Impact of Discretization Noise of the Dependent variable on Machine Learning Classifiers in Software Engineering

Gopi Krishnan Rajbahadur, Shaowei Wang, Yasutaka Kamei, and Ahmed E. Hassan

Abstract—Researchers usually discretize a continuous dependent variable into two target classes by introducing an artificial discretization threshold (e.g., median). However, such discretization may introduce noise (i.e., discretization noise) due to ambiguous class loyalty of data points that are close to the artificial threshold. Previous studies do not provide a clear directive on the impact of discretization noise on the classifiers and how to handle such noise. In this paper, we propose a framework to help researchers and practitioners systematically estimate the impact of discretization noise on classifiers in terms of its impact on various performance measures and the interpretation of classifiers. Through a case study of 7 software engineering datasets, we find that: 1) discretization noise affects the different performance measures of a classifier differently for different datasets; 2) Though the interpretation of the classifiers are impacted by the discretization noise on the whole, the top 3 most important features are not affected by the discretization noise. Therefore, we suggest that practitioners and researchers use our framework to understand the impact of discretization noise on the performance of their built classifiers and estimate the exact amount of discretization noise to be discarded from the dataset to avoid the negative impact of such noise.

Index Terms—Discretization noise, Discretization, Classifiers, Feature Importance Analysis, Performance, Random Forest, Logistic Regression, Decision Trees, KNN.

1 Introduction

Machine learning classifiers are widely used throughout software engineering studies. Some of the most common uses of classifiers include predicting defects [23], [38], [52], [55], bug-fix times [33], understanding the features that impact the defect proneness of a software system [9], [44], [48], [50].

Usually, classifiers are trained on labeled data points and are used to predict the target class of the unlabeled data points. In the absence of pre-defined class labels and the availability of only the continuous dependent variable, researchers usually discretize[1] the continuous **dependent** variable into artificial target classes. Such discretization might be based on domain knowledge [25], a phenomenon that the study wishes to observe [33], or in many cases when there are no imposed target classes, an artificial discretization threshold is used to discretize the target variable into binary (or n-ary) classes [13], [16], [67], [69].

However a plethora of prior studies note that the discretization of the continuous dependent variable could be detrimental to the performance of classifier and may produce misleading results [10], [12], [15], [56]. One alternative approach to avoid such discretization noise would be to train regression models on the continuous dependent variable and then discretize the predicted outcome afterwards [55]. But as Rajbahadur *et al.* [55] observe, only when there is a significant class imbalance in the dataset, classification through regression would yield better results. Therefore, the discretization of continuous dependent variable with artificial discretization thresholds is still widely practiced in software engineering as evidenced by [22], [25], [28], [31], [33], [59], [67], [69].

The other problem with such a discretization approach is that the data points that are very close to the discretization threshold (e.g., median) get class labels that might not be reflective of the true class to which they belong. While many previous studies explore the harmful impacts of discretizing the continuous dependent variables on the performance of

- *Gopi Krishnan Rajbahadur, Shaowei Wang and Ahmed E. Hassan are with Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Canada.*
  E-mail: {krishnan, shaowei, ahmed}@cs.queensu.ca
- *Yasutaka Kamei is with Principles of Software Languages (POSL) Lab, Graduate School and Faulty of Information Science and Electrical Engineering, Kyushu University, Japan.*
  Email: kamei@ait.kyushu-u.ac.jp
- *Shaowei Wang is the corresponding author*

[1] Discretization is the process of turning numerical data into discrete data with finite intervals [21]

the classifiers [10], [12], [15], [56], the problem of data points with ambiguous class labels and its impact on the classifiers is completely unexplored. For instance, consider the example of determining whether a bug was closed fast or slow. A domain-expert might decide that any bug closed within a week is a fast-closed bug. However, such a discretization rule for the dependent variable would lead to noise for data points close to that 7-days threshold. For instance, a bug that is closed within 7 days and 1 min would be considered as a slow-closed bug. Such discretization introduces noise in the data that is used for training the classifiers. We define such noise as the *discretization noise* and the data points whose ambiguous class labels that generate the discretization noise as the *noisy points*.

In summary, discretization of continuous dependent variable is both problematic and generates discretization noise. However, the practice of discretizing the continuous dependent variable still remains a widely used practice in software engineering without any consideration to the generated discretization noise.

Therefore the main goal of our study is two-fold: first, to introduce awareness among the software engineering researchers and practitioners about previously unexplored discretization noise. Second, we provide them with a framework - a systematic and rigorous method for exploring the impact of discretization noise on their classifier of choice for any given dataset.

We highlight the capability of our framework by conducting our study on four different types of software engineering datasets (Q&A websites data (4 websites), Linux patch acceptance time data, bug-fix delay data, mobile app ratings data), as a binary classification problem, where the discretization noise is caused by the discretization of a continuous dependent variable around artificial discretization thresholds. We applied our proposed framework to four different families of classifiers (i.e., Random forest classifier (RFCM), Logistic regression (LR), Classification and Regression Trees (CART), and K-Nearest Neighbors (KNN)) to analyse the impact of discretization noise on the various common performance measures (i.e., *Accuracy, Precision, Recall, Brier score, AUC, F-Measure, MCC*). In addition, we also analyse the impact of discretization noise on the interpretation of classifiers in terms of the derived feature importance ranks. The derived feature importance ranks is a rank list that reports the features of the dataset in the order of their influence on the classification.

We highlight our findings and suggestions as follows:

1) **The impact of discretization noise is inconsistent across multiple performance measures for different datasets across all the studied classifiers.** Though the impact on Recall is the most pronounced (up to 139%), other performance measures - Precision, Brier score, F-Measure, and MCC are also impacted at least up to 43.19% due to the inclusion or exclusion of discretization noise (both positively and negatively). **Therefore, we urge the researchers and practitioners to use our framework to analyse if (and how much) discretization noise exists and how to address it.**
2) **Though the overall derived feature importance ranks are impacted by discretization noise, the importance ranks of the top three important features are not affected.** Therefore, in absence of any impact on the interpretation of the top x features detected by our framework, one could include or discard the discretization noise in their datasets as recommended by our framework. Especially without being worried about the discretization noise's impact on the interpretation of the classifier.

Finally, we also provide the framework as an R package (and guidelines for using our framework) to provide automated support to others who wish to revisit their prior results or to consider discretization in the future studies.

Paper organization. Section 2 places our contribution relative to prior work. Section 3 outlines our framework. Section 4 presents the impact of discretization noise on the performance of the classifiers and on the derived feature importance ranks of several software engineering datasets. Section 5 explores why discretization noise is sometimes detrimental and sometimes useful. In Section 6 provide a user guideline for our framework. We then discuss the threats to the validity of our observations in Section 7. Finally, we conclude our study in Section 8.

2 Related Work

In this section, we discuss two groups of prior work: efforts that attempt to establish the impact of noise on classifiers and efforts that investigate problems with discretization on classifiers.

2.1 Impact of Noise on Classifiers

Discretization noise while being similar to class noise [75], differs from it because class noise is caused by data points with wrong class labels due to contradictory examples and random misclassifications [75]. Whereas, we view discretization noise as data points with ambiguous class labels due to their proximity to the discretization threshold. While discretization noise is unexplored, a number of studies have explored the impact of noise on classifiers [20], [35], [36], [60], [75]. For instance, Kim *et al.* [36] explored the impact of class noise on defect classifiers and provided insights on acceptable levels of noise in the data when building classifiers and the sensitivity of the various classifiers to class noise. Whereas Folleco *et al.* [20] explored the impact of class noise on various classifiers when using software quality data and found that the performance of the random forest classifiers is the most robust. Similarly, studies outside software engineering have also found class noise to be detrimental to the performance of a classifier [54], [76]. On the contrary, Tantithamthavorn *et al.* [63] demonstrated that the Precision of a classifier is not impacted by class noise and the most influential features (i.e., features in the top 3 ranks) remain the same irrespective of the amount of class noise that is present in the data. They suggested that the insights from the constructed classifiers can be used without worrying about the noise. However, they did note that filtering the noise increases the Recall of the classifiers. Different from prior studies, we are not concerned with the impact of class noise and noise in the dataset that is generated at random [26] or during data collection [49]. Rather, we provide a framework to analyse the impact of the discretization noise that is generated due to the discretization of the dependent variable (in the absence of labeled data) on the performance of classifiers and the derived feature importance ranks of a classifier in a systematic and repeatable manner.

2.2 Impact of Discretization on Classifiers

In software engineering, discretization has been used to transform both continuous independent and dependent variables into discrete classes [32], [42], [46], [47], [55], [74].

TABLE 1: Details of datasets used in the study

Dataset	#Size	#Features	R(dependent variable)
Stack Overflow	55,853	28	0-9,981.40 mins
Mathematics	70,336	27	0-30,073.72 mins
Ask Ubuntu	7,134	26	0-31,638.55 mins
Super User	10,776	27	0-51,376.33 mins
Patch	20,000	22	0-1,266.92 days
Bug-delay	2,434	23	-1,319.06*-1,990.27 days
App-rating	7,365	22	1.41-4.97 stars

*The dependent variable has negative value since some developers started fixing a bug before the bug was reported.
R(x) - Range(x)

2.2.1 Discretization of the independent variable

Yang *et al.* [71] and Garcia *et al.* [21] have shown that the discretization of the independent variables improves the performance of machine learning classifiers. True to that result, many studies in software engineering discretize the independent variables as part of data pre-processing before constructing classifiers [42], [46]. For instance, Jiang *et al.* investigated and found that while discretization of independent variables improves the performance of some classifiers, it did not universally benefit all classifiers [32]. Contrary to those studies, Nam and Kim used a median based discretization of independent variables to assign class labels for un-labeled data points to train a cross-project defect prediction model [53]. However, except for univariate discretization techniques, the discretization techniques used in the aforementioned studies cannot be used in our study, as they focus on discretizing an independent variable based on the information that is contained in the other independent variables or the class distribution of the dependent variable. But in our study, we focus on studying the impact of discretizing the dependent variable into meaningful target classes based on itself and domain knowledge.

2.2.2 Discretization of the dependent variable

Mockus [49] suggested that many of the software engineering datasets have poor data quality which affects the insights that are offered by the software analytical models. This noise can arise from various sources in the case of labeled data. But when the class labels for the data are not available and a continuous dependent variable is discretized using a threshold to generate class labels [13], [16], [67], [74], we end up with discretization noise. But many researchers actively have argued against the practice of discretization. For instance, Altman *et al.* [5] and Cohen [10] pointed out that discretization at the median of a continuous variable leads to a loss of information, and discretization at other cut points away from the center lead to a much greater information loss. In addition, several prior studies argued against the use of any data-driven cutpoint to discretize dependent variable as it introduced noise and bias [12], [56], [57]. MacCallum [43] and DeCoster *et al.* [15] further state that discretization is rarely ever justifiable and it is almost always safer not to discretize. However, several researchers also note that it is acceptable to discretize sometimes [14], [15], [17], [37]. In summary, though there are some benefits associated with discretization, the majority of the research agrees that it is not a safe practice and should mostly be avoided.

2.2.3 Discretization in software engineering

While a majority of the research community agrees that discretization is not a safe practice as highlighted in the prior section, discretization of the dependent variable still continues to be an accepted practice in software engineering and other fields. For instance, many software engineering studies like [22], [25], [28], [31], [33], [59], [67], [69] discretize the dependent variable to generate the outcome classes on which machine learning classifiers are trained. Of these, some of the studies like [28], [67], [69] discard the data around the discretization threshold due to its ambiguous class loyalties and use only the top and bottom x% to train classifiers. But even while doing so, they do not provide any clear theoretical or empirical reason for doing so. Whereas some other studies like [22], [25], [31], [33], [59] split the continuous dependent variable on the various criterion and include all the data points without any consideration for the noise around the discretization threshold.

Therefore, our study is the first study in the field of software engineering to investigate the effect and impact of discretization noise on the performance and interpretation of classifiers. For the researchers and practitioners who continue to discretize the dependent variable artificially to build classifiers, we propose a framework. They can use our proposed framework to analyse the impact that the generated discretization noise has on their classifiers. Furthermore, we also shine the spotlight on the noise that is generated due to the discretization of the dependent variable for building classifiers - which generally is ignored in the field of software engineering.

3 Framework for Understanding the Impact of Discretization Noise

An overview of our framework for understanding the impact of discretization noise is presented in Figure 1. The framework consists of six steps. Due to space constraints, we detail the steps below (with more explanations where needed) with a running example in Appendix B to better demonstrate the use of our framework.

The individual steps of our approach are explained in detail below.

3.1 Step 1: Correlation & Redundancy Analysis

In this study, we collect data based on two criteria: 1) the dependent variable is continuous; 2) the dependent variable lacks a clear-cut threshold for discretization. Based on these two criteria, we collect 4 types of data (7 datasets): Q&A websites data [69], Linux patch acceptance time (Patch) data [33], Bug-fix delay time (Bug-delay) data [72], and Mobile app rating (App-rating) data [67] (More details about the datasets are given in Appendix A). Table 1 contains basic information about the number of data points and the independent features of each of the studied datasets.

We perform correlation and redundancy analysis on the independent features of a studied dataset to remove correlated and redundant features from the dataset, thereby not biasing our feature importance results [62]. We do so instead of using other common and state of the art dimensionality reduction techniques like PCA, since, dimensionality reduction techniques like PCA combine and transform the original features into principal components, which are no longer directly interpretable. Finally, a recent MSR study by Ghothra *et al.* [24] showed that correlation-based feature selection is very robust for software engineering datasets. Though we use and recommend correlation and redundancy analysis, our framework supports the use of other methods. We do not pre-process the independent features of the dataset any further in our study. However, practitioners can perform other data pre-processing steps like imputation if required.

3.2 Step 2: Discretization

Threshold estimation: The primary objective of our framework is to understand the impact of discretization noise. We discretize the dependent variable with respect to an artificial threshold (a.k.a a cutpoint) into two response classes: "class1" and "class2". We then assign the "class1" class label to all the data points with a dependent variable that has a value that is less than or equal to the chosen discretization threshold (e.g., median). The remaining data points are assigned the class label "class2". The artificial threshold for such a discretization could be chosen in multiple ways. The threshold could be domain specific and be defined by the experts (e.g., ideal bug-fix time for a specific project as defined by the software engineers working on the project). Alternatively, in the absence of such an established domain specific discretization threshold, many of the prior studies have resorted to various heuristic, intuitive and alternate thresholds for discretization [5], [13], [16], [67], [69]. But irrespective of the choice of the discretization threshold, the data points close to the discretization threshold produce discretization noise. Our framework analyses the impact of discretization noise generated by any such discretization threshold.

In this study, to demonstrate the generalizability and applicability of our framework, we use three artificial discretization thresholds.

***M**edian based discretization **T**hreshold (**MT**):* Many prior studies use median for discretizing the dependent variable into binary classes [13], [16] and it is often used in the absence of explicit domain knowledge about the classes of a dependent variable [5].

*Univariate **C**lustering based **T**hreshold (**CT**):* Univariate clustering is an automated technique for discretization. Univariate clustering splits the dependent variable into multiple groups in an optimal fashion. We use Wang and Song's implementation *optimal k-means clustering in one dimension* (ckmeans.1d.dp[2]) here [68]. The ckmeans.1d.dp divides data in one dimension into k clusters so that the sum of squares of within-cluster distances from each element to its corresponding cluster mean is minimized [68]. We set k equal 2 since we wish to divide the dependent variable into two classes.

*CA**RT** based discretization **T**hreshold (**RTT**):* We use the regression tree approach as described by Breiman [7].[3] Here, we use the continuous dependent variable of our dataset as both the independent and the target variable for the regression tree. We then use the generated regression tree's root node as the threshold for discretization since we attempt to split the dependent variable into two classes.

The generated discretization threshold is used for discretizing the continuous dependent variable into binary classes.

Noisy area estimation: Once the dataset is discretized, we need to define the area of the dataset which contains discretization noise as the noisy area. Domain experts could

[2] https://cran.r-project.org/web/packages/Ckmeans.1d.dp/index.html
[3] https://cran.r-project.org/web/packages/rpart/index.html

TABLE 2: Estimated discretization threshold, limits and % of data points in the noisy area for the datasets considered in the study.

Dataset	MT			CT			RTT			*step_size*
	Threshold	Noisy area (%)	Limit	Threshold	Noisy area (%)	Limit	Threshold	Noisy area (%)	Limit	
SO	21.83 Mins	29	55	136.18 Mins	34	85	214.81 Mins	53	95	5
MA	30.28 Mins	41	70	154.48 Mins	38	85	502.98 Mins	44	95	5
AU	39.74 Mins	38	70	329.28 Mins	54	95	338.93 Mins	53	95	5
SU	30.14 Mins	31	60	193.06 Mins	62	95	262.53 Mins	56	95	5
PH	1.31 Days	10	30	0.06 Days	4	40	9.48 Days	22	60	5
BD	0.67 Days	6	50	0 Days	*	*	47.29 Days	54	100	5
AR	4.03 Stars	43	7	3.86 Stars	50	10	3.74 Stars	63	15	0.5

*The automated noisy area estimation algorithm found no data points in the noisy area
MT- **M**edian based discretization **T**hreshold, **CT**- Univariate **C**lustering based discretization **T**hreshold, **RTT**- CA**RT** based discretization **t**hreshold
Datasets: SO- Stack Overflow, MA- Mathematics, AU- Ask Ubuntu, SU- Super User, PH- Patch, BD- Bug-delay, AR- App-rating

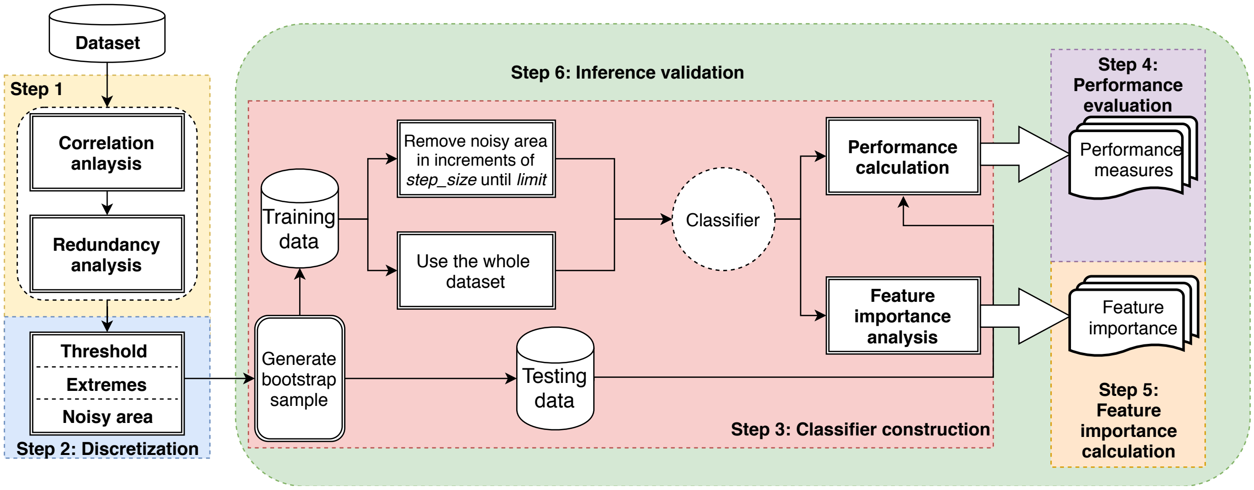

Dataset
Step 1
Correlation anlaysis
Redundancy analysis
Threshold
Extremes
Noisy area
Step 2: Discretization
Step 6: Inference validation
Training data
Generate bootstrap sample
Remove noisy area in increments of step_size until limit
Use the whole dataset
Testing data
Classifier
Performance calculation
Feature importance analysis
Step 3: Classifier construction
Step 4: Performance evaluation
Performance measures
Feature importance
Step 5: Feature importance calculation

Fig. 1: Overview of our framework.

determine a specific range of values around the discretization threshold to be noisy and this could be used as the noisy area. But as we lack deep domain expertise of the datasets considered in this study (which might be the case for many practitioners), we present Algorithm 1: an automated algorithm for estimating the noisy area in a given dataset.

We define the *noisy area* as the data points whose class loyalties are hard to discern due to their proximity to the artificial discretization threshold. Such a hypothesis follows from the rationale of prior studies where the data points around the discretization threshold were discarded to provide better class separation in the training data [1], [2], [34], [67], [69]. Algorithm 1 takes the dataset, cutpoint (i.e., the discretization threshold), and *step_size* as input parameters. *step_size* controls the granularity of the analysis (i.e., the size of the increment from the cut point) - a smaller value of the *step_size* allows for a finer estimation of the noisy area, whereas a larger value provides a coarse estimation of the noisy area. The *step_size* used for all the datasets in this study is given the Table 2.

Line 1 to 3 of the algorithm establishes the initial candidate noisy area, by selecting the area around the cutpoint. More specifically, we consider the points within the area $cutpoint \pm cutpoint * 100\%$ as the candidate noisy area. We do so for two reasons: 1) most of the discretization noise would be concentrated around the discretization threshold due to its proximity to the threshold. 2) if we consider more data, we might not be able to ascertain if the impact of the noisy area on the performance and interpretation of a classifier is due to discretization noise or the high volume of data that is lost. Through line 4 to line 9 we incrementally subset the dataset into the quantum of size given by $cutpoint \pm cutpoint * setp\_size$ and compute the non-linearity of the quantum.

Non-linearity is one of the complexity measures defined by Ho and Basu [29]. Non-linearity score attempts to quantify how hard it might be for a classifier to classify the data points (please refer Section 5 and Table 4 in the appendix for more details about complexity measures). Once we establish the non-linearity for all the quanta, we take the quantum with the maximum non-linearity as the noisy area for our analysis and the step_size that yielded the quantum as the $limit$, which we use to demarcate the noisy area. We use the maximum non-linearity to demarcate the noisy area as

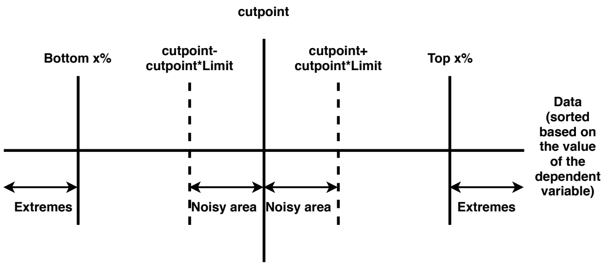


Fig. 2: Extremes and noisy area definitions of a dataset.

it indicates the quantum with the highest data complexity (thereby harder for the classifier). Figure 2 demonstrates how the limit value is used to demarcate the noisy area in a dataset. We present the $limit$ that is generated for demarcating the noisy area of all the studied datasets for various discretization thresholds in Table 2.

Extremes estimation: Finally we establish the data points with the least discretization noise as the extremes. These data points typically have high discriminative power as they are the furthest away from the discretization threshold. Extremes are typically the data points that are associated with the top and bottom x% of the sorted continuous dependent variable. Prior studies usually consider the top and bottom $x\%$ as data points that are devoid of noise and use them for constructing the classifier [67], [69]. We use x as 10% in this study. But the framework allows using any value without any further change to the overall methodology.

Algorithm 1: Automated noisy area estimation algorithm

```
Input: dataset, cutpoint, step_size
Output: limit
Result: Estimates the noisy area in the data automatically by
        computing the limit
$lower\_limit = cutpoint - cutpoint * 100\%$
$upper\_limit = cutpoint + cutpoint * 100\%$
$noisy\_area =$
  $SUBSET(dataset, lower\_limit, upper\_limit)$
while $((cutpoint \pm cutpoint * step\_size) \leq$
  $upper\_limit\ AND\ \geq lower\_limit)$ do
    $quanta$ = SUBSET($noisy\_area,\ cutpoint - cutpoint *$
      $step\_size, cutpoint + cutpoint * step\_size$)
    $nl\_score$ = COMPUTE_NON_LINEARITY($quanta$)
    $results[step\_size] = (nl\_score)$
    $step\_size \mathrel{+}= step\_size$
end
$limit = Indexof$ MAX($results$)
```

3.3 Step 3: Classifier construction

To study the impact of discretization noise, we construct a classifier on the whole dataset and on the dataset with the noisy area removed. One can choose any classifier of their choice in this step. In our study, we consider the 6 classifiers considered by Rajbahadur *et al.* [55]. From the 6 classifiers, we choose the classifiers that have a default feature importance computation method as our framework studies the impact of discretization noise on both the performance and feature importance. Therefore we demonstrate the capability of our framework to analyse the impact of discretization noise on random forest classifier (RFCM), Logistic Regression (LR), Classification and Regression Tree (CART), and K-Nearest Neighbour (KNN). All of the chosen classifiers are hyper parameter tuned to ensure the best and stable performance. We used the method used by Tantithamthavorn *et al.* [64] to hyper-parameter tune all of our classifiers.

Though we use the four aforementioned classifiers, one can use other classifiers instead of these classifiers without any changes to the other steps in the framework.

3.4 Step 4: Performance Evaluation

In this step, the desired classifier performance evaluation measures are chosen. In this study, we observe and evaluate the performance of the constructed classifiers on *Accuracy, Precision, Recall, Brier score, Area Under the receiver operator characteristic Curve (AUC), F-measure, and Mathew's Correlation Coefficient (MCC)*, since many prior studies studied the performance of classifiers using these measures [6], [8], [73]. We calculate these measures with "class 1" as the relevant (positive) class.

Though we demonstrate our framework on the aforementioned performance measures, our framework allows users to use any performance evaluation measures (by themselves or in combination with other measures).

3.5 Step 5: Feature Importance Calculation

We use the default feature importance calculation technique that is associated with each of the studied classifiers to compute the feature importance for each classifier. We use the variable importance computation method **VarImp()** of **caret** package to compute the feature importance of the studied classifiers.

3.6 Step 6: Inference Validation

To ensure that the conclusions that we draw about our classifiers are statistically robust, we use the 100 out-of-sample bootstrap validation technique, which yields an optimal balance between the bias and variance as suggested in the recent study of Tantithamthavorn *et al.* [65].

The out-of-sample bootstrap process is repeated 100 times. After the bootstrap validation, 100 performance measures and 100 lists of derived feature importance ranks are generated. We carry out further analysis on these generated performance measures and the derived feature importance ranks to investigate our research questions.

3.7 Framework Deployment

We use our framework of 6 steps on any given dataset to analyse the impact of discretization noise (as demonstrated in Section 4) along with performance and interpretation on the chosen classifier. Step 1 removes the correlation and redundancy among the features in a dataset, while step 2 is pivotal for estimating the noisy area and extremes for a chosen discretization threshold. Steps 3 to 6 are repeated by incrementally discarding data points in increments of the $step\_size$ parameter (smaller *step_size* enables finer analysis and vice versa) in the noisy area around the threshold until all of the data points in the noisy area are discarded. Such an incremental analysis helps the framework identify the impact of discretization noise and determine the exact amount of data from the noisy area that needs to be discarded for a given dataset, discretization threshold and classifier of choice. We also provide an R package[4] of our framework to enable others and practitioners automated support to use our framework with trivial effort.

4 Understanding the Impact of Discretization Noise on the Performance and Interpretation of a Classifier

4.1 Studying the Impact of Discretization Noise on the Performance of a Classifier

Motivation: It is intuitive to expect that discretization noise might impact the performance of a classifier. Ferri *et al.* show that different performance measures are impacted differently by different types of noise in a dataset [18]. Therefore, first, it is essential to establish if discretization noise impacts the performance of a classifier like other noises. Second, if the discretization noise does impact the performance of a classifier, we need to analyse how the discretization noise in a dataset impacts the performance of a classifier (either positively/negatively) in different performance measures. Finally, it is essential to establish how much data do we have to discard to avoid the impact of discretization noise (as opposed to using only the top and bottom x%).

In order to enable researchers and practitioners to perform such an analysis in a generalizable fashion, we propose our framework. Our framework enables researchers and practitioners to examine the impact of discretization noise on the performance of various classifiers using a variety of software engineering datasets, across a multitude of performance measures.

Approach: We employ our proposed framework (see Section 3) to perform an incremental analysis as mentioned in Section 3.7 to estimate the impact of discretization noise on the performance of a chosen classifier. We specifically draw attention to the classifier construction (step 3) of the framework (see Figure 1). In order to ascertain the performance impact of discretization noise on the classifiers, we train the chosen classifier on data after excluding incremental amounts of discretization noise. More specifically, we discard data points in windows which are defined as $cutpoint \pm cutpoint * x/100$ and use the retained data as the training data to build a classifier, where x varies from 0 to *limit* in increments of $step\_size$ (as mentioned in Step 2 of our framework (See Section 3.2)). The $step\_size$ can be different for different datasets depending on the *limit* used to define the noise area for a particular dataset. Table 2 presents the various limit and *step _ size* values used for different datasets in our study. We perform this incremental analysis (see Fig 1 of Appendix) on all the studied datasets for the three different discretization thresholds (MT, CT, and, RTT) considered in the study as given in Section 3.2 for all the four chosen classifiers (RFCM, LR, CART, and, KNN).

To measure whether the performance of a chosen classifier is impacted by removing data points in the noisy area (data containing discretization noise), we use a Wilcoxon signed-rank test [70], since it is a non-parametric test without any assumptions about the underlying distribution. Furthermore, to quantify the magnitude of the performance differences between the performance of the classifier with no data points removed and the classifier with data points in the noisy area removed, we use Cohen's d effect size test [11]. The threshold for analyzing the magnitude is as follows: $|d| \leq 0.2$ means magnitude is negligible, $|d| \leq 0.5$ means small, $|d| \leq 0.8$ means medium and $|d| > 0.8$ means large.

We perform these statistical tests between the performance measures of the classifier that is constructed on the whole data and the classifier constructed on each step of the incremental analysis (where noisy points in the noisy area is incrementally removed). We do so to estimate how much data needs to be discarded to observe a statistically significant impact on the performance of a classifier. The x value (of the $cutpoint \pm cutpoint * x/100$ used for discarding data in the noisy area) for which different performance measures get significantly impacted is reported. If discarding the whole of the noisy area does not create a statistically significant impact for a particular performance measure then 0 is reported instead of x to signify that the discretization noise does not have a significant impact on that particular performance measure for the studied classifier.

Results: The impact of discretization noise on the performance of different classifiers varies across datasets. Similar performance impacts could be observed for the discretization noise generated by all the discretization thresholds considered in the study. Therefore, we only report the impact of discretization noise generated by the median

[4] https://github.com/SAILResearch/suppmaterial-19-gopi-discretization_noise_impact

TABLE 3: Percentage of improvement in median performance of various classifiers with the noisy area removed over classifiers with no data removed across various performance measures (The x value for which the performance impact first occurs for the given measure is also provided).

Classifier	**Dataset**	**ACC (%)**		**PRC (%)**		**RCL (%)**		**BS (%)**		**AUC (%)**		**F-M (%)**		**MCC (%)**	
		Mag	x	**Mag**	x	**Mag**	x	**Mag**	x	**Mag**	x	**Mag**	x	**Mag**	x
RF	SO	**-0.65**	50	**5.07**	15	**-12.42**	15	**-8**	5	0	0	**-3.69**	25	-0.054$^{\#}$	55
	MA	**-3.5**	45	**8.96**	15	**-32.2**	15	**-11.77**	5	-1.23$^{\#}$	70	**-11.61**	20	**-6.38**	50
	AU	**-7.76**	40	**12.09**	30	**-99.02**	25	**-7.79**	5	0	0	**-43.19**	25	**-18.64**	45
	SU	-0.53§	0	**5.88**	25	**-20.4**	20	**-7.82**	5	0§	45	**-7.59**	30	0.36	0
	PH	**-1.64**	10	**1.36**	25	**-6.64**	10	**-2.04**	5	**-2.22**	10	**-2.62**	10	**-4.08**	10
	BD	0.47§	0	0.72	0	0.75	0	-0.66$^{\#}$	40	0§	0	0.74§	0	1.81§	0
	AR	-0.78§	0	**-4.23**	3	**13.17**	2	**-2.89**	0.5	0	0	**4.52**	2	-2.93§	0
LR	SO	-0.76$^{\#}$	55	**6.24**	15	**-20.81**	20	**-7.73**	5	0§	0	**-7.04**	25	-0.83	0
	MA	**-1.78**	15	**12.33**	10	**-46.87**	10	**-12.02**	5	0$^{\#}$	70	**-15.28**	15	**-2.59**	25
	AU	**-5.54**	50	**11.47**	30	**-139.29**	25	**-10.42**	10	1.56§	0	**-59.78**	25	**-16.3**	60
	SU	**-1.17**	60	**6.42**	30	**-48.2**	20	**-5.87**	10	0	0	**-18.92**	20	**-4.61**	60
	PH	-0.08§	0	**2.23**	15	**-4.68**	10	**-2.59**	5	0 §	0	**-0.97**	15	**-1.28**	20
	BD	-0.2	0	-0.39	0	1.63§	0	-0.78$^{\#}$	40	0	0	0.98	0	-0.18	0
	AR	-0.36§	0	**-2.83**	4	**10.82**	2	**-8.03**	0.5	0	0	**3.95**	2	0.44	0
CART	SO	**1.41**	10	**8.58**	15	**-16.77**	20	-2.77$^{\#}$	30	**3.85**	20	**-3.42**	35	**5.36**	15
	MA	**-1.67**	10	**11.26**	20	**-37.16**	20	**-6.54**	60	0	30	**-11.93**	30	-1.27$^{\#}$	10
	AU	-0.56§	0	**7.34**	30	**-40.49**	30	**-9.55**	50	0	50	**-15.97**	40	2.47§	0
	SU	**1.46**	20	**6.93**	25	**-21.36**	45	**-7.32**	55	**2.99**	25	**-7.05**	50	**9.52**	20
	PH	**-1.12**	10	**1.76**	25	**-6.77**	15	-1.79§	0	**-1.79**	15	**-2.45**	10	**-3.21**	10
	BD	1.2§	0	1.33§	0	0.67	0	-0.77§	0	1.54§	0	1.69§	0	6.53§	0
	AR	0.33	0	-1.06$^{\#}$	6.5	**9.92**	3.5	-3.62§	0	1.64§	0	**4.35**	3.5	3.79§	0
KNN	SO	**2.64**	10	**6.02**	10	**-9.88**	25	**-3.72**	5	**4.29**	5	**-1.81**	10	**12.14**	10
	MA	**1.55**	10	**10.45**	10	**-30.51**	20	**-6.74**	5	**4.29**	10	**-9.91**	10	**11.61**	10
	AU	-0.15	0	**4.12**	50	**-46.12**	25	**-4.52**	10	1.75$^{\#}$	55	**-21.44**	30	4.56§	0
	SU	**1.78**	20	**4.28**	20	**-19.06**	30	**-3.04**	15	**1.69**	15	**-7.01**	40	**17.31**	20
	PH	**-1.13**	20	-0.33	0	**-4.09**	15	**-0.59**	10	**-1.37**	15	**-2.18**	15	**-4.61**	20
	BD	0.77	0	0.86	0	0.44	0	-0.22§	0	1.85	0	0.67	0	17.43	0
	AR	0.16	0	-0.62§	0	**11.54**	2	**-1.19**	1.5	0	0	**5.6**	2	2.05	0

1. **Performance Measures:** ACC- Accuracy, PRC- Precision, RCL- Recall, BS- Brier Score, F-M- F-Measure
2. **Datasets:** SO- Stack Overflow, MA- Mathematics, AU- Ask Ubuntu, SU- Super User, PH- Patch, BD- Bug-delay, AR- App-rating
3. Mag- Magnitude of the performance impact $|x$- % of data of the noisy area when dropped starts statistically impacting the given performance measure
4. **Cohen's d effect size:** Negligible - No formatting, Small -§, Medium -$^{\#}$, Large - **bold**
5. ’$-$‘ indicates performance measure decreases due to removal of noisy area; ’$+$‘ indicates performance measure increases due to removal of noisy area
6. All the values with small, medium or large effect size are statistically significant with $p \leq 0.05$
7. ’$-$‘in cases of Brier score indicates an actual increase in the Brier score and ’$+$‘ a decrease in Brier score (Lower the Brier score, the lesser the error)

based discretization threshold (MT) on various performance measures for all the four classifiers in Table 3 for brevity and space constraints (See Table 1 and Table 2 of the appendix for the performance impact due to other discretization thresholds on the studied classifiers).

Table 3 shows that the discretization noise impacts different classifiers differently. For instance, in the case of both CART and KNN classifiers, for all the datasets except for the patch dataset, the removal of discretization noise improves the performance in terms of AUC. However, for the patch dataset removal of discretization noise negatively impacts the AUC measure. For the same classifiers, even while the AUC and the Precision measures are positively impacted by the removal of discretization noise for (5/7 for CART and 6/7 for KNN), the Recall measure is negatively impacted for 5/7 datasets. Similarly, for LR classifier, while we observe no large impact on the AUC, we could observe up to 139% impact on the Recall as we can observe from Table 3. Furthermore, while the removal of discretization noise negatively impacts the accuracy of the RFCM and LR classifier, for the Stack Overflow dataset, it positively impacts the accuracy of the CART and KNN classifiers. Such a varied impact on the different performance measures for the different classifiers can be observed throughout (see Table 3). **Therefore we assert that different classifiers are impacted differently (either positively or negatively) on the studied performance measures and datasets.**

Additionally, in Table 3 we also provide the x, which tells us the percentage of data from the noisy area, that when dropped, starts statistically impacting the given performance measure. This value aids the users of our framework to know how much data they need to discard in order to avoid the performance impact of the noise on a particular performance measure for a studied classifier.

We find that the removal of discretization noise has both a positive and negative impact on different performance measures for different classifiers. In addition, we do not observe a generalizable trend in how the discretization noise affects different performance measures. For instance, from Table 3, we see that the removal of the noisy area from datasets negatively impacts the performance measures of an RFCM for 6/7 datasets on Accuracy, Recall, Brier score, F-measure, 2/7 datasets on AUC, and 5/7 datasets on MCC, most of which in a statistically significant fashion with and a large effect size. However, for 6/7 datasets, removal of noisy area positively impacts the Precision in a statistically significant fashion with a large effect size (in most cases). Furthermore, while removal of noisy area negatively impacts the performance measures for most datasets, it improves the Accuracy, Precision, Recall, F-Measure, and MCC for the App-rating dataset in a statistically significant fashion with a large effect size. These varied impacts of the discretization noise on the different performance measures highlight the need for our frameworks to study the peculiarities of each case study as **the discretization noise affects the different performance measures differently**

Finally, we also note that **magnitude of the performance impact due to discretization noise varies for different performance measures.** For instance, in all of the classifiers, for most of the datasets, we could see that while Recall is impacted heavily (up to 139% for LR in Ask Ubuntu), other performance measures even if impacted significantly, are not at the same magnitude (e.g, only -5.54% for accuracy in LR for Ask Ubuntu).

Discussion: We find that the magnitude of the impact of some performance measures is much greater than other performance measures. Also, from Table 3 we note that for a given dataset and a classifier, some performance measures are impacted even for small amounts of discretization noise (given by the x in the Table 3). Whereas, some other performance measures are more resilient. For instance, in the case of the Mathematics dataset for the RFCM classifier, discarding 15% of the data in the noisy area significantly impacts both Precision and Recall with a large magnitude varying from 8.96% to -32.2%. Whereas even with a drop of 70% data from the noisy area the AUC gets impacted marginally by -1.23%, which indicates that **some performance measures are more resistant to discretization noise than others**.

The different degrees to which different performance measures are impacted can be attributed to the different nature of each performance measure and what they seek to capture. For instance, Precision, Recall, and F-measure focus on capturing how good a classifier performs in predicting one of the classes [61]. Therefore, a potential imbalance in the dataset caused by discarding different amounts of data in the noisy area, even if it is discretization noise (as such noisy points contain useful information too), could impact these measures greatly. Furthermore, Flach shows that these measures are easily impacted by class imbalance [19]. These explain the large impacts that we observe for the Precision, Recall, and F-measures (as opposed to Accuracy which takes both the classes into consideration). Especially, with all of our datasets having a skewed distribution of the dependent variable (average Skewness of the dependent variable of all of the studied datasets is 12.99 and average Kurtosis is 309.4). For example, let us consider the Ask Ubuntu dataset with MT as the discretization threshold. When the whole dataset is used (without any removal of data), the number of data points belonging to each class are equal. However, discarding the data in the noisy area according to Table 2 for MT induces a class imbalance in the dataset. The number of data points belonging to "class1" only makes up 35% of the dataset. Such class imbalance produced by discarding data induces a high degree of impact on class-specific performance measures for the Ask Ubuntu dataset as shown in Table 3.

However, the more balanced measures like AUC and MCC are more robust and insensitive to class distributions [39]. Therefore, AUC and MCC are impacted much less severely by the discretization noise in the dataset. As we can observe from Table 3, the magnitude to which measures like AUC and MCC are impacted is lesser than the class-specific measures like Precision, Recall, and F-measure. For instance, while Recall is impacted by as much as 139%, AUC is impacted only by at most 4.29% across all the datasets and classifiers. Even for the Ask Ubuntu dataset, while the impact on class-specific performance measures is high, the AUC is seldom impacted. A similar trend could be observed for MCC as they consider both false positives and false negatives, even though they are slightly more sensitive to the discretization noise than AUC. As Huang *et al.* [30] and Mossman [51] show balanced measures like AUC are very stable and insensitive to noise hence even a small impact in terms of magnitude (if the effect size is large) could be significant. Therefore though the absolute magnitude of the impact is small for some performance measures like AUC and Accuracy, the true nature of the impact needs to established by the practitioner. Our framework seeks to provide a means for finding and measuring the impact. Finally, similar to other balanced measures we observe that the impact on Brier score is moderate (within 13%) when compared to class-specific performance measures. Brier score being an error metric calculates the mean squared difference between the actual outcome and the assigned probability, which elucidates the distance between the classifier's predictions and the actual classes in probability scale. Table 3 shows that Brier score is universally negatively impacted for a RFCM (signifying an increase in Brier score). Because, though the noisy points contain discretization noise, they also contain useful information (see Section 5.1) and the removal of such information could universally affect the predicted probability scores of classifiers, especially when they are robust to noise [20]. However the Table 1 and Table 2 provided in Appendix C (for performance impact due to discretization noise that is generated by CTT and RT) shows a varied but moderate impact (within 14%) for LR, CART and KNN, which is similar to other balanced performance measures.

In summary, unlike the class-specific performance measures, balanced measures are impacted moderately by the discretization noise.

To conclude, discretization noise impacts different performance measures differently for various datasets with varying magnitudes for different classifiers (0-139%) as shown in Table 3. We also note that **balanced performance evaluation measures are more resilient to the discretization noise whereas class-specific performance evaluation measures are greatly impacted by the discretization noise in the dataset**. Thus the inclusion of discretization noise in the construction of any chosen classifier could be either beneficial or detrimental depending on the performance measure of interest and the dataset at hand. **Such unpredictability demonstrates the need for our framework to better understand when it is advisable to remove the noisy points, and how much of the data in the noisy area is to be removed to avoid impact on the studied performance measure for a chosen classifier.**

The impact of noisy discretization data points is inconsistent across multiple performance measures for different datasets and different. Though the impact on class-specific measures (Precision, Recall,F-measure) is the most pronounced (up to 139%) other performance measures are also consistently impacted at least up to 60% due to the inclusion or exclusion of discretization noise (both positively and negatively). Hence, it is advisable to use our framework beforehand to carefully understand if the noisy discretization data points are to be used or discarded and how much of them needs to be discarded.

4.2 Studying the Impact of Discretization Noise on the Interpretation of a Classifier

Motivation: Many prior studies use classifiers to understand the impact of features on the dependent variable [45], [66]. From Section 4.1, we observe that the data from the noisy area sometimes impacts the performance of a classifier differently for different datasets. This could be because the data in the noisy area contains useful information along with the discretization noise. Therefore removing/including such data may also lead to a misleading interpretation of a classifier. Therefore, we seek to observe how the discretization noise impacts the derived feature importance with our proposed framework to decide if such noisy data points can be safely discarded or should they be included nevertheless.

Approach: We demonstrate the capability of our framework to analyse the impact of discretization noise on the derived feature importance ranks. We adopt the same incremental analysis approach that we adopted in the Section 4.1. But instead of measuring the performance of the classifiers that are constructed by incrementally discarding data from the noisy area, we note the feature importance values of these classifiers (see step 3 of Section 3). We perform an incremental analysis on all the studied datasets for the three different discretization thresholds (MT, CT, RTT) considered in the study in Section 3.2 (see Fig 1 of Appendix) for all the four chosen classifiers (RFCM, LR, CART, KNN).

We measure the derived feature importance values of the chosen classifier trained on various data configurations. We use the Scott-Knott ESD test to rank [23], [41], [64], [65] the features with their feature importance values. To observe whether the derived feature importance ranks vary significantly, we compare the derived feature importance ranks of the classifier that is trained on the whole data and the classifier with $x = limit$ (see Table 2) data points removed from the noisy area for each of studied dataset across all the discretization thresholds. We compute the difference between the derived feature importance ranks for each feature in the dataset and compare them to the null distribution (where the rank difference of each feature is zero) to see if removing the noisy area impacts the derived feature importance ranks of a classifier [55].

For instance, lets consider the Stack Overflow dataset with MT as the discretization threshold when used for training an RFCM. We wish to study the impact of discretization noise on the interpretation of the constructed RFCM. For simplicity, lets consider that the Stack Overflow dataset has only 5 features. For the RFCM that is intially trained on the whole dataset, a derived feature importance ranks is generated for its 5 features ($F_{\text{whole}} = 3, 1, 5, 4, 2$). Following which, an RFCM is trained on the Stack Overflow dataset devoid of the noisy area ($F_{\text{noisy\_area\_removed}} = 2, 1, 3, 2, 4$). The difference between F_{whole} and $F_{\text{noisy\_area\_removed}}$ is *difference* = $1, 0, 2, 2, 2$. If the F_{whole} and $F_{\text{noisy\_area\_removed}}$ are the same, then the *difference* generated would be zero and would imply that the discretization noise does not impact the interpretation of a classifier. If the difference is not 0 (as in our example), we compare the *difference* against a zero distribution (*null_ distribution*= $0, 0, 0, 0, 0$) with a Wilcoxon-signed rank test and Cohen's effect size test to determine whether if the difference is significant or otherwise.

In addition, even if the discretization noise impacts the overall interpretation of a classifier, most researchers and practitioners care only about the top x most important features [27], [40]. Therefore, in this section we present the results of the top 3 features as a showcase. To measure how likely the rank of a feature could shift due to the removal of the discretization noise, we compute the likelihood of rank shifts for top 3 ranks. The importance rank of a feature is ascertained by the median rank of the derived feature importance ranks for a feature of the dataset.

We use a bootstrap analysis to compute the likelihood of rank shifts similar to prior study [64]. Figure 3 outlines the process of estimation of the likelihood of a rank shift. The feature importance ranks that are generated from each

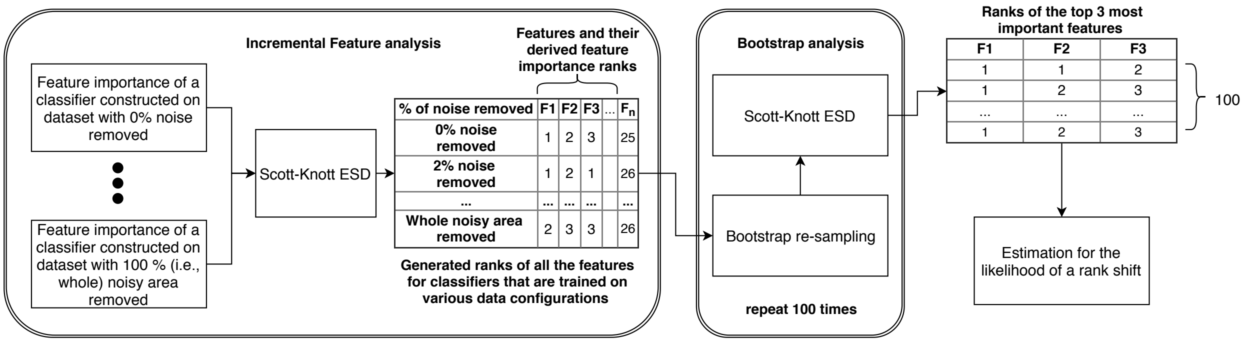

Incremental Feature analysis
Feature importance of a classifier constructed on dataset with 0% noise removed
Feature importance of a classifier constructed on dataset with 100 % (i.e., whole) noisy area removed
Scott-Knott ESD
Features and their derived feature importance ranks
% of noise removed
F1
F2
F3
...
Fn
0% noise removed
1
2
3
25
2% noise removed
1
2
1
26
...
Whole noisy area removed
2
3
3
26
Generated ranks of all the features for classifiers that are trained on various data configurations
Bootstrap analysis
Scott-Knott ESD
Bootstrap re-sampling
repeat 100 times
Ranks of the top 3 most important features
F1
F2
F3
1
1
2
1
2
3
...
...
...
1
2
3
100
Estimation for the likelihood of a rank shift

Fig. 3: The procedure for estimation of the likelihood of a rank shift.

classifier in the incremental feature analysis are taken as the input. These derived ranks are then re-sampled with replacement (i.e., Bootstrap re-sampling). The bootstrap re-sampled feature rank distribution is generated for all the features of each of the studied datasets and they are re-ranked using the Scott-Knott ESD test as shown in the "Bootstrap analysis" part of Figure 3. This process is repeated 100 times. The intuition behind such a procedure is that the bootstrap re-sampling and re-ranking would alleviate possible minor and insignificant fluctuations in the derived feature importance ranks while highlighting the pattern of significant fluctuations in the derived feature importance ranks for a feature, and thereby bringing out its true rank. Now we have 100 derived feature importance ranks for all the features in each of the studied datasets.

These 100 ranks (the output of the "Bootstrap analysis" part of Figure 3) for a feature are used for estimating the likelihood of a rank shift. The likelihood of a rank shift for rank x feature is computed as the percentage of how many ranks are not equal to x. For example, for rank 1 feature, out of 100 times 2 ranks are not equal to 1, then the likelihood of a rank shift for that feature is 0.02. The estimation for the likelihood of rank shifts is done for the features in the top 3 ranks of all the studied datasets. If the likelihood values are high for a particular rank in a dataset, it indicates that discretization noise impacts the derived feature importance and the feature(s) reported at that rank should be interpreted with caution in that dataset.

Results: The overall derived feature importance ranks are impacted by the discretization noise for most of the studied datasets. We only report the impact of discretization noise generated by median based discretization threshold (MT) on the derived feature importance ranks for RFCM due to space constraints, However similar results are noted on all the other classifiers (LR, CART, KNN) on all the studies discretization thresholds (please refer to Table 3, 4, and 5 of Appendix C for other results). We show a comparison of the derived feature importance ranks of an RFCM that is trained on the dataset with and without the noisy area removed in Table 4. The results highlight that for most of the studied datasets (6/7 datasets in the case of the RFCM classifier and for all the datasets for other classifiers), the overall derived feature importance ranks of all the features in a dataset, are significantly impacted (i.e., $p-$value < 0.05) with a non-negligible effect size.

TABLE 4: The likelihood of rank shifts in the top 3 most important ranks (column A) and the comparison of the derived feature importance ranks of an RFCM trained on the whole dataset (Rank$_W$) and the dataset with the noisy area removed (Rank$_{NR}$) (column B).

Dataset	Rank shift likelihood (A)			Rank$_W$ vs. Rank$_{NR}$ (B)	
	Rank 1	Rank 2	Rank 3	p-value	Cohen's d
Stack Overflow	0	0	0	0	-1.29 (L)
Mathematics	0	0	0	0	-2.14 (L)
Ask Ubuntu	0	0	0	0	-2.84 (L)
Super User	0	0	0	0.01	-0.74 (M)
Patch	0	0	0	0.03	-0.62 (M)
Bug-delay	0	0	0	0	-1.39 (L)
App-rating	0	0	0	1	-0.30 (S)

However, the derived feature importance ranks of the top 3 features are not impacted by the discretization noise. To specifically understand how much of an impact that the discretization noise has on the most important features, we computed the likelihood of a rank shift for the top 3 important features. We present the results in Table 4. **For all the datasets, the ranks of the most important features (i.e., all features at rank 1, rank 2 and rank 3) are not impacted due to discretization noise**. We further observe that these trends are not specific to RFCM and the discretization noise generated by MT. From Table 3, 4, and 5 (please see Appendix C) we observe that the most important features are not impacted by the discretization noise generated in the dataset for any of the studied classifiers.

In summary, though the overall ranks of derived feature importance are impacted by the discretization noise in the dataset, the top 3 (most important) features that most researchers and practitioners focus on [27], [40], [55], [63] are not impacted by the discretization noise. **Therefore, we suggest that the decision of either including or removing**

the data in the noisy area could be exclusively arrived at from the results of Section 4.1 without being worried much about its impact on interpretation. Nevertheless these results might vary for other settings (e.g., other datasets or classifiers) and our framework is able to provide a case by case guidance.

The discretization noise (generated by our studied discretization thresholds) does not impact the derived feature importance of any of the top 3 features yet it impacts the overall derived feature importance ranks.

5 Discussion

5.1 Why does a classifier trained on the whole dataset (with discretization noise) sometimes perform better than the classifier trained on data devoid of discretization noise?

From the Section 4, we observe that excluding data from the noisy area, i.e., data with discretization noise, sometimes negatively impacts the performance of a classifier. We seek to understand this counter intuitive phenomena. Furthermore, in this section, we also remark about why the top 3 important features of a classifier are not impacted by discretization noise.

We hypothesize that a classifier in some cases is able to capture the signal from the noisy points in spite of the discretization noise. Doing so, allows the classifier to capture more information from the noisy data points, in addition to information available from the clean data. Therefore, discarding those noisy points negatively impacts the performance of a classifier. To examine our hypothesis, we start by constructing classifiers with data points from the noisy area (generated with MT) and test them on the data from the extremes and noisy areas. Such an experiment helps us understand whether the data with the discretization noise contains any useful information.

We follow the steps outlined in our framework (see Section 3) to construct the classifiers on the noisy area and generate the out-of-sample test sets from the extremes and noisy area separately. For this experiment, we construct an RFCM and observe its performance on the AUC measure. We do so just on RFCM as our intention is only to analyse if the noisy area contains useful information and our results on RFCM could help us test it succinctly. Furthermore, in this section we report only the AUC measure as from Table 3 we could observe that AUC measure is the most resilient performance measure that has the least impact across all the studied classifiers and we wish to report the impact on the most resilient measure. A high AUC would therfore objectively justify the presence of useful information. However, all the other performance measures follow the same trend.

The noisy area contains useful information that might help improve the performance of RFCMs. From Table 5, we observe that the RFCMs that are trained on the noisy area perform extremely well on the extremes for 5 out of the 7 studied datasets. Three datasets have an AUC that is larger than 0.95. For instance, in the Stack Overflow dataset, the RFCMs that are trained on the noisy area have an AUC of 0.96 when tested on the extreme areas.

While the previous experiments show that noisy area contains useful information, we cannot conclusively establish if the contained information in the noisy points amidst the discretization noise could be successfully used by the classifiers. To test if the studied classifiers can use the information contained in the noisy area, we construct classifiers that are trained with extremes and data from the noisy area (in contrary to the previous experiment, where we trained only on the noisy area) and then add increasing amounts of data from the noisy area. If the performance of a classifier does not degrade significantly (some degradation should be expected) with the increased amount of noise, it may indicate that the classifier is capable of capturing a signal as long as there is enough information in the data. Therefore, we could infer that despite the presence of discretization noise, the data points in the noisy area provide an additional signal to the classifier and the exclusion of the noisy area, negatively impacts the performance of the classifier. On the other hand, if there is a drastic and significant performance degradation of the classifier, it would then invalidate our hypothesis that classifiers are able to learn an additional signal in spite of discretization noise.

We perform the experiment by setting up a simulation study with the help of our framework similar to the previous experiment, where we train all the classifiers on the extremes with different amount data from the noisy area. We train our classifiers on four different data configurations which are given by $(extremes + (noisyarea + over\_sample\% * (noisyarea)))$, where the $over\_sample$ takes values of $0, 100, 200, 300$. We oversample different amounts of data from the noisy area while keeping the amount of the data from extremes constant. We build the classifiers on four data configurations i.e., $(extremes + (noisyarea + 0\% * (noisyarea)))$, $(extremes + (noisyarea + 100\% * (noisy\_area)))$, $(extremes + (noisyarea + 200\% * (noisy\ area)))$, and $(extremes + (noisyarea + 300\% * (noisy\ area)))$. For instance, in the Ask Ubuntu dataset from our study for MT, the extremes have 1,427 data points and the noisy area has 2,711 data points as shown in the Table 2. Therefore for the first configuration, we would have 4,138 data points, of which 65% is comprised of noisy points. Therefore for Ask Ubuntu dataset, our four data configuration consist of 65%, 79%, 85% and 88% noisy points respectively along with the clean data from the extremes. (See Figure 3 of Appendix C explaining the overall experimental setup)

TABLE 5: Median performance (AUC) of the RFCMs on the different regions of the data.

Training data → Testing data	Stack Overflow	Mathematics	Ask Ubuntu	Super User	Patch	Bug-delay	App-rating
Noisy area → Extremes	0.96	0.96	0.86	0.86	0.98	0.69	0.76

The performance of a classifier that is constructed on the aforementioned data configurations is evaluated on the out-of-sample test data from the extremes. Note that the out-of-sample test data that is obtained from the extremes, is not used in the training phase and is only used for testing the constructed classifier. The experimental setup for constructing a classifier is similar to that of our framework, as outlined in Section 3. Finally, we also capture the derived feature importance ranks and observe the likelihood of rank shifts for the top 3 most important features as outlined in Section 3 and Section 4.2.

Adding data from the noisy area to the training data does not greatly impact the performance of a classifier. We report the results in Table 6. The columns in Table 6 correspond to different data configurations that we discussed earlier. We observe that the median AUC of a classifier that is trained on the dataset without noise is quite close to the AUC of a classifier that is trained on data with 300% noise. For RFCM, LR and CART classifiers, even an addition of 300% of data from the noisy area only impacts the AUC within 6% as we can observe from Table 6. Even in the case of the KNN classifier, which is an instance-based classifier that is traditionally more sensitive to noise in the data [3], gets impacted only by 11% in terms of AUC even with the addition of up to 300% data points from the noisy area. These results signify that the performance of the constructed classifiers on the extremes does not degrade significantly even when there is 300% (at least X% of the data is noisy) data from the noisy area in addition to data from the extremes.

Furthermore, we also note that the likelihood of rank shifts for top 3 ranks between classifiers that are trained on the first configuration (0%) and the last configuration (300%) is 0. Which further reinforces the validity of our hypothesis that the classifiers are able to capture the signal in the noisy points despite the discretization noise and the most important features contributed by the true signal in the underlying data are not perturbed by the discretization noise.

In summary, We establish that the noisy area does contain some useful information. Further, we observe only a maximum performance drop of 11% across 7 datasets for all of the studied classifiers (with less than 6% performance drop for RFCM, LR and CART) with the addition of as much as 300% of data from the noisy area, where at least more than 67% of the dataset is noisy. This suggests that a classifier is able to capture the information in the data in spite of the noise, thereby explaining why the exclusion of data points from the noisy area sometimes impacts the performance of a classifier. In addition, the likelihood of rank shift for the top 3 most important features is 0 for all the classifiers, signifying that the discretization noise even in such high quantities does not impact the interpretation of the classifiers.

TABLE 6: Performance comparison (in AUC) of classifiers that are trained on different data configurations.

Classifier	Dataset	0% Noise	100% Noise	200% Noise	300% Noise
RF	SO	0.96	0.96	0.96	0.96
	MA	0.96	0.96	0.96	0.96
	AU	0.86	0.85	0.84	0.84
	SU	0.86	0.86	0.87	0.85
	PT	0.99	0.99	0.99	0.99
	BD	0.69	0.68	0.66	0.65
	AR	0.76	0.77	0.76	0.76
LR	SO	0.92	0.93	0.92	0.92
	MA	0.93	0.92	0.92	0.92
	AU	0.78	0.77	0.76	0.76
	SU	0.79	0.79	0.78	0.77
	PT	0.98	0.98	0.98	0.97
	BD	0.68	0.68	0.66	0.66
	AR	0.72	0.72	0.71	0.71
CART	SO	0.89	0.86	0.84	0.83
	MA	0.87	0.86	0.87	0.85
	AU	0.74	0.69	0.67	0.66
	SU	0.72	0.70	0.69	0.68
	PT	0.94	0.89	0.90	0.90
	BD	0.64	0.62	0.60	0.60
	AR	0.64	0.63	0.62	0.62
KNN	SO	0.80	0.75	0.71	0.69
	MA	0.80	0.75	0.72	0.70
	AU	0.62	0.61	0.59	0.58
	SU	0.69	0.64	0.62	0.61
	PT	0.83	0.82	0.81	0.80
	BD	0.55	0.52	0.51	0.50
	AR	0.60	0.57	0.56	0.55

Datasets: SO- Stack Overflow, MA- Mathematics, AU- Ask Ubuntu, SU- Super User, PH- Patch, BD- Bug-delay, AR- App-rating

5.2 Why does inclusion of discretization noise sometimes negatively impacts the performance of a classifier?

Contrary to Section 5.1, in this section, we seek to understand why the inclusion of discretization noise negatively impacts the performance of some classifiers. From Table 3 we observe that for all the studied classifiers, the inclusion of discretization noise sometimes negatively impacts the performance of a classifier even though Section 5.1 shows that data in the noisy area has useful information and classifiers are capable of leveraging it. From Table 3 we also observe that for all the studied classifiers and datasets, at least one of the performance measure is negatively impacted. We hypothesize that such a negative impact could be due to the high complexity (less discriminative power) of the noisy points around the discretization threshold,

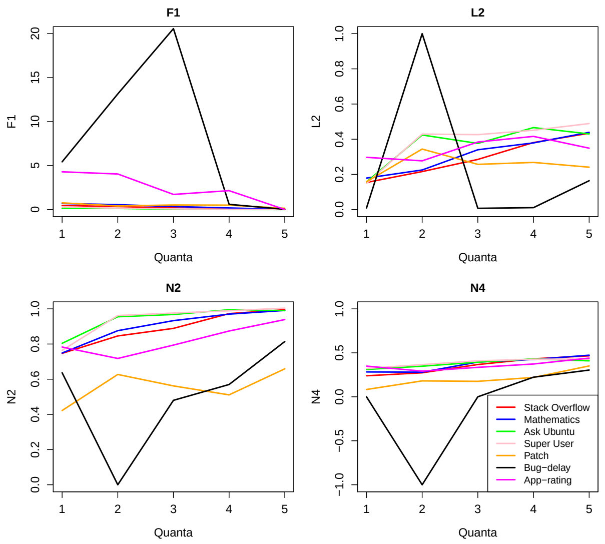


Fig. 4: Data complexity across quantum for the studied datasets.

despite containing useful information. We arrive at such a hypothesis as prior studies show that it is difficult for the classifiers to perform well if the complexity of the data is high, irrespective of the contained information [4], [29]. Thus, we are interested in exploring if the negative impact in the performance of a classifier due to the inclusion of discretization noise is because of the high complexity of the data points around the discretization threshold (noisy area).

Ho and Basu [29] provide complexity metrics to measure the complexity of data. We use these complexity metrics to measure the complexity of different regions of our data. From the multiple methods that are proposed by Ho and Basu [29], we choose Fisher's discriminant ratio (F1), linear separability (L2), mixture identifiability (N2) and nonlinearity (N4) as they are simple to explain and easy to interpret. We briefly explain the metrics that we choose in the Table 6 of Appendix C. See the study by Ho and Basu [29] for more details about the computation of these metrics.

We use the abovementioned measures to compute how the complexity of the dataset changes across data points as we move from the extremes to the noisy areas. We first discretize the data into "class 1" and "class 2" classes as outlined in Section 3.2. We then transform the continuous dependent variable using the Box-Cox transformation [58] to alleviate the skew and increase the spread of the distribution of the dependent variable. We then split the data into 5 quanta for each class using the **bin** function in R. We do so to compartmentalize the data in relation to the continuous dependent variable and analyse the changes to the complexity of the data points as we move closer to the discretization threshold (we choose MT) for our case study. We would not be able to observe how the complexity changes in different areas of the data without such compartmentalization of the data. The choice of using 5 quanta is so that the compartmentalization is neither too granular nor too encompassing. The 1stquantum contains most of the data from extremes and the 5th quantum contains most of the data from the noisy area, whereas 2nd to 4th quantum roughly contain an equivalent amount of data points in between. Finally, we compute the above-mentioned data complexity metrics for the data points in each of these quantum and plot the results. (See Figure 2 of Appendix C explaining the overall experimental setup)

From Figure 4, we observe that as we move from the extremes (1st quantum) towards the noisy area (5th quantum), we see a steady increase in data complexity across all complexity measures for all the datasets except for the Bug-delay dataset. We see that all four complexity measures (i.e., Fischer's discriminant ratio, linear separability, mixture identifiability, and nonlinearity) are very high for the data points in the noisy area compared to the data points in the extremes, and the inclusion of such complex data makes it very hard for the classifiers to perform well. The steady increase in data complexity as we move across the quantum can be attributed to the steady increase of the discretization noise in the dataset as we move from the 1st quantum to the 5th quantum (the extremes to the noisy area). Therefore, when the discretization noise (the data points in the noisy area with high complexity) is discarded, the performance of some of the classifiers increases.

The lower complexity in the 2nd quantum for the Bug-delay dataset does not impact our findings. It is due to the way the dataset is split, the BoxCox transformation aims to spread the dependent variable sufficiently so that the class-wise binning yields data in all quantum. But for the Bug-delay dataset, when we split the data into quantum, we observe that the 2nd quantum has data points that only belong to "class 2" and not "class 1" because the quantum 2 for the Bug-delay dataset contains only data points belonging to "class 2", its complexity is very low, which is reflected in Figure 4. But this phenomenon has no bearing on our findings that the quantum containing high volumes of discretization noise (q5) is more complex than the quantum containing extremes data (q1) and thereby discarding them sometimes improves the performance of the classifiers.

Hence, the presence of a high volume of discretization noise in the noisy area increases the data complexity, which in turn results in the decreased performance of a classifier that is trained with discretization noise, despite containing useful information. Therefore, in some cases, the performance of a classifier benefits from discarding the data points with from the noisy area.

6 Guidelines for Using Our Framework

We explain in detail our framework in Section 3 and demonstrate how it is used to study the impact of discretization noise on the performance and interpretation of a classifier in Section 4. Furthermore this section, we provide practical guidelines on how to use our framework and the best practices to follow.

Figure 5 shows the involved steps, step-wise outputs, user considerations at each step and the overall workflow of our framework. A user can follow the steps one by one when they are given a dataset to study.

6.1 Performance impact estimation

A classifier constructed with increasing amounts of discretization noise being removed is constantly compared against the classifier constructed on the whole dataset with a Wilcoxon signed-rank test and a Cohen's effect size test as outlined in Section 4.1. If for the chosen performance measure, the impact is statistically significant with non-negligible effect size, then the amount of noisy points to be discarded and the magnitude of the performance impact due to the discretization noise is reported to the user. If the discretization noise does not impact the chosen performance measure then our framework would output 0 (suggesting no data needs to discarded) and recommend the use of the whole dataset as outlined in the workflow of Figure 5.

However, the choice of the performance measure to focus on and how much of an improvement/impact that one should consider actionable depends entirely on the context. For instance, in a dataset of 100 data points, if 90 data points belong to "class 1" and 10 data points belong to "class 2", then accuracy (w.r.t "class 1") would be 90% even if the classifier always predicts "class 1" for all the 100 data points. Therefore, other balanced performance measures like AUC might be required.

6.2 Interpretation impact estimation

The derived feature importance ranks of the classifiers constructed on datasets with varying amounts of discretization noise being removed is computed. These computed ranks are compared to see if the derived feature importance ranks change between the classifiers that are constructed on the whole dataset and the ones that are constructed on the dataset without discretization noise (please see Section 4.2 and Figure 5). Our framework then checks if the differences of the derived feature importance ranks between the classifiers that are trained on the dataset with and without the discretization noise are statistically significant. If they are, our framework also calculates the likelihood of rank shifts for the top n features (between the classifier trained on the whole dataset and the data with the framework recommended amount of data from noisy area removed). In summary, our framework reports if there is an impact of discretization noise on the overall interpretation and the likelihood of rank shifts in for the top n features to the user.

6.3 Best practices

In this section, we recommend the key best practices for others to follow when they are discretizing the data using an artificial threshold. From Section 4.1 we note that performance of all the classifiers is impacted across all performance measures differently and that class-specific performance measures (e.g., Precision and Recall) are more sensitive to discretization noise than others. Therefore, we recommend the following best practices for the researchers and practitioners:

1) Irrespective of the choice of a classifier, if one intends to discretize the continuous dependent variable into artificial classes, one should use our framework to analyse if they should use the whole dataset or discard the discretization noise.
2) Class-specific performance measures are more sensitive to discretization noise. Therefore, instead of discarding a fixed amount of data like some of the previous studies [28], [67], [69], we recommend the use of our framework to estimate how much discretization noise that one should discard to avoid any negative impacts on their performance measure of choice. Hence, one could avoid the unwanted loss of data.
3) If our framework reports a high likelihood of a rank shift for one of the top n features, we recommend not to trust the feature importance rank for that particular feature and seek the opinion of the domain expert. However, if our framework detects any impact in the overall interpretation along with the performance, then we recommend the use of the interpretation of the best performing model.

7 Threats to Validity

External Validity Many of the prior studies highlight that different classifiers have different performance on the same data [23], [55]. So the choice of classifiers might impact the findings of our study, as we only use four classifiers (RFCM, LR, CART, and KNN) in our analysis. However, the chosen classifiers represent a diverse range of families: statistical family, nearest neighbor family, Decision tree family and ensemble family, i.e., 4/6 of the common classifier families as outlined by Lessmann *et al.* [39]. We left out representative approaches from the neural networks family and the support vector machine family. We did so, as classifiers from these families typically do not have a default feature importance measure.

Construct Validity Threats to construct validity pertains to the suitability of the measures that are used in our study. In our study, we study the impact of discretization noise that is generated in the dataset by using three different discretization thresholds as mentioned in Section 3 and the results might vary when another threshold is used. However, the three chosen discretization threshold computation methods

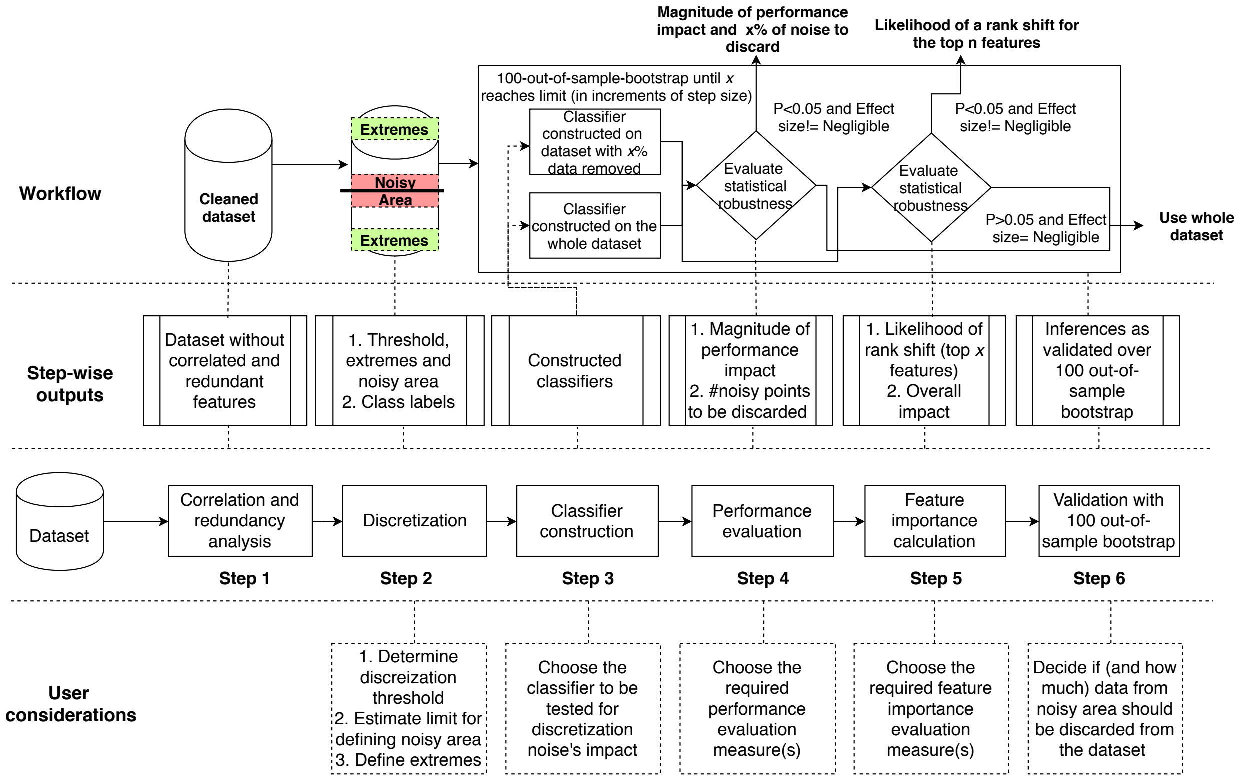

Magnitude of performance impact and x% of noise to discard
Likelihood of a rank shift for the top n features
100-out-of-sample-bootstrap until x reaches limit (in increments of step size)
Classifier constructed on dataset with x% data removed
Classifier constructed on the whole dataset
P<0.05 and Effect size!= Negligible
Evaluate statistical robustness
P<0.05 and Effect size!= Negligible
Evaluate statistical robustness
P>0.05 and Effect size= Negligible
Use whole dataset
Workflow
Cleaned dataset
Extremes
Noisy Area
Extremes
Step-wise outputs
Dataset without correlated and redundant features
1. Threshold, extremes and noisy area
2. Class labels
Constructed classifiers
1. Magnitude of performance impact
2. #noisy points to be discarded
1. Likelihood of rank shift (top x features)
2. Overall impact
Inferences as validated over 100 out-of-sample bootstrap
Dataset
Correlation and redundancy analysis
Discretization
Classifier construction
Performance evaluation
Feature importance calculation
Validation with 100 out-of-sample bootstrap
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
User considerations
1. Determine discreization threshold
2. Estimate limit for defining noisy area
3. Define extremes
Choose the classifier to be tested for discretization noise's impact
Choose the required performance evaluation measure(s)
Choose the required feature importance evaluation measure(s)
Decide if (and how much) data from noisy area should be discarded from the dataset

Fig. 5: User considerations and workflow that are associated with each of the steps of our framework.

(MT, CT, RTT) discretize the dependent variables differently and represent the most common ways of unsupervised discretization of the dependent variable. Our framework enables others to explore other discretization thresholds in a systematic manner.

Another construct validity in our study is the choice of the $limit$ parameter for deciding the size of the noisy area in each dataset. We used the limit values that are generated by our automated noisy area estimation algorithm as given in Section 3. Though such an algorithm estimates the noisy area quantitatively based on a complexity measure, it might not consider the inherent dataset characteristics and bias. We acknowledge that it could be a potential threat and we urge researchers to explore various limits, or with limits that are established by domain experts and ratify our findings. Future studies should use our framework to test different values for the limit.

Finally, in this section, we wish to reiterate to the readers that our framework enables the researchers and practitioners to fiddle with any components and try a variety of combinations. We only define the needed analysis that is to be done, so that the drawn observations are valid.

8 Conclusion

In this paper, we propose a framework to systematically and rigorously analyse the impact of discretization noise on the performance and interpretation of a classifier within the context of their own domain. We perform a case study on a variety of software engineering datasets and we find that:

1) Discretization noise impacts the different performance measures of classifiers differently across the different datasets. We observe that discretization noise leads to an up to 139% performance differences across various performance measures across all the studied classifiers. Hence it is very important for researchers and practitioners to use our framework to analyse the impact of discretization noise on the classifier's for before either including or discarding it in their analysis.
2) When discretization noise negatively impacts the performance of a classifier, our framework provides a systematic and statistically robust way to estimate exactly how much data should be discarded to avoid discretization noise without incurring unwarranted data loss.
3) Though discretization noise impacts the overall derived feature importance ranks of a classifier, it does not impact the derived feature importance ranks of the top 3 ranks for our case studies. Our framework provides a

case by case guidance for others who wish to explore its use for their own case studies.

R package and User Guideline: We provide an R package to enable others to use our framework to analyse the impact of discretization noise. Furthermore, we provide a user guideline, a step-by-step walkthrough and the best practices of using our framework in Section 6.

References

[1] W. Abdelmoez, M. Kholief, and F. M. Elsalmy, "Bug fix-time prediction model using naïve bayes classifier," in *Computer Theory and Applications (ICCTA), 2012 22nd International Conference on*. IEEE, 2012, pp. 167–172.

[2] M. Abdelwahab and C. Busso, "Supervised domain adaptation for emotion recognition from speech," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5058–5062.

[3] D. W. Aha and D. F. Kibler, "Noise-tolerant instance-based learning algorithms." in *IJCAI*. Citeseer, 1989, pp. 794–799.

[4] C. O. Alm, D. Roth, and R. Sproat, "Emotions from text: Machine learning for text-based emotion prediction," in *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP-05)*. Association for Computational Linguistics, 2005, pp. 579–586.

[5] D. G. Altman and P. Royston, "The cost of dichotomising continuous variables," *Bmj*, vol. 332, no. 7549, p. 1080, 2006.

[6] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PloS one*, vol. 12, no. 6, p. e0177678, 2017.

[7] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[8] G. W. Brier, "Verification of forecasts expressed in terms of probability," *Monthey Weather Review*, vol. 78, no. 1, pp. 1–3, 1950.

[9] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, 2009.

[10] J. Cohen, "The cost of dichotomization," *Applied psychological measurement*, vol. 7, no. 3, pp. 249–253, 1983.

[11] J. Cohen, *Statistical power analysis for the behavioral sciences . Hilsdale*. Hillsdale, N.J. : L. Erlbaum Associates, 1988, vol. 2.

[12] N. V. Dawson and R. Weiss, "Dichotomizing continuous variables in statistical analysis a practice to avoid," 2012.

[13] M. A. de Almeida, H. Lounis, and W. L. Melo, "An investigation on the use of machine learned models for estimating correction costs," in *Proceedings of the 20th IEEE/ACM international conference on Software engineering (ICSE-98)*. IEEE, 1998, pp. 473–476.

[14] J. DeCoster, M. Gallucci, and A.-M. R. Iselin, "Best practices for using median splits, artificial categorization, and their continuous alternatives," *Journal of experimental psychopathology*, vol. 2, no. 2, pp. jep–008 310, 2011.

[15] J. DeCoster, A.-M. R. Iselin, and M. Gallucci, "A conceptual and empirical examination of justifications for dichotomization." *Psychological methods*, vol. 14, no. 4, p. 349, 2009.

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

[17] D. P. Farrington and R. Loeber, "Some benefits of dichotomization in psychiatric and criminological research," *Criminal behaviour and mental health*, vol. 10, no. 2, pp. 100–122, 2000.

[18] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27–38, 2009.

[19] P. A. Flach, "The geometry of roc space: understanding machine learning metrics through roc isometrics," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 194–201.

[20] A. Folleco, T. M. Khoshgoftaar, J. Van Hulse, and L. Bullard, "Software quality modeling: The impact of class noise on the random forest classifier," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-08)*. IEEE, 2008, pp. 3853–3859.

[21] S. Garcia, J. Luengo, J. A. Saez, V. Lopez, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, 2012.

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

[23] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE-15)*. IEEE, 2015, pp. 789–800.

[24] B. Ghotra, S. McIntosh, and A. E. Hassan, "A large-scale study of the impact of feature selection techniques on defect classification models," in *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE, 2017, pp. 146–157.

[25] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE-10)*, vol. 1. IEEE, 2010, pp. 495–504.

[26] F. E. Harrell, *Missing Data*. New York, NY: Springer New York, 2001, pp. 41–52.

[27] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 263–272.

[28] S. Hassan, C.-P. Bezemer, and A. E. Hassan, "Studying bad updates of top free-to-download apps in the google play store," *IEEE Transactions on Software Engineering*, 2018.

[29] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 289–300, 2002.

[30] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[31] O. Jalali, T. Menzies, and M. Feather, "Optimizing requirements decisions with keys," in *Proceedings of the 4th international workshop on Predictor models in software engineering*. ACM, 2008, pp. 79–86.

[32] Y. Jiang, B. Cukic, and T. Menzies, "Can data transformation help in the detection of fault-prone modules?" in *Proceedings of the 2008 workshop on Defects in large software systems*. ACM, 2008, pp. 16–20.

[33] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast? case study on the linux kernel," in *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR-13)*. IEEE, 2013, pp. 101–110.

[34] T. Judd, K. Ehinger, F. Durand, and A. Torralba, "Learning to predict where humans look," in *Computer Vision, 2009 IEEE 12th international conference on*. IEEE, 2009, pp. 2106–2113.

[35] T. M. Khoshgoftaar, S. Zhong, and V. Joshi, "Enhancing software quality estimation using ensemble-classifier based noise filtering," *Intelligent Data Analysis*, vol. 9, no. 1, pp. 3–27, 2005.

[36] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceedings of the 33rd ACM/IEEE International Conference on Software Engineering (ICSE-11)*, 2011, pp. 481–490.

[37] H. C. Kraemer, A. Noda, and R. O'Hara, "Categorical versus dimensional approaches to diagnosis: methodological challenges," *Journal of psychiatric research*, vol. 38, no. 1, pp. 17–25, 2004.

[38] R. Krishna, T. Menzies, and L. Layman, "Less is more: Minimizing code reorganization using xtree," *Information and Software Technology*, vol. 88, pp. 53–66, 2017.

[39] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[40] 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 *Proceedings of the 2013 international conference on Software Engineering*. IEEE Press, 2013, pp. 372–381.

[41] H. Li, W. Shang, Y. Zou, and A. E. Hassan, "Towards just-in-time suggestions for log changes," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1831–1865, 2017.

[42] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[43] R. C. MacCallum, S. Zhang, K. J. Preacher, and D. D. Rucker, "On the practice of dichotomization of quantitative variables." *Psychological methods*, vol. 7, no. 1, p. 19, 2002.

[44] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th IEEE/ACM Working Conference on Mining Software Repositories (MSR-14)*. ACM, 2014, pp. 192–201.

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

[46] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaganeli, "The inductive software engineering manifesto: principles for industrial data mining," in *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*. ACM, 2011, pp. 19–26.

[47] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.

[48] T. Menzies, D. Owen, and J. Richardson, "The strangest thing about software," *Computer*, vol. 40, no. 1, 2007.

[49] A. Mockus, *Missing Data in Software Engineering*. Springer, 2008, pp. 185–200.

[50] A. Mockus, "Organizational volatility and its effects on software defects," in *Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of software engineering (FSE-10)*. ACM, 2010, pp. 117–126.

[51] D. Mossman, "Assessing predictions of violence: being accurate about accuracy." *Journal of consulting and clinical psychology*, vol. 62, no. 4, p. 783, 1994.

[52] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, 2017.

[53] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 452–463.

[54] D. F. Nettleton, A. Orriols-Puig, and A. Fornells, "A study of the effect of different types of noise on the precision of supervised learning techniques," *Artificial intelligence review*, vol. 33, no. 4, pp. 275–306, 2010.

[55] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," in *Proceedings of the 14th IEEE/ACM International Conference on Mining Software Repositories (MSR-17)*, 2017, pp. 135–145.

[56] P. Royston, D. G. Altman, and W. Sauerbrei, "Dichotomizing continuous predictors in multiple regression: a bad idea," *Statistics in medicine*, vol. 25, no. 1, pp. 127–141, 2006.

[57] D. D. Rucker, B. B. McShane, and K. J. Preacher, "A researchers guide to regression, discretization, and median splits of continuous variables," *Journal of Consumer Psychology*, vol. 25, no. 4, pp. 666–678, 2015.

[58] R. Sakia, "The box-cox transformation technique: a review," *The statistician*, vol. 41, no. 2, pp. 169–178, 1992.

[59] J. Schumann, K. Gundy-Burlet, C. Pasareanu, T. Menzies, and T. Barrett, "Software v&v support by parametric analysis of large software simulation systems," in *2009 IEEE Aerospace Conference*, 2009.

[60] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data," *Information Sciences*, vol. 259, no. NA, pp. 571–595, 2014.

[61] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation," in *Australasian joint conference on artificial intelligence*. Springer, 2006, pp. 1015–1021.

[62] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 2018, pp. 286–295.

[63] 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 *Proceeding of the 37th IEEE/ACM International Conference on Software Engineering (ICSE-15)*, 2015, p. 812823.

[64] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th IEEE/ACM International Conference on Software Engineering (ICSE-16)*, 2016, pp. 321–332.

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

[66] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th IEEE/ACM International Conference on Software Engineering (ICSE-16)*. IEEE, 2016, pp. 1039–1050.

[67] 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 *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME-15)*. IEEE, 2015.

[68] H. Wang and M. Song, "Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming," *The R journal*, vol. 3, no. 2, p. 29, 2011.

[69] S. Wang, T.-H. Chen, and A. E. Hassan, "Understanding the factors for fast answers in technical q&a websites," *Empirical Software Engineering*, vol. PP, 2017.

[70] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[71] Y. Yang, G. I. Webb, and X. Wu, "Discretization methods," in *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 101–116.

[72] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE-12)*. IEEE, 2012, pp. 225–234.

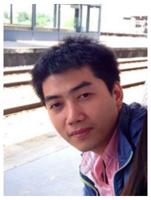
[73] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 182–191.

[74] Q. Zheng, A. Mockus, and M. Zhou, "A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 637–648.

[75] X. Zhu and X. Wu, "Class noise vs. attribute noise: A quantitative study," *Artificial Intelligence Review*, vol. 22, no. 3, pp. 177–210, 2004.

[76] X. Zhu, X. Wu, and Q. Chen, "Eliminating class noise in large datasets," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. AAAI, 2003, pp. 920–927.

Gopi Krishnan Rajbahadur is currently a Ph.D. student in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. He received his BE in computer Science and Engineering from SKR Engineering college, Anna University, India. He also spent close to five years working as a data scientist in various software corporations in both India and Canada. His research interests include interpretable machine learning, mining software repositories and safe data science. More information at: http://gopikrishnanrajbahadur.me/



Shaowei Wang is a postdoctoral fellow in the Software Analysis and Intelligence Lab (SAIL) at Queens University, Canada. He obtained his PhD from Singapore Management University, and BSc from Zhejiang University. His research interests include code mining and recommendation, software maintenance, developer forum analysis, and mining software repositories. More information at:https://sites.google.com/site/wswshaoweiwang/

Yasutaka Kamei is an assistant professor at Kyushu University in Japan. He received his BE degree in informatics from Kansai University, and PhD degree in information science from the Nara Institute of Science and Technology. He was a research fellow of the JSPS (PD) from July 2009 to March 2010. From April 2010 to March 2011, he was a postdoctoral fellow at Queens University in Canada. His research interests include empirical software engineering, open source software engineering, and mining software repositories (MSR). He is a member of the IEEE More information at http://posl.ait.kyushu-u.ac.jp/ kamei/

Ahmed E. Hassan is an IEEE fellow and member, ACM influential educator, an NSERC Steacie Fellow, a Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queens University, Canada. His industrial experience includes helping architect the Blackberry wireless platform, and working for IBM Research at the Almaden Research Lab and the Computer Research Lab at Nortel Networks. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Dr. Hassan serves on the editorial board of the IEEE Transactions on Software Engineering, the Journal of Empirical Software Engineering, and PeerJ Computer Science. He spearheaded the organization and creation of the Mining Software Repositories (MSR) conference and its research community. More information at https://sail.cs.queensu.ca/