# The Impact of Feature Importance Methods on the Interpretation of Defect Classifiers

Gopi Krishnan Rajbahadur, Shaowei Wang, Gustavo A. Oliva, Yasutaka Kamei, *Senior Member, IEEE*, and Ahmed E. Hassan, *Fellow, IEEE*

**Abstract**—Classifier specific (CS) and classifier agnostic (CA) feature importance methods are widely used (often interchangeably) by prior studies to derive feature importance ranks from a defect classifier. However, different feature importance methods are likely to compute different feature importance ranks even for the same dataset and classifier. Hence such interchangeable use of feature importance methods can lead to conclusion instabilities unless there is a strong agreement among different methods. Therefore, in this paper, we evaluate the agreement between the feature importance ranks associated with the studied classifiers through a case study of 18 software projects and six commonly used classifiers. We find that: 1) The computed feature importance ranks by CA and CS methods do not always strongly agree with each other. 2) The computed feature importance ranks by the studied CA methods exhibit a strong agreement including the features reported at top-1 and top-3 ranks for a given dataset and classifier, while even the commonly used CS methods yield vastly different feature importance ranks. Such findings raise concerns about the stability of conclusions across replicated studies. We further observe that the commonly used defect datasets are rife with feature interactions and these feature interactions impact the computed feature importance ranks of the CS methods (not the CA methods). We demonstrate that removing these feature interactions, even with simple methods like CFS improves agreement between the computed feature importance ranks of CA and CS methods. In light of our findings, we provide guidelines for stakeholders and practitioners when performing model interpretation and directions for future research, e.g., future research is needed to investigate the impact of advanced feature interaction removal methods on computed feature importance ranks of different CS methods.

**Index Terms**—Model interpretation, model agnostic interpretation, built-in interpretation, feature Importance analysis, variable importance

## 1 Introduction

Defect classifiers are widely used by many large software corporations [1], [2], [3], [4] and researchers [5], [6], [7]. Defect classifiers are commonly interpreted to uncover insights to improve software quality. Such insights help practitioners formulate strategies for effective testing, defect avoidance, and quality assurance [8], [9]. Therefore it is pivotal that these generated insights are reliable.

When interpreting classifiers, prior studies typically employ a feature importance method to compute a ranking of feature importances (a.k.a., feature importance ranks) [8], [10], [11], [12], [13]. These feature importance ranks reflect the order in which the studied features contribute to the predictive capability of the studied classifier [14]. These feature importance methods can be divided in two categories: classifier-specific (CS) and classifier-agnostic (CA) methods. A classifier-specific (CS) method makes use of a given classifier's internals to measure the degree to which each feature contributes to a classifier's predictions [15]. We note, however, that a CS method is not always readily available for a given classifier. For example, complex classifiers like SVMs and deep neural networks do not have a widely accepted CS method(s) [16].

For cases such as those, or when a universal way of comparing the feature importance ranks of different classifiers is required [17], [18], classifier-agnostic (CA) methods are typically used. Such CA methods measure the contribution of each feature towards a classifier's predictions. For instance, some CA methods measure the contribution of each feature by effecting changes to that particular feature in the dataset and observing its impact on the outcome. The primary advantage of CA methods is that they can be used for any classifier (i.e., from interpretable to black box classifiers).

Despite computing feature importance ranks using different ways, CS and CA methods are indiscriminately and interchangeably used in software engineering studies (Table 1). For instance, to compute feature importance ranks for a random forest classifier, Treude and Wagner [19] and Yu *et al.* [20] use CS methods: the Gini importance and the Breiman's importance methods respectively. On the other hand, Mori and Uchihira [13] and Herzig [21] use CA methods: Partial Dependence Plot (PDP) and filterVarImp

- *Gopi Krishnan Rajbahadur is with the Centre for Software Excellence, Huawei, Canada. E-mail: gopi.krishnan.rajbahadur1@huawei.com.*
- *Gustavo A. Oliva and Ahmed E. Hassan are with the Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada. E-mail: {gustavo, ahmed}@cs.queensu.ca.*
- *Shaowei Wang is with the Department of Computer Science, University of Manitoba, Winnipeg, MB R3T 2N2, Canada. E-mail: shaowei@cs.umanitoba.ca.*
- *Yasutaka Kamei is with the Principles of Software Languages (POSL) Lab, Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan. E-mail: kamei@ait.kyushu-u.ac.jp.*

*Manuscript received 19 Feb. 2020; revised 27 Jan. 2021; accepted 31 Jan. 2021. Date of publication 4 Feb. 2021; date of current version 18 July 2022. (Corresponding author: Shaowei Wang.) Recommended for acceptance by E. Murphy-Hill. Digital Object Identifier no. 10.1109/TSE.2021.3056941*

0098-5589 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Different Feature Importance Methods Used for Interpreting Various Classifiers in the Software Engineering Literature

| Classifier Family | Papers using CS | Used CS methods | Papers using CA | Used CA methods |
|---|---|---|---|---|
| **Statistical Techniques** | [29][30]✓ [13, 31–36] | Regression coefficients, ANOVA | [10, 21, 37, 38] | Boruta*, filterVarImp† |
| **Rule-Based Techniques** | [39]✓,[40] | Interpreting rules, varImp✇ | [37] | Boruta* |
| **Neural Networks** | [41]✖ [42]✓ | MODE[42] | [37] | Boruta* |
| **Decision Trees** | [43][44]✓[45] | Decision branches, Gini importance | [10, 21, 37, 38] | Boruta*, filterVarImp† |
| **Ensemble methods-Bagging** | [19, 20, 46] [11]✖ [47, 48] [49]✖ [12, 50, 51] [52]✖ | Permutation importance, Gini importance | [10, 13, 21, 37, 38, 53, 54] | Boruta*, filterVarImp†, PDP, Marks method[55], BestFirst★ |
| **Ensemble methods-Boosting** | - | - | [10, 21, 37] | Boruta*, filterVarImp† |

✖-*The used method for computing the feature importance ranks is not mentioned*
✓-*Papers in which the rationale for choosing a given feature importance method is specified*
∗-*https://cran.r-project.org/web/packages/Boruta/index.html*
†-*https://www.rdocumentation.org/packages/caret/versions/6.0-84/topics/filterVarImp*
★-*https://www.rdocumentation.org/packages/FSelector/versions/0.31/topics/best.first.search*
✇-*https://www.rdocumentation.org/packages/caret/versions/6.0-84/topics/varImp*

respectively. Since these methods compute feature importances differently (Section 3.3), different CS or CA methods are likely to compute different feature importance ranks for the same classifier. Yet, we observe that the rationale for choosing a given feature importance method is rarely motivated by prior studies (Section 2.1).

The interchangeable use of feature importance methods is acceptable only if the feature importance ranks computed by these methods do not differ from each other. Therefore, in order to determine the extent to which the importance ranks computed by different importance methods agree with each other, we conduct a case study on 18 popularly used software defect datasets using classifiers from six different families. We compute the feature importance ranks using six CS and two CA methods on these datasets and classifiers. The list of CS methods is summarized in Table 6, available in the online supplemental material. The two CA methods are: permutation importance (Permutation) and SHapley Additive ExPlanations (SHAP). Finally, we compute Kendall's Tau, Kendall's W, and Top-k ($k \in \{1, 3\}$) overlap to quantify the agreement between the computed feature importance ranks by the different studied feature importance methods for a given classifier and dataset. While Kendall's measures compute differences across the different feature importance ranks, the Top-K overlap measure focuses on the top-k items of these rankings (more details in Section 3.5). We highlight our findings below:

- The computed feature importance ranks by CA and CS methods do not always strongly agree with each other. For two of the five studied classifiers, even the most important feature varies across CA and CS methods.
- The computed feature importance ranks by the studied CA methods exhibit a strong agreement including the features reported at top-1 and top-3 ranks for a given dataset and classifier.
- On a given dataset, even the commonly used CS methods yield vastly different feature importance ranks, including the top-3 and the top-1 most important feature(s).

We then investigate why the agreement between the different CS methods and CA and CS methods remains weak, while the agreement between the computed feature importance ranks by CA methods is strong. Through a simulation study we find that, as hinted by prior studies [22], [23], [24], [25], [26], feature interactions present in the studied datasets impact the computed feature importance ranks of CS methods. We then investigate if removal of feature interaction in a given dataset (through a simple method like Correlation-based Feature Selection (CFS) [27], [28]) improves the agreement between the computed feature importance ranks of studied feature importance methods. We find that removal of feature interaction, significantly improves the agreement between the computed feature importance ranks of CA and CS methods. However, the improvement in agreement between the computed feature importance ranks of the studied CS methods remains marginal. In light of these findings, we suggest that future research on defect classification should:

1) Identify (e.g., using methods like Friedman's H-statistic (more details in Section 5.1)) and remove feature interactions present in the dataset (e.g., using simple methods like CFS) before interpreting the classifier, as they hinder the classifier interpretation.
2) One should always specify the used feature importance method to increase the reproducibility of their study and the generalizability of its insights.

*Paper Organization.* Section 2 presents the motivation of our study and the related work. Section 3 explains how we conducted our case study. In Section 4, we present the results of our case study which examines the extent to which the feature importance ranks computed by different feature importance methods vary. In Section 5, we investigate the impact of feature interactions on the computed feature importance ranks by studied interpretation methods. Section 6 presents the implications of our results and avenues for future research. Section 7 lists the threats to the validity of our study. Finally, Section 8 concludes our study.

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

## 2 Motivation and Related Work

In this section, we motivate our study based on how prior studies employed feature importance methods (Section 2.1). Next, we situate our study relative to prior related work (Section 2.2).

### 2.1 Motivation

We conduct a literature survey of the used feature importance methods in prior studies. To survey the literature, we searched Google Scholar with the terms "software engineering", "variable importance", "feature importance" and the name of each classifier that is studied in our paper (Section 3.4). We searched the Google Scholar multiple times, once for each studied classifier. We eliminated all the papers that were from before the year 2000 to restrict the scope of our survey to recent studies. We read each paper from the search results in order to check if they employed any feature importance method(s) to generate insights. We consider all the studies presented in the google scholar and do not filter based on venues. However, we do not include the papers in which one of the authors of this current study was involved to avoid potential confirmation bias. A summary of our literature survey is shown in Table 1.

We observe that studies rarely specify the reason for choosing their feature importance method – only four out of the 29 surveyed studies provide a rationale for choosing their used feature importance method.

We note that both CA and CS methods are widely used. For instance, from Table 1, we see that both Gini importance and filterVarImp have been used to interpret a random forest classifier. However, given that (i) feature importance methods are typically used to generate insights and (ii) different methods compute the feature importance using different approaches, such interchangeable usage of methods on a given classifier in prior studies is troublesome [58].

For instance, Zimmermann and Nagappan [31] used an F-Test on the coefficients of a logistic regression classifier (a CS method) to show that there exists a strong empirical relationship between Social Network Analysis (SNA) metrics and the defect proneness of a file. Later, Premraj and Herzig [40] used filterVarImp (a CA method) and logistic regression classifier to show that empirical relationship between SNA metrics and the defective files are negligible. Given that CS and CA methods can produce different feature importance ranks, it is unclear whether the aforementioned conflicting result is due to absence of an empirical relationship in the data or simply due to the differing feature importance methods and as such leads to conclusion instability.

More generally, the interchangeable use of feature importance methods (i.e., CS and CA methods), when replicating a study, is acceptable only if the computed ranks by different methods are not vastly different on a given dataset. Otherwise, it raises concerns about the stability of conclusions across the replicated studies. Hence, we investigate the following research question:

> (RQ1) For a given classifier, how much do the computed feature importance ranks by CA and CS methods differ across datasets?

Similar concerns exist regarding the interchangeable use of different CA methods, even for the same classifier. From Table 1, we observe that, within each classifier family, different studies use different CA methods. The rationale for choosing a given CA method (for instance, filterVarImp) over another (for instance, PDP) is rarely provided. For instance, none of the studies using a CA method in Table 1 provide reasons for choosing one CA method over another. Yet, the extent to which these CA agree with each other is unclear. Such a concern becomes particularly relevant with the recent rise of complex classifiers for defect prediction [7], [59], [60], as these classifiers do not have a universally agreed-upon or popular CS method. Hence, we investigate the following research question:

> (RQ2) For a given dataset and classifier, how much do the computed feature importance ranks by the different CA methods differ?

A number of general prior studies already note that feature importance ranks differ vastly between CS methods [61]. However, such a comparison among CS methods pertaining to different classifier (i.e., CS method associated with a decision tree classifier and a random forest classifier) has not been studied, in the context of defect prediction and software engineering. Such a study is extremely important to understand the limits of reproducibility and generalizability of prior studies. For instance, Jahanshahi *et al.* [12] replicate the study of McIntosh and Kamei [62] using random forest (and the CS methods of random forest classifier) as opposed to the non-linear logistic regression classifier and its associated CS method. Jahanshahi *et al.* [12] observe that their feature importance ranks differ from those of the original study. In particular, different CS methods are likely to compute feature importances differently and the difference in insight could be attributed to the used CS method rather than the underlying phenomena (e.g., just-in-time defect prediction) that is being studied. Therefore, we study the following research question along with the previous ones:

> (RQ3) On a given dataset, how much do the computed feature importance ranks by different CS methods differ?

### 2.2 Related Work

As summarized in Section 2.1, both CA and CS methods have been widely used by the software engineering researchers to compute feature importance ranks. In the following, we describe related work regarding (i) usage of feature importance methods in software engineering (ii) the problems associated with widely used feature importance methods and (iii) sensitivity of feature importance methods:

*Usage of Feature Importance Methods in Software Engineering*. Both CA and CS methods have been widely used by software engineering researchers to compute feature importance ranks. For instance, McIntosh *et al.*[63] and Morales *et al.* [34] construct regression models and use ANOVA (a

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

CS method) to understand which aspects of code review impact software quality. In turn, Fan *et al.* [52] use the CS methods that are associated with the random forest classifier to identify the features that distinguish between merged and abandoned code changes. Similarly, various CS methods that are associated with the random forest classifier have been used to identify features that are important for determining who will leave a company [64], who will become a long time contributor to an open source project [53], code metrics that signal defective code [11], popularity of a mobile app [65], likelihood of an issue being listed in software release notes [66]. Furthermore, CS methods that are associated with logistic regression and decision trees have also been used to generate insights on similar themes [39], [41], [67], [68], [69], [70]. Correspondingly, previous studies also use CA methods to interpret classifiers. For example, Tantithamthavorn *et al.* [18] and Rajbahadur *et al.* [17] use the permutation CA method to study the impact of data pre-processing on a classifier's feature importance ranks. Furthermore, Dey and Mockus [54] use partial dependence plots (PDP) to identify why certain metrics are not important for predicting the change popularity of an npm package, whereas Mori and Uchihira [13] use PDP to compute the feature importance of random forest classifiers. More recently, Jiarpakdee *et al.* [71] demonstrated how instance-level CA methods like LIME-HPO (Locally Interpretable Model-Agnostic Explanations with Hyperparameter Optimization) and Breakdown [72] can be used to identify features that are important in determining whether a given module will become defective. They further show that these instance-level feature importances mostly agree with the traditional feature importance ranks.

*Problems Associated With Widely Used Feature Importance Methods.* Prior studies investigated the potential problems and concerns regarding the widely used feature importance methods. Strobl *et al.* [61] find that CS methods associated with the widely used random forest classifier are biased and that different CS methods might yield different feature importance ranks when features are correlated. To deal with such correlations, Strobl *et al.* [73] propose a conditional permutation importance method. Similarly, Candes *et al.* [74] propose ways to mitigate to potential false discoveries caused by the feature importance method of generalized linear models. Lundberg *et al.* [26] and Gosiewska and Biecek [72] identify that several popular CA methods produce imprecise feature importance ranks.

*Sensitivity of Feature Importance Methods.* While the aforementioned studies focus on finding potential problems with existing methods, very few studies compare the existing and widely used feature importance methods. For instance, Grömping [75] compares the feature importance methods of linear regression and logistic regression and identifies both similarities and differences, whereas, Auret and Aldrich [76] compare several tree-based feature importance methods and find that CS method associated with conditional bagged inference trees to be robust.

Jiarpakdee *et al.* [8] analyze the impact of correlated features on the feature importance ranks of a defect classifier. They find that including correlated features when building a defect classifier results in generation of inconsistent feature importance ranks. In order to avoid that, Jiarpakdee *et al.* [8] recommend practitioners to remove all the correlated features before building a defect classifier. Through a different study, Jiarpakdee *et al.* [77] propose an automated method called AutoSpearman that helps practitioners to automatically remove correlated and redundant features from a dataset. They demonstrate that the AutoSpearman method helps one to avoid the harmful impact of these correlated metrics on the computed feature importance ranks of a defect classifier. Similarly, Tantithamthavorn *et al.* [78] find that noise introduced in the defect datasets due to the mislabelling of defective modules influences the computed feature importance ranks. They show that, among the features reported in the top-3 ranks, the noise introduced by mislabelling does not impact the feature reported at rank 1. However, it influences the features reported at rank 2 and rank 3 across several defect classifiers that they study.

Similarly, several prior studies from Tantithamthavorn *et al.* [18], [79], [80] investigate how various experimental design choices impact the computed feature importance ranks of a classifier. For instance, Tantithamthavorn *et al.* [18] investigated whether class rebalancing methods impact the computed feature importance ranks of a classifier. They observe that using class rebalancing methods can introduce *concept drift* in a defect dataset. This concept drift, in turn, impacts the computed feature importance ranks of a defect classifier constructed on the rebalanced defect dataset. Tantithamthavorn *et al.* [80] also investigate the impact of hyperparameter optimization on the computed feature importance ranks of a classifier. They find that the features reported at the top-3 features importance ranks differ significantly between the hyperparameter tuned classifiers and untuned classifiers. Rajbahadur *et al.* [17] investigate if the feature importance ranks computed for the regression-based random forest classifiers vary when computed by a CS method and Permutation CA method.

However, to the best of our knowledge, our study is the first work to empirically measure the agreement between the feature importance ranks computed by CA and CS methods across 18 datasets, six classifiers, and 8 feature importance methods, especially in the context of defect prediction. Our study enables the software engineering community to assess the impact of using various feature importance methods interchangeably. As it raises concerns about the stability of conclusions across replicated studies – as such conclusion instabilities might be due primarily to changes in the used feature importance methods instead of characteristics that are inherent in the domain.

## 3 Case Study Setup

In this section, we describe the studied datasets (Section 3.1), classifiers (Section 3.2), and feature importance methods (Section 3.3). We then describe our case study approach (Section 3.4), as well as the evaluation metrics that we employ (Section 3.5).

### 3.1 Studied Datasets

We use the software project datasets from the PROMISE repository [81]. The data set contains the defect data of 101 software projects that are diverse in nature. Use of such

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

varied software projects in our study helps us successfully mitigate the researcher bias identified by Shepperd *et al.* [82], [83]. In addition, similar to the prior studies by Rajbahadur *et al.* [17] and Tantithamthavorn *et al.* [18], we further filter the datasets to study based on two criteria. We remove the datasets with EPV less than 10 and the datasets with defective ratio less than 50. After filtering the 101 datasets from PROMISE with the aforementioned criteria, we end up with 18 datasets for our study: Poi-3.0, Camel-1.2, Xalan-2.5, Xalan-2.6, Eclipse34_debug, Eclipse34_swt, Pde, PC5, Mylyn, Eclipse-2.0, JM1, Eclipse-2.1, Prop-5, Prop-4, Prop-3, Eclipse-3.0, Prop-1, Prop-2. Table 5 in Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2021.3056941, shows various basic characteristics about each of the studied datasets.

### 3.2 Studied Classifiers

We construct classifiers to evaluate our outlined research questions from Section 2.1. We choose the classifiers based on two criteria. First, the classifiers should be representative of the eight commonly used machine learning families in Software Engineering literature as given by Ghotra *et al.* [84]. The goal behind this criterion is to foster the generalizability and applicability of our results. Second, the chosen classifiers should have a CS method. We only choose classifiers with a CS method, so that we can compare and evaluate the computed feature importance ranks by the CA methods against the CS feature importance ranks for a given classifier (RQ1). In addition, such a choice enables us to compare the agreement between the computed feature importance ranks between different classifiers (RQ3). After the application of these criteria, we eliminate three machine learning families (clustering based classifiers, support vector machines, and nearest neighbour), as the classifiers from those families do not have a CS method. Furthermore, we split the ensemble methods family given by Ghotra *et al.* [84] into two categories to include classifiers belonging to both bagging and boosting families. The classifiers we finally choose are: Regularized Logistic Regression (glmnet), C5.0 Rule-Based Tree (C5.0Rules), Neural Networks (with model averaging) (avNNet), Recursive partitioning and Regression Trees (rpart), Random Forest (rf), Extreme Gradient Boosting Trees (xgbTree). Table 6 in Appendix A.2, available in the online supplemental material, shows the studied classifiers, their hyperparameters and machine learning families to which they belong.

We choose one representative classifier from each of the machine learning family from the *caret*[1] package in R. Table 6, available in the online supplemental material, also shows the caret function that was used to build the classifiers. The selected classifiers have a CS method, that is given by the *varImp()* function in the caret package.

Inherently interpretable classifiers (e.g., fast-and-frugal trees, naive-bayes classifiers and simple decision trees) do not benefit as much from feature importance methods. Hence, such classifiers are out of the scope of this study. Nevertheless, we strongly suggest that the future studies should also explore the reliability of the insights that is derived from simple interpretable classifiers.

### 3.3 Studied Feature Importance Methods

#### 3.3.1 Classifier Specific Feature Importance (CS) methods

The CS methods typically make use of a given classifier's internals to compute the feature importance scores. These methods are widely used in software engineering to compute feature importance ranks as evidenced from Table 1.

We use six CS methods that are associated with the six classifiers that we study, namely: Logistic Regression FI (LRFI), C5.0 Rule-Based Tree FI (CRFI), Neural Networks (with model averaging) FI (NNFI), Recursive Partitioning and Regression Trees FI (RFI), Random Forest FI (RFFI), and Extreme Gradient Boosting Trees FI (XGFI). Table 7 in Appendix A.3, available in the online supplemental material, provides a brief explanation about the inner working of these CS methods on a given classifier. For a more detailed explanation we refer the readers to Kuhn [85].

#### 3.3.2 Classifier Agnostic Feature Importance (CA) Methods

CA methods compute the importance of a feature by treating the classifier as a "black-box", i.e., without using any classifier-specific details. In this study, we use the permutation feature importance (Permutation) and SHapley Additive exPlanation (SHAP) CA methods. We use these two CA methods instead of others for the following reasons. First, Permutation is one of the oldest and most popularly used CA methods in both machine learning and software engineering communities [17], [18], [86], [87], [88], [89], [90]. It was first introduced by Breiman [91] as way of measuring the feature importance ranks of a random forest classifier and later was adopted as a CA method. Second, we consider SHAP, as it one of the more recent global feature importance method that is theoretically guaranteed to produce optimal feature importance ranks (more details are given below) [92]. Though SHAP was proposed by Lundberg and Lee [93] only in 2017, it has already garnered over 2,000 citations. Furthermore, SHAP is being increasingly adopted in the software engineering community to compute feature importance ranks, as evidenced by its usage in recent studies [94], [95]. Finally, both of these feature importance methods do not require any hyperparameter optimization unlike the CA techniques used by Jiarpakdee *et al.* [71] and Peng and Menzies [96]. Appendix A.4, available in the online supplemental material, describes Permutation and SHAP in more detail.

### 3.4 Approach

Fig. 1 provides an overview of our case study approach. We use this approach to answer all of our aforementioned research questions in Section 2.2.

#### 3.4.1 Data Pre-Processing

*Correlation and Redundancy Analysis.* We perform correlation and redundancy analysis on the independent features of the studied defect datasets, since the presence of correlated or redundant features impacts the interpretation of a classifier

1. https://cran.r-project.org/web/packages/caret/index.html

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

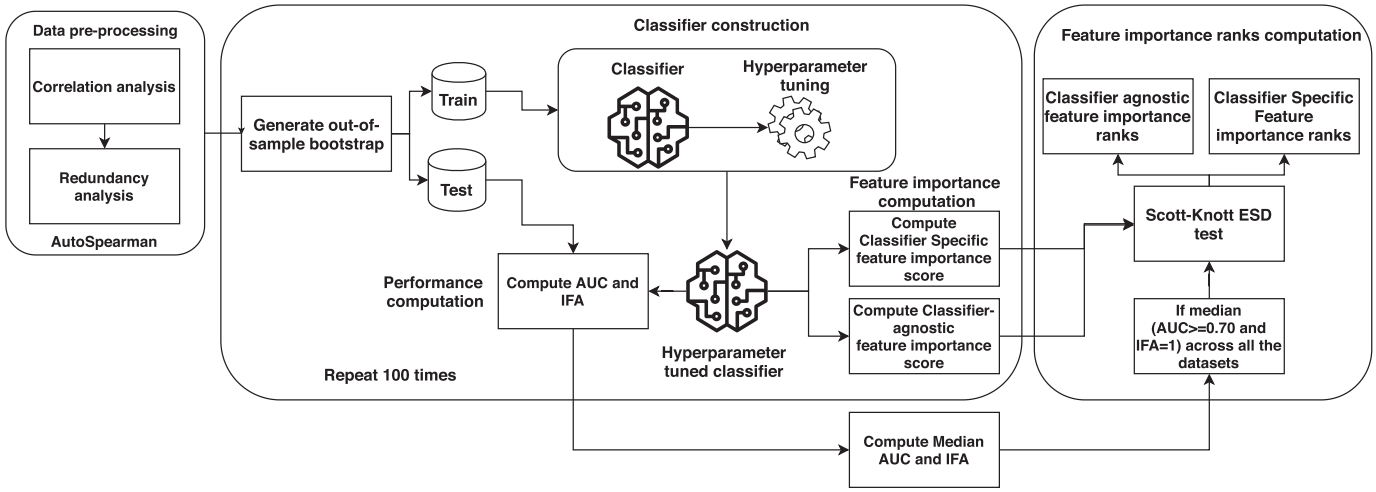


Fig. 1. Overview of our case study approach.

and yields unstable feature importance ranks [8], [97], [98]. Similar to Jiarpakdee *et al.* [71] we employ the AutoSpearman technique [77] using `AutoSpearman` function from the `Rnalytica` R package to remove the correlated and redundant features from our studied datasets.

### 3.4.2 Classifier Construction

*Out-of-Sample Bootstrap.* To ensure the statistical validity and robustness of our findings, we use an out-of-sample bootstrap method with 100 repetitions to construct the classifiers [79], [99]. More specifically, for each studied dataset, every classifier is trained 100 times on the 100 resampled *train* sets, then these classifiers are used for computing the 100 feature importance scores. The performance of these trained classifiers are also evaluated on the 100 out-of-sample *test* sets. Appendix A.5, available in the online supplemental material, describes the out-of-sample bootstrap method in more detail.

*Classifier Construction With Hyperparameter Tuning.* Several prior studies [18], [100] show that hyperparameter tuning is pivotal to ensure that the trained classifiers fit the data well. Furthermore, Tantithamthavorn *et al.* [18] show that feature importance ranks shift between hyperparameter tuned and untuned classifiers. Therefore, we tune the hyperparameters for each of the studied classifiers using random search [101] in every bootstrap iteration using `caret` R package [102]. In every iteration, we pass the *train* set, classifier and the associated hyperparameters outlined in Table 6, available in the online supplemental material, to `train` function of the `caret` package. similar to Jiarpakdee *et al.* [71], we then use the automated hyperparameter optimization option provided by train function to conduct a random search (with 10-fold cross-validation) for the optimal hyperparameters that maximizes AUC measure. Once the optimal hyperparameters that maximizes the AUC measure for the given classifier are found, we use these hyperparameters to construct the classifier on the train set. Finally, we use these hyperparameter tuned classifiers to conduct the rest of our study.

### 3.4.3 Performance Computation

Similar to a recent study by Jiarpakdee *et al.* [71], we compute the AUC (Area Under the Receiver Operator Characteristic Curve) and the IFA (Initial False Alarm) to measure the performance of our classifiers. We choose these two performance measures in particular over other performance measures for the following reasons. First, several prior studies recommend the use of the AUC measure over other performance measures to quantify the discriminative capability of a classifier [84], [97], [103], [104]. Second, as Parnin and Orso [105] and Huang *et al.* [106] argue, the IFA of a classifier being low is extremely important for a classifier to be adopted in practice. For these reasons, we choose the AUC and IFA measures to evaluate the performance of our classifiers. In Appendix A.6, available in the online supplemental material, we describe provide more details about AUC and IFA, including how they are calculated.

### 3.4.4 Computation of Feature Importance Scores

We use both the CS and CA methods to computer feature importance scores, as detailed in Section 3.3, for all the studied classifiers in each bootstrap iteration. For CA methods: we use the `vip` package and the method outlined by Rajbahadur *et al.* [17] to compute the PDP and Permutation CA methods feature importance scores respectively. For the CS computation, we use the `VarImp()` function of the `caret` R package [85].

### 3.4.5 Computation of Feature Importance Ranks

We use the Scott-Knott Effect Size Difference (SK-ESD) test (v2.0) [107] to compute the feature importance ranks from the feature importance scores computed in the previous step, as done by prior studies [8], [104]. For each dataset and studied classifier, three feature importance scores are computed (one CS score and two CA scores) for each bootstrap iteration. The SK-ESD test is applied on these scores to compute three feature importance rank lists (one CS rank list and two CA rank lists) for all the 6 studied classifiers on each dataset. The process of feature importance rank computation from the feature importance scores is depicted in the right-hand side of Fig. 1. Also, we note that we only compute the feature importance ranks for classifiers that simultaneously have a median AUC greater than 0.7 and a median IFA = 1. We do so, as Chen *et al.* [7] and Lipton [108] argue, a classifier should have a good operational

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

performance for the computed feature importance ranks to be trusted. Due to this constraint, we discarded the classifier C5.0Rules from our studied classifiers.

### 3.5 Evaluation Metrics

We measure the difference between the different feature importance rank lists by measuring how much they agree with each other.

*Kendall's Tau Coefficient ($\tau$)* [109] is a widely used non-parametric rank correlation statistic that is used to compute the similarity between *two* rank lists [110], [111]. Kendall's $\tau$ ranges between -1 to 1, where -1 indicates a perfect disagreement and 1 indicates a perfect agreement. We use the interpretation scheme suggested by Akoglu [112]:

$$\text{Kendall's } \tau \text{ Agreement} = \begin{cases} \text{weak}, & \text{if } |\tau| \leq 0.3 \\ \text{moderate}, & \text{if } 0.3 < |\tau| \leq 0.6 \\ \text{strong} & \text{if } |\tau| > 0.6 \end{cases}.$$

*Kendall's W coefficient* [109] is typically used to measure the extent of agreement among *multiple* rank lists given by different raters (CS methods in our case and raters $\geq 2$). The Kendall's W ranges between 0 to 1, where 1 indicates that all classifiers agree perfectly with each other and 0 indicates perfect disagreement. We use the Kendall's W in RQ3 to estimate extent to which the different feature importance ranks that are computed by CS methods agree across all the studied classifiers for a given dataset. We use the same interpretation scheme for Kendall's W as we use for Kendall's Tau.

*Top-3 overlap* is a simple metric that computes the amount of overlap that exists between features at the top-3 ranks in relation to the total number of features at the top-3 ranks across n feature importance rank lists. This metric does not consider the ordinality of the features in the top-3 ranks, i.e., the order in which a given feature appears in the top-3 ranks. Rather, it only checks if a given feature appeared in all of the top-3 rank lists. Top-3 overlap is adapted from the popular Jaccard Index [113] for measuring similarity. We compute the top-3 overlap among $n$ feature importance lists with the equation 1 ($k = 3$), where $list_i$ is the $i$th feature list and $n$ is the total number of lists of features to compare (for in RQ1 and RQ2, $n = 2$, whereas in RQ3, $n = 5$).

$$Top-k\ overlap = \frac{\bigcap_{i\geq 2}^{n} Features\ at\ top\ k\ ranks\ of\ list_i}{\bigcup_{i\geq 2}^{n} Features\ at\ top\ k\ ranks\ of\ list_i}. \quad (1)$$

We define the interpretation scheme for Top 3 overlap as follows, which aims to enable easier interpretation of the results:

$$\text{Top-3 Agreement} = \begin{cases} \text{negligible}, & \text{if } 0.00 \leq \text{top-3 overlap} \leq 0.25 \\ \text{small}, & \text{if } 0.25 < \text{top-3 overlap} \leq 0.50 \\ \text{medium}, & \text{if } 0.50 < \text{top-3 overlap} \leq 0.75 \\ \text{large} & \text{if } 0.75 < \text{top-3 overlap} \leq 1.00 \end{cases}$$

For example, assume that the top-3 features for CS and CA on a given dataset and classifier are $Imp_{CS}(Top\ 3) = \{cbo, loc, pre\}$ and $Imp_{CA}(Top\ 3) = \{loc, lcom3, dit\}$ respectively. Then the top-3 overlap corresponds to $1/5 = 0.2$ ( as $n = 2, k = 3$).

*Top-1 overlap* is analogous to the Top-3 overlap metric (Equation (1), with $k = 1$). We define the interpretation scheme for Top-1 overlap as follows: if top-1 overlap is $\leq 0.5$ then agreement is low, otherwise agreement is deemed high.

## 4 Case Study Results

In this section, we detail the results of our case study with regards to our research questions from Section 2.

### 4.1 (RQ1) For a given classifier, how much do the computed feature importance ranks by CA and CS methods differ across datasets?

*Approach*. For each of the five constructed classifiers with (median AUC $> 0.7$ and median IFA $\leq 1$ across the studied datasets), we compare the feature importance ranks that are computed by the CA and CS methods across all the studied datasets. For each classifier, on a given dataset, we compare the feature importance ranks computed by SHAP and Permutation CA methods with the feature importance ranks that are computed by the studied CS method of a classifier. We quantify the agreement between the two rank lists in terms of Top-1 overlap, Top-3 overlap and Kendall's Tau. We compute the Top-1 and Top-3 overlap in addition to the Kendall's Tau because some of the prior work primarily examines the top $x$ important features [1], [7], [114]. Finally, we aggregate the comparisons with respect to each classifier across the studied datasets.

For instance, for the avNNet classifier, we first compare the feature importance ranks that are computed by SHAP (CA method) with those that are computed by the CS method of avNNet (i.e. NNFI, see Table 7, available in the online supplemental material) on the `eclipse-2.0` dataset. Next, we determine the agreement between the two lists according to Top-1, Top-3 overlap, and Kendall's Tau. We then repeat this step for every dataset and plot the distribution for each agreement metric. An analogous process is followed in order to compare the Permutation method with the NNFI method.

The goal of this RQ is to determine the extent to which the feature importance ranks that are computed by CA methods differ from the more widely used and accepted CS methods for each classifier. If the studied CA methods consistently have a high agreement with the CS methods for each classifier and across all the studied datasets, then one can use both CS methods and CA methods interchangeably.

*Results: Result 1) The SHAP and Permutation CA methods have a low median top-1 overlap with the CS methods on two of the five studied classifiers.* The leftmost lane in Fig. 2 shows the top-1 overlap between the feature importance rank lists that are computed by the CS and CA methods for each classifier and across all the studied datasets. We observe that the median top-1 overlap between the studied CA methods and the CS method of a classifier is low for two classifiers, namely rpart and avNNet. In other words, even the most important feature varies between the rankings that are computed by the CA and CS methods for two of the studied classifiers.

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

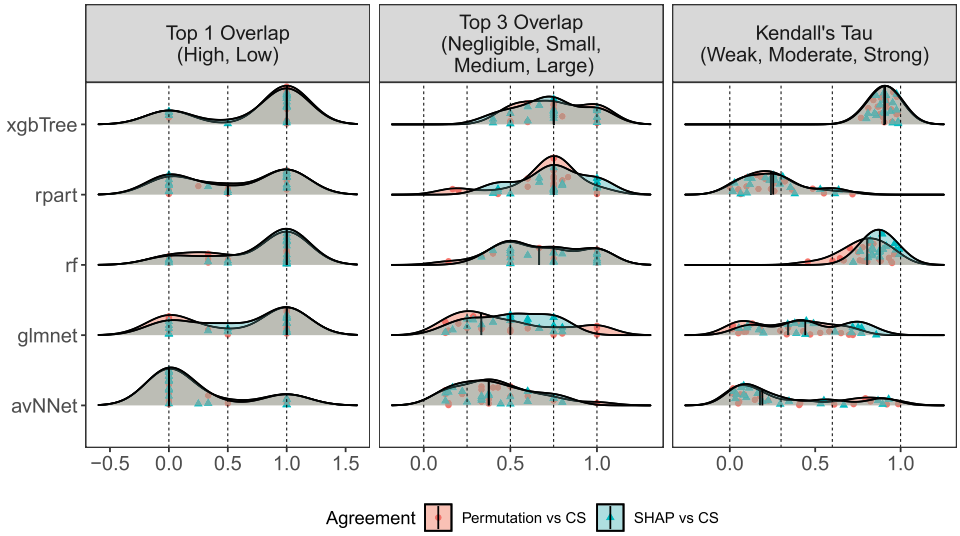


Fig. 2. A density plot of Top-1 Overlap, Top-3 Overlap, and Kendall's Tau between CA and CS methods for each classifier across the studied datasets. The circles and triangles correspond to individual observations. The dotted lines correspond to the metric-specific interpretation scheme outlined in Section 3.5. The vertical lines inside the density plots correspond to the median of the distributions.

*Result 2) Both CA methods have a small median top-3 overlap with CS methods on two of the five studied classifiers.* We see from the middle lane of Fig. 2 that the features that are reported at the top-3 ranks by both the SHAP and Permutation method do not exhibit a large overlap with the feature importance ranks that are computed by the CS method for *any* of the studied classifiers. Furthermore, from Fig. 2, we observe that even on cases where the median overlap is medium, the spread of the density plot is also large (i.e., several datasets exhibit small and even negligible top-3 overlap).

*Result 3) For three out of the five studied classifiers, the Kendall's Tau agreement between CA and CS methods is only moderate at best.* Kendall's Tau values between the feature importance ranks that are computed by CA and CS methods for each classifier and across all the studied datasets are depicted as density distributions in Fig. 2. For both rpart and glmnet, the median Kendall's Tau agreement is weak. Even for avNNet, where the median agreement between the CA and CS methods is moderate, the spread of the density lot is large with several datasets exhibiting negligible agreement. We observe that the median Kendall's Tau indicates a strong agreement for only for two of the six studied classifiers, namely xgbTree and rf (note the vertical bars inside the density plots in the right-most lane).

In summary, the CA and CS methods do not always exhibit strong agreement for the computed feature importance ranks across the studied classifiers. Therefore, we discourage the interchangeable use of CA and CS methods in general. In particular, we suggest that unless agreement between the CA and CS methods can be improved (we present a potential solution in Section 5.2), whenever possible, future defect prediction studies should preferably choose the same feature importance method when replicating or seeking to validate a prior study.

> The computed feature importance ranks by CA and CS methods do not always strongly agree with each other. For two of the five studied classifiers, even the most important feature varies across CA and CS methods.

### 4.2 (RQ2) For a given dataset and classifier, how much do the computed feature importance ranks by the different CA methods differ?

*Approach.* For each of the studied datasets, we check the extent to which the feature importance ranks computed by SHAP and Permutation CA methods agree with each other for all the five studied classifiers. Similarly to the previous RQ, in order to quantify agreement, we compute the Top-1 Overlap, Top-3 Overlap, and Kendall's Tau measures.

*Result 4) SHAP and Permutation methods have a high median top-1 overlap and a strong agreement (in terms of Kendall's Tau) across all the studied datasets.* Furthermore, SHAP and Permutation CA methods do not have a small or negligible overlap across any of the studied datasets. Except on Prop-4 and PC5 datasets, the feature importance ranks computed by SHAP and Permutation CA methods have a large median Top-3 overlap. Furthermore, from the rightmost lane in Fig. 3 we observe that except for the rf classifier on prop-4 dataset, the Kendall's Tau agreement values are consistently strong for all the studied classifiers.

> The computed feature importance ranks by the studied CA methods exhibit a strong agreement including the features reported at top-1 and top-3 ranks for a given dataset and classifier.

### 4.3 (RQ3) On a given dataset, how much do the computed feature importance ranks by different CS methods differ?

*Approach.* For each of the datasets, we obtain the computed feature importance ranks by the studied CS methods of each of the five studied classifiers. We then calculate the Kendall's W between the six feature importance rank lists that are computed by the studied CS method of each classifier. Unlike the previous RQs, we compute the Kendall's W instead of Kendall's Tau, as Kendall's W is able to measure agreement among multiple feature importance rank lists

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

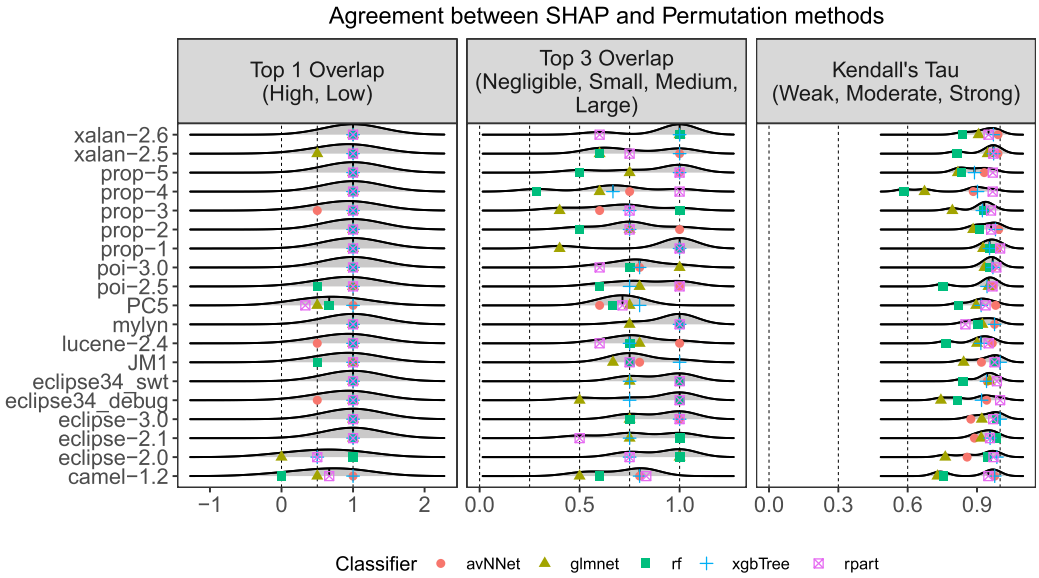


Fig. 3. A density plot of Top-1 Overlap, Top-3 Overlap, and Kendall's Tau values between the SHAP and Permutation CA methods for each of the studied classifiers across all the studied datasets. The dotted lines correspond to the metric-specific interpretation scheme outlined in Section 3.5. The vertical lines inside the density plots correspond to the median of the distributions. Please note the empty space in the rightmost figure is due to the observed range of Kendall's Tau values in this case varying between 0.5 and 1. In other words, across all the studied datasets the computed feature importance ranks of SHAP and Permutation at least had a moderate agreement.

TABLE 2
Top-1 Overlap, Top-3 Overlap, and Kendall's W Among the Computed Feature Importance Ranks by the CS Method of Each Classifier

| Dataset | Top-1 overlap | Top-3 overlap | Kendall's W |
|---|---|---|---|
| poi-3.0 | Low (0) | Negligible (0) | Weak (0.13) |
| camel-1.2 | Low (0) | Negligible (0) | Weak (0.22) |
| xalan-2.5 | Low (0) | Negligible (0.10) | Weak (0.18) |
| xalan-2.6 | Low (0) | Negligible (0.16) | Weak (0.25) |
| eclipse34_debug | Low (0) | Negligible (0.14) | Weak (0.26) |
| eclipse34_swt | Low (0) | small (0.33) | **Strong (0.62)** |
| pde | Low (0) | **Small (0.4)** | **Strong (0.70)** |
| PC5 | Low (0) | Negligible (0) | Weak (0.18) |
| mylyn | Low (0) | **Small (0.29)** | Weak (0.16) |
| eclipse-2.0 | Low (0) | Negligible (0.17) | Weak (0.26) |
| JM1 | Low (0) | Negligible (0.20) | **Moderate (0.31)** |
| eclipse-2.1 | Low (0) | Negligible (0.17) | **Moderate (0.31)** |
| prop-5 | Low (0) | Negligible (0) | Weak (0.17) |
| prop-4 | Low (0) | Negligible (0.14) | **Moderate (0.37)** |
| prop-3 | Low (0) | Negligible (0.09) | **Moderate (0.32)** |
| eclipse-3.0 | Low (0) | **Small (0.28)** | **Moderate (0.32)** |
| prop-1 | Low (0) | Negligible (0.16) | **Moderate (0.36)** |
| prop-2 | Low (0) | Negligible (0) | Weak (0.28) |

*Best results for each metric are shown in bold.*

(Section 3.5). Furthermore, we also calculate the Top-3 overlap among all the five feature importance rank lists. We do so for all the studied datasets. A high Kendall's W and a high Top-3 overlap across all the studied datasets among the constructed classifiers would indicate high agreement between the computed feature importance ranks by different classifiers and the studied CS method of each classifier.

*Results: Result 5) The computed feature importance ranks by different CS methods vary extensively.* None of CS methods agree on the most important feature (as evidenced by the results presented in Table 2). Furthermore, the maximum top-3 overlap is only small and it happens for only three out of 18 datasets. Finally, we also observe that, on a given dataset, only on two occasions the feature importance rank lists computed by the different CS methods strongly agree with each other. We summarize the Top 1 overlap, the Top-3 overlap and Kendall's W among the computed feature importance ranks for all the six studied classifiers across the studied datasets in Table 2.

From Table 2, we observe that both the Kendall's W and top-3 overlap among the feature importance ranks that are computed by studied CS methods associated with each of the classifier – which are widely used in the software engineering community– is very low. Such a small Top-3 overlap and Top-1 overlap for all the datasets indicates that computed feature importance ranks by CS methods differ substantially among themselves. Hence, different classifiers and their associated CS methods cannot be used interchangeably.

> On a given dataset, even the commonly used CS methods yield vastly different feature importance ranks, including the top-3 and the top-1 most important feature(s).

## 5 Discussion

### 5.1 Why do Different Feature Importance Methods Produce Different Top-3 Features on a Given Dataset?

*Motivation.* From the results presented for RQ1 (Section 4.1) we observe that, on a given dataset and classifier the CA methods and CS methods produce different feature importance ranks (including the top-3 ones). Similarly, from the results presented in RQ3 (Section 4.3), on a given dataset, even the widely used CS methods produce vastly different feature importance ranks. Such a result is in spite of us having removed the correlated and redundant features from the datasets in a pre-processing step using a state-of-the-art technique like AutoSpearman [77]. However, in contrast to the results presented in RQ1 and RQ3, in RQ2 (Section 4.2), we observe that the studied CA methods (i.e., Permutation

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

and SHAP methods) produce similar feature importance ranks on a given dataset and classifier.

We hypothesize that different CS methods produce different top-3 feature importance ranks even on the same dataset and classifier (and when compared to the feature importance ranks computed by the CA methods) because of feature interactions that are present in the studied datasets. Feature interactions can be defined as a phenomenon where the effect of the independent features on a dependent feature is not purely additive [15], [23]). We arrive at such a hypothesis as many of the prior studies show that the presence of feature interactions in a given dataset can affect the different feature importance methods differently and make them assign different feature importance ranks to features [22], [23], [24], [25], [26]. Therefore, in this section, we seek to find out if different feature importance methods (in addition to being inherently different) yield different feature importance ranks due to the presence of feature interactions in the dataset.

We further note that computed feature importance ranks by the CA methods exhibit a high agreement and overlap as both Permutation and SHAP are typically not impacted by the feature interactions present in a dataset [15].

*Approach.* To test our hypothesis and detect if any of the features presented in a given dataset interact with other features in that dataset, we compute the *Friedman H-Statistic* [115] for each feature against all other features. The Friedman H-statistic works as follows. First, a classifier (any classifier - we use random forest as it captures interactions well [116]) is constructed using the given dataset. For instance, consider the Eclipse-2.0 dataset and that we wish to compute the Friedman H-Statistic between the feature `pre` and all the other features. We first compute the partial dependence between `pre` and the dependent variable with respect to the random forest classifier (`PD_pre`) for all the data points. Following which, partial dependence between all the features (as a single block) in Eclipse-2.0 (except `pre`) is computed (`PD_rest`) for all the data points. If there is no feature interaction between `pre` and the other features in Eclipse-2.0, the outcome probability of the constructed classifier can be expressed as a sum of `PD_pre` and `PD_rest`. Therefore, the difference between the outcome probability of the classifier and the sum of `PD_pre` and `PD_rest` is computed and the variance of this difference is reported as the Friedman H-Statistic. We compute the Friedman H-Statistic for all the studied datasets 10 times, as Friedman H-Statistic is known to exhibit fluctuations because of the internal data sampling [15]. We then consider the median score of the Friedman H-Statistic for each dataset. We use the R package `iml`[2] to compute the Friedman H-Statistic.

The Friedman H-Statistic is a numeric score that ranges between 0 and 1. However, it can sometimes exceed 1 when the higher order interactions are stronger than the individual features [15]). A Friedman H-Statistic of 0 or closer to 0 indicates that no interaction exists between the given feature and the rest of the features and a Friedman H-Statistic of 1 indicates extremely high levels of interaction. For a more theoretical and detailed explanation we refer the readers to [15], [115]. In this study, we consider a feature to exhibit interactions with other features if the Friedman H-Statistic is $\geq 0.3$. We choose 0.3 as a cut-off, to only indicate the existence of a feature interaction, but not to qualify the strength of the interaction, because the presence of feature interactions irrespective of the strength could potentially impact feature importance ranks [22], [23], [24], [25], [26]. In addition we also report the results for the number of features that exhibit a Friedman H-statistic $\geq 0.5$. We choose to report the results on multiple thresholds to present a comprehensive depiction of the feature interactions. Furthermore, there is no established guideline in the literature regarding how one should interpret the Friedman H-statistic or how thresholds should be selected.

*Construction of a Synthetic Dataset Without Any Feature Interactions.* Next, to determine if the absence of feature interactions enables the different CS methods to compute the same top-3 features, we simulate a dataset with no feature interactions. We do so instead of using a real dataset as it is difficult to find a real-world defect dataset without any feature interactions. We generate a dataset with 1,500 data points and 11 independent features of which five features carry the signal: `signal = {x1,x2,x3,x4,x5}` and six features are just noise i.e., does not exhibit any relationship to the dependent feature `noise = {n1, n2, n3, n4, n5, n6}`. We add noise features to make our simulated dataset similar to that of a real-world defect dataset. All the signal features and `n1, n5, n6` are generated by randomly sampling the normal distribution with mean = 0 and standard deviation = 1. Similarly, we sample the uniform distribution between 0 and 1 to generate the values for `n2, n3, n4`. We use both the normal and the uniform distributions for generating the noise features to ensure the presence of different types of noise in our simulated dataset. Next, to construct our dependent feature for the dataset, we construct $y_{signal}$ with the signal variables as given in Equation (2). We assign different weights to the different signal features when constructing the $y_{signal}$ to ensure that we know the true importance of each of the signal features. We then convert the $y_{signal}$ in to a probability vector $y_{prob}$ with a sigmoid function as given in Equation (3). Finally, we generate the dependant feature by sampling the binomial distribution to generate the dependent feature $y_{dependant}$ with $y_{prob}$ as given in Equation (4).

$$y_{signal} = 20x1 + 10x2 + 5x3 + 2.5x4 + 0.5x5 \tag{2}$$

$$y_{prob} = \frac{1}{1 + e^{-y_{signal}}} \tag{3}$$

$$y_{dependant} = Binomial(1500, y_{prob}) \tag{4}$$

*Construction of Classifiers and Calculation of Top-1 and Top-3 Overlaps.* We then construct all of the studied classifiers on the simulated dataset with $y_{dependent}$ as the dependent feature. We construct all the classifiers with 100-Out-of-sample bootstrap on the simulated dataset and compute the feature importance ranks computed by the CA and CS methods as outlined in Section 3.4. For each of the studied classifiers, we calculate the top-1 and top-3 overlap between the feature importance ranks computed by the CA and CS methods

2. https://cran.r-project.org/web/packages/iml/index.html

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

(similarly to RQ1). Furthermore, we also calculate the top-1 and top-3 overlaps between feature importance ranks computed by the studied CS methods (similarly to RQ3). We then check if they exhibit a top-1 and top-3 overlap close to 1 for all the classifiers between the computed feature importance ranks of the CS and the CA methods. In addition we also check the top-1 and top-3 overlap among the computed feature importance ranks of the various CS methods. If in both the cases they exhibit a top-1 and top-3 overlap close to 1, we can then assert that the feature interactions in the dataset affects the top-3 features computed by the different CS methods and vice versa.

*Determining Whether Feature Interactions Impact interpretation.* To verify that the feature interactions present in the dataset impact only the studied CS methods and not the studied CA methods, we compute the feature importance ranks for all the studied classifiers with the studied CS and CA methods on a simulated dataset with feature interactions. To simulate a dataset with interactions, we take the simulated dataset from earlier (the one without any interactions) and introduce interactions by modifying the Equation (2). We modify the $y_{signal_with_interactions}$ to depend on hidden interactions as given by Equation (5). The rest of the data generation process remains the same as earlier i.e., the independent features are still given by `independent features = {signal,noise}` where `signal = {x1,x2,x3,x4,x5}` and `noise = {n1, n2, n3, n4, n5, n6}`. Finally, $y_{dependant_with_interactions}$ is generated using Equations (3) and (4).

$$y_{signal_with_interactions} = y_signal + x1 * x3 + x2 * x3 + x2 * x1. \tag{5}$$

We then construct all of the studied classifiers with $y_{dependent_with_interactions}$ on the simulated dataset with interactions using 100-Out-of-sample bootstrap. Next, we compute the feature importance ranks using the studied CA and CS methods. Finally, we calculate the top-1 and top-3 overlap between the computed feature importance ranks of CA and CS methods, respectively (similarly to RQ2 and RQ3). If feature interactions do not impact the CA methods, we should observe high top-1 and top-3 overlaps (close to 1) between the computed feature importance ranks of the CA methods and vice versa. Similarly, if the feature interactions impact the computed feature importance ranks of CS methods, we should observe low top-1 and top-3 overlaps between the computed feature importance ranks of the different CS methods and vice versa.

*Results: Result 6) At least two features and as many as eight features interact with the rest of the features in all of the 18 studied datasets (i.e., Friedman H-Statistic $\geq 0.3$). Furthermore, we find that 14 of the 18 datasets have at least two features with a Friedman H-Statistic $\geq 0.5$.* We present the number of features in each dataset with a Friedman H-statistic $\geq 0.3$ and $\geq 0.5$ in Table 8 (Appendix B, available in the online supplemental material). From Table 8, available in the online supplemental material, we observe that all datasets contain more than two features that interact with the other features. Though Friedman H-Statistic only computes if a given feature interacts with the rest of the features and excludes other feature interactions like second-order interactions, pairwise interactions, and higher-order interactions, it gives us a hint as to the presence or absence of feature interactions in a dataset.

*Result 7) The top-3 and the top-1 overlap between the feature importance ranks computed by the CS and CA methods on each of the classifiers is 1 on our simulated dataset devoid of feature interactions.* In addition, the top-3 and top-1 overlap between the computed feature importance ranks of the CS methods on the simulated dataset without interaction is 1. Such a result indicates that in the dataset without feature interactions, all the studied feature importance methods identify the same top-3 features. Furthermore, we also observe that all the studied important features are identified x1, x2, x3 as the top-3 features in the same order of importance. Thus, we assert that the different feature importance methods are able to assign the feature importance ranks correctly when independent features' contribution to the dependent feature is additive without any interactions.

*Result 8) The top-3 and the top-1 overlap between the feature importance ranks computed by the studied CA methods is 1 on the simulated dataset with interactions.* In turn, we find that the top-1 and top-3 overlap between the feature importance ranks computed by the different CS methods is 0 on the simulated dataset with interactions. Such a result indicates that studied CA methods (i.e., SHAP and Permutation) are not impacted by the feature interactions in the simulated dataset. However, the computed feature importance ranks of the studied CS methods are heavily influenced by the presence of feature interactions in the dataset.

Hence, we conclude that the presence of feature interactions in the studied defect datasets could be the reason why different CS methods produce a different top-3 set of features. In addition, we conclude that alongside the fact that different feature importance methods compute feature importances differently, feature interactions in the datasets can also be a key confounder that affects the computed feature importance ranks by the studied feature importance methods.

### 5.2 Can we Mitigate the Impact of Feature Interactions?

*Motivation.* From the results presented in Section 5.1 we observe that feature interactions impact feature importance ranks computed by the CS methods. Such a result indicates that CS methods cannot be interchangeably used. Though comprehensively identifying and removing all the feature interactions from a dataset is still an open area of research, there exist several simple methods like Correlation-based Feature Selection (CFS) [27], wrapper methods [27] that allows us to remove lower order feature interactions [117], [118], [119].

Therefore, in this discussion, we investigate if removing the feature interactions present in a dataset through a simple method like CFS would increase the agreement between the computed feature importance ranks of the studied CS methods on a given dataset. We also investigate if removing feature interactions yields an improved agreement between the computed feature importance ranks of CA and CS methods. Finally we also study if the removal of feature interactions has any impact on the agreement between the feature

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
The Median Top-1 and Top-3 Overlap Improvements Upon Removal of Feature Interactions (FI) Between the Computed Feature Importance Ranks of the Studied CA and CS Methods

| Classifier | Top-1 Overlap | | Top-3 Overlap | |
|---|---|---|---|---|
| | RQ1 results | After removal of FI | RQ1 results | After removal of FI |
| xgbTree | High | High | Medium | **Strong** |
| rpart | Low | **High** | Medium | **Strong** |
| rf | High | High | Medium | **Strong** |
| glmnet | High | High | Small | **Moderate** |
| avNNet | Low | Low | Small | **Moderate** |

TABLE 4
The Median Top-1 and Top-3 Overlap Improvements Upon Removal of Feature Interactions (FI) Between the Computed Feature Importance Ranks of the Studied CS Methods

| Dataset | Top-1 Overlap | | Top-3 Overlap | |
|---|---|---|---|---|
| | RQ3 results | After removal of FI | RQ3 results | After removal of FI |
| poi-3.0 | Low | Low | Negligible | Negligible |
| camel-1.2 | Low | Low | Negligible | **Small** |
| xalan-2.5 | Low | Low | Negligible | **Medium** |
| xalan-2.6 | Low | **High** | Small | Small |
| eclipse34_debug | Low | **High** | Small | Small |
| eclipse3_swt | Low | Low | Negligible | **Small** |
| pde | Low | Low | Small | Small |
| PC5 | Low | Low | Negligible | Negligible |
| mylyn | Low | Low | Small | Small |
| eclipse-2.0 | Low | Low | Negligible | Negligible |
| JM1 | Low | Low | Negligible | **Small** |
| eclipse-2.1 | Low | Low | Negligible | **Small** |
| prop-5 | Low | **High** | Negligible | Negligible |
| prop-4 | Low | **High** | Negligible | **Small** |
| prop-3 | Low | Low | Negligible | Negligible |
| eclipse-3.0 | Low | Low | Small | Small |
| prop-1 | Low | Low | Negligible | **Small** |
| prop-2 | Low | Low | Negligible | **Small** |

importance ranks computed by the studied CA methods. If it does, then we can recommend researchers and practitioners to remove the feature interactions in their datasets using CFS prior to building the machine learning classifiers.

*Approach.* We use the CFS method to remove the feature interactions across the studied datasets. We use the CFS method in lieu of other methods like Wrapper in the dataset for the following reasons. First, several prior studies show that, in addition to eliminating correlation between the features, CFS method is also useful for mitigating feature interactions [27], [28]. Second, CFS method has been widely used in the software engineering community (though for removing correlated features) [71]. Third, it is simple to implement and is extremely fast. Finally, since its a filter type method, it does not make any assumptions about the dataset or the subsequent classifier that is to be built.

Therefore, for each of the studied datasets, following the removal of correlated and redundant features using AutoSpearman, we apply the CFS [27] method to remove the feature interactions. The CFS method chooses a subset of features that exhibits the strongest relationship with the dependent feature while exhibiting a minimal correlation among themselves. It is important to note that we apply the CFS method to the features that were not flagged as correlated/redundant by AutoSpearman technique (as opposed to applying CFS method by itself to eliminate both the observed inter-feature correlation and feature interaction), as Jiarpakdee *et al.* [71] argue CFS method might not remove all the correlated features from the dataset effectively.

After removing the feature interactions using CFS methods, we re-run the analysis that we conducted in RQ1 (Section 4.1) and RQ3 (Section 4.3) using the same approach. We then evaluate whether the agreement in terms of top-1 and top-3 overlap increases between the CA and CS methods (compared to the results presented in Section 4.1). Similarly, we also evaluate whether the agreement between the computed feature importance ranks of different CS methods on a given dataset increases in terms of top-1 and top-3 overlaps (compared to the results presented in Section 4.3).

*Results: Result 9) Removing the feature interactions increases the median top-1 and top-3 overlap between the studied CA and CS methods across all the five studied classifiers.* From Table 3 we observe that across all the classifiers, the top-1 and top-3 overlaps improve by at least one level. Such a result indicates that, removing feature interactions with CFS generally yields a higher agreement between the CA and CS methods with regards to feature ranking (i.e., promotes stability of the results).

*Result 10) Removing the feature interactions increases the median top-1 and top-3 overlap between the studied CS methods on four and five of the 18 studied datasets respectively.* Table 4 depicts the improvements in top-1 and top-3 overlaps between the studied CS methods across all the studied datasets. From Table 4 we observe that the removal of feature interactions with CFS yields only small improvements. Therefore, we suggest that researchers and practitioners should be cautious when using different CS methods interchangeably even after removing feature interactions. However, our inference is exploratory in nature and thus further research should be conducted to understand advanced feature interaction removal methods may help improve the agreement across different CS methods.

*Result 11) After removing the feature interactions, SHAP and Permutation yield a strong agreement on all datasets.* From the results presented in Section 4.2 we observe that the Permutation and SHAP had less than large overlap only on the Prop-4 and PC5 datasets. However, upon removal of feature interactions from these two datasets, the feature importance ranks computed by the studied CA methods have a large top-1 and top-3 overlap across all the studied datasets and classifiers.

## 6 Implications

In this section, we outline the implications that one can derive from our results, including potential pitfalls to avoid and future research opportunities.

*Implication 1) Researchers and practitioners should be aware that feature interactions can hinder classifier interpretation. We recommend these stakeholders to detect feature interactions in their datasets (e.g., by means of the Friedman's H-statistic) and, in the positive scenario, remove these interactions if possible (e.g.,*

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

*by preprocessing the dataset with the CFS method).* From Sections 5.1 and 5.2, we find that removing the feature interactions, even with a simple method like CFS, increases the agreement between the feature importance rankings produced by the studied feature importance methods. In other words, once feature interactions are removed, the final feature ranking tends to change. Hence, we consider that feature interactions hinder the interpretability of machine learning models in defect prediction. We thus encourage researchers and practitioners to detect feature interactions in their dataset (e.g., by means of the Friedman's H-statistic). In case interactions are discovered, we encourage these stakeholders to remove them if possible (e.g., by preprocessing the dataset with the CFS method).

*Implication 2) The lack of clear specification of the feature importance method employed in software engineering studies seriously threatens the reproducibility of these studies and the generalizability of their insights.* 14 percent of the studies listed in Table 1 do not specify their employed feature importance method to arrive at their insights. The absence of the specification of the feature importance method employed is more prevalent for random forest classifiers (3/11 studies) – the classifier that is widely used in software engineering. This poses a serious threat, as many random forest implementation across the popular data mining toolboxes come with many different ways of computing the feature importance. For instance, random forest implementation in the R package *randomForest*[3] has 3 feature importance methods available and the R package *partykit*[4] has 2 implementations of feature importance methods for random forest.

*Implication 3) Future research should evaluate the extent to which SHAP and Permutation can be used interchangeably.* We conjecture that using either SHAP or Permutation would lead to similar rankings for defect datasets that have similar characteristics to those studied by us. We also conjecture that the feature importance rankings of prior studies in defect classification would not change much if Permutation were to be replaced with SHAP or vice-versa. Nonetheless, we do emphasize that even small changes in rankings could be meaningful in practice and lead to completely different action plans (e.g., in terms of prioritizing pieces of code to be tested or reviewed). Hence, despite the similar rankings produced by SHAP and Permutation, concluding that they can be used interchangeably would be an overstatement. We thus invite future work to further evaluate the differences in the rankings produced by SHAP and Permutation (e.g., by evaluating defect datasets that have different characteristics compared to the ones that we studied).

*Implication 4) Future research should consider investigating whether more advanced feature interaction removal methods can increase the agreement between the feature interaction rankings produced by different feature importance methods.* By means of a simple feature interaction removal method like CFS, we are able to improve the agreement between the feature importance rankings produced by CS and CA methods (Section 5.2). Even between the computed feature importance ranks of CS methods, removing feature interactions helps to improve agreement in some cases. Such a result suggests that more sophisticated feature interaction removal methods have the potential to further improve the agreement between the computed feature importance ranks of different feature importance methods.

## 7 Threats to Validity

In the following, we discuss the threats to the validity of our study.

*Internal Validity.* We choose classifier families who has a CS method. As previous studies show that different classifiers may have different performance on a given dataset [17], [84], this could be a potential threat. However, we choose representative classifiers from 6 of the 8 commonly used classifier families as outlined by Ghotra *et al.* [84].

We use the AUC and IFA performance measures to shortlist the classifiers used in this study. However, using different performance measures like $P_{opt20}$, MCC, and F-Measure to shortlist the classifiers to include in our study might potentially bias the conclusion of our study and we declare it as a threat to internal validity. We invite the future studies to revisit our study by using different performance measures to shortlist the classifiers to be used to compare feature importance ranks computed by different feature importance methods.

The classifiers that we choose to study are either probabilistic or stochastic in nature. We do not include any deterministic classifiers like Naive Bayes in our study and it could be a potential threat to the internal validity of our study. Particularly since, Wu *et al.* [120] point out that computed feature importance ranks of classifiers like decision trees can be particularly sensitive to data characteristics. We invite the future research to revisit our findings on deterministic and stable learners.

*Construct Validity.* In our study, we choose classifiers where all the studied datasets achieved an AUC above 0.70. According to Muller *et al.* [121], an AUC score above 0.70 indicates the fair discriminative capability of a classifier. Furthermore, these datasets have been used in many of the studies as outlined in Table 1.

We use a simple random search method to hyperparameter tune our studied classifiers. Our decision stems from the work of Tantithamthavorn *et al.* [80] who show that, irrespective of the performance measure considered, different hyperparameter tuning methods (including grid search, random search, differential evolution based methods and genetic algorithm based methods) yield similar performance improvements. The authors suggest that, as far as performance improvements are concerned, researchers and practitioners can safely use any of the aforementioned automated hyperparameter tuning methods to tune defect classifiers. Therefore, we consider that employing different or more advanced hyperparameter optimization methods would not necessarily result in better hyperparameters for our studied classifiers. Nonetheless, we invite future work to revisit our findings by using more advanced hyperparameter tuning methods to tune the classifiers.

The hyperparameters that we tune for our studied classifiers are limited. Similar to Jiarpakdee *et al.* [71], we use the automated hyperparameter optimization option provided by caret package to tune the hyperparameters of the studied

3. https://cran.r-project.org/web/packages/randomForest/index.html

4. https://cran.r-project.org/web/packages/partykit/index.html

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

classifiers. caret package only supports tuning a limited number of hyperparameters (please see Table 6, available in the online supplemental material). Tuning a wider range of hyperparameters for the studied classifiers might potentially change our findings. We consider it a threat and invite the future studies to revisit our findings by tuning a wider range of hyperparameters for the studied classifiers.

*External Validity.* In this study, we choose 18 datasets that represent software projects across several corpora (e.g., NASA and PROMISE) and domains (both proprietary and open-source). However, our results likely do not generalize to all software defect datasets. Nevertheless, the datasets that we use in our study are extensively used in the field of software defect prediction [2], [8], [17], [31], [40], [79] and is representative of several corpora and domains. Therefore we argue that our results will still hold. However, future replication across different datasets using our developed methodology might be fruitful.

Second, we only consider one defect prediction context in our study (i.e., within-project defect prediction). Yet, there are multiple defect prediction contexts such as Just-In-Time defect prediction [122], [123] and cross-project defect prediction [2]. Hence future studies are needed to explore these richer contexts.

Finally, we study a limited number of CS and CA methods and therefore, our results might not readily generalize to other feature importance methods. For instance, there are recent developments like LIME [124] that have been proposed in the machine learning community for generating feature importance ranks. Nevertheless, the approach and the metrics that we use in our study are applicable to any feature importance method. Therefore, we invite future studies to use our approach to re-examine our findings on other (current and future) feature importance methods.

## 8 Conclusion

Classifiers are increasingly used to derive insights from data. Typically, insights are generated from the feature importance ranks that are computed by either CS or CA methods. However, the choice between the CS and CA methods to derive those insights remains arbitrary, even for the same classifier. In addition, the choice of the exact feature important method is seldom justified. In other words, several prior studies use feature importance methods interchangeably without any specific rationale, even though different methods compute the feature importance ranks differently. Therefore, in this study, we set out to estimate the extent to which feature importance ranks that are computed by CS and CA methods differ.

By means of a case study on 18 defect datasets and 6 defect classifiers, we observe that while the computed feature importance ranks by different CA methods strongly agree with each other, the computed feature importance ranks of CS methods do not. Furthermore, the computed feature importance ranks of studied CA and CS methods do not strongly agree with each other either. Except when using different studied CA methods, even the feature reported as the most important feature differs for many of the studied classifiers – raising concerns about the stability of conclusions across replicated studies.

We further find that the commonly used defect datasets are rife with feature interactions and these feature interactions impact the computed feature importance ranks of the CS methods (not the CA methods). We also demonstrate that removing these feature interactions, even with simple methods like CFS improves agreement between the computed feature importance ranks of CA and CS methods. We end our study by providing several guidance for future studies, e.g., future research is needed to investigate the impact of advanced feature interaction removal methods on computed feature importance ranks of different CS methods.

## Acknowledgments

This work was partially supported by JSPS KAKENHI Japan grant no. JP18H03222 and JSPS International Joint Research Program with SNSF (Project "SENSOR").

## References

[1] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead, Jr, "Does bug prediction support human developers? Findings from a Google case study," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 372–381.

[2] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proc. 7th Joint Meeting Europ. Softw. Eng. Conf., ACM SIGSOFT Symp. Foundations Softw. Eng.*, 2009, pp. 91–100.

[3] B. Caglayan, B. Turhan, A. Bener, M. Habayeb, A. Miransky, and E. Cialini, "Merits of organizational metrics in defect prediction: An industrial replication," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 89–98.

[4] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proc. ACM SIGSOFT 20th Int. Symp. Foundations Softw. Eng.*, 2012, Art. no. 62.

[5] F. Zhang, A. E. Hassan, S. McIntosh, and Y. Zou, "The use of summation to aggregate software metrics hinders the performance of defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 43, no. 5, pp. 476–491, May 2017.

[6] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: A study of breakage and surprise defects," in *Proc. 19th ACM SIGSOFT Symp., 13th Eur. Conf. Foundations Softw. Eng.*, 2011, pp. 300–310.

[7] D. Chen, W. Fu, R. Krishna, and T. Menzies, "Applications of psychological science for actionable analytics," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2018, pp. 456–467.

[8] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 320–331.

[9] C. Theisen, K. Herzig, P. Morrison, B. Murphy, and L. Williams, "Approximating attack surfaces with stack traces," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 199–208.

[10] K. Herzig, S. Just, and A. Zeller, "The impact of tangled code changes on defect prediction models," *Empir. Softw. Eng.*, vol. 21, no. 2, pp. 303–336, 2016.

[11] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *Proc. 15th Int. Symp. Softw. Rel. Eng.*, 2004, pp. 417–428.

[12] H. Jahanshahi, D. Jothimani, A. Başar, and M. Cevik, "Does chronology matter in jit defect prediction?: A partial replication study," in *Proc. 15th Int. Conf. Predictive Models Data Analytics Softw. Eng.*, 2019, pp. 90–99.

[13] T. Mori and N. Uchihira, "Balancing the trade-off between accuracy and interpretability in software defect prediction," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 779–825, 2019.

[14] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "A benchmark for interpretability methods in deep neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2019, pp. 9737–9748.

[15] C. Molnar, "Interpretable machine learning," Lulu.com, 2020.

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

[16] S. Chakraborty *et al.*, "Interpretability of deep learning models: A survey of results," in *Proc. IEEE SmartWorld Ubiquitous Intell. Comput. Advanced Trusted Computed Scalable Comput. Commun. Cloud Big Data Comput., Internet People Smart City Innovation*, 2017, pp. 1–6.

[17] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories*, 2017, pp. 135–145.

[18] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1200–1219, Nov. 2020.

[19] C. Treude and M. Wagner, "Predicting good configurations for github and stack overflow topic models," in *Proc. 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 84–95.

[20] T. Yu, W. Wen, X. Han, and J. Hayes, "Conpredictor: Concurrency defect prediction in real-world applications," *IEEE Trans. Softw. Eng.*, vol. 45, no. 6, pp. 558–575, Jun. 2019.

[21] K. Herzig, "Using pre-release test failures to build early post-release defect prediction models," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, 2014, pp. 300–311.

[22] A. B. de González *et al.*, "Interpretation of interaction: A review," *Ann. Appl. Statist.*, vol. 1, pp. 371–385, 2007.

[23] G. Freeman, "The analysis and interpretation of interactions," *J. Appl. Statist.*, vol. 12, no. 1, pp. 3–10, 1985.

[24] A. Fisher, C. Rudin, and F. Dominici, "All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously," vol. 20, no. 177, pp. 1–81, 2019.

[25] S. Devlin, C. Singh, W. J. Murdoch, and B. Yu, "Disentangled attribution curves for interpreting random forests and boosted trees," 2019, *arXiv: 1905.07631*.

[26] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," 2018, *arXiv: 1802.03888*.

[27] M. A. Hall, "Correlation-based feature selection for machine learning," Univ. Waikato Hamilton, 1999.

[28] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 6, pp. 1437–1447, Nov./Dec. 2003.

[29] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects," *IEEE Trans. Softw. Eng.*, vol. 29, no. 4, pp. 297–310, Apr. 2003.

[30] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 481–494, Jun. 2003.

[31] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proc. ACM/IEEE 30th Int. Conf. Softw. Eng.*, 2008, pp. 531–540.

[32] L. Angelis, I. Stamelos, and M. Morisio, "Building a software cost estimation model based on categorical data," in *Proc. 7th Int. Softw. Metrics Symp.*, 2001, pp. 4–15.

[33] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 284–292.

[34] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? A case study of the QT, VTK, and ITK projects," in *Proc. IEEE 22nd Int. Conf. Softw. Anal. Evol. Reeng.*, 2015, pp. 171–180.

[35] T. A. Ghaleb, D. A. da Costa, and Y. Zou, "An empirical study of the long duration of continuous integration builds," *Empir. Softw. Eng.*, 2019, pp. 1–38.

[36] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2015, pp. 111–120.

[37] J. Zhou, S. Wang, C.-P. Bezemer, Y. Zou, and A. E. Hassan, "Studying the association between bountysource bounties and the issue-addressing likelihood of github issue reports," *IEEE Trans. Softw. Eng.*, early access, Feb. 17, 2020, doi: 10.1109/TSE.2020.2974469.

[38] S. Wang, T.-H. Chen, and A. E. Hassan, "Understanding the factors for fast answers in technical Q&A websites," *Empir. Softw. Eng.*, vol. 23, no. 3, pp. 1552–1593, 2018.

[39] F. Calefato, F. Lanubile, and N. Novielli, "An empirical assessment of best-answer prediction models in technical Q&A sites," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 854–901, 2019.

[40] R. Premraj and K. Herzig, "Network versus code metrics to predict defects: A replication study," in *Proc. Int. Symp. Empir. Softw. Eng. Meas.*, 2011, pp. 215–224.

[41] G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet, "Automatically finding the control variables for complex system behavior," *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 439–468, 2010.

[42] L. B. Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, and A. D. Brucker, "Time for addressing software security issues: Prediction models and impacting factors," *Data Sci. Eng.*, vol. 2, no. 2, pp. 107–124, 2017.

[43] J. Santos and O. Belo, "Estimating risk management in software engineering projects," in *Proc. Ind. Conf. Data Mining*, 2013, pp. 85–98.

[44] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: automated neural network model debugging via state differential analysis and input selection," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Foundations Softw. Eng.*, 2018, pp. 175–186.

[45] K. El-Emam, D. Goldenson, J. McCurley, and J. Herbsleb, "Modelling the likelihood of software process improvement: An exploratory study," *Empir. Softw. Eng.*, vol. 6, no. 3, pp. 207–229, 2001.

[46] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Proc. Int. Workshop Mining Softw. Repositories*, 2006, pp. 119–125.

[47] O. Malgonde and K. Chari, "An ensemble-based model for predicting agile software development effort," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 1017–1055, 2019.

[48] M. Haran *et al.*, "Techniques for classifying executions of deployed software to support software engineering tasks," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 287–304, May 2007.

[49] G. Gousios, M. Pinzger, and A. V. Deursen, "An exploratory study of the pull-based software development model," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 345–355.

[50] R. Niedermayr and S. Wagner, "Is the stack distance between test case and method correlated with test effectiveness?," in *Proc. Eval. Assessment Softw. Eng.*, 2019, pp. 189–198.

[51] D. Martens and W. Maalej, "Towards understanding and detecting fake reviews in app stores," *Empir. Softw. Eng.*, vol. 24, no. 6, pp. 3316–3355, 2019.

[52] Y. Fan, X. Xia, D. Lo, and S. Li, "Early prediction of merged code changes to prioritize reviewing tasks," *Empir. Softw. Eng.*, vol. 23, no. 6, pp. 3346–3393, 2018.

[53] L. Bao, X. Xia, D. Lo, and G. C. Murphy, "A large scale study of long-time contributor prediction for GitHub projects," *IEEE Trans. Softw. Eng.*, early access, May 23, 2019.

[54] T. Dey and A. Mockus, "Are software dependency supply chain metrics useful in predicting change of popularity of NPM packages?," in *Proc. 14th Int. Conf. Predictive Models Data Anal. Softw. Eng.*, 2018, pp. 66–69.

[55] A. Dehghan, A. Neal, K. Blincoe, J. Linaker, and D. Damian, "Predicting likelihood of requirement implementation within the planned iteration: An empirical study at IBM," in *Proc. 14th Int. Conf. Mining Softw. Repositories*, 2017, pp. 124–134.

[56] K. Blincoe, A. Dehghan, A.-D. Salaou, A. Neal, J. Linaker, and D. Damian, "High-level software requirements and iteration changes: a predictive model," *Empir. Softw. Eng.*, vol. 24, no. 3, pp. 1610–1648, 2019.

[57] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proc. 7th Int. Conf. Predictive Models Softw. Eng.*, 2011, Art. no. 11.

[58] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," *Empirical Softw. Eng*, vol. 17, no. 1–2, pp. 1–17, 2012.

[59] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," in *Proc. 40th Int. Conf. Softw. Eng.: New Ideas Emerg. Results*, 2018, pp. 53–56.

[60] J. Hihn and T. Menzies, "Data mining methods and cost estimation models: Why is it so hard to infuse new ideas?" in *Proc. 30th IEEE/ACM Int. Conf. Autom. Softw. Eng. Workshop*, 2015, pp. 5–9.

[61] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, "Bias in random forest variable importance measures: Illustrations, sources and a solution," *BMC Bioinf.*, vol. 8, no. 1, 2007, Art. no. 25.

[62] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 5, pp. 412–428, May 2018.

[63] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 2146–2189, 2016.


Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

[64] L. Bao, Z. Xing, X. Xia, D. Lo, and S. Li, "Who will leave the company?: A large-scale industry study of developer turnover by mining monthly work report," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories*, 2017, pp. 170–181.

[65] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? A case study on free android applications," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2015, pp. 301–310.

[66] S. L. Abebe, N. Ali, and A. E. Hassan, "An empirical study of software release notes," *Empir. Softw. Eng.*, vol. 21, pp. 1107–1142, 2016.

[67] L. C. Briand, J. Daly, V. Porter, and J. Wust, "Predicting fault-prone classes with design measures in object-oriented systems," in *Proc. 9th Int. Symp. Softw. Rel. Eng.*, 1998, pp. 334–343.

[68] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Trans. Softw. Eng.*, vol. 35, no. 6, pp. 864–878, Nov./Dec. 2009.

[69] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in *Proc. 19th ACM SIGSOFT Symp., 13th Eur. Conf. Foundations Softw. Eng.*, 2011, pp. 4–14.

[70] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? An empirical case study of windows vista," in *Proc. 31st Int. Conf. Softw. Eng.*, 2009, pp. 518–528.

[71] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, early access, Mar. 23, 2020, doi: 10.1109/TSE.2020.2982385.

[72] A. Gosiewska and P. Biecek, "ibreakdown: Uncertainty of model explanations for non-additive predictive models," *CoRR*, vol. abs/1903.11420, 2019.

[73] C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis, "Conditional variable importance for random forests," *BMC Bioinf.*, vol. 9, no. 1, 2008, Art. no. 307.

[74] E. Candes, Y. Fan, L. Janson, and J. Lv, "Panning for gold:'model-x'knockoffs for high dimensional controlled variable selection," *J. Roy. Statist. Soc. Ser. Methodol.*, vol. 80, no. 3, pp. 551–577, 2018.

[75] U. Grömping, "Variable importance assessment in regression: linear regression versus random forest," *Amer. Statistician*, vol. 63, no. 4, pp. 308–319, 2009.

[76] L. Auret and C. Aldrich, "Empirical comparison of tree ensemble variable importance measures," *Chemometrics Intell. Lab. Syst.*, vol. 105, no. 2, pp. 157–170, 2011.

[77] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "Autospearman: Automatically mitigating correlated software metrics for interpreting defect models," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2018, pp. 92–103.

[78] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 812–823.

[79] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 1–18, Jan. 2017.

[80] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 45, no. 7, pp. 683–711, Jul. 2019.

[81] J. Sayyad Shirabad and T. Menzies, "The PROMISE repository of software engineering databases," School Inf. Technol. Eng., Univ. Ottawa, Canada, 2005. [Online]. Available: Available: http://promise.site.uottawa.ca/SERepository

[82] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, Jun. 2014.

[83] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Comments on "Researcher bias: The use of machine learning in software defect prediction"," *IEEE Trans. Softw. Eng.*, vol. 42, no. 11, pp. 1092–1094, Nov. 2016.

[84] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 789–800.

[85] M. Kuhn, "Variable importance using the caret package," 2012.

[86] A. Avati, K. Jung, S. Harman, L. Downing, A. Ng, and N. H. Shah, "Improving palliative care with deep learning," *BMC Med. Inform. Decis. Making*, vol. 18, no. 4, 2018, Art. no. 122.

[87] S. Janitza, C. Strobl, and A.-L. Boulesteix, "An AUC-based permutation variable importance measure for random forests," *BMC Bioinf.*, vol. 14, no. 1, 2013, Art. no. 119.

[88] J. Jiarpakdee, "Towards a more reliable interpretation of defect models," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: Companion*, 2019, pp. 210–213.

[89] A. Husain, "Understanding how developers work on change tasks using interaction history and eye gaze data," Ph.D. dissertation, Youngstown State Univ., Youngstown, Ohio, 2015.

[90] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 3590–3638, 2020.

[91] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[92] I. Covert, S. Lundberg, and S.-I. Lee, "Understanding global feature contributions with additive importance measures," in *Advances Neural Inf. Process. Syst.*, vol. 33, pp. 17212–17223, 2020.

[93] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 4765–4774.

[94] M. Viggiato and C.-P. Bezemer, "Trouncing in dota 2: An investigation of blowout matches," in *Proc. AAAI Conf. Artif. Intell. Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 294–300.

[95] G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions," *Automated Softw. Eng.*, vol. 27, no. 3, pp. 369–392, 2020.

[96] K. Peng and T. Menzies, "How to improve ai tools (by adding in se knowledge): Experiments with the timelime defect reduction tool," 2020, *arXiv: 2003.06887*.

[97] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proc. 40th Int. Conf. Softw. Eng.: Softw. Eng. Practice*, 2018, pp. 286–295.

[98] F. E. Harrell, Jr, *Regression Modeling Strategies: with Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Berlin, Germany: Springer, 2015.

[99] B. Efron, "Estimating the error rate of a prediction rule: Improvement on cross-validation," *J. Amer. Statist. Assoc.*, vol. 78, no. 382, pp. 316–331, 1983.

[100] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?," *Inf. Softw. Technol.*, vol. 76, pp. 135–146, 2016.

[101] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[102] M. Kuhn *et al.*, "Building predictive models in r using the caret package," *J. Statist. Softw.*, vol. 28, no. 5, pp. 1–26, 2008.

[103] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul./Aug. 2008.

[104] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering," *IEEE Trans. Softw. Eng.*, early access, Jun. 24, 2019, doi: 10.1109/TSE.2019.2924371.

[105] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?," in *Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 199–209.

[106] Q. Huang, X. Xia, and D. Lo, "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2017, pp. 159–170.

[107] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 1–18, 2017.

[108] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery," *Queue*, vol. 16, no. 3, pp. 31–57, 2018.

[109] M. G. Kendall, "Rank correlation methods," Griffin, 1948.

[110] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Towards a framework for software measurement validation," *IEEE Trans. Softw. Eng.*, vol. 21, no. 12, pp. 929–944, Dec. 1995.

[111] A. Bachmann and A. Bernstein, "When process data quality affects the number of bugs: Correlations in software engineering datasets," in *Proc. 7th IEEE Working Conf. Mining Softw. Repositories*, 2010, pp. 62–71.

[112] H. Akoglu, "User's guide to correlation coefficients," *Turkish J. Emergency Med.*, vol. 18, pp. 91–93, 2018.


Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.

[113] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Societe Vaudoise Sci. Naturelles.*, vol. 37, pp. 547–579, 1901.
[114] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *Proc. 21st IEEE Int. Conf. Softw. Maintenance*, 2005, pp. 263–272.
[115] J. H. Friedman *et al.*, "Predictive learning via rule ensembles," *Ann. Appl. Statist.*, vol. 2, no. 3, pp. 916–954, 2008.
[116] M. N. Wright, A. Ziegler, and I. R. König, "Do little interactions get lost in dark random forests?," *BMC Bioinf.*, vol. 17, no. 1, 2016, Art. no. 145.
[117] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Syst. Softw.*, vol. 83, no. 1, pp. 2–17, 2010.
[118] J. Cahill, J. M. Hogan, and R. Thomas, "Predicting fault-prone software modules with rank sum classification," in *Proc. 22nd Australian Softw. Eng. Conf.*, 2013, pp. 211–219.
[119] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empir. Softw. Eng.*, vol. 17, no. 4–5, pp. 531–577, 2012.
[120] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, 2008.
[121] M. P. Muller *et al.*, "Can routine laboratory tests discriminate between severe acute respiratory syndrome and other causes of community-acquired pneumonia?," *Clin. Infect. Dis.*, vol. 40, no. 8, pp. 1079–1086, 2005.
[122] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction," in *Proc. 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 34–45.
[123] Y. Kamei *et al.*, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2012.
[124] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?: Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.
[125] L. S. Shapley, "A value for n-person games," *Contributions Theory Games*, vol. 2, no. 28, pp. 307–317, 1953.
[126] C. Halimu, A. Kasem, and S. S. Newaz, "Empirical comparison of area under roc curve (AUC) and mathew correlation coefficient (MCC) for evaluating machine learning algorithms on imbalanced datasets for binary classification," in *Proc. 3rd Int. Conf. Mach. Learn. Soft Comput.*, 2019, pp. 1–6.
[127] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng.*, 2006, pp. 8–17.
[128] C. Bird *et al.*, "Fair and balanced?: Bias in bug-fix datasets," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf., ACM SIGSOFT Symp. Foundations Softw. Eng.*, 2009, pp. 121–130.

**Gopi Krishnan Rajbahadur** received the BE degree in computer science and engineering from SKR Engineering College, Anna University, India, and the PhD degree in computer science from Queen's University, Canada. He is a senior researcher with the Centre for Software Excellence at Huawei, Canada. His research interests include software engineering for artificial intelligence (SE4AI), artificial intelligence for software engineering (AI4SE), mining software repositories (MSR) and explainable AI (XAI).

**Shaowei Wang** received the BSc degree from Zhejiang University. He is an assistant professor with the Department of Computer Science, University of Manitoba. His research interests include software engineering, machine learning, data analytics for software engineering, automated debugging, and secure software development. He is one of four recipients of the 2018 Distinguished Reviewer Award for the Springer EMSE (SE's highest impact journal). For more information, please visit https://sites.google.com/site/wswshaoweiwang.

**Gustavo A. Oliva** received the MSc and PhD degrees from the University of São Paulo (USP), in Brazil under the supervision of Prof. Dr. Marco Aurélio Gerosa. He is a research fellow with the School of Computing, Queen's University, in Canada under the supervision of Prof. Dr. Ahmed E. He leads the blockchain research team with the Software Analysis and Intelligence Lab (SAIL), where he investigates blockchain technologies through the lens of Software Engineering. In addition, he also conducts research on software ecosystems, explainable AI, and software maintenance. For more information, please visit https://www.gaoliva.com

**Yasutaka Kamei** (Senior Member, IEEE) received the BE degree in informatics from Kansai University, and the PhD degree in information science from the Nara Institute of Science and Technology. He is an associate professor with Kyushu University in Japan. He was a research fellow with the JSPS (PD) from July 2009 to March 2010. From April 2010 to March 2011, he was a postdoctoral fellow with Queen's University in Canada. His research interests include empirical software engineering, open source software engineering, and mining software repositories (MSR). He serves on the editorial boards of Springer Journal of Empirical Software Engineering. For more information, please visit http://posl.ait.kyushu-u.ac.jp/∼kamei

**Ahmed E. Hassan** (Fellow, IEEE) received the PhD degree in computer science from the University of Waterloo. He is an ACM SIGSOFT Influential Educator, an NSERC Steacie fellow, the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair with the School of Computing, Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He spearheaded the creation of the Mining software repositories (MSR) conference and its research community. He also serves/d on the editorial boards of *IEEE Transactions on Software Engineering*, *Springer Journal of Empirical Software Engineering*, and *PeerJ Computer Science*. For more information, please visit http://sail.cs.queensu.ca

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Queen's University. Downloaded on July 24,2023 at 14:22:13 UTC from IEEE Xplore. Restrictions apply.