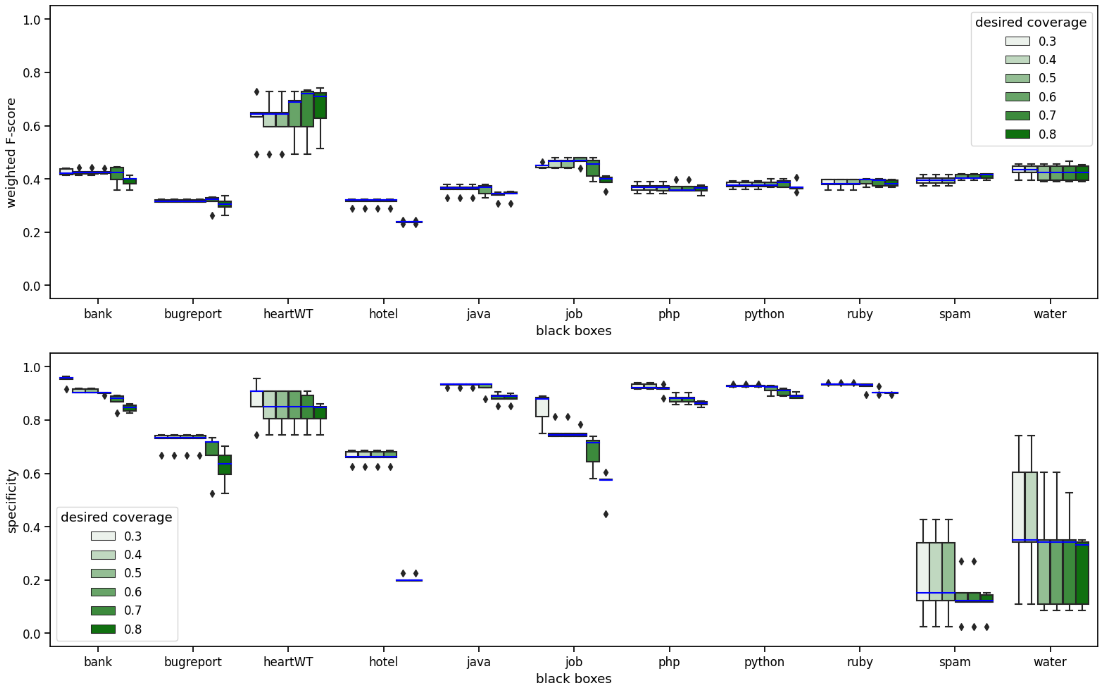


Figure 19: Weighted F-scores and specificity for decision trees (rule set).

for many cases the F-score and specificity stay the same for an increasing desired coverage. This happens, when the rule constructed to clear a lower coverage already covers enough mispredicted data instances to reach the higher coverage values. Comparing the Bayesian optimization approach to the baseline, it performs similar or better in 6 cases. Same as shown above, when constructing and evaluating only a single rule, explanations generated by Bayesian optimization have a very low specificity for the spam data set. Likewise, the same is true for MMD in combination with the hotel data set. Our decision tree approach improves the performance of constructed misprediction explanations in all but one case, in both weighted F-score and specificity. While the F-score for the spam data set is also comparable, the specificity in that case is far below the baseline. In summary decision trees performed by far the best in constructing misprediction explanations when employing the whole data set with all features.

Summary The experimental results show that utilizing our approach to extract rules from decision trees instead of using rule induction like in MMD improves performance of the resulting single rule in nine of eleven cases and resulting misprediction explanation in ten of eleven cases. The efficient and effective technique to find good data splitting features and thresholds by considering Gini impurity, provides better results even in this simple form, with still more potential for improvement. For both, single rule and rule set construction, Bayesian optimization could improve on the baseline only in a few cases, while performing alike or worse

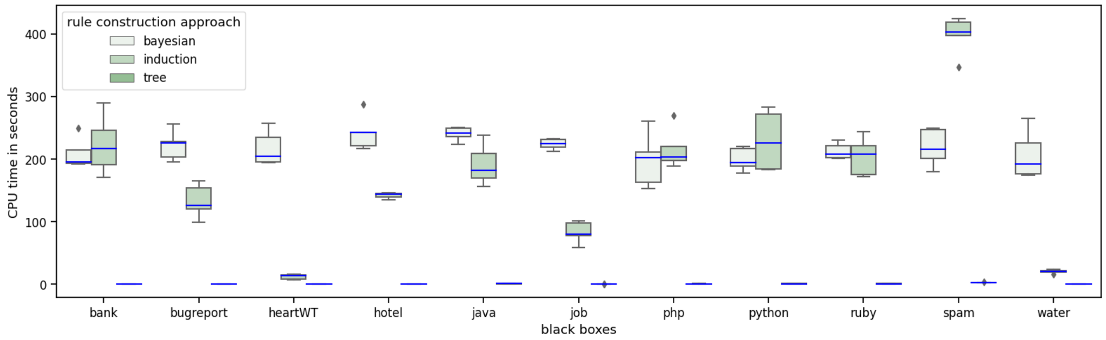


Figure 20: CPU time required by Bayes, MMD and Trees (single rule).

in many. These results could possibly be improved with more optimization cycles, but as we will show later, would result in a high increase in computational demand. Sadly, ISLearn could only be evaluated for single rule generation, but was able to convincingly improve upon the baseline in 6 of 11 cases. One unique strong point of ISLearn is its ability to find the best rule that guarantees the desired coverage, even for single rule generation, and get as close as possible to it. This seems to improve results especially for data sets for which MMD struggles to provide rules that not also cover most of the correctly predicted data instances.

Partial Answer to RQ1 Based on the experimental results, we conclude the following:

RQ1 Based on our evaluation, we conclude that our decision tree approach to construct misprediction explanations improves performance compared to the baseline in 9 of 11 cases for single rule generation and in 10 of 11 cases for full rule sets satisfying a desired coverage. Additionally, ISLearn was able to improve one additional single rule case that decision trees could not advance.

6.1.2 Computational Demand

Figure 20 through 22 show the computational demand, in form of CPU time in seconds, for the different rule construction approaches, when generating a single rule or a full misprediction explanation satisfying a desired coverage. This is done for all 11 data sets and its five repetitions. Since execution time is always slightly different even for the exact same code, we opted to also run each repetition five times and calculate the average CPU time required.

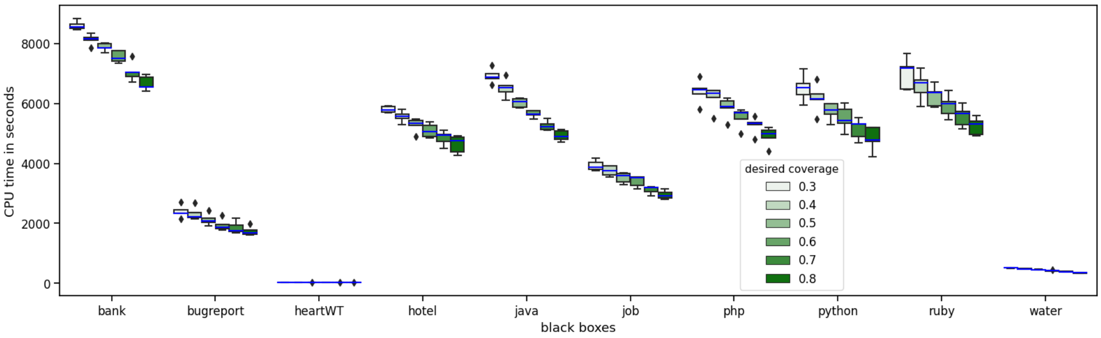


Figure 21: CPU time required by ISLearn (single rule).

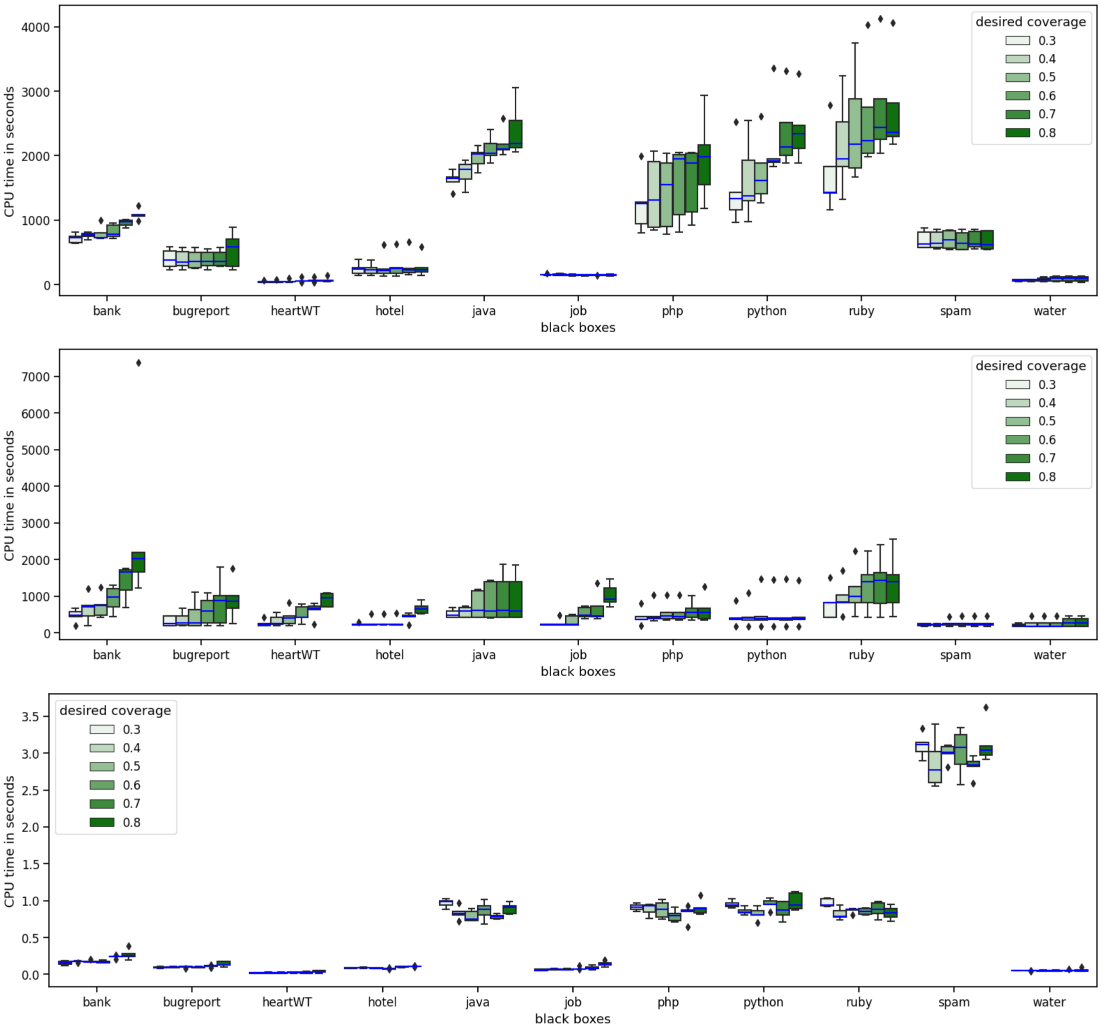


Figure 22: CPU time required by MMD (top), Bayes (middle) and Trees (bottom) (rule set).

Single Rule Figure 20 displays the CPU time required for single rules constructed by MMD, Bayesian optimization and decision trees for the 11 data sets and five repetitions. Figure 21 contains the computational demand for the ISLearn approach. Overall, a clear dependency between data set size (instances as well as features) and CPU time required is shown in the visualized data for MMD, decision trees and ISLearn. Demonstrated by the results for the heartWT and spam data sets, which are by far the smallest and biggest respectively. For Bayesian optimization the computational demand stays relatively stable for all cases, because the workload for this approach is somewhat fixed, based on the number of optimization cycles, initial points and points to sample. Compared to the baseline this helps for the big spam data set, but is a disadvantage for smaller ones. Bayesian optimization could only improve upon the baseline in 3 cases, while being significantly worse for many. The results for ISLearn show the already mentioned problem of high computational cost for feature rich data sets. For the spam data set with around 100 features even single rule generation was not feasible anymore in a reasonable time. It is considerably worse in every case then the baseline and can only beat Bayesian optimization for the small heartWT data set. One interesting detail is, that the execution time is reduced for a higher desired coverage. The reason is probably, that ISLearn tries to find the best rule, while getting as close as possible to the required coverage, thus a lower coverage means more possibilities to reach that coverage. In contrast to ISLearn, decision trees are extremely fast in providing a rule. In all but one case the rule construction is finished in less than a second of CPU time. The exception is the big spam data set for which rule construction took around three seconds. For every case the decision tree approach required significantly less CPU time than the baseline.

Rule Set Figure 22 depicts CPU time required for misprediction explanations adhering to a given desired coverage, constructed by MMD, as well as with our Bayesian optimization and decision tree approaches. The connection, mentioned above, between computational demand and data set size is still true, but since it is now possible that multiple rules are generated iteratively to satisfy a desired coverage, the number of rules required becomes also a significant factor. If the median execution time stays the same for increasing coverage values, the reason for that is, that in the process of trying to cover for example 40 percent of the mispredictions, the next rule added to the rule set already increases coverage to above 80 percent, thus satisfying all other options. For Bayesian optimization the execution time stays pretty predictable: The time required to construct a single rule, multiplied by the number of rules needed to reach the coverage threshold. For decision trees every iterative step of adding another rule to the explanation becomes faster than the last one, because the tree needs to be trained on a smaller data set. Bayesian optimization was considerably quicker than the baseline in 5 cases: the four merge conflict prediction data sets and the spam data set. A big drawback is, that every additional rule for the rule set adds substantial computational demand.

For the bank data set Bayesian optimization is faster than the baseline for a low desired coverage, but when more rules need to be added the approach falls behind. The outlier for the bank data set and coverage of 80 percent is from a repetition we forcible stopped after two hours because the rule set generation was stuck in an endless loop. For 880 executions this happened 3 times. The reason was not obvious but likely a missed edge case when implementing the approach. Our decision tree approach eclipsed the results of the baseline exactly like for single rule generation. In ten of eleven cases the misprediction explanation was generated in less or around one second of CPU time.

Summary The experimental results show that utilizing our approach to extract rules from decision trees drastically reduces computational demand of both, single rule and full misprediction explanation generation for every case. Single rule and rule set construction taken together, Bayesian optimization again could improve on the baseline only in a few cases, while performing alike or worse in many. The results would get even worse when trying to improve explanation performance by increasing optimization cycles, initial points or points to sample. ISLearn could not compete and the results show it is very computationally expensive.

Partial Answer to RQ2 Based on the experimental results, we conclude the following:

> **RQ2** Based on our evaluation, we conclude that our decision tree approach to construct misprediction explanations drastically reduces computational demand compared to the baseline for single rule generation, as well as for full rule sets satisfying a desired coverage, in every case.

6.1.3 Explanation Length

Figure 23 shows the length of generated misprediction explanations, in number of predicates contained within, for the different rule construction approaches. This is done for all 11 data sets and their five repetitions. The visual data shows, that in many cases the median rule set length does not change much for increasing coverage thresholds. The reason is the same, we already mentioned above: The explanation constructed for a lower coverage might already clear higher thresholds. The occasionally big variance in rule set length for different repetitions shows the big impact a certain split into training set and test set can have on the results. In the most cases MMD provided the shortest misprediction explanations or at least similar length, while the rule sets constructed with Bayesian optimization usually contained the most predicates.

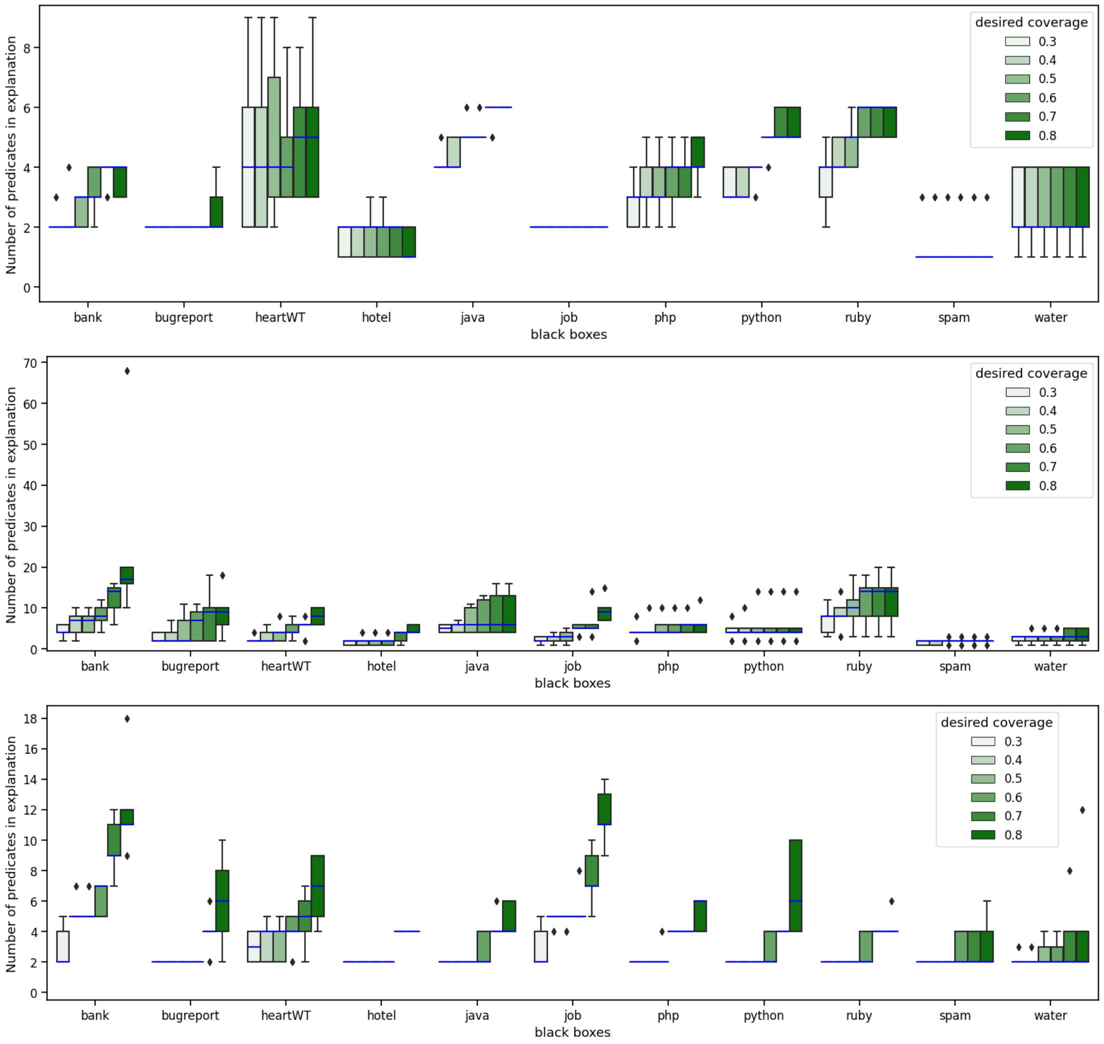


Figure 23: Number of predicates in explanation created by MMD (top), Bayes (middle) and Trees (bottom).

Partial Answer to RQ3 Based on the experimental results, we conclude the following:

RQ3 Based on our evaluation, we conclude that our decision tree and Bayesian optimization approaches did not produce misprediction explanations with reduced rule set length compared to the baseline.

6.2 Experimental Results with most influential Input Features

In this Section we evaluate how our proposed rule creation approaches perform, compared to the baseline of MMD [Cit+21], when only utilizing a set number of most influential input features when constructing the misprediction explanation. We also assess the effects on MMD and use the same three evaluation categories as in Section 6.1.

6.2.1 Performance Metrics

Figure 24 through 31 show the weighted F-score and specificity for the different rule construction approaches, when generating a single rule or a full misprediction explanation with either a changing coverage and five influential input features, or with a desired coverage of 60 percent and a variable number of features. Since displaying the results for single rules and a changing amount of influential input features requires eight graphs, these can be found in the Appendix as Figure 39 through 42 on page 74 and 75.

Single Rule First, we present how utilizing only a number of influential input features impacts the performance when generating single rules with the different approaches. Figure 24 displays the weighted F-score and specificity for single rules constructed by MMD, Bayesian optimization and decision trees when using the five most influential input features of every data set. Figure 25 contains the statistical measures for the ISLearn approach. The visualized data shows, that only with a few exceptions, the mean performance of the generated misprediction explanations is very similar to the results when utilizing all input features (page 40). This is true for both weighted F-score and specificity. Additionally, the variance in performance for different repetitions is reduced in many cases. Using only a small number of important features also allowed us to use ISLearn together with the spam data set, which was not possible with all features. When observing the performance change when relying on two, three, four or six influential features (Figure 39 through 42) it is apparent, that for single rules there are only small differences in many cases. Bayesian optimization is somewhat of an exception, since in some cases a significant drop of the F-score can be noticed for a higher number of features.

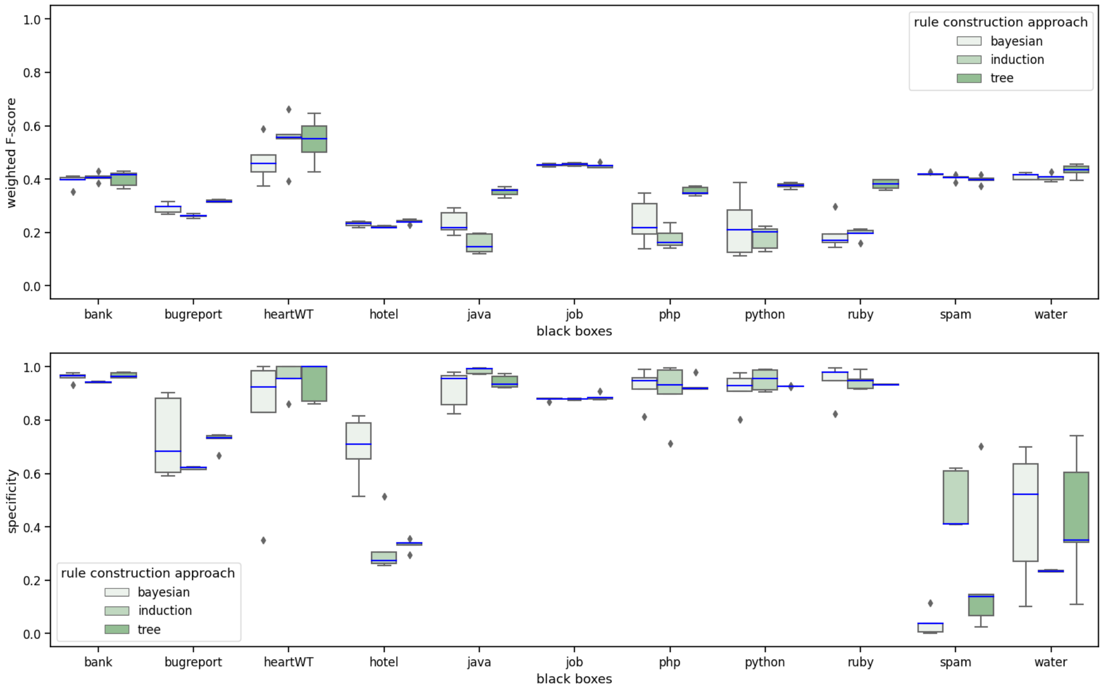


Figure 24: Weighted F-scores and specificity for MMD, Trees and Bayes (single rule, five features).

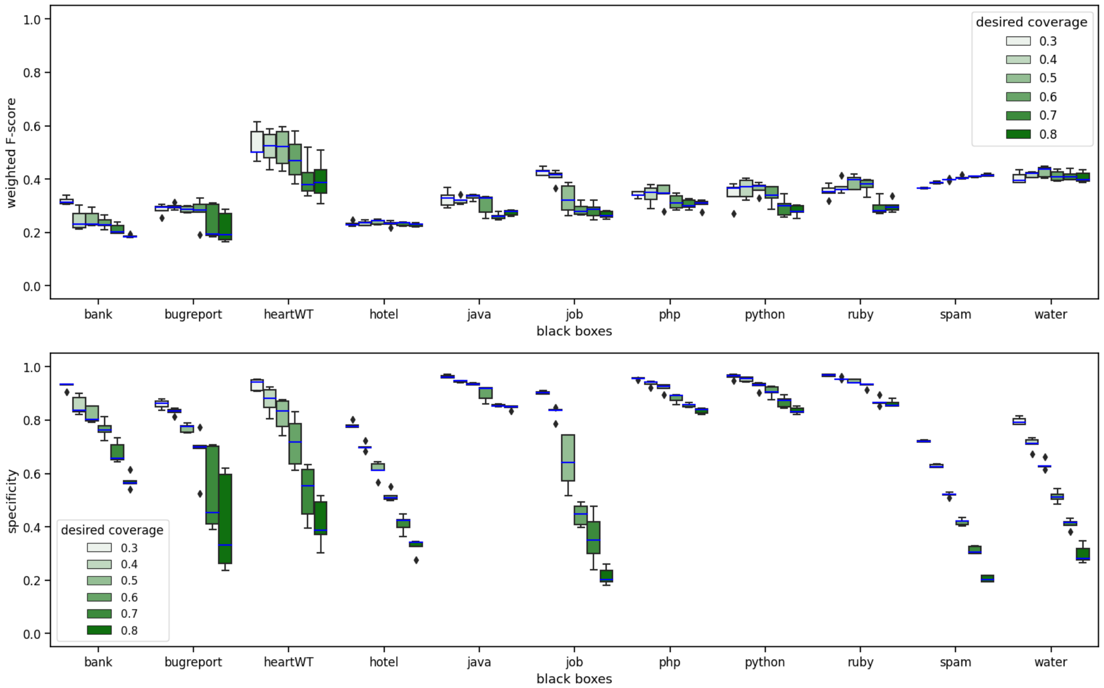


Figure 25: Weighted F-scores and specificity for ISLearn (single rule, five features).

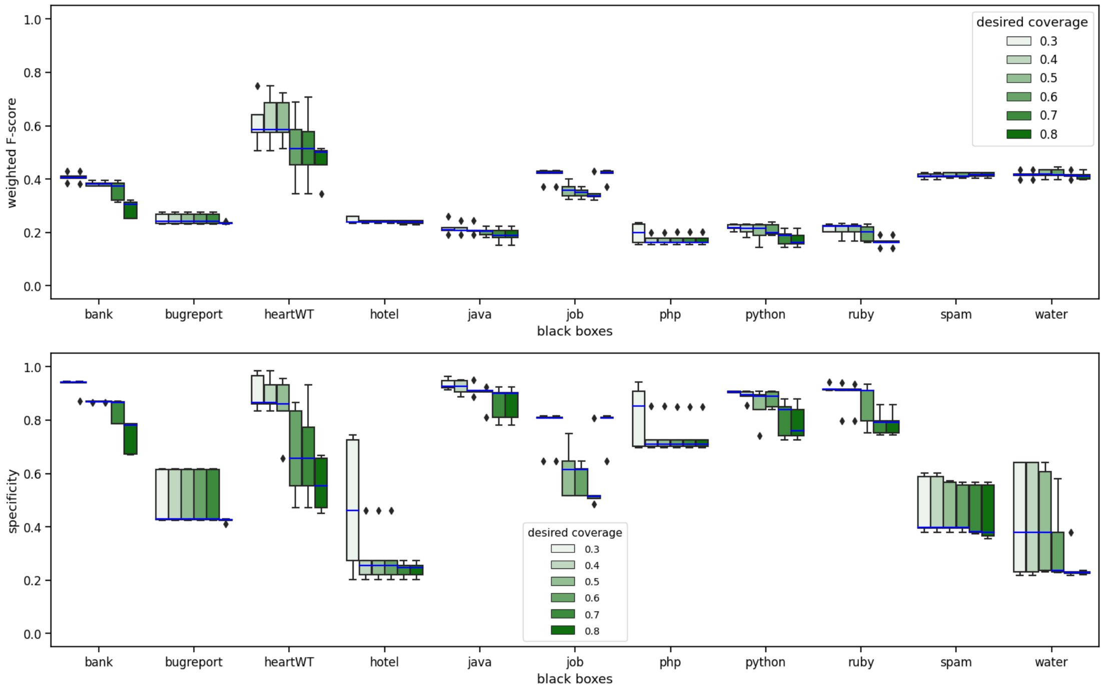


Figure 26: Weighted F-scores and specificity for MMD (rule set, five features).

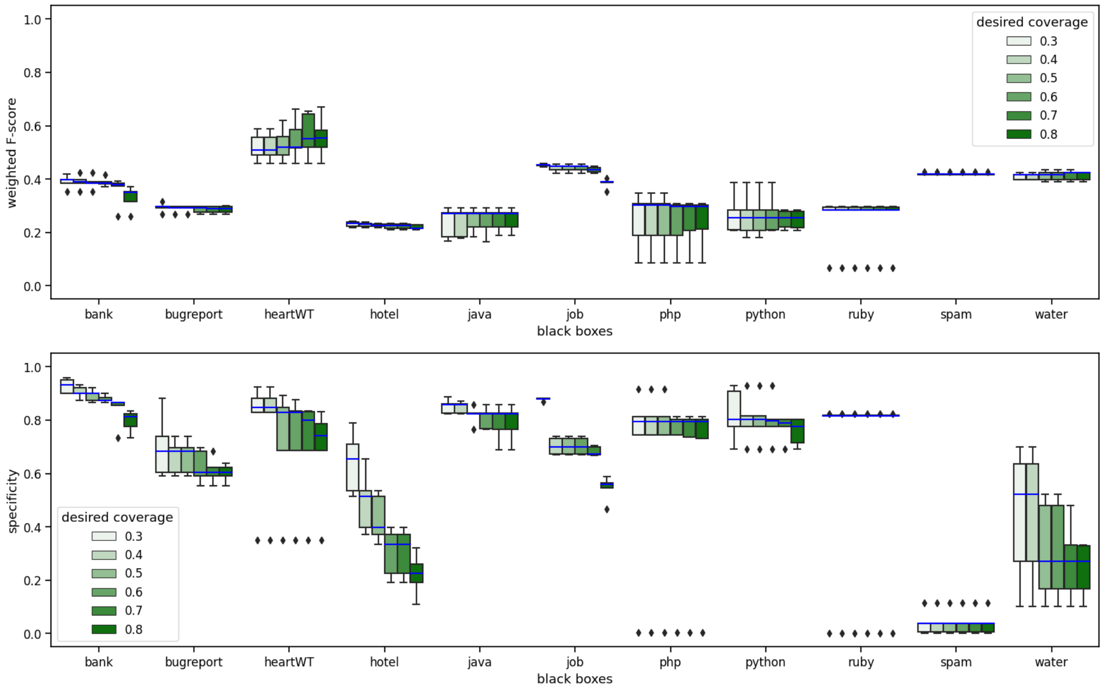


Figure 27: Weighted F-scores and specificity for Bayesian optimization (rule set, five features).

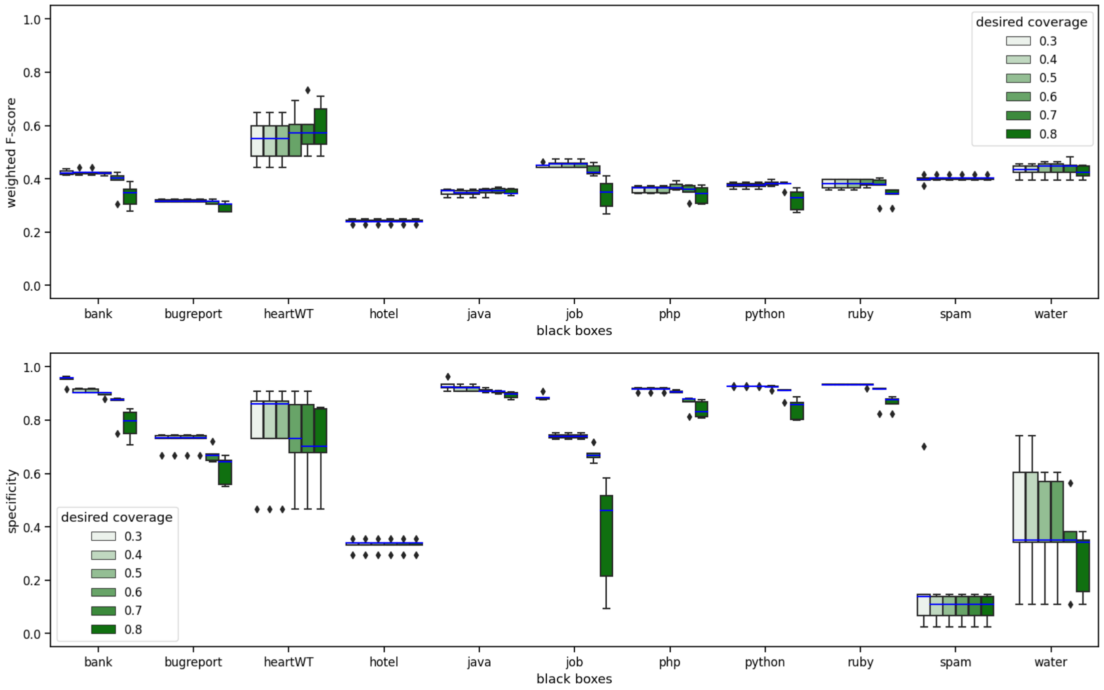


Figure 28: Weighted F-scores and specificity for decision trees (rule set, five features).

Variable Coverage Next, we show the performance of full misprediction explanations adhering to a given desired coverage, when utilizing only the five most influential input features for rule construction. Figure 26 through 28 depict the weighted F-score and specificity of MMD, as well as for Bayesian optimization and decision trees. The visual data shows, just like with single rule construction, the performance of the rule sets, build with only five important features, is in most cases as good or only slightly worse than using all input features. A slight loss in performance is not surprising, since most of the pruned features also have a non-zero importance in causing mispredictions, thus at least some information is lost. We hoped the performance of Bayesian optimization would increase when presented with less features, since this would reduce wasting limited optimization cycles on low impact features. In some cases this worked and increased performance significantly, but not as universally as we hoped. In many cases, the weighted F-score remained nearly the same. Overall, the decision tree approach provided the best results in most (8 of 11) cases, with Bayes being ahead in two and one case tied between MMD and Trees.

Variable Feature Count Finally, for performance we inspect full misprediction explanations with a desired coverage of 60 percent, when using a changing amount of influential input features for rule construction. Figure 29 through 31) show the weighted F-score and specificity of MMD, as well as for Bayesian optimization

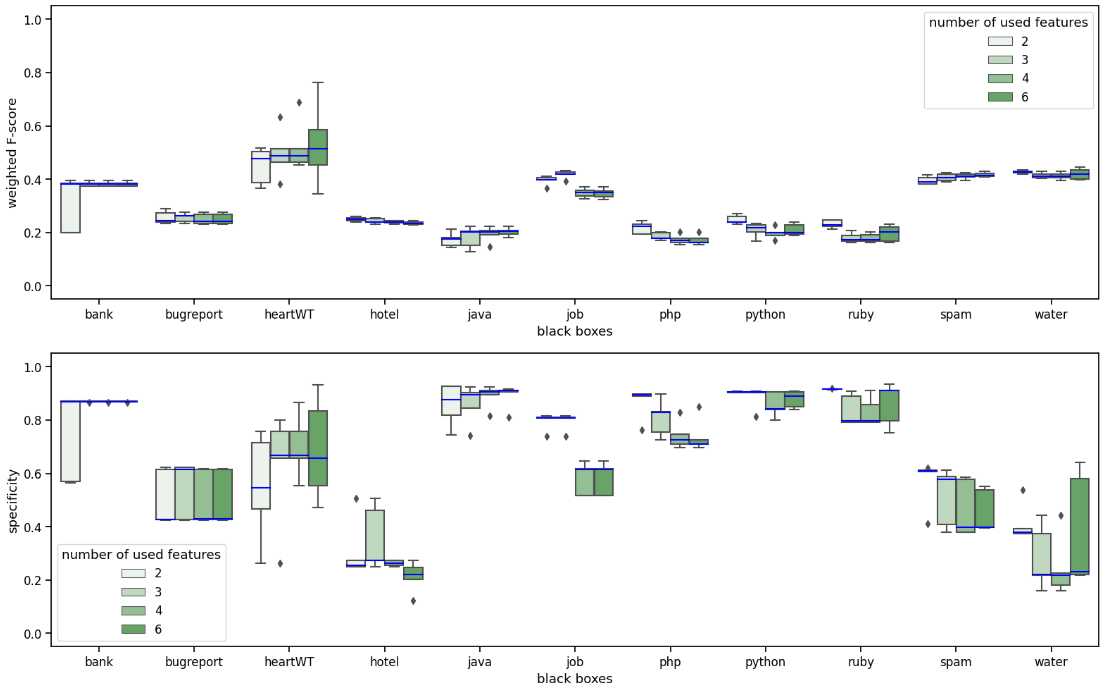


Figure 29: Weighted F-scores and specificity for MMD (rule set, coverage: 0.6).

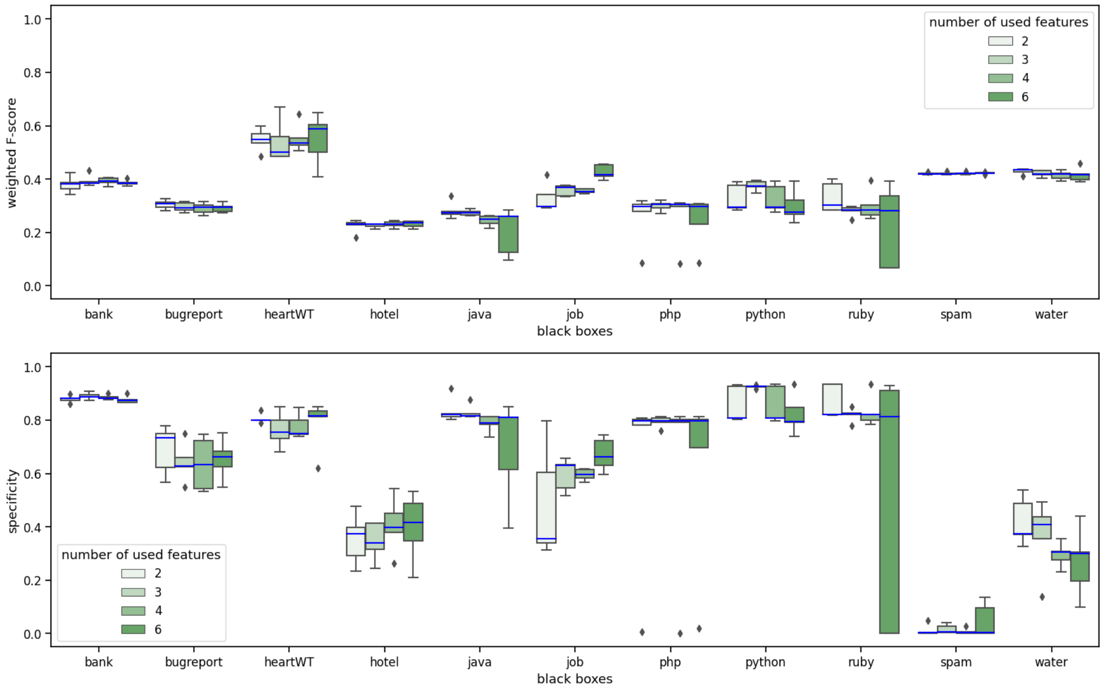


Figure 30: Weighted F-scores and specificity for Bayesian optimization (rule set, coverage: 0.6).

and decision trees. The results show, that even when generating multiple rules to build a rule set reaching the coverage threshold, the number of available influential features to work with does not seem to have a clear impact, or in most cases nearly no impact at all. In many cases utilizing only the two most influential input features is enough to reach similar or only slightly worse results, while for Bayesian optimization the performance even increases for some cases.

Summary The experimental results show that utilizing our approach to identify input features with high impact on causing model mispredictions, and using only these influential features to construct single rules or full misprediction explanations leads to similar or only slightly reduced performance in most cases, compared to considering all input features. The slight reduction in performance results from the loss of information, when removing less influential input features, since they still have a non-zero importance in causing mispredictions. The benefit is a reduced performance variance between different repetitions in many cases. Bayesian optimization is an exception, because for this approach, a significant improvement of performance can be observed for some cases. The reason is, that Bayesian optimization has a fixed amount of optimization cycles, and by already filtering out input features with low importance, more cycles can be used to pin point better feature and threshold combinations. The number of influential features used to construct the misprediction explanations in most cases had no significant impact on the performance or did not reveal a clear trend.

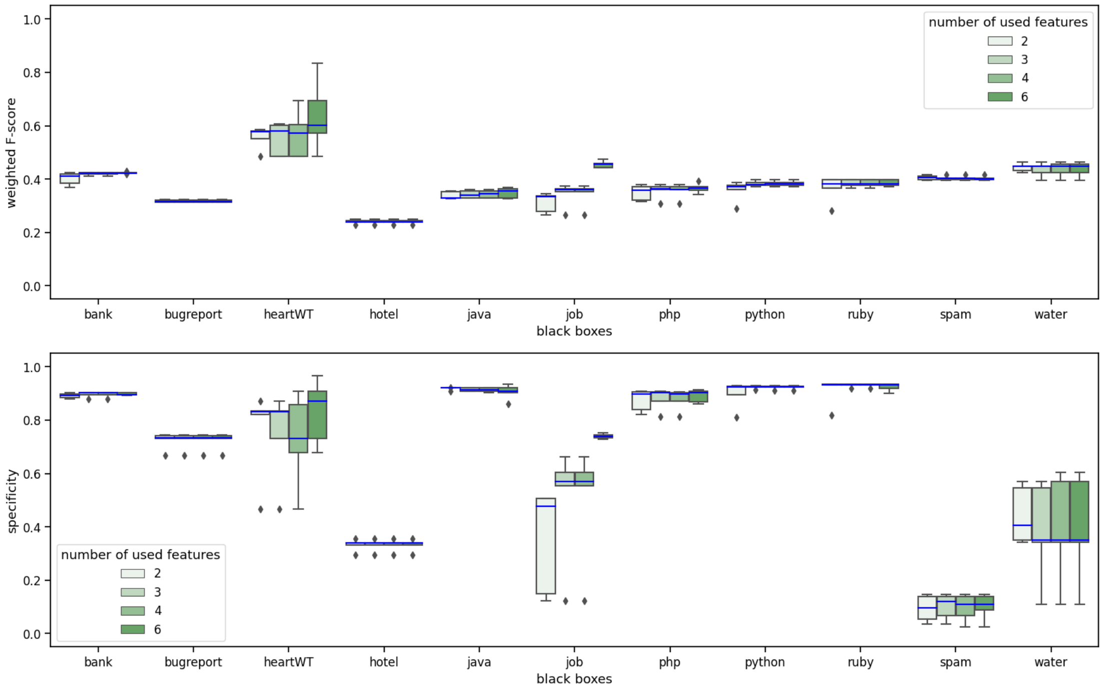


Figure 31: Weighted F-scores and specificity for decision trees (rule set, coverage: 0.6).

Partial Answer to RQ1 Based on the experimental results, we conclude the following:

RQ1 Based on our evaluation, we conclude that our approach to identify input features with high impact on causing model mispredictions, is not able to increase performance for MMD and decision trees, while improving results for Bayesian optimization only in a few cases.

6.2.2 Computational Demand

Figure 32 through 35 show the computational demand, in form of CPU time in seconds, for the different rule construction approaches, when generating a single rule or a full misprediction explanation with either a changing coverage and 5 influential input features, or with a desired coverage of 60 percent and a variable number of features. The results for single rules and a changing amount of influential input features can be found in the Appendix as Figure 38 on page 73.

Single Rule Figure 32 displays the CPU time required for single rules constructed by MMD, Bayesian optimization and decision trees with the five most influential input features. Figure 33 contains the computational demand for the ISLearn approach. The results show, that our approach, to only utilize influential input features, significantly reduces the computational demand of MMD and ISLearn for every case. Making it even feasible to generate rules for the spam data set with ISLearn. For Bayesian optimization the execution time stays in the same range of 200 to 250 seconds, exactly like when using all features. This is not surprising, as we already mentioned, that the computational demand of the Bayesian optimization approach is not dependent on the data set size. Our decision tree approach required more time for every case, when also employing the feature reduction. The reason is, that the explanation construction is already extremely quick with all features, and the time saved by using less features cannot offset the time required to train a

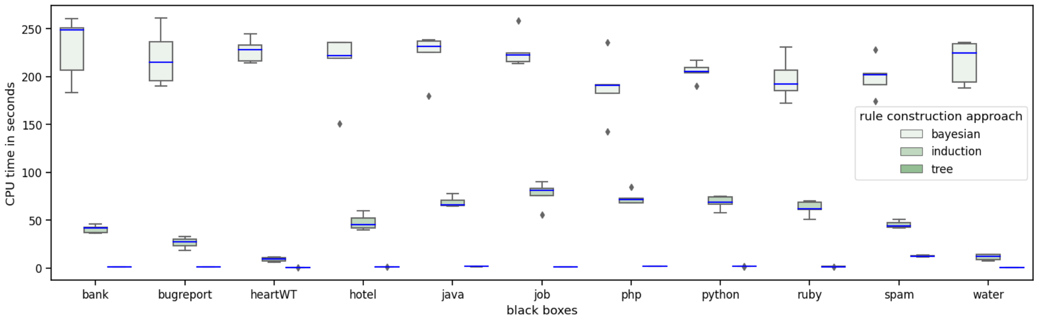


Figure 32: CPU time required by MMD, Trees and Bayes (single rule, five features).

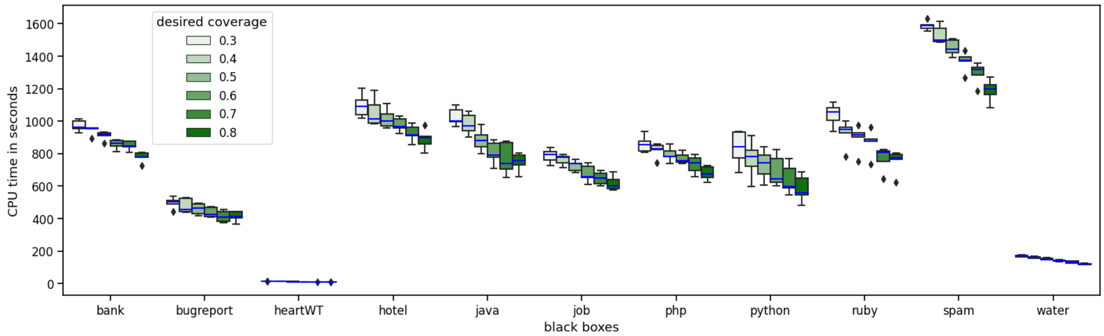


Figure 33: CPU time required by ISLearn (single rule, five features).

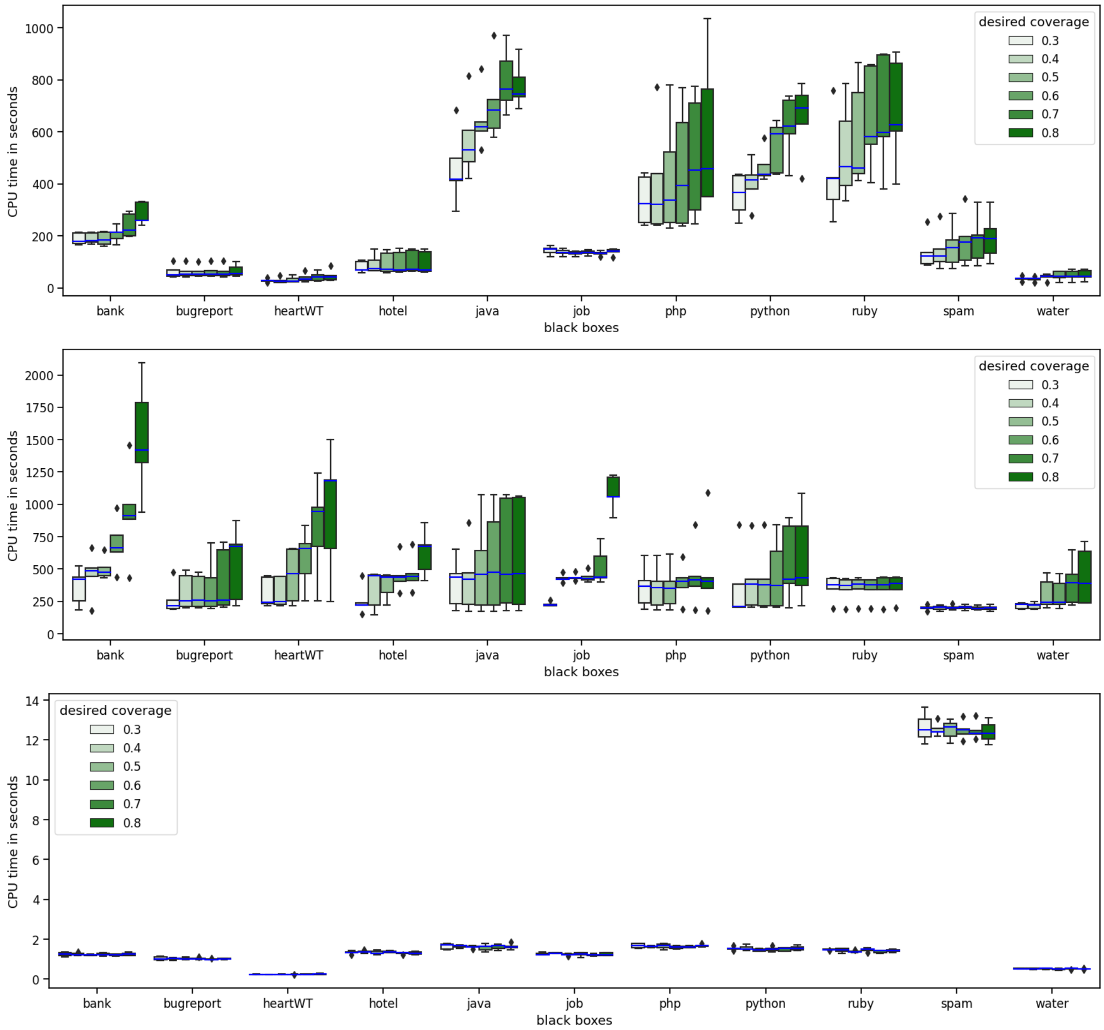


Figure 34: CPU time required by MMD (top), Bayes (middle) and Trees (bottom) (rule set, five features).

random forest to predict mispredictions and then identify influential features with feature importance. The visualized data for varying number of features (Figure 38) clearly demonstrates the reduction of computational demand for MMD and ISLearn when using less features. For the decision tree approach the difference is negligible.

Variable Coverage Figure 34 depicts the CPU time required for building full misprediction explanations with the five most influential input features. Most of the observations made for single rule generation are true for rule set construction as well. The computational demand for MMD is considerably reduced in every case. The execution time of the decision tree approach is still longer than just using all features, even when multiple trees would be trained on less data. Whether Bayesian optimization is faster or slower depends on the number of rules needed to reach the desired coverage, when using only a reduced number of features. On

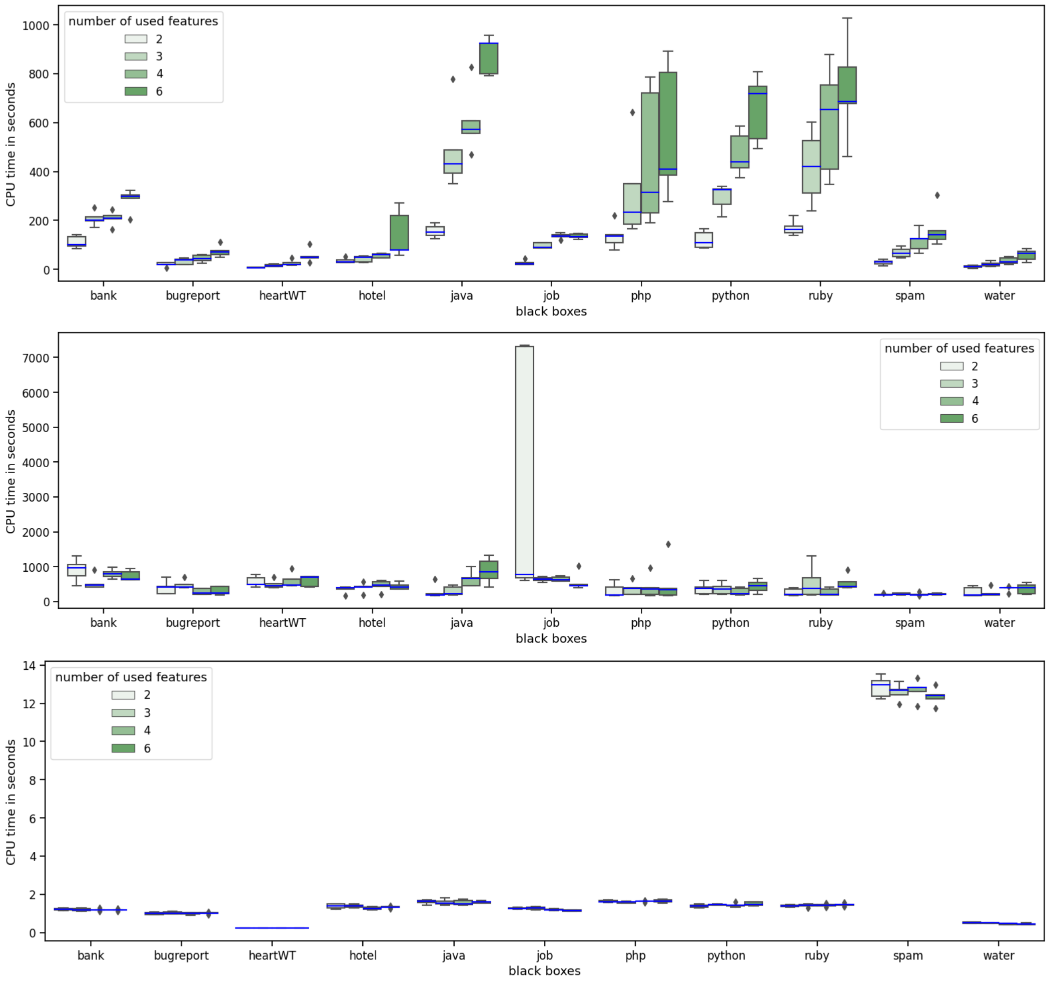


Figure 35: CPU time required by MMD (top), Bayes (middle) and Trees (bottom) (rule set, coverage: 0.6).

average, execution time is shorter for six cases, and higher in the remaining five, thus it depends on the data set and no clear statement can be made.

Variable Feature Count Figure 35 shows the CPU time required for building full misprediction explanations with a varying number of important input features, when adhering to a desired coverage of 60 percent. For MMD it is again very clear, that reducing the number of utilized input features considerably reduces the median CPU time required in nearly every case. For the decision tree approach the difference in execution time is again insignificant. The results for Bayesian optimization do not show a distinct connection between computational demand and number of influential features used. The outlier for two influential features and the job data set contains two of the three repetitions that we terminated after two hours, because they got stuck in an endless loop of adding the same rule to the rule set.

Summary The experimental results show that utilizing our approach to identify input features with high impact on causing model mispredictions, and using only these influential features to construct single rules or full misprediction explanations, leads to significantly reduced computational demand for MMD and ISLearn in every case. Since the execution time of the Bayesian optimization approach is not dependent on the data set size and mostly based on the number of iterative explanation construction steps needed to reach the desired coverage, no general statement can be made when comparing results to the baseline. The decision tree approach does not benefit from utilizing only the most influential input features, because it is already extremely quick when using all features, and the time saved from working with less features, does not outweigh the time spent to identify important features. Still, even with the slightly increased execution time and the reduced computational demand of MMD and ISLearn, the decision tree approach is by far the fastest in providing single rules as well as full misprediction explanations. Additionally, a clear correlation between computational demand and number of features used can be observed for MMD and ISLearn, while the effect for decision trees is insignificant.

Partial Answer to RQ2 Based on the experimental results, we conclude the following:

RQ2 We conclude that our approach to identify input features with high impact on causing model mispredictions, is able to significantly reduce computational demand for MMD and ISLearn in every case, while providing mixed results for Bayesian optimization, and increased execution times for decision trees.

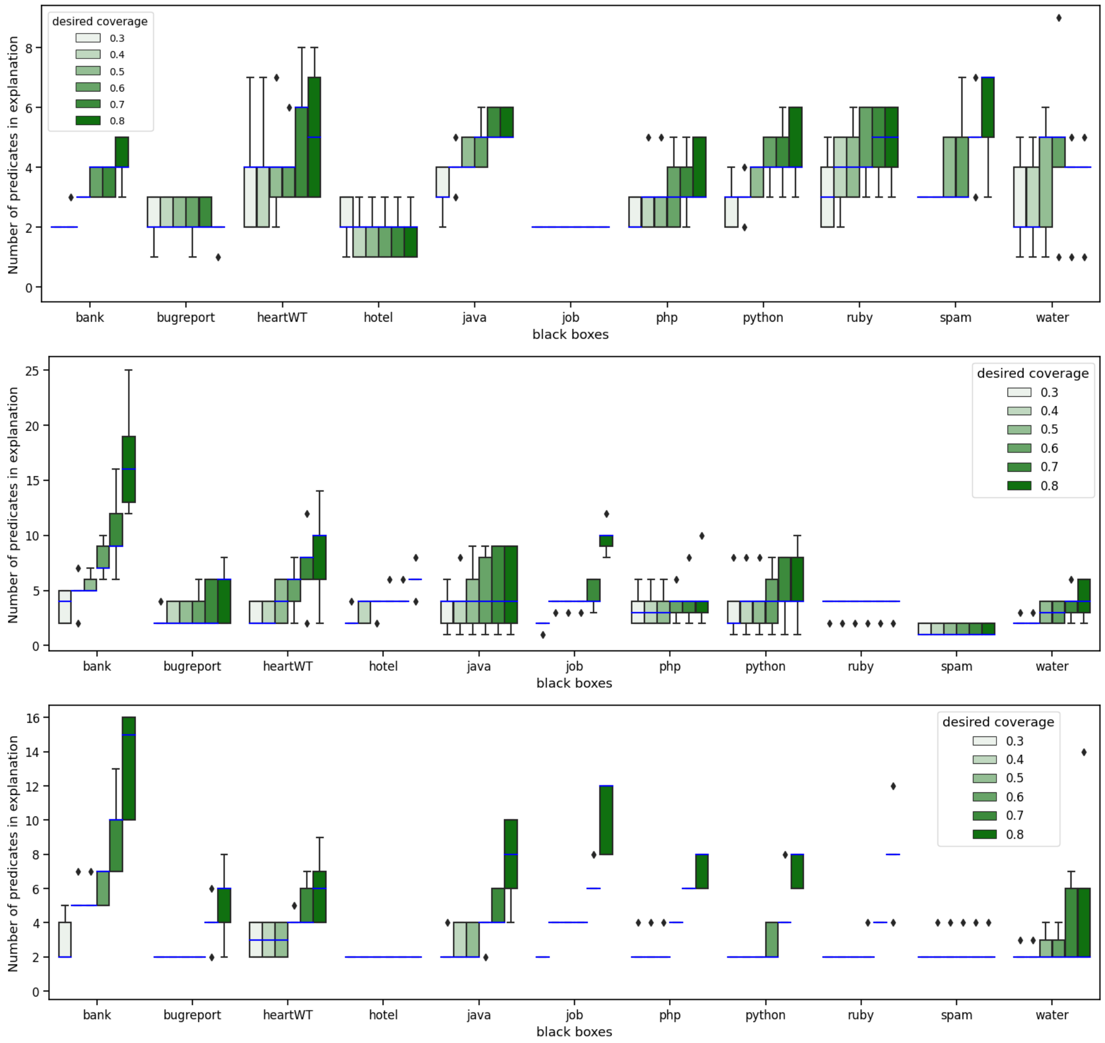


Figure 36: Number of predicates in explanation created by MMD (top), Bayes (middle) and Trees (bottom) (five features).

6.2.3 Explanation Length

Figure 36 and 37 show the length of generated misprediction explanations, in number of predicates contained within, for the different rule construction approaches, when generating full misprediction explanations with either a changing coverage and five influential input features, or with a desired coverage of 60 percent and a variable number of features.

Variable Coverage The visualized data in Figure 36 shows, when utilizing only the five most important features, MMD produces explanations with nearly the same length compared to using all features, with exceptions in case of the spam and water data sets. This is not surprising since the explanation length is also considered by MMD when constructing the rule set, keeping the explanations short. For Bayesian optimization the results match the findings for computational demand. A longer rule set leads to a proportional increase in execution time. In that vein it is also true, that like for computational demand, Bayesian optimization cannot improve upon the baseline consistently. For some cases the explanations are shorter, while for others slightly longer rule sets are constructed, but in most cases the difference in length is rather small. The same can be observed for decision trees, where the rule set length rarely differs more than a few predicates. Overall, the findings observed for explanation length with all features are still true: On average MMD produces the shortest rule sets, while Bayesian optimization provides the longest.

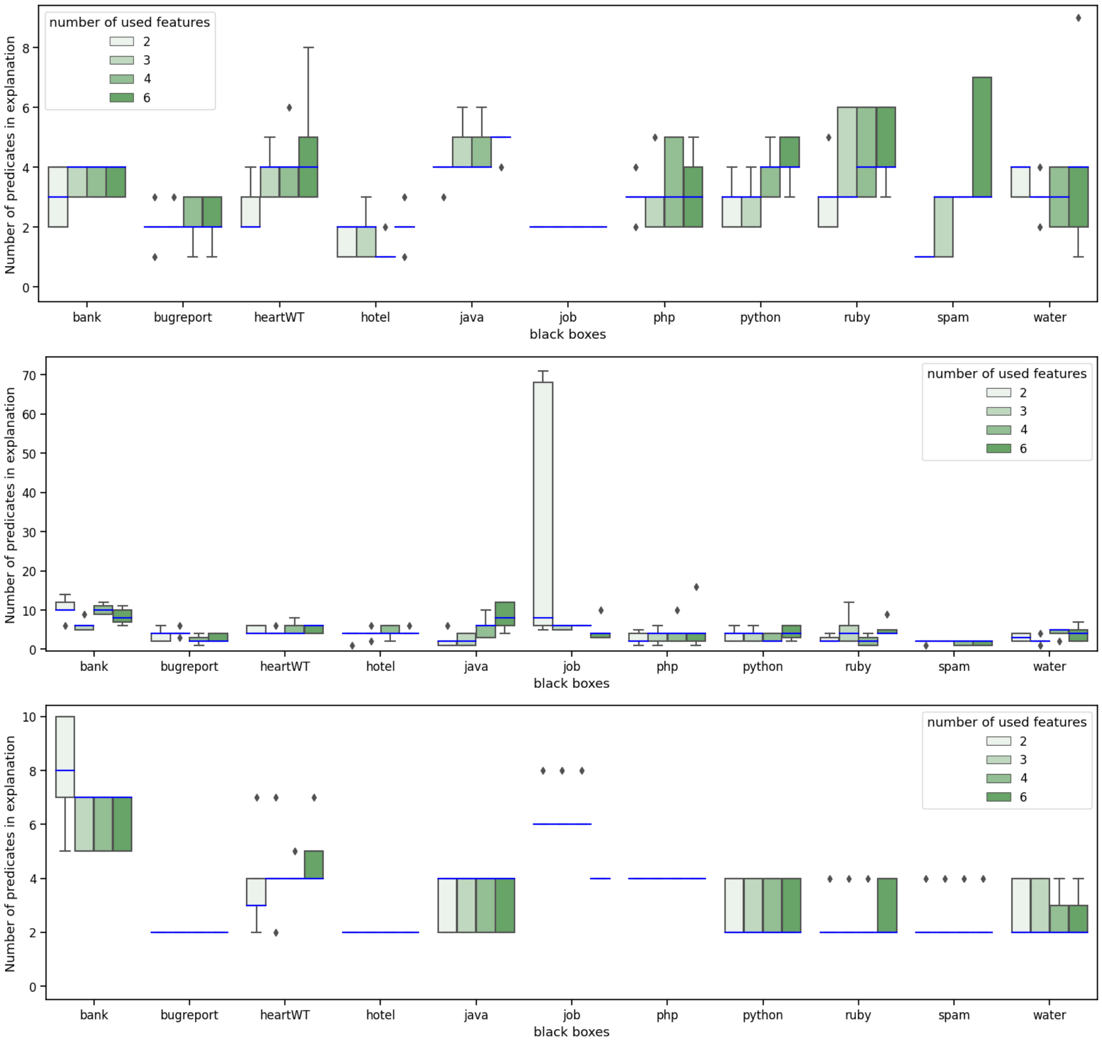


Figure 37: Number of predicates in explanation created by MMD (top), Bayes (middle) and Trees (bottom) (coverage: 0.6).

Variable Feature Count Figure 37 shows, that for MMD and decision trees, the number of influential features used had no noteworthy impact on the median explanation length. That is also true for Bayesian optimization in most cases, except for the bank and java data sets. The outliers for the job data set are again from the same two faulty repetitions mentioned in Section 6.2.2 paragraph “Variable Feature Count”.

Partial Answer to RQ3 Based on the experimental results, we conclude the following:

RQ3 Based on our evaluation, we conclude that our approach to identify input features with high impact on causing model mispredictions is not able to consistently reduce misprediction explanation length, but in most cases the length stays close to the same or only slightly increases.

Evaluation Summary Altogether, the main conclusions of our evaluation are: The decision tree approach was able to improve upon the baseline’s performance in most cases, while also reducing computational demand drastically. ISLearn has serious scalability problems, and Bayesian optimization could not reliably outdo the results of MMD. When utilizing our approach to construct misprediction explanations exclusively with influential input features, computational demand is reduced significantly for rule construction approaches, whose execution time is influenced by the number of input features. The loss of information by only using a small number of input features only slightly reduced explanation performance and slightly increased rule set length.

6.3 Threats to Validity

In this thesis, we rely on non-deterministic processes to construct misprediction explanations for a black box machine learning model. Therefore, we acknowledge possible internal and external threats to the validity of our work.

Internal Since machine learning algorithms intrinsically contain non-deterministic characteristics, utilizing them always poses a major threat to internal validity. It is essential to be certain, that results are not just random occurrences. For that reason, a thorough statistical evaluation is required, which we did by training five models for every data set with different training sets and running every experiment five times. We use the same data sets as Gesi et al. [Ges+23], whose research topic is very similar to ours. Additionally, we did not tune the input parameters of the baseline we compare against (MMD) or changed hyperparameters to influence the results of our approaches. We kept the default input parameters when training machine learning models with the exception of limiting the maximum decision tree depth to two, when the tree is used to extract rules with two predicates. Finally, we automated the data collection as well as the statistical evaluation, to avoid human error.

The baseline determines performance of an explanation with precision and recall. Because our used data sets display a noticeable difference in class distribution, we opted to also measure specificity, which in turn enables us to assess the quality of generated misprediction explanation with more confidence.

External The central threat to external validity is, whether our experimental results can be generalized for other black box models, based on the restricted number of evaluated models we trained. Still, since we utilized models based on a variety of real-life data set, we are convinced that our approaches will induce similar results for all kind of data sets. In addition, our techniques are fully model-agnostic and consequently work for any type of machine learning model.

7 Discussion and Limitations

In the following, we discuss the results of this thesis and detail certain limitations of our proposed approach.

Requirement of Test Data To generate a misprediction explanation for a given black box machine learning model, a corresponding test data set is required, because data instances with known ground truth are needed to be able to check for mispredictions. For most cases this also makes it impossible to easily generate additional data instances, since the ground truth is not known. Additionally, if the constructed misprediction explanation performs well for new data is very dependent on the quality of the test data set. Our experimental results for repetitions with different training-test data splits already show considerable differences in performance and rule set length of the generated explanations. Consequently, the test set needs to have a reasonable split between correct predictions and mispredictions, as well as contain data instances that represent the full value range of every feature if possible. Otherwise, the misprediction explanation might only perform well for a certain input space covered by the test data set.

Precision, Recall and Length A balance must be struck between precision recall and explanation length. Focusing too much on two characteristics always worsens the third. Maximizing precision and recall leads to long, overfit rules, while maximizing precision and keeping the rule set short causes a low recall. The third option is a low precision when focusing on recall and length. Under these constraints it is important to adjust the weights of these characteristics depending on the use case.

Rule Construction Approaches MMD utilizes rule induction with beam search to construct the rules for misprediction explanations and we proposed three more possible approaches, employing ISLearn, decision trees and Bayesian optimization. Each option has advantages and disadvantages, which need to be considered. Since MMD takes rule length into account, it consistently provides short explanations, but in return the suggested rule sets could not always satisfy the desired coverage and MMD becomes expensive for bigger data sets. ISLearn is the only approach, able to adhere to a coverage threshold for single rules. I addition ISLearn tries to produce an explanation with a coverage close to the threshold, while the other approaches tend to overshoot the target. In return ISLearn has even bigger scalability problems than MMD making it extremely computationally expensive in its current state. Bayesian optimization's execution time is not affected by data set size at all, but performance of produced explanations likely decreases for an increasing number of input features, when the number of optimization cycles is not increased as well. Furthermore, in our experiments, rule sets constructed with Bayesian optimization contained the most rules. Our decision tree approach is able to produce misprediction explanations extremely fast, while also providing the best explanation performance in most cases. Without tweaking the training parameters, rule sets tend to be longer than MMD, since it can happen, that rules with very high precision but low recall are added to the explanation.

Rule Performance Measure When constructing a misprediction explanation, the manner in which the performance of rules is compared against each other is very influential on the final result. MMD considers a mix of precision, recall and rule length. Since we limited the rule length to a maximum of two predicates for our thesis, we utilized weighted F-score to assess, which rule to add to the explanation next. In this way, both precision and recall matter, but we were able to increase the weight of precision to double that of recall. In most cases this prevented both: very long rule sets, by adding rules with high precision but only covering a few mispredictions, and many false positives, by using high recall rules with low precision. Still, the exact weights are arbitrarily chosen, just to favor precision. A more sophisticated way to measure rule performance is desirable and even required when incorporating the possibility of different rule lengths.

Feature Importance Measure For our approach to identify the most influential input features we needed an explainable machine learning technique to extract, which input features have overall the most impact on a model's prediction. We tried three options for random forests with scikit-learn: the build-in impurity-based feature importance, feature importance based on feature permutation and feature importance with SHAP. The misprediction explanations constructed when employing SHAP performed in many cases slightly better than the one generated when using impurity-based feature importance, but SHAP is so computationally expensive (especially for bigger data sets), that the overall execution times increased, rendering the feature reduction meaningless. Impurity based feature importance can be susceptible for high cardinality features and feature importance based on feature permutation is an alternative that overcomes these limitations, but is more costly. The two approaches certainly deemed different input features as most influential, but when constructing misprediction explanations with them, the ones relying on permutation importance performed far worse, even when we tried different amount of feature permutations. For these reasons we settled for impurity-based feature importance in this thesis, but the whole approach would unquestionably benefit from a fast but more reliable feature importance measure.

Baseline We chose to compare our results against the explanations provided by MMD, since the research by Cito et al. [Cit+21] presents the idea of misprediction explanations we tried to improve upon, in both performance and computational demand. During our work on this thesis, new research was published by Gesi et al. [Ges+23], also aiming to improve upon MMD by leveraging feature bias to select important features. Since, we use a different approach to select important features, we considered comparing our results to theirs, and chose to use the same data sets to make that possible. We distanced ourselves from the idea after we could not even get close to reproduce the performance they calculate as the baseline for MMD and checking the data set coverage of one example explanations presented in the paper. One example rule set given in the paper is for the ruby merge conflict data set, which has around 40.000 data instances. The given rule set only covers eight data instances from the full data set. Even considering, that all these eight are split into the test set and mispredicted by the trained model, we are unable to comprehend how the precision and recall values of more than 90 percent are calculated for the rule set.

8 Conclusion and Future Work

In this thesis, we proposed different approaches to improve upon the idea of misprediction explanations presented by Cito et al. [Cit+21]. We aimed to boost explanation performance by utilizing different rule construction techniques and to enhance scalability with a procedure to identify input features with the most impact on causing mispredictions. Instead of rule induction with beam search, we evaluated the use of ISLearn, Bayesian optimization and decision trees to generate rules. To reduce the number of input features, we trained a random forest to predict mispredictions and employed impurity-based feature importance, to extract the significant ones.

We evaluated the effectiveness of our approaches by constructing misprediction explanations for machine learning black boxes trained on 11 real-world data sets and compared the results to MMD as a baseline, in three essential characteristics: (*i*) Rule Set Performance, (*ii*) Computational Demand and (*iii*) Rule Set Length. The experiments were repeated with and without our feature reduction approach, and for construction of single rules or full misprediction explanations. Because of severe scalability problems ISLearn could only be evaluated when generating single rules (no disjunctions). Evaluating performance, the results for ISLearn and Bayesian optimization suggest no consistent improvement, performing worse for some and better for other cases, compared to the baseline. The decision tree approach however was able to improve upon the baseline in most cases. Considering computational demand, ISLearn could not compete, while Bayesian optimization could beat the baseline only in a few cases. In contrast, decision trees eclipsed the results of the baseline drastically reducing computational demand in every case. In some cases, up to more than 99.9 percent, producing a better performing explanation in around one second instead of more than 30 minutes. Evaluating rule set length, MMD already constructs short rule sets and both Bayesian optimization and decision trees could not improve upon that in most cases. The experimental results show, that explanation performance is usually only slightly reduced and rule set length just slightly increased, when utilizing our approach to construct the explanation exclusively with influential input features. In return computational demand is reduced significantly for rule construction approaches, whose execution time is influenced by the number of input features. An exception is our decision tree approach, which already is very fast, and identifying the features to use takes more time than is saved by training the decision trees on less features.

In conclusion, determining and utilizing only influential input features can significantly reduce computational demand of constructing misprediction explanations, when a slight reduction in performance is acceptable. Furthermore, the results for our decision tree approach are very promising, being able to provide slightly better performing explanations than the baseline in a fraction of the time, without the need to limit input features for scalability.

Future Work There are several opportunities for improvement with future research, some already mentioned in Section 7. All rule set construction approaches would benefit greatly from the introduction of a more sophisticated rule performance measure. MMD uses weights for precision, recall and rule length, while our approaches utilize weighted F-score to quantify rule performance. Both are somewhat rudimentary and considering more rule and rule set characteristics might improve explanation quality as a whole. When reducing the number of input features to improve scalability, it is crucial to select the correct ones to keep. We use impurity-based feature importance to extract the most influential features from a random forest, which is trained to predict mispredictions. However, impurity-based feature importance has flaws and a more reliable technique would be desirable, but needs to be computationally cheap to be feasible. If the technique is model agnostic, this would also allow to train more complex machine learning models than random forests, to better predict mispredictions. Since employing decision trees to generate rules had very promising results, even without influencing training parameters, the approach can likely be improved with a more sophisticated training process. Exploring different options for training parameters or varying data subsets, in a random forest like fashion, might discover better rules than the current straight forward method. Finally, an unexplored topic is the relation between the composition of the test set used to construct a misprediction explanation and the performance of the rule set for new unseen data. No universally useful misprediction explanation can be built from a test data set only containing instances with minimal differences, if the black box model was trained on more varied feature values. Thus, the question arises: What are necessary requirements regarding test sets, to construct misprediction explanations useful in practice?

References

[AJ18] David Alvarez-Melis and Tommi S. Jaakkola. *On the Robustness of Interpretability Methods.* 2018. DOI: `10.48550/ARXIV.1806.08049`.

[Atz15] Martin Atzmueller. "Subgroup discovery". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.1 (2015), pp. 35–49.

[Bot+20] Tiago Botari, Frederik Hvilshøj, Rafael Izbicki, and Andre CPLF de Carvalho. "MeLIME: meaningful local explanation for machine learning models". In: *arXiv preprint arXiv:2009.05818* (2020).

[Bre96] Leo Breiman. "Bagging predictors". In: *Machine learning* 24 (1996), pp. 123–140.

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[Bre+84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees.* CRC press, 1984.

[Cas+17] Marco Castelluccio, Carlo Sansone, Luisa Verdoliva, and Giovanni Poggi. "Automatically analyzing groups of crashes for finding correlations". In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* 2017, pp. 717–726.

[Che+20] Simin Chen, Soroush Bateni, Sampath Grandhi, Xiaodi Li, Cong Liu, et al. "DENAS: automated rule generation by knowledge extraction from neural networks". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2020, pp. 813–825.

[CG19] Ambika Choudhury and Deepak Gupta. "A survey on medical diagnosis of diabetes using machine learning techniques". In: *Recent Developments in Machine Learning and Data Analytics: IC3 2018.* Springer. 2019, pp. 67–78.

[Cit+21] Jürgen Cito, Isil Dillig, Seohyun Kim, Vijayaraghavan Murali, and Satish Chandra. "Explaining mispredictions of machine learning models using rule induction". In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2021, pp. 716–727.

[cit22] citostyle. *MMD implementation.* 2022. URL: `https://github.com/facebookresearch/mmd` (visited on 08/06/2023).

[CN89] Peter Clark and Tim Niblett. "The CN2 induction algorithm". In: *Machine learning* 3 (1989), pp. 261–283.

[Coh95] William W Cohen. "Repeated incremental pruning to produce error reduction". In: *Machine Learning Proceedings of the Twelfth International Conference ML95.* 1995.

[Fra18] Peter I Frazier. “A tutorial on Bayesian optimization”. In: *arXiv preprint arXiv:1807.02811* (2018).

[Fri01] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[Ges+23] Jiri Gesi, Xinyun Shen, Yunfan Geng, Qihong Chen, and Iftekhar Ahmed. “Leveraging Feature Bias for Scalable Misprediction Explanation of Machine Learning Models”. In: *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. 2023.

[Hal+17] Patrick Hall, Navdeep Gill, Megan Kurka, and Wen Phan. “Machine learning interpretability with h2o driverless ai”. In: *H2O. ai* (2017).

[Hu+18] Linwei Hu, Jie Chen, Vijayan N Nair, and Agus Sudjianto. “Locally interpretable models and effects based on supervised partitioning (LIME-SUP)”. In: *arXiv preprint arXiv:1806.00663* (2018).

[Jag17] Venkata Jagannath. *Diagram of a random decision forest*. 2017. URL: `https://en.wikipedia.org/wiki/Random_forest#/media/File:Random_forest_diagram_complete.png` (visited on 07/29/2023).

[Kim+20] Edward Kim, Divya Gopinath, Corina Pasareanu, and Sanjit A Seshia. “A programmatic and semantic approach to explaining and debugging neural network based object detectors”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11128–11137.

[KK06] Sotiris Kotsiantis and Dimitris Kanellopoulos. “Discretization techniques: A recent survey”. In: *GESTS International Transactions on Computer Science and Engineering* 32.1 (2006), pp. 47–58.

[Lak+] H Lakkaraju, E Kamar, R Caruana, and J Leskovec. “Interpretable and explorable approximations of black box models. 2017”. In: *arXiv preprint arXiv:1707.01154* ().

[LBL16] Himabindu Lakkaraju, Stephen H Bach, and Jure 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.

[Lee+20] Sangmin Lee, Younghoon Kim, Hyungu Kahng, Soon-Kyo Lee, Seokhyun Chung, et al. “Intelligent traffic control for autonomous vehicle systems based on machine learning”. In: *Expert Systems with Applications* 144 (2020).

[LHK19] Ilse van der Linden, Hinda Haned, and Evangelos Kanoulas. “Global aggregations of local explanations for black box models”. In: *arXiv preprint arXiv:1907.03039* (2019).

[LL17] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Advances in neural information processing systems* 30 (2017).

[Ma+18] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. "MODE: automated neural network model debugging via state differential analysis and input selection". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2018, pp. 175–186.

[Mol22] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.* 2nd ed. 2022.

[MCB20] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. "Interpretable machine learning–a brief history, state-of-the-art and challenges". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer. 2020, pp. 417–431.

[Ni+20] Jianjun Ni, Yinan Chen, Yan Chen, Jinxiu Zhu, Deena Ali, et al. "A survey on theories and applications for self-driving cars based on deep learning methods". In: *Applied Sciences* 10.8 (2020), p. 2749.

[Qia+20] Rebecca Qian, Yang Yu, Wonhee Park, Vijayaraghavan Murali, Stephen Fink, et al. "Debugging crashes using continuous contrast set mining". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice.* 2020, pp. 61–70.

[Qui87] J Ross Quinlan. "Decision trees as probabilistic classifiers". In: *Proceedings of the Fourth International Workshop on Machine Learning.* Elsevier. 1987, pp. 31–37.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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.

[rin22] rindPHI. *ISLearn implementation.* 2022. URL: `https://github.com/rindPHI/islearn/` (visited on 08/07/2023).

[Riv87] Ronald L Rivest. "Learning decision lists". In: *Machine learning* 2 (1987), pp. 229–246.

[SR19] Sharath M Shankaranarayana and Davor Runje. "ALIME: Autoencoder based approach for local interpretability". In: *International conference on intelligent data engineering and automated learning.* Springer. 2019, pp. 454–463.

[Sha+53] Lloyd S Shapley et al. "A value for n-person games". In: (1953).

[Sla+20] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu 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.

[Son+06] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair. "Software defect association mining and defect correction effort prediction". In: *IEEE Transactions on software engineering* 32.2 (2006), pp. 69–82.

[SZ22] Dominic Steinhöfel and Andreas Zeller. "Input invariants". In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2022, pp. 583–594.

[SH77] Philip H Swain and Hans Hauska. "The decision tree classifier: Design and potential". In: *IEEE Transactions on Geoscience Electronics* 15.3 (1977), pp. 142–147.

[VBC20] Giorgio Visani, Enrico Bagli, and Federico Chesani. *OptiLIME: Optimized LIME Explanations for Diagnostic Computer Algorithms.* 2020. DOI: 10.48550/ARXIV.2006.05714.

[Vis+22] Giorgio Visani, Enrico Bagli, Federico Chesani, Alessandro Poluzzi, and Davide Capuzzo. "Statistical stability indices for LIME: Obtaining reliable explanations for machine learning models". In: *Journal of the Operational Research Society* 73.1 (2022), pp. 91–101.

[WLS15] Pengfei Wei, Zhenzhou Lu, and Jingwen Song. "Variable importance analysis: A comprehensive review". In: *Reliability Engineering & System Safety* 142 (2015), pp. 399–432.

[Wu+19] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. "Errudite: Scalable, reproducible, and testable error analysis". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* 2019, pp. 747–763.

Appendix

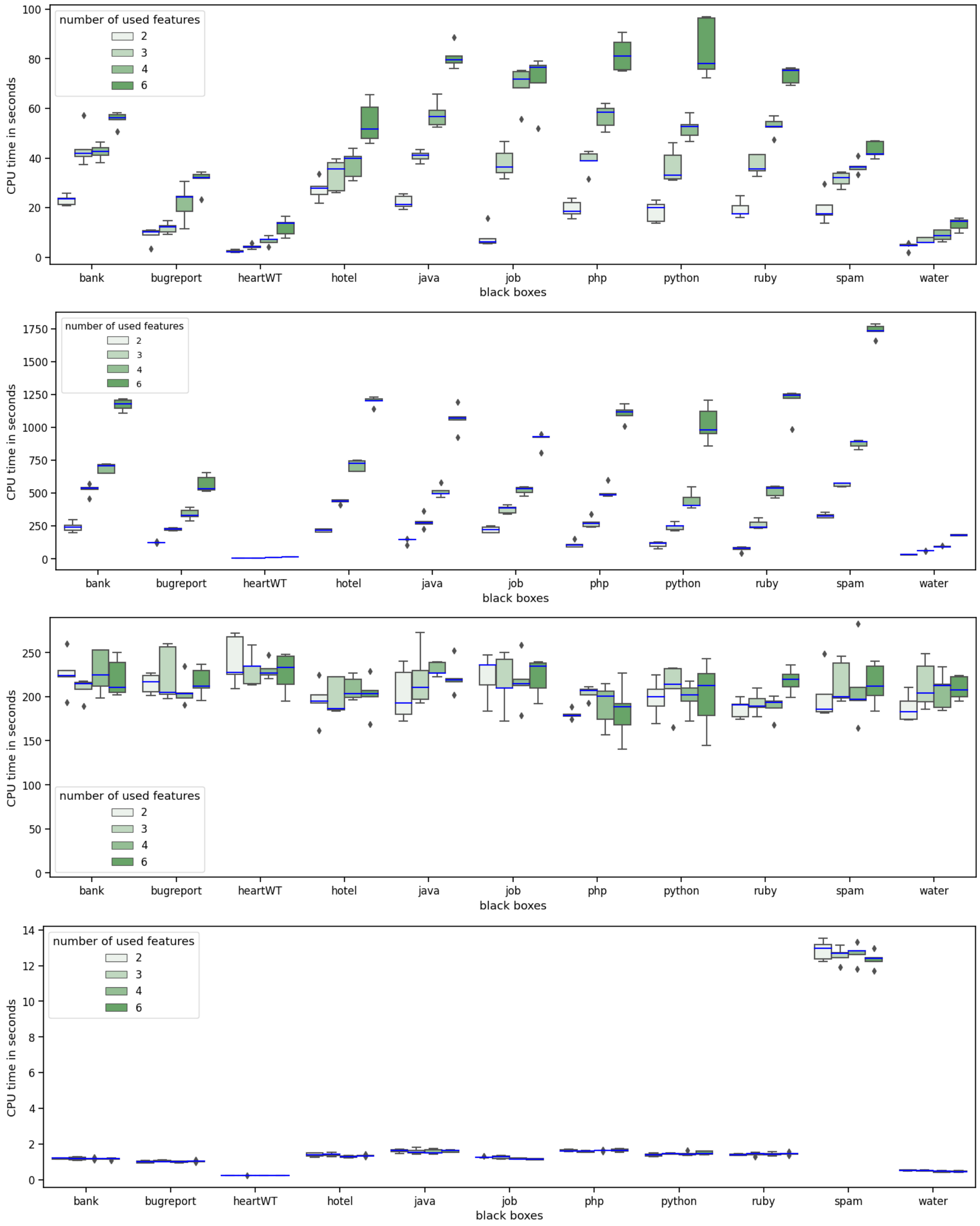


Figure 38: CPU time required by top to bottom: MMD, ISLearn, Bayes and Trees (rule set, coverage: 0.6).

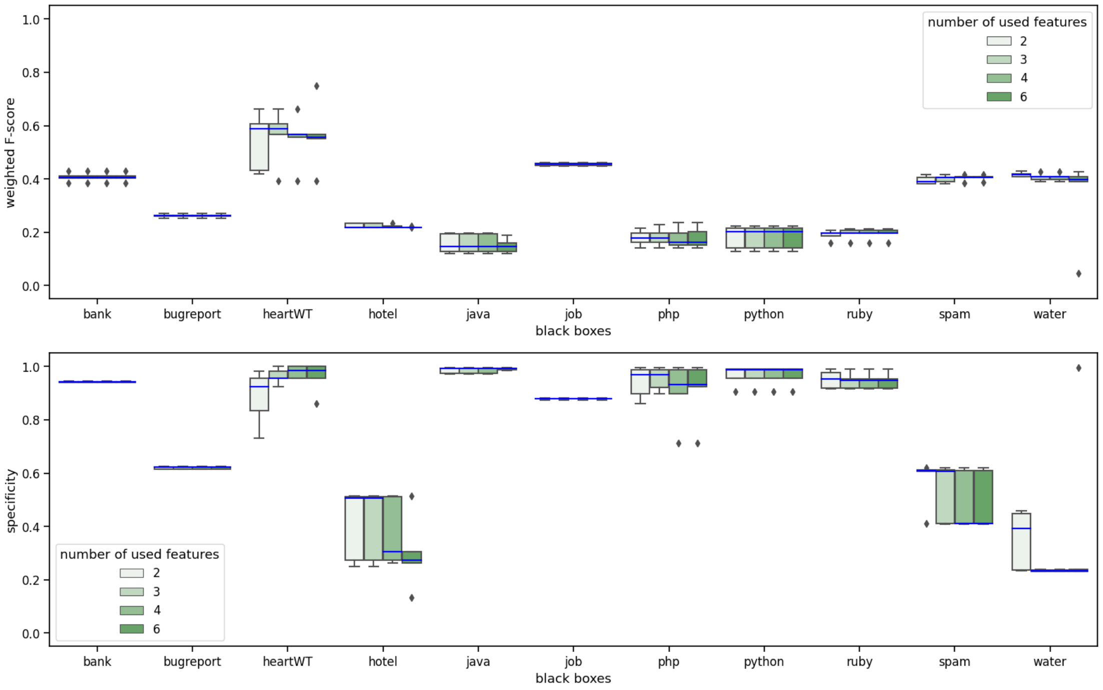


Figure 39: Weighted F-scores and specificity for MMD (single rule, coverage: 0.6).

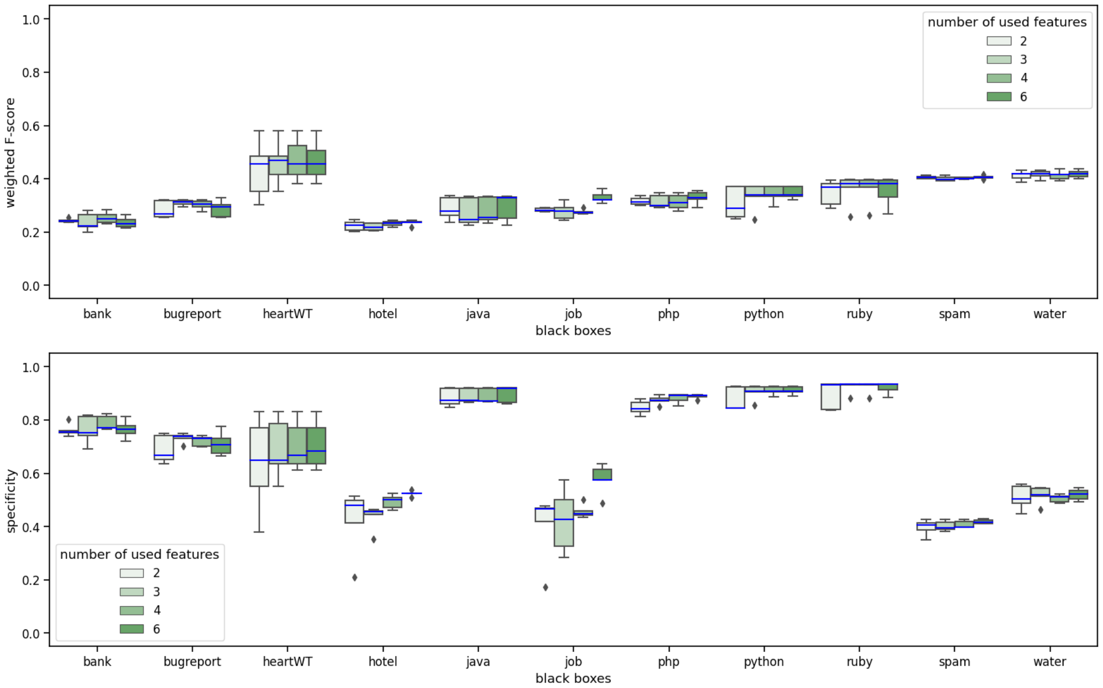


Figure 40: Weighted F-scores and specificity for ISLearn (single rule, coverage: 0.6).

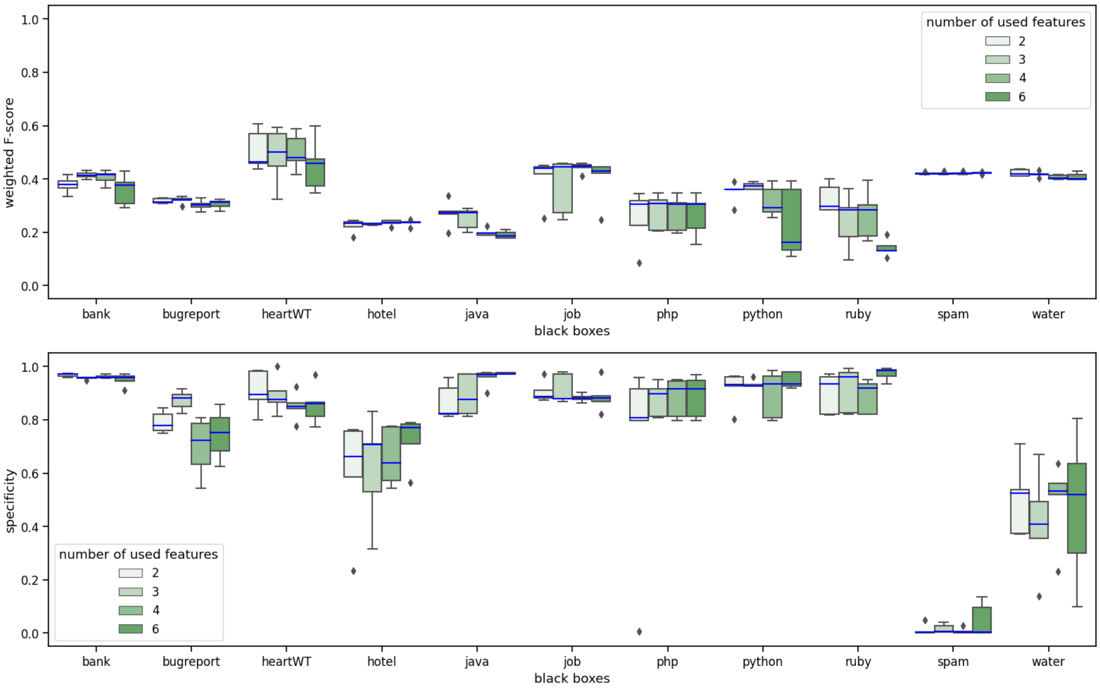


Figure 41: Weighted F-scores and specificity for Bayes (single rule, coverage: 0.6).

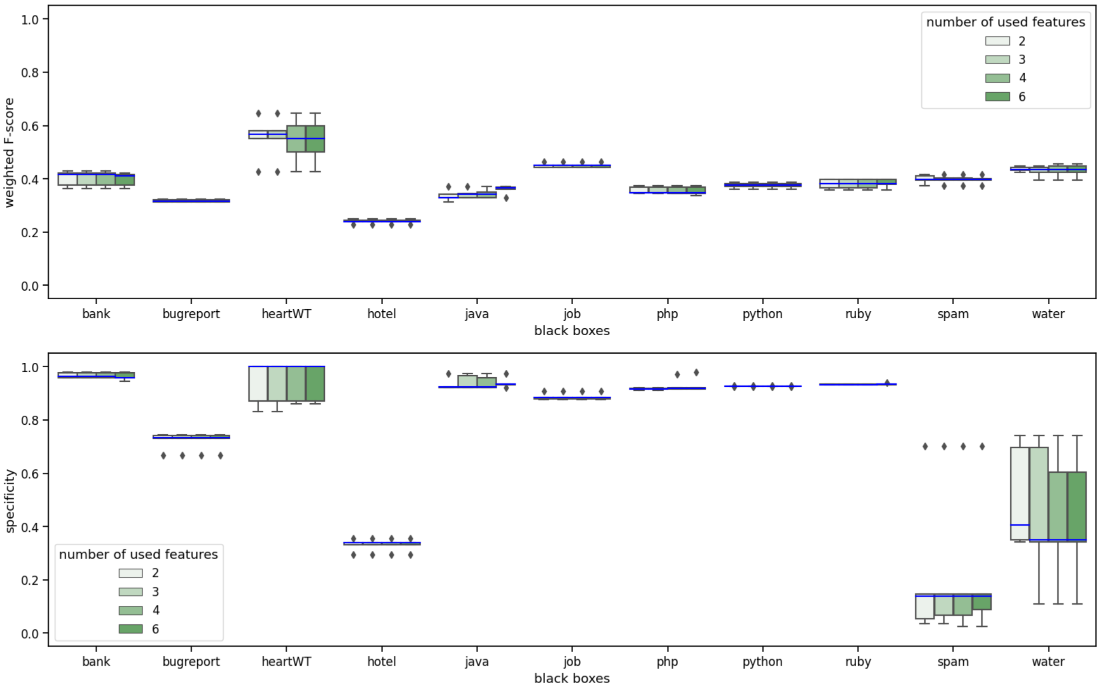


Figure 42: Weighted F-scores and specificity for Trees (single rule, coverage: 0.6).

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 14. September 2023 ....