Leveraging Feature Bias for Scalable Misprediction Explanation of Machine Learning Models

Jiri Gesi
Donald Bren School of ICS
University of California, Irvine
Irvine, USA
fjiriges@uci.edu

Xinyun Shen
Donald Bren School of ICS
University of California, Irvine
Irvine, USA
xinyuns@uci.edu

Yunfan Geng
Donald Bren School of ICS
University of California, Irvine
Irvine, USA
yunfang2@uci.edu

Qihong Chen
Donald Bren School of ICS
University of California, Irvine
Irvine, USA
chenqh@uci.edu

Iftekhar Ahmed
Donald Bren School of ICS
University of California, Irvine
Irvine, USA
iftekha@uci.edu

***Abstract*—Interpreting and debugging machine learning models is necessary to ensure the robustness of the machine learning models. Explaining mispredictions can help significantly in doing so. While recent works on misprediction explanation have proven promising in generating interpretable explanations for mispredictions, the state-of-the-art techniques "blindly" deduce misprediction explanation rules from all data features, which may not be scalable depending on the number of features. To alleviate this problem, we propose an efficient misprediction explanation technique named Bias Guided Misprediction Diagnoser (BGMD), which leverages two prior knowledge about data: a) data often exhibit highly-skewed feature distributions and b) trained models in many cases perform poorly on subdataset with under-represented features. Next, we propose a technique named MAPS (Mispredicted Area UPweight Sampling). MAPS increases the weights of subdataset during model retraining that belong to the group that is prone to be mispredicted because of containing under-represented features. Thus, MAPS make retrained model pay more attention to the under-represented features. Our empirical study shows that our proposed *BGMD* outperformed the state-of-the-art misprediction diagnoser and reduces diagnosis time by 92%. Furthermore, *MAPS* outperformed two state-of-the-art techniques on fixing the machine learning model's performance on mispredicted data without compromising performance on all data. All the research artifacts (i.e., tools, scripts, and data) of this study are available in the accompanying website [1].**

***Index Terms*—machine learning, data imbalance, rule induction, misprediction explanation**

I. Introduction

Machine learning (ML) techniques, similar to other fields, have been gaining popularity in software engineering tasks. Defect prediction [25], [29], [30], automatic code completion [15], [51], predicting merge conflicts [44], and synthesizing and repairing programs [48], [53], [56], [58] are some examples. While these models' overall performance is good, interpreting and debugging them is a challenge, which also impedes the real-world usage of these models [22], [37].

Specific characteristics of ML systems make them difficult to debug. The opacity of the learned models, high dimensionality of the input data, dependence on the data quality [8], [12] are a few of them. Data often exhibits highly-skewed class distributions (class imbalance), i.e., most data belong to the majority class, and the minority class only contains a small number of instances [52]. To complicate things even more, imbalance not only happens at class level but also on data features [25]. Since ML models are usually trained by minimizing average training loss on all data, which is also known as Empirical Risk Minimization (*ERM*), a feature imbalance can lead to models that achieve low test error but still incur high error on instances that contain under-represented features. For example, Gesi et al. [25] showed that in software defect prediction tasks, comparing with most commits, the prediction model often performs significantly worse for the commits, which involve a large number of modified files since the number of training instances with a large number of modified files is very few during training. The similar situation has been observed in other fields as well, such as a vehicle recognition model usually fails to detect crashed cars as a car because of very few crashed car instances in the training dataset [57]. These varying granularities of imbalance (i.e., class vs. feature) severely impact the robustness of models.

To ensure the robustness of the models, the explanation generation technique has been proven to be one of the most effective ways as it can help in explaining the rationale for a prediction [18], [19], [24], [45], [49]. Researchers have been trying various explanation generation techniques [45], [49] to shed light on the global behavior of a model either by highlighting which features are the most important or by constructing a surrogate and simpler model that emulates a complex model. However, except for [16], [17], none of the work focused on explaining the mispredictions of a ML model.

In the most recent work, Cito et al. [17] proposed a technique named *EXPLAIN*, which generates a set of decision rules based on features and mispredicted instances to explain the reasons for mispredictions, i,e., Misprediction Explanation (*ME*) rules. However, one of the limitations of *EXPLAIN* is that its generated *ME* rules are deduced "blindly" from all features. Since ML models can have thousands or even millions of features [6], without guiding the ME rule generation by incorporating some form of prior knowledge, techniques like *EXPLAIN* will suffer from scalability issues. Furthermore, due to data and model drift over time [34], models must be retrained, and ME rules also must be regenerated. Additional time requirements for approaches "blindly" relying on all features for rule deduction would quickly add up when done many times over the lifetime of a model.

In another related line of work, researchers introduced various methods to improve the model's prediction performance on the instances containing under-represented features [25], [57]. However, previous approaches typically require additional annotations [17]. For example, adding additional annotated code commits that modify a large number of files or adding annotated crashed car pictures in the vehicle detection dataset. While these approaches have been successful at improving the model's performance for instances containing under-represented features, the required additional annotated training data is often expensive [39].

Having Observed these limitations of the existing techniques, in this paper, we propose a technique called **B**ias **G**uided **M**isprediction **D**iagnoser (**BGMD**), which leverages feature imbalance as prior knowledge for generating rules to explain misprediction. Then, we use generated rules from *BGMD* to guide a novel upweight sampling method that can improve ML model's performance on mispredicted data without requiring additional annotated instances, named ***MAPS*** (**M**ispredicted **A**rea U**P**weight **S**ampling).

Figure 1 shows the high-level overview of how *BGMD* and *MAPS* work together to resolve the aforementioned limitations of existing techniques. Figure. 1-(a) presents a trained model that classifies black and white points based on two features (x-axis and y-axis coordinates). The model predicts points in green region as black points and white in blue region. Next, *BGMD* identifies two regions (red square area in Figure. 1-(b)) that contain instances that are prone to misprediction. Then, *MAPS* improves the weight of instances within the identified regions (Figure. 1-(c)) so that the retrained model pays more attention to these part of instances. The retrained model result presents in Figure. 1-(d), which could perform better on the instances that were identified by *BGMD*. A detailed description of the *MAPS* algorithm is in Section IV.

We empirically compared *BGMD* with the state-of-the-art *EXPLAIN* [17] technique and the result shows that *BGMD* not only outperformed *EXPLAIN* in generating *ME* rules in trms of rule coverage but also reduced 92% in rule generation time. Furthermore, we also investigated if *MAPS* can successfully improve the ML model's performance, specifically for instances containing under-represented features that are prone to misprediction. We empirically evaluated *MAPS* on three software engineering tasks and five general classification tasks

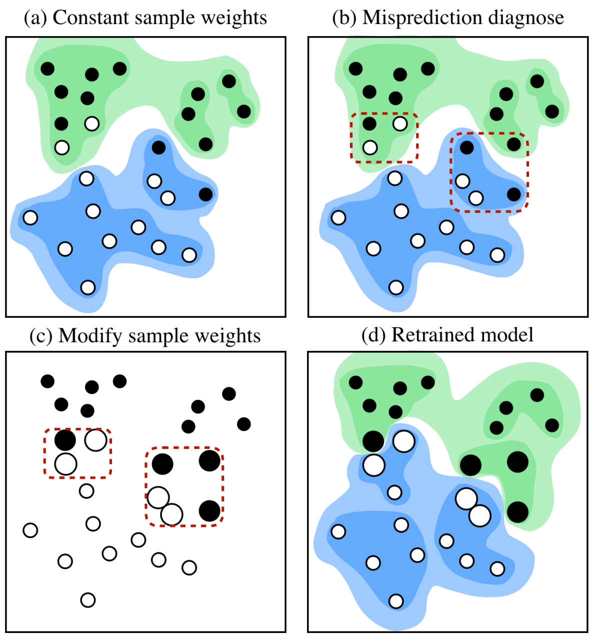

(a) Constant sample weights
(b) Misprediction diagnose
(c) Modify sample weights
(d) Retrained model

Fig. 1: Overview of Mispredicted Area Upweight Sampling

TABLE I: Samples from a dataset used to train a *ML* model that predicts whether a merge commit is likely to lead conflict

commit num	added file num	parallel changed file num	developer num	...	conflicted	pred
3	7	0	12	...	True	True
1	2	3	4	...	False	False
1	3	2	3	...	True	False
5	13	0	8	...	False	True

and the result shows that *MAPS* can significantly improve the model's performance without requiring extra annotation data.

The *key contributions* of this study are:

- Introduces a scalable ML model misprediction explanation rule generation technique named *BGMD*.
- Introduces a new upweight sampling method that improves model performance on data prone to be mispredicted without requiring extra annotated training data named *MAPS*.
- Empirically evaluates new proposed techniques with corresponding state-of-the-art techniques.

The rest work is structured as follows. In Sec. II, we introduce the necessary preliminary information. Then, in Sec. III, we introduce *BGMD*. In Sec. IV, we describe how *MAPS* works. In Sec. V, we show empirical evaluations and results. Then, In Sec. VI, we make further discussions. In Sec. VII, we review some of the related works close to our problem. In Sec. VIII, we present threats to validity, and finally, in Sec. IX, the conclusions are drawn.

II. Preliminaries

In this section, we describe what a misprediction explanation (*ME*) rule generation technique is and how the generated rules can be used to explain mispredictions of a *ML* model.

Imagine training a model to predict whether a merge commit is likely to cause a conflict. The model may be based on features such as number of commits ("commit num"), number of added files ("added file num"), number of changed files parallelly ("parallel changed file num"), number of involved developers ("developer num") and potentially dozens of additional features. Table I provides a small subset of the entire dataset, including the true label ("conflicted") and the model's prediction ("pred"). We will use it as a running example for the rest of this section.

We use instances $x \in \mathcal{X}$ and corresponding labels $y \in \mathcal{Y}$ to train a *ML* model. Let $\mathcal{D} : \mathcal{X} \to \mathcal{Y}$ be the ground truth for the dataset. Given instances $(x_1, y_1), ..., (x_n, y_n) \in \mathcal{X}$, a trained *ML* model $\mathcal{M}_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by θ, we define a misprediction indicator $\mathcal{I} : x \to \{0, 1\}$:

$$\mathcal{I}(x) = \begin{cases} 1 & \text{if } \mathcal{D}(x) \neq M_\theta(x) \\ 0 & \text{if } \mathcal{D}(x) = M_\theta(x) \end{cases} \tag{1}$$

In other words, $\mathcal{I}(x)$ is 1 iff when the *ML* model $\mathcal{M}_\theta$ predicts the wrong label for instance x.

Misprediction coverage: *ME* technique's goal is to generate a decision list Φ, i.e., *ME* rules. These rules are generated based on model training features. In the case of our running example, these features would be all available features shown in table I. Then *ME* technique generated rules identifies a sub-dataset $\Phi(x)$, in which most of the instances are prone to be mispredicted by the trained model:

$$P\left(\Phi(x) = 1 \,|\, \mathcal{I}(x) = 1, x \in \mathcal{X}\right) \tag{2}$$

We refer to the value of Equation 2 as the *ME* coverage of rule Φ where $\Phi(x) = 1$ when the *ME* rule covers an instance x that is mispredicted by the model. The larger value of Equation 2 means the more mispredicted instances are explained by decision list Φ. And decision list Φ is composed of a set of rules:

$$\Phi = \{\phi_1 \wedge \phi_2 \wedge ...\phi_n\} \tag{3}$$

where ϕ_i is a predicate based on feature i and defined as:

$$\phi \to x_c = c \,|\, x_c \neq c \,|\, x_n \leq c \,|\, x_n > c \tag{4}$$

Where each condition is a conjunction of the atomic predicate of the form "$x\, op\, c$" where x is a feature and c is a variable. The notation x_c indicates categorical features, and x_n indicates numeric features. For example, in the running example, the best rule list is when $\Phi = \{$*commit num* $>$ *28 & added file num* $>$ *15 & developer num* $<=$ *15 & developer* $>$ *9*$\}$ which has a precision of 82% and a recall of 46%. This means that 82% of the instances identified by the above-mentioned rule are mispredicted by the model, and the identified instances contain 46% of all mispredicted instances. A good *ME* rule should have a higher misprediction coverage, which means both high precision and recall.

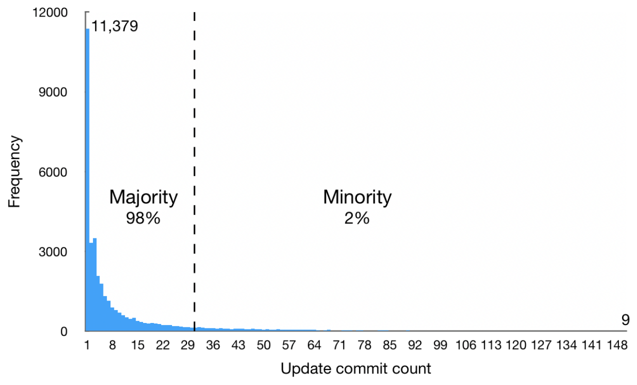


Fig. 2: Commit count frequency for dataset [43]

III. BGMD: Bias Guided Misprediction Diagnoser

In this section, we present our proposed *ME* rule generation technique *BGMD*. First, we show an example of feature imbalance that occurs in merge conflict prediction datasets [43]. We then introduce how *BGMD* exploits feature imbalances in ML models to achieve scalable *ME*.

A. Data Feature Imbalance

ML model performance heavily relies on data quality [8]. However, data often exhibit highly-skewed feature distribution. For example, figure 2 shows the frequency of the *Updated commit count* feature in a merge conflict prediction data set (we only present the *Updated commit count* between 1 and 150 because of the space limitation).

From figure 2, we observe that 11,379 merge commit instances contain one update commit, but only nine merge commit instances have 150 update commits. Additionally, instances with less than 30 *Updated commit count* accounted for 98% of all data. Thus, in the merge conflict prediction dataset, instances with *Updated commit count* less than 30 belong to the majority group with respect to *Updated commit count* feature, while instances with *Updated commit count* over 30 belong to the minority group. The minority group of data is usually under-represented during model training, and as a result, the trained model is biased towards the majority group, causing the model to perform poorly on data containing under-represented features [25], [57]. Despite such bias, these features should not be removed because that might negatively impact the model's overall performance. For example, in case of the data shown in figure 2, *Updated commit count* is one of the most important features for merge conflict prediction [36]. So removing the *Updated commit count* will adversely impact the overall model's performance. Therefore, directly removing the biased features is not advised in literature [13], [39].

B. Bias Guided Misprediction Diagnoser

The general *ME* rule generation for *ML* model is formulated in Section II. At a high level, the first step of *BGMD* is to select a subset of features whose part of data are prone to be mispredicted based on imbalanced features, such as the *Updated commit count* feature in the merge conflict prediction data set

Algorithm 1 BGMD $(D, \mathcal{A}, M, \delta)$

```
Input: Labeled dataset $D : \mathcal{X} \rightarrow \mathcal{Y}$;
       ML model $M : \mathcal{X} \rightarrow \mathcal{Y}$;
       Data attributes: $\mathcal{A}$;
       Target coverage: $\delta$.
Output: Misprediction explaination for model $M$.
 1: $\mathcal{I} \leftarrow I\{D, M(\mathcal{A})\}$
 2: $\mathcal{BA} \leftarrow ExtractBiasFeatures(\mathcal{A}, \mathcal{I})$
 3: $Atom \leftarrow GenAtoms(\mathcal{BA})$
 4: $\Phi \leftarrow [\,]$
 5: $cvg \leftarrow 0$
 6: $cur \leftarrow D$
 7: while $cvg \leq \delta$ do
 8:     $\Phi \leftarrow LearnRule(Atom, cur)$
 9:     $cur \leftarrow Filter(\Phi, cur)$
10:     $cvg \leftarrow ComputeCoverage(\mathcal{I}, cur)$
11: end while
    return Misprediction Explanation $\Phi$
```

(Section III-A). Then, *BGMD* deduces a list of explanation rules to explain when a data contains what particular features that the model tends to mispredict. Note that, to the best of our knowledge, *BGMD* is the first method to use the feature imbalance for model *ME* rule generation.

Algorithm 1 presents the procedure of *BGMD* method. This procedure takes labeled data set D containing ground truth label, all attributes $\mathcal{A}$, an ML model M, and a target *ME* coverage δ (percentage of mispredicted data) as inputs. We now explain the procedure of BGMD.

Construct misprediction indication vector. The first step (line 1 in Algorithm 1) is to build a misprediction indication vector $\mathcal{I} : \mathcal{X} \rightarrow 0, 1$ for the model M, such that:

$$\mathcal{I}(x) = 1 \Leftrightarrow (D(x) \neq M(x))$$

In other words, the extracted indication vector $\mathcal{I}$ maps each input in D to a boolean value indicating whether the instance is mispredicted by the given model M.

Extract biased features. Next, our algorithm calls a procedure named *ExtractBiasFeatures* (line 2 in Algorithm 1) to select a subset of features that the trained model is biased on, i.e., the model M performs significantly better on the feature's majority group than its minority group. The detail of the procedure *ExtractBiasFeatures* is in Algorithm. 2.

First, *ExtractBiasAttributes* separates all data into *mispredicted* and *correctly predicted* groups. Then, iterate each feature in $\mathcal{A}$ and evaluate whether there is a significant feature distribution difference (Mann-Whitney test, $\alpha < 0.05$) between the *mispredicted* and *correctly predicted* groups. We use the non-parametric Mann-Whitney test since the data population usually is not normally distributed. We consider the model is biased towards a feature if the Mann-Whitney test shows there is a significant difference ($\alpha < 0.05$) between *mispredicted* and *correctly predicted* instances.

Generate atomic predicates. Next, *BGMD* calls *GenAtoms*

Algorithm 2 ExtractBiasFeatures $(\mathcal{A}, \mathcal{I}, \alpha)$

```
Input: Data features: $\mathcal{A}$;
       Mispredict indicator: $\mathcal{I}$;
       Significance threshold: $\alpha$.
Output: Biased feature list $\mathcal{BA}$.
 1: $\mathcal{BA} \leftarrow [\,]$
 2: mispredicted $\leftarrow \mathcal{I}(x) = 1$
 3: correctly-predicted $\leftarrow \mathcal{I}(x) = 0$
 4: for $feature$ in $\mathcal{A}$ do
 5:     MG $\leftarrow$ mispredicted[$feature$]
 6:     CG $\leftarrow$ correctly-predicted[$feature$]
 7:     P-value $\leftarrow$ Mann-Whitney(MG, CG)
 8:     if P-value $< \alpha$ then
 9:         $\mathcal{BA}.insert$(feature)
10:     end if
11: end for
    return Biased feature list $\mathcal{BA}$
```

TABLE II: Example of universe atomic predicates based on the dataset in Table I

Atomic Predicates		
commit num > 3	*add file num > 5*	*developer num > 4*
commit num <= 3	*add file num <= 5*	*developer num <= 4*
commit num > 18	*add file num > 15*	*developer num > 9*
commit num <= 18	*add file num <= 15*	*developer num <= 9*
commit num > 28	*add file num > 30*	*developer num > 15*
commit num <= 28	*add file num <= 30*	*developer num <= 15*

procedure (line 3 in Algorithm 1) to generate candidate atomic predicates of the form "x op c", where x is a feature and c is a constant value. If x is a categorical variable, we generate predicates of the form $x_c = c_j$ and $x_c \neq c_j$, where $c_j \in \mathcal{BA}$. For numerical features, we use operators $\leq, >$ and generate constant c_j using equal frequency binning [33]. For instance, if we have a numerical feature containing values $V = \{v_1, v_2, ..., v_n\}$, we first partition the (sorted) set V into k bins where each bin has roughly equal size. The value of k is a hyper-parameter and is set to 4 by default. Then, we use the highest value in each bin as one of the constants in our predicates to generate atoms of the form "x_n op c". Table II shows the universe of atomic predicates that are generated based on the features illustrated in Table I.

Rule Learning. During rule learning (line 8 in Algorithm 1), we want to learn rules that are correlated with mispredictions. This problem is equivalent to maximizing the following objective function:

$$precision = \frac{|x \in \mathcal{X}|\Phi(x) \wedge I(x) = 1|}{|x \in \mathcal{X}|\Phi(x)|} \tag{5}$$

which tries to make identified instances by Φ contain a higher percentage of mispredicted instances, and it corresponds to the precision value of *ME* rule.

However, if our rule learning algorithm solely aims to maximize precision, the *BGMD* may lead to a small rule size that takes many iterations to converge. Moreover, it may produce an over-fitted rule to a specific mispredicted instance (100% precision). Generating many rules to meet the coverage threshold would also result in producing a large number of sub-rules in Φ. This ultimately compromises the interpretability. Hence, instead of optimizing only on *precision*, our rule learning algorithm also takes *rule size* and *recall* into account. The recall is shown below:

$$recall = \frac{|x \in \mathcal{X} | \Phi(x) \wedge I(x) = 1|}{|x \in \mathcal{X} | I(x) = 1|} \quad (6)$$

which corresponds to the ratio between the identified mispredicted instances by generated rules and all mispredicted instances by the given model.

Thus, our final rule learning optimization objective function is a linear combination of *precision*, *recall*, and *rule size*:

$$Obj = \lambda_1 \cdot precision + \lambda_2 \cdot recall + \lambda_3 \cdot \frac{1}{size(\phi)} e \quad (7)$$

where parameters λ_1, λ_2, and λ_3 are tunable hyper-parameters and they are depends on the context and set to 1 by default. Precision is the primary factor that identifies mispredictions instances density, i.e., reducing the number of correctly predicted instances in identified instances. And recall controls the coverage of all mispredicted instances, i.e., increasing the number of identified mispredicted instances. Furthermore, rule size is mainly used for accelerating convergence and improving the explainability of generated rules.

Main learning loop. After the initialization phase (line 1 to 6), the algorithm enters a loop (line 7 to 11) that iteratively adds previously generated atomic predicate into a decision list until the learned rules achieve the desired coverage δ. The learned decision lists Φ is a list of predicates. For example, the list $\Phi = [\phi_1, \phi_2]$ corresponds to the following explanation:

$$if\,(\phi_1)\; then\; 1\; else\, if\,(\phi_2)\; then\; 1\; else\; 0$$

At a high level, the learning loop synthesizes the target decision list using a standard *sequential covering* method [10]. In particular, it first learns a rule ϕ_1 for the whole data set, then filters out instances satisfying ϕ_1, then learns another rule ϕ_2 for the remaining instance, and so on, until the target coverage is reached. Intuitively, the predicate in the i'th branch is the best predictor for the mispredictions in the subset of the data not covered by the earlier predicates. The algorithm terminates only when misprediction coverage cvg exceeds target coverage δ, thus the output of the *BGMD* procedure is guaranteed to satisfy the coverage constraint.

C. Implementation

We implemented *BGMD* as a Python library that can be installed using *pip* command. It takes a Pandas dataframe, a target coverage, and a set of optional parameters and returns a set of decision lists paired with precision, recall, F1 score, and coverage metrics. An implementation is available in accompany website [1]

IV. MAPS: MISPREDICTED AREA UPWEIGHT SAMPLING

In this section, we present Mispredicted Area uPweight Sampling (*MAPS*), which leverages the *ME* information generated by *BGMD* to improve the *ML* model's performance on instances that contains under-represented features.

A. Overview of the baseline algorithms

In this study, we use a standard *ML* model training method and two sampling algorithms as baselines.

Empirical Risk Minimization (**ERM**) is a standard approach to train a *ML* model by minimizing the average training loss. *ERM* is trying to minimize the following loss function:

$$L_{ERM}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i; \theta) \quad (8)$$

where θ is the parameter of the trained model.

Synthetic Minority Oversampling TEchnique (**SMOTE**) [13] is one of the most popular oversampling methods to improve the model's robustness by synthesizing instances in minority groups. The intuition of *SMOTE* is that it tries to balance the number of instances between majority group and minority group by synthesizing artificial instances in the minority group. So that the trained model can pay more attention to the instances in the minority group.

SMOTE is parameterized with K neighbors (the number of nearest neighbors it will consider) and the number N of new instances that it wishes to create. The way *SMOTE* synthesizes an instance is : (1) Randomly selects an instance in the minority group. (2) Randomly selects any of its K nearest neighbors belonging to the same class and generates a temporary new instance X_{temp} using the average of selected K neighbors. (3) Randomly specifies a value lambda in the range [0, 1]. (4) Generates and places a new instance on the vector between the original and X_{temp}, located lambda percent of the way from the original instance. In this work, we consider the *BGMD* identified data groups as the minority group since these groups are under-represented during model training.

Just Train Twice (**JTT**) [39] is a upweight sampling technique that was proposed in PMLR'21 [5], whose goal is to improve model's robustness via fixing model's performance on the mispredicted instances. We selected JTT as one of the baseline because JTT has been proven to be the state-of-the-art technique which has been compared with several upweight and reweight methods such as CVaR DRO [23], and Group DRO [47]. *JTT* has two-stages. In the first stage, it trains a *ML* model $\hat{M}$ on training data and then constructs a misprediction indication vector I on the validation data using equation 1, such that:

$$\mathcal{I}(x) = 1 \Leftrightarrow (D(x) \neq \hat{M}(x))$$

where D is the ground truth label for validation data.

Next, *JTT* retrains a final model M with validation data by upweighting all instances in the validation data that were mispredicted by the first trained model:

$$L_{JTT}(\theta, I) = \left(\lambda_{up} \sum_{I(x_i)=1} \ell(x_i, y_i; \theta) + \sum_{I(x_i)=0} \ell(x_i, y_i; \theta) \right) \tag{9}$$

where $\lambda_{up} \in R_+$ is a tunable hyperparameter. The intuition of *JTT* is that for instances that the first model mispredicted, the final model should pay more attention to them. However, increasing the model's weight only for mispredicted data instances can make the model overfit to them. This could also result in the previously correctly predicted instances to be mispredicted by the final model. This problem was also found in our experiments and details are in Section V.

B. MAPS: Mispredicted Area uPweight Sampling

MAPS is a novel upweight sampling method proposed in this paper based on the empirical observation that *ML* models tend to perform poorly on subsets of data containing under-represented features [25]. Therefore, *MAPS* first utilizes the *ME* rules generated by *BGMD* to identify a subset of the dataset that is prone to misprediction due to feature under-representation and then uses up-weight sampling to make the new model more aware of data with under-represented features. Unlike *JTT*, the retrained model using *MAPS* avoid focusing too much on a small subset of data that can lead to overfitting, and instead focus more on balancing saliency and under-represented features. The *MAPS* details presents in algorithm 3

Stage 1: Mispredicted Area identification. *MAPS* first trains a normal *ML* model $\hat{M}$. Then it identifies groups of instances that tend to be mispredicted by using misprediction diagnosing techniques, such as *BGMD* (Section III).

$$\Phi = BGMD(x_i, y_i,\ \hat{M}) \tag{10}$$

Stage 2: Upweighting. After identifying the groups of instances that first model tends to mispredict, *MAPS* retrains a final model M_{final} by upweighting the identified instances during model training, using below loss function:

$$L_{MAPS}(\theta, \Phi) = \left(\lambda_{up} \sum_{x_i \in \Phi}^{n} \ell(x_i, y_i; \theta) + \sum_{x_i \notin \Phi} \ell(x_i, y_i; \theta) \right) \tag{11}$$

Implementation. The *MAPS* training method is described in Algorithm 3. To implement the upweighted objective (equation 11), we multiply a upweight value λ_{up} on identified subset of data. However, it's challenging to determine a universal upweight value λ_{up} for all models, so we tried various upweight values and used the best performed retrained model. Similar upweight value λ_{up} selection method was also used for *JTT*. In addition, we also analyzed the impact of different upweight value λ_{up} on *MAPS* in Section V-C.

V. Evaluation

In this section, we present empirical evaluation results that aim to answer the following research questions:

Algorithm 3 MAPS training

```
Input: Training set $\mathcal{D}$ and hyperparameter $\lambda_{up}$.
Stage one: Mispredict area identification
1. Train $\hat{M}$ on $\mathcal{D}$ via ERM (equation 8).
2. Extract the misprediction explaining rules $\Phi$ (equation 10).
Stage two: Upweighting points meet rules
3. Construct upweighted dataset $\mathcal{D}_{up}$ containing the training instances that meet the misprediction explain rules $\Phi$.
4. Set $\lambda_{up}$ times in loss function for $\mathcal{D}_{up}$ training instances and one for other examples (equation 11).
5. Train final model $M_{final}$ using $L_{MAPS}$ as the loss function.
```

- **RQ1:** How does *BGMD* perform compared to the state-of-the-art *ME* rule generation method? (Section V-A)
- **RQ2:** Can *MAPS* help improve the performance of *ML* models? (Section V-B)
- **RQ3:** How do different upweight values affect the performance of the model when using *MAPS*? (Section V-C)

To answer first research question, we compared *BGMD* with the state-of-the-art *ME* rule generation method *EXPLAIN* [17] on two SE tasks and five Kaggle [4] classification tasks. And to answer the second question, we compared *MAPS* method with a popular oversampling method (*SMOTE*) and a state-of-the-art upweight-sampling method (*JTT*) [39].

A. ME rule generation technique comparison

ME techniques need to ensure (1) high model *ME* coverage (quality) by the generated rules, and (2) less rule generation time (efficiency).

In terms of *ME coverage* metric, the subset data covered by generated *ME* rules should ensure that: (i) the majority of the covered data is mispredicted by the given model (*precision*), and (ii) the covered data should account for as many mispredicted data by given model as possible (*recall*). Thus, both *precision* (equation 5) and *recall* (equation 6) are important for a good *ME coverage* metrci. Therefore, when comparing *MAPS* with the state-of-the-art *ME* rule generation technique *EXPLAIN* [17], we use F1 score as it is a harmonically balanced value of precision and recall.

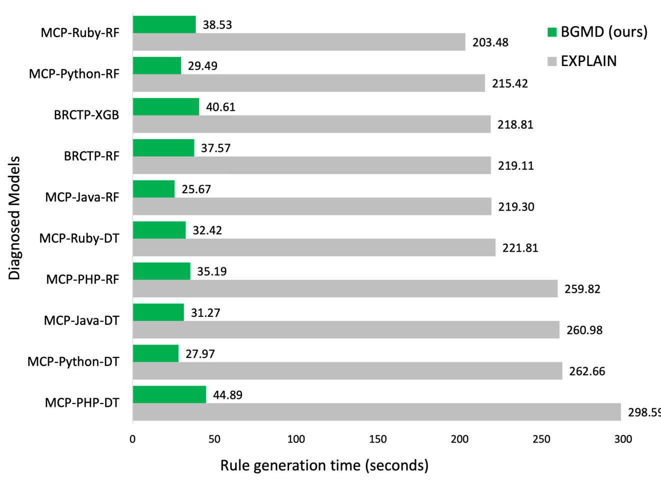
$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{12}$$

In terms of *ME* rule generation efficiency, we use rule generation time as the evaluation metric, i.e., the less time spent in the rule generation process, the more efficient the technique is.

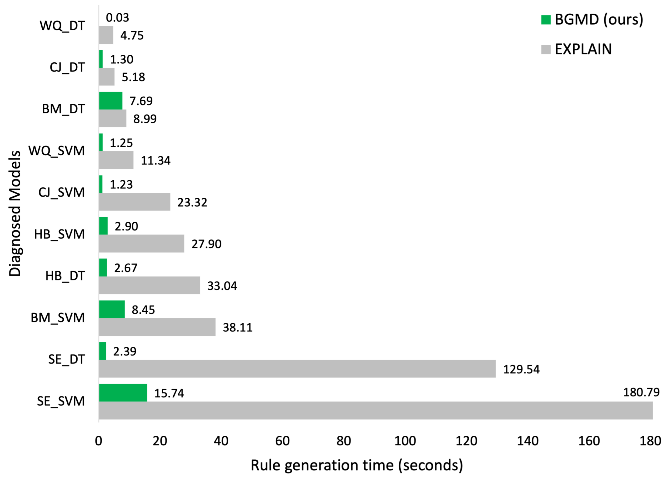
Evaluation Subjects. For evaluation, we selected the two models that performed best in the study that proposed *EXPLAIN* [17], i.e., Decision Tree (*DT*) and Random Forest (*RF*). To replicate the study conducted by Cito et al. [17] and to evaluate whether our approach can be extended to Non-SE models, we also evaluated on two publicly available models from Kaggle [4]. We select the Kaggle models that still have room for improvement. Thus, we select Support Vector

TABLE III: Generated misprediction explanation rule coverage metrics by BGMD and EXPLAIN. DT represents decision tree, RF represents random forest, SVM for Support Vector Machine.

Software Engineering		**EXPLAIN**			**BGMD (ours)**			**Kaggle**		**EXPLAIN**			**BGMD (ours)**		
Task	**Model**	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**	**Task**	**Model**	**Prec.**	**Recall**	**F1**	**Prec.**	**Recall**	**F1**
Merge Conflict	DT	33.08	62.46	43.25	58.40	72.99	64.89	Spam	SVM	49.24	82.60	61.70	49.24	82.60	61.70
Pred. (Ruby)	RF	93.02	94.83	93.91	92.06	95.84	93.91	Email	DT	46.38	61.62	52.93	46.20	61.35	53.40
Merge Conflict	DT	67.64	71.45	69.49	64.56	78.76	70.96	Hotel	SVM	33.13	98.58	49.60	33.13	98.58	49.60
Pred. (Python)	RF	98.74	95.58	97.13	98.74	95.58	97.13	Booking	DT	18.96	99.92	31.87	18.96	99.92	31.87
Merge Conflict	DT	54.47	67.14	60.15	61.06	64.33	62.65	Bank	SVM	45.90	38.48	41.86	45.90	38.48	41.86
Pred. (Java)	RF	89.26	94.32	91.72	92.62	90.83	91.72	Marketing	DT	33.33	35.92	34.58	33.33	35.92	47.83
Merge Conflict	DT	65.25	63.17	64.19	60.25	68.68	64.19	Change	SVM	42.19	89.54	57.36	42.19	89.54	59.51
Pred. (PHP)	RF	85.33	92.96	88.98	83.33	95.45	88.98	Job	DT	20.28	44.21	59.51	20.28	44.21	59.51
Bug Report	DT	30.60	19.53	23.84	31.89	31.00	31.44	Water	SVM	55.51	47.87	49.74	48.95	83.93	62.55
Close Time Pred.	RF	7.51	38.14	12.55	41.97	40.69	41.32	Quality	DT	19.50	50.43	28.13	22.75	87.83	36.14
Average		**62.49**	**69.96**	**64.52**	**68.49**	**73.42**	**70.72**	Average		**36.44**	**68.66**	**47.10**	**36.99**	**70.50**	**51.92**



(a) SE models ("MCP" represents Merge Conflict Prediction; "BRCTP" represents Bug Rreport Close Time Prediction).



(b) Non-SE models ("WQ" represents Warter Quality; "CJ" represents Change Job; "BM" represents Bank Market; "HB" represents Hotel Booking; and "SE" represents Spam Email).

Fig. 3: Misprediction explanation rule generation time comparison

Machine (*SVM*) and Decision Tree (*DT*). Since our goal is to compare the effectiveness of various *ME* rule generation approaches, we did not conduct hyper-parameter tuning so that the models have room for improvement and there are mispredicted data that can be identified by the *ME* rules. So we used default hyper-parameters in all models. In total, there are 57 hyper-parameters for the three models used in this paper. Due to space constraints, we list them in the companion website [3]. To remove the model variance, we used five-fold cross-validation (train-80%, test-20%) and repeated it five times with random seeds, and finally reported the median value, which is a common approach used by other studies [11].

Three SE data were evaluated in *EXPLAIN*, which are private to the Meta company and inaccessible to us. So, we use two publicly accessible SE datasets: merge conflict prediction [43] and bug report close time prediction [27]. For non-SE tasks, *EXPLAIN* was evaluated on two publicly available datasets from Kaggle [4]. However, the datasets they evaluated are too small to highlight the efficiency or scalability of different *ME* rule generation techniques. So we decided to use larger datasets (Spam Email, Water Quality, Bank Marketing, Change Job, and Hotel Booking) to compare our approach with *EXPLAIN*. Details of these datasets are in the companion website [2].

TABLE IV: Representative rule from each technique

	Feature #	Rule	Prec.	Rec.	F1.	Time(s)
BGMD	**8**	**If** line_removed > 332 & developer_num > 45 & commit_num>372 & parallel_changed_file_num > 12 **elseif** line_removed>332 & developer_num > 88 & commit_num>229 & parallel_changed_file_num>12	0.92	0.95	0.94	**38.53**
EXPLAIN	**29**	**If** line_removed>332 & developer_num>88 & parallel_changed_file_num>12 **elseif** developer_num>45.0 & commit_num > 229 & parallel_changed_file_num>12	0.93	0.94	0.93	**203.48**

Results: Table III shows the results of applying *EXPLAIN* and *BGMD* on five SE data sets and five non-SE data sets collected from Kaggle. In SE related models, *BGMD* outperformed *EXPLAIN* on all three metrics. From Table III, we can observe that *BGMD* improved the explanation rule result's *precision* from 62.49 to 68.49, *recall* from 69.96 to 73.42, and *F1 score* from 64.52 to 70.72 on average. In addition, we also got similar results on non-SE models. On average, the precision of rules generated by *BGMD* is 36.99, recall is 70.5, and F1 score is 51.92. However, the precision of *EXPLAIN* generated rules is 36.44, recall is 68.66, and F1 score is 47.1.

For merge conflict prediction models, the *BGMD*'s F1 score improves on three out of four *DT* learners. For example, On Ruby data, *BGMD*'s F1 score is 64.89, but *EXPLAIN*'s F1 score is 43.25. However, for RF-based merge conflict prediction models, the performance of *BGMD* and *EXPLAIN* are similar since *EXPLAIN* already achieved a very high F1 score (close to 0.9), leaving small room for improvement. For bug report close time prediction models, *EXPLAIN* got 23.84 F1 score with *DT* learner and 12.55 with *RF* learner. However, *BGMD* received 31.44 and 41.32, respectively. In summary, *BGMD* generated rules from a smaller number of biased features and performed better than *EXPLAIN* in terms of rule coverage. Similar results can be observed for the Kaggle dataset in Table III.

In addition, one thing to note is that both *BGMD* and *EXPLAIN* take an input parameter that denotes target recall (i.e., percentage of mispredicted instances covered by generated explanation). Thus, the generated rules prioritize improving the recall values, which hurts the precision score. This is visible in table III where the average recall is 68.66 for *EXPLAIN* for non-SE tasks, but its precision is only 36.44. The same is visible for SE tasks. This holds true also for *BGMD*.

Figure 3 presents the rule generation time comparison between *BGMD* and *EXPLAIN*. The average *ME* rule generation time by *BGMD* on SE models was 34 seconds. In contrast, the average rule generation time by *EXPLAIN* was 238 seconds. The results are significantly different (Mann-Whitney test, p-value$<$5.02e-14) [41], and the effect size is large (Cohen's D = 9.28) [20]. In terms of non-SE models, *BGMD* spent four seconds to generate *ME* rules, but *EXPLAIN* needed 60 seconds in average. The results are statistically significantly different for *BGMD* (Mann-Whitney test, p-value$<$5.02e-14) [41], and the effect size is large (Cohen's D=9.28) [20].

Table IV provides an illustrative overview of the representative *ME* rules for merge conflict prediction model produced by *BGMD* and *EXPLAIN*. *BGMD* spent 38.53 seconds to generate the rules, but *EXPLAIN* needed 203.48 seconds. This happened because *BGMD* induced rules from the identified eight biased features out of 29. In contrast, *EXPLAIN* tried to infer rules from all 29 features. Furthermore, after running a large number of models, we observed that the rules inferred by *EXPLAIN* contain the same features considered by *BGMD*.

Observation: Compared to state-of-the-art *EXPLAIN*, *BGMD* reduces misprediction explanation rule generation time by up to 92% without affecting *ME* coverage.

B. *Effectiveness of Mispredicted Area Upweight Sampling*

In this section, we present the results to answer the research question: *Can MAPS fix the model's performance*? We present the evaluation results on five SE tasks that were used in RQ1 in table V. Due to space constraints, we report the evaluation results for non-SE tasks in the companion website [1]. We used models trained with *ERM* as the baseline, which is named as "default" in table V. In addition, we used oversampling method *SMOTE* and upweight sampling method *JTT* for comparison.

To compare *SMOTE*, *JTT*, and *MAPS* performance, it is important that these methods should not only improve the model's performance on mispredicted data, but also ensure model's performance on all data. Thus, in table V, we present each method's performance on both *Mispredicted data* and *All data*. To remove the model variance, we used 5 fold cross-

TABLE V: "Default" denotes off-the-shelf model; "SMOTE" is trained with SMOTE [13]; "JTT" is trained with JTT [39]; "MAPS" is trained with this paper proposed algorithm. The darker the color, the higher the value.

Task	Model	Algo	Mispredicted Data			All data		
			Pre.	Rec.	F1.	Pre.	Rec.	F1
Merge Conflict Predic. (Ruby)	DT	Default	0.54	0.57	0.55	0.63	0.68	0.64
		SMOTE	0.51	0.52	0.52	0.7	0.68	0.69
		JTT	0.53	0.55	0.55	0.72	0.7	0.71
		MAPS	0.56	0.58	0.57	0.78	0.79	0.79
	RF	Default	0.71	0.49	0.58	0.67	0.81	0.71
		SMOTE	0.62	0.69	0.65	0.69	0.9	0.78
		JTT	0.72	0.6	0.59	0.7	0.86	0.77
		MAPS	0.72	0.52	0.6	0.7	0.83	0.76
Merge Conflict Predic. (Java)	DT	Default	0.56	0.59	0.57	0.7	0.74	0.72
		SMOTE	0.56	0.61	0.58	0.7	0.75	0.73
		JTT	0.58	0.58	0.58	0.7	0.72	0.7
		MAPS	0.58	0.62	0.6	0.75	0.76	0.75
	RF	Default	0.75	0.57	0.65	0.82	0.81	0.81
		SMOTE	0.64	0.73	0.68	0.78	0.81	0.79
		JTT	0.75	0.56	0.64	0.82	0.84	0.83
		MAPS	0.76	0.58	0.66	0.83	0.86	0.84
Merge Conflict Predic. (Python)	DT	Default	0.44	0.46	0.44	0.6	0.59	0.59
		SMOTE	0.42	0.47	0.46	0.55	0.55	0.55
		JTT	0.45	0.46	0.45	0.57	0.61	0.61
		MAPS	0.46	0.48	0.48	0.64	0.62	0.63
	RF	Default	0.69	0.37	0.48	0.74	0.59	0.66
		SMOTE	0.54	0.55	0.55	0.63	0.73	0.69
		JTT	0.68	0.36	0.47	0.75	0.59	0.67
		MAPS	0.69	0.38	0.49	0.78	0.61	0.72
Merge Conflict Predic (PHP)	DT	Default	0.52	0.54	0.53	0.73	0.82	0.77
		SMOTE	0.5	0.56	0.53	0.72	0.76	0.74
		JTT	0.54	0.55	0.54	0.72	0.71	0.71
		MAPS	0.53	0.56	0.55	0.76	0.86	0.81
	RF	Default	0.7	0.5	0.58	0.7	0.88	0.75
		SMOTE	0.59	0.69	0.64	0.71	0.95	0.84
		JTT	0.71	0.51	0.59	0.65	0.86	0.74
		MAPS	0.72	0.52	0.61	0.73	0.91	0.82
Bug Report Close Time Predic.	RF	Default	0.69	0.66	0.68	0.71	0.67	0.69
		SMOTE	0.65	0.72	0.69	0.7	0.73	0.72
		JTT	0.69	0.71	0.7	0.74	0.73	0.73
		MAPS	0.71	0.7	0.71	0.75	0.71	0.73
	XGB	Default	0.8	0.63	0.69	0.8	0.76	0.78
		SMOTE	0.78	0.7	0.73	0.76	0.84	0.81
		JTT	0.82	0.64	0.72	0.81	0.76	0.78
		MAPS	0.84	0.66	0.74	0.81	0.78	0.8

TABLE VI: Summarized information of comparing MAPS with SMOTE [13], JTT [39] based on the result in table V

	Mispredicted data			All data		
	Won on Prec.	Won on Rec.	Won on F1	Won on Prec.	Won on Rec.	Won on F1
SMOTE	0	6	4	0	6	3
JTT	2	0	0	2	1	1
MAPS	9	5	6	10	4	6

validation (train-80%, test-20%) and repeated 10 times with random seeds and finally reported the median, which is a common approach used by other studies [11].

Table VI summarizes the win times of each method on various metrics in table V. If more than one techniques get the highest value on a metric, we consider they all win for that metric. For example, MAPS received highest precision on all ten models, and JTT on two. However, in terms on Recall, SMOTE received best performance on six, but MAPS got the highest recall on four models. Since *SMOTE* balances the data via synthesizing instances in minority groups, the trained model is biased towards the data that has been "duplicated" many times, i.e., the previous minority data. Moreover, the new trained model performs worse on the majority group that performed well before the retraining. This results in *SMOTE* gaining in recall but lowering the precision. Similar affect has been observed in prior studies involving *SMOTE* [13], [38], [42]. While SMOTE outperforms MAPS in terms of recall, MAPS has better combined results in terms of the overall performance measured using F1. Not only does it improve the model's performance on mispredicted data, it doesn't corrupt data that was previously correctly predicted. Based on the evaluation results shown in table VI, *MAPS* won more times compared to *SMOTE* (Mann-Whitney test, p-value$<$3.8e-2) [41] and *JTT* (Mann-Whitney test, p-value $<$ 2.5e-4) [41].

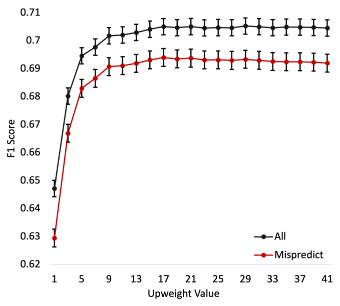
Observation: *MAPS* significantly outperforms both state-of-the-art techniques in improving model performance, especially in terms of precision and F1.

C. Impact of Upweight Value on MAPS

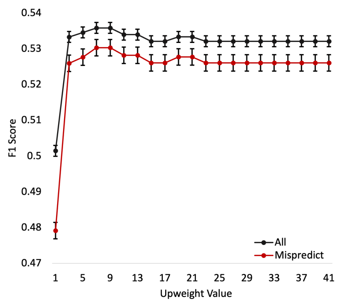
MAPS algorithm contains an important hyper-parameter: *upweight value* (λ_{up}) in equation 11, which is a number multiplied by the *ME* rule identified instances. The higher the *upweight value*, the retrained model pays more attention to the identified instances. However, the best *upweight value* has to be empirically determined. Thus, we investigated the impact of weight hyper-parameter on *MAPS* algorithm.

Figure 4 shows four representative F1-score change patterns when increasing the *upweight value* in *MAPS*. Note that when the *upweight value* is equal to one, all data have the same weight during model training. So for each figure in Figure 4, the left most pair of dots is the result for the default model without using *MAPS*. When the *upweight value* is equal to five, the instances identified by *ME* rule have fives times weight than others during model training. The higher the *upweight value*, the trained *ML* model pays more attention to the instances that identified by *ME* rule generation tools. In each subfigure, we present the model's F1 score changes when using *MAPS* for *all data* and *mispredicted* data to show the various upweight value's impact when using *MAPS*.

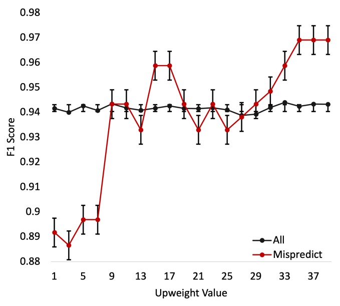
Figure 4(a) is the F1 score changes for all data and mispredicted data in the Merge Conflict Prediction (Ruby) dataset. The chart shows that F1 score grow gradually to a plateau. Figure 4(b) is the F1 score changes for the Hotel Booking dataset, and both F1 scores come to a plateau faster than Figure 4(a). These two patterns are the most common patterns when increasing *upweight value* in *MAPS* algorithm. In addition, we also observe other F1 score change patterns. Such as Figure 4(c), as the *upweight value* increases, the mispredicted data's F1 score increases, but all data's F1 score only changes a little. Figure 4(d) shows another interesting pattern. When the weight multiply is one, the F1 score of all data is higher than mispredicted data. But their performance drops drastically when the *upweight value* is equal to three, and all data's performance is even worse than mispredicted data. Then, as the weight increases, the two F1 scores begin to grow together and exceed the initial value until they enter a platform together. Figure 4 show the most common F1 score change patterns. Although their patterns are different, main trends are similar to the F1 scores increase and plateau after a particular point.



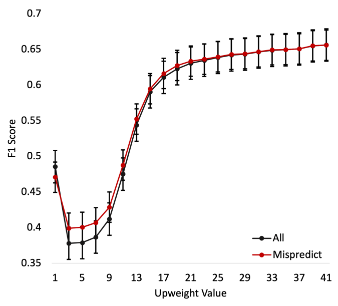
(a) MCP. (Ruby)



(b) Hotel Booking



(c) BRCTP



(d) Spam Email

Fig. 4: F1 score change patterns when increasing weight times value in *MAPS*

Observation: *MAPS* improves model performance with increasing upweight values. After a certain point, the improvement stops as the model converges in performance.

VI. Discussion

In this section, we first discuss why our proposed *BGMD* could perform better than the state-of-the-art model *ME* and the benefits of using *MAPS* to improve the model's robustness.

A. Why BGMD works better?

Focus only on useful features. *BGMD* deduces the *ME* rules only on biased features instead of "blindly" trying on all features. For example, in table IV, *BGMD* generated rules from eight biased features. In contrast, *EXPLAIN* generated rules from 29 features. Although *EXPLAIN* tried to explain misprediction using 29 features, their generated explanation rules were based on the same eight features that *BGMD* focused on. A similar situation happened on all evaluated models that *EXPLAIN* tried to explain mispredictions using all available features, but the constituent features for its generated rules were all considered by *BGMD*.

Make more attempts. Cito et al. [17] showed that with more granular predicates on features, the generated *ME* rules can get better results on misprediction coverage but need more computation time. Thus, given the same computation time, *BGMD* is able to deduce better *ME* rules. *BGMD* ignores many features that are not helpful to explain the mispredictions. For example, in Spam Email dataset, it contains 232 features, and *BGMD* only focuses on 23 features associated with the corresponding model's mispredictions. Thus, shown in Figure 3(b), *EXPLAIN* spent 180.79 seconds to generate *ME* rules. In contrast, *BGMD* only took 15.74 seconds.

B. Why MAPS is a good method to fix models?

Competitive performance. According to our empirical evaluation results in table VI, *MAPS* outperformed the popular oversampling method *SMOTE* [13] and state-of-the-art method *JTT* [39] on ten models in terms of precision and F1 score on mispredicted data. Thus, *MAPS* is a competitive method to improve the model's robustness.

Uncompromising performance for all data: Table V shows *MAPS* not only improved the model's performance on mispredicted data but also on all data. In contrast, although *JTT* used a similar upweight sampling approach as *MAPS*, it reduced the performance of four models on all data. We attribute our success in this regard to making retrained model pay more attention to the under-represented features instead of focusing more on particular mispredicted instances. Thus, *MAPS* can improve the model's misprediction performance without compromising all data prediction performance.

No extra computation: Table VI shows that SMOTE also did well in helping the model fix performance on mispredicted data, especially on improving *recall*. Furthermore, if possible, adding more manually annotated data in minority groups might improve model performance even more than SMOTE. However, adding more data means more computation during model training. One benefit of *MAPS* is that it does not require extra annotated or synthesized data, which does not add computation overhead during model training. Note that *MAPS* is not an alternative to SMOTE, but a complement. Because table VI shows that *SMOTE* performed best on improving the model's recall on both mispredicted and all data.

Model agnostic: *MAPS* entirely focuses on identified data groups that are prone to be mispredicted to fix the model's performance on them. There are works where optimization algorithms have been used to modify models [17]. These works are model specific and, most of the time, combined with internal model logic. Thus *MAPS* is much more general as it can be used for any kind of model.

VII. Related Work

The study in the paper relates to several topics below.

Interpret ML Models. Interpreting *ML* models has been a popular topic for the past couple of years. Local interpretability techniques, such as LIME [45] and Integrated Gradients [49], use several simple, explainable models to simulate complex models. However, the problem with local interpretability techniques is that they can not completely represent the complex models. On the other hand, global interpretability techniques, such as GALE [50], DENAS [14] and BETA [35] help to understand the distribution of the target outcome based on the features. Some techniques try to explain the models by generating counterfactual explanations via modifying the inputs [18], [40], [46]. However, only a few studies investigated explaining model mispredictions. Cito et al. [17] proposed *EXPLAIN* based on rule generation. However, *EXPLAIN*'s rule deducing efficiency is low because it "blindly" analyzes all features. Thus, we proposed an efficient model *ME* method that leverages data feature bias in this paper.

Debugging ML Models. The goal of debugging a model is to identify the specific groups of data on which *ML* model is likely to fail and then fix the model's performance on identified data. Tongshuang et al. [55] presented an error analysis for NLP models called *Errudite*. However, *Errudite* requires users to tune the parameters in order to perform error analyses. On the contrary, our proposed method automatically identifies the groups of data that are prone to be mispredicted and uses a simple and effective method to fix the model's performance on them. Kim et al [32] is close to our work. However, their approach is tailored to Computer Vision, and adopting their approach for text/source code is not trival due to the difference between CV and text/source code. They can create permutations of features (i.e., weather, car model, etc.). If the object of interest remains intact, they can create new images without changing the meaning. In our case, the meaning is changed if the context is changed.

Select data for upweight sampling. Increasing part of data instance's weight during model training has been proved as an efficient approach to improve model's performance [9], [21], [26], [39]. For instance, Karan et al. [26] isolates features that differentiate subgroups within a class and then augment the minority groups. Jonathon et al. [9] tweaks L2 regularization to produce the correct weighting effect on minority groups. Fereshte et al. [31] improves fairness and robustness by halving the loss across all the groups. Another group of studies identifies the groups based on fairness [7], [28], [31], [54]. In addition, *JTT* identifies mispredicted instances from a validation set through a trained model and then retrains a model via upweighting only on mispredicted instances. In other words, *JTT* is trying to make the retrained model focus more on mispredicted instances in the first model. In contrast, our approach identifies the groups of data that tend to be mispredicted because of some under-represented features during model training. Then, we increase these identified data weights during model training and make retrained models pay more attention to those under-represented features.

VIII. Threats to Validity

We have taken care to ensure that our results are unbiased and tried to eliminate the effects of random noise, but it's possible that mitigation strategies may not have been effective.

Bias due to dataset: Our findings may not generalize to all software projects since we evaluate using 10 datasets. However, all these datasets are publicly available and have been used in previous studies. Moreover, r considered projects are large and significantly different in size, programming languages, complexity. So we believe that the selected projects adequately address the concern.

Bias due to models: This work is based on binary classification and tabular data, which are very common in *ML* software. We select the models that have been used by the papers that introduced the dataset. In the future, we will test how our method performs in complex neural network models.

IX. Conclusion

We propose an efficient model-agnostic technique for generating useful and interpretable misprediction explanations for machine learning models. We demonstrate through case studies that our proposed bias-guided misprediction explanation technique is significantly more efficient than the state-of-the-art technique and generates explanation rules that have higher misprediction explanation capability. In addition, we introduce a mispredicted area upweight sampling algorithm to improve the model's robustness via fixing the model's performance on incorrectly predicted instances containing under-represented features. Our results show that our proposed method outperforms the state-of-the-art techniques. We plan to conduct studies on a broader range of tasks and datasets in the future.

References

[1] "Bias guided misprediction explanation companion code and experimental results website," https://github.com/Jirigesi/BGMD_MAPS, accessed: 2022-08-27.

[2] "Dataset details for experiments in this study," https://github.com/Jirigesi/BGMD_MAPS/blob/main/README.md#:~:text=0.0%2C%20max_samples%3DNone-,Data,5%2C940,-Footer, accessed: 2022-08-27.

[3] "Default parameters of support vector machines, decision trees and random forest learners used in this study," https://github.com/Jirigesi/BGMD_MAPS/blob/main/README.md#:~:text=SVM,0.0%2C%20max_samples%3DNone, accessed: 2022-08-27.

[4] "Kaggle machine learning competitions," https://www.kaggle.com/, accessed: 2022-08-27.

[5] "Proceedings of machine learning research," https://proceedings.mlr.press/, accessed: 2022-08-27.

[6] "University of california at irvine machine learning dataset repository," https://archive.ics.uci.edu/ml/datasets.php?format=&task=cla&att=&area=&numAtt=&numIns=&type=&sort=attDown&view=table, accessed: 2022-08-27.

[7] A. Agarwal, A. Beygelzimer, M. Dudík, J. Langford, and H. Wallach, "A reductions approach to fair classification," in *International Conference on Machine Learning*. PMLR, 2018, pp. 60–69.

[8] S. Barocas, M. Hardt, and A. Narayanan, "Fairness and machine learning: Limitations and opportunities," 2018.

[9] J. Byrd and Z. Lipton, "What is the effect of importance weighting in deep learning?" in *International Conference on Machine Learning*. PMLR, 2019, pp. 872–881.

[10] J. Cendrowska, "Prism: An algorithm for inducing modular rules," *International Journal of Man-Machine Studies*, vol. 27, no. 4, pp. 349–370, 1987.

[11] J. Chakraborty, S. Majumder, and T. Menzies, "Bias in machine learning software: why? how? what to do?" in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 429–440.

[12] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: a way to build fair ml software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 654–665.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[14] S. Chen, S. Bateni, S. Grandhi, X. Li, C. Liu, and W. Yang, "Denas: automated rule generation by knowledge extraction from neural networks," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 813–825.

[15] N. Chirkova and S. Troshin, "Empirical study of transformers for source code," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 703–715.

[16] Y. Chung, T. Kraska, N. Polyzotis, K. H. Tae, and S. E. Whang, "Automated data slicing for model validation: A big data-ai integration approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 12, pp. 2284–2296, 2019.

[17] J. Cito, I. Dillig, S. Kim, V. Murali, and S. Chandra, "Explaining mispredictions of machine learning models using rule induction," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 716–727.

[18] J. Cito, I. Dillig, V. Murali, and S. Chandra, "Counterfactual explanations for models of code," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2022, pp. 125–134.

[19] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does bert look at? an analysis of bert's attention," *arXiv preprint arXiv:1906.04341*, 2019.

[20] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988.

[21] E. Creager, J.-H. Jacobsen, and R. Zemel, "Environment inference for invariant learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 2189–2200.

[22] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 2018, pp. 53–56.

[23] J. Duchi, P. Glynn, and H. Namkoong, "Statistics of robust optimization: A generalized empirical likelihood approach," *arXiv preprint arXiv:1610.03425*, 2016.

[24] A. Galassi, M. Lippi, and P. Torroni, "Attention in natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4291–4308, 2020.

[25] J. Gesi, J. Li, and I. Ahmed, "An empirical examination of the impact of bias on just-in-time defect prediction," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–12.

[26] K. Goel, A. Gu, Y. Li, and C. Ré, "Model patching: Closing the subgroup performance gap with data augmentation," *arXiv preprint arXiv:2008.06775*, 2020.

[27] M. Habayeb, S. S. Murtaza, A. Miranskyy, and A. B. Bener, "On the use of hidden markov model to predict the time to fix bugs," *IEEE Transactions on Software Engineering*, vol. 44, no. 12, pp. 1224–1244, 2017.

[28] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," *Advances in neural information processing systems*, vol. 29, 2016.

[29] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "Deepjit: an end-to-end deep learning framework for just-in-time defect prediction,"

in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 34–45.

[30] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "Cc2vec: Distributed representations of code changes," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 518–529.

[31] F. Khani, A. Raghunathan, and P. Liang, "Maximum weighted loss discrepancy," *arXiv preprint arXiv:1906.03518*, 2019.

[32] E. Kim, D. Gopinath, C. Pasareanu, and S. A. Seshia, "A programmatic and semantic approach to explaining and debugging neural network based object detectors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 128–11 137.

[33] S. Kotsiantis and D. Kanellopoulos, "Discretization techniques: A recent survey," *GESTS International Transactions on Computer Science and Engineering*, vol. 32, no. 1, pp. 47–58, 2006.

[34] P. Kourouklidis, D. Kolovos, N. Matragkas, and J. Noppen, "Towards a low-code solution for monitoring machine learning model performance," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–8.

[35] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec, "Interpretable & explorable approximations of black box models," *arXiv preprint arXiv:1707.01154*, 2017.

[36] O. Leßenich, J. Siegmund, S. Apel, C. Kästner, and C. Hunsen, "Indicators for merge conflicts in the wild: survey and empirical study," *Automated Software Engineering*, vol. 25, no. 2, pp. 279–313, 2018.

[37] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead, "Does bug prediction support human developers? findings from a google case study," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 372–381.

[38] K. Li, W. Zhang, Q. Lu, and X. Fang, "An improved smote imbalanced data classification method based on support degree," in *2014 international conference on identification, information and knowledge in the internet of things*. IEEE, 2014, pp. 34–38.

[39] E. Z. Liu, B. Haghgoo, A. S. Chen, A. Raghunathan, P. W. Koh, S. Sagawa, P. Liang, and C. Finn, "Just train twice: Improving group robustness without training group information," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6781–6792.

[40] N. Madaan, I. Padhi, N. Panwar, and D. Saha, "Generate your counterfactuals: Towards controlled counterfactual generation for text," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 15, 2021, pp. 13 516–13 524.

[41] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[42] M. Naseriparsa and M. M. R. Kashani, "Combination of pca with smote resampling to boost the prediction rate in lung cancer dataset," *arXiv preprint arXiv:1403.1949*, 2014.

[43] M. Owhadi-Kareshk, S. Nadi, and J. Rubin, "Predicting merge conflicts in collaborative software development," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2019, pp. 1–11.

[44] R. Pan, V. Le, N. Nagappan, S. Gulwani, S. Lahiri, and M. Kaufman, "Can program synthesis be used to learn merge conflict resolutions? an empirical analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 785–796.

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

[46] A. Ross, A. Marasović, and M. E. Peters, "Explaining nlp models via minimal contrastive editing (mice)," *arXiv preprint arXiv:2012.13985*, 2020.

[47] S. Sagawa, P. W. Koh, T. B. Hashimoto, and P. Liang, "Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization," *arXiv preprint arXiv:1911.08731*, 2019.

[48] D. Song, W. Lee, and H. Oh, "Context-aware and data-driven feedback generation for programming assignments," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 328–340.

[49] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

[50] I. van der Linden, H. Haned, and E. Kanoulas, "Global aggregations of local explanations for black box models," *arXiv preprint arXiv:1907.03039*, 2019.

[51] Y. Wang and H. Li, "Code completion by modeling flattened abstract syntax trees as graphs," *Proceedings of AAAIConference on Artificial Intellegence*, 2021.

[52] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–34, 2020.

[53] C.-P. Wong, P. Santiesteban, C. Kästner, and C. Le Goues, "Varfix: balancing edit expressiveness and search effectiveness in automated program repair," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 354–366.

[54] B. Woodworth, S. Gunasekar, M. I. Ohannessian, and N. Srebro, "Learning non-discriminatory predictors," in *Conference on Learning Theory*. PMLR, 2017, pp. 1920–1953.

[55] T. Wu, M. T. Ribeiro, J. Heer, and D. Weld, "Errudite: Scalable, reproducible, and testable error analysis," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[56] Y. Xiong and B. Wang, "L2s: A framework for synthesizing the most probable program under a specification," *TOSEM: ACM Transactions on Software Engineering and Methodology*, 2021.

[57] Y. Yao, M. Xu, Y. Wang, D. J. Crandall, and E. M. Atkins, "Unsupervised traffic accident detection in first-person videos," *arXiv preprint arXiv:1903.00618*, 2019.

[58] B. Yu, H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, and J. Zhao, "Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment," *IEEE Transactions on Reliability*, 2021.