Robust Learning of Deep Predictive Models from Noisy and Imbalanced Software Engineering Datasets

Zhong Li
lizhong@smail.nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Minxue Pan*
mxp@nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Yu Pei
csypei@comp.polyu.edu.hk
Department of Computing, The Hong Kong Polytechnic University
Hong Kong, China

Tian Zhang
ztluck@nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Linzhang Wang
lzwang@nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

Xuandong Li
lxd@nju.edu.cn
State Key Lab for Novel Software Technology, Nanjing University
Nanjing, China

ABSTRACT

With the rapid development of Deep Learning, deep predictive models have been widely applied to improve Software Engineering tasks, such as defect prediction and issue classification, and have achieved remarkable success. They are mostly trained in a supervised manner, which heavily relies on high-quality datasets. Unfortunately, due to the nature and source of software engineering data, the real-world datasets often suffer from the issues of sample mislabelling and class imbalance, thus undermining the effectiveness of deep predictive models in practice. This problem has become a major obstacle for deep learning-based Software Engineering.

In this paper, we propose RobustTrainer, the first approach to learning deep predictive models on raw training datasets where the mislabelled samples and the imbalanced classes coexist. RobustTrainer consists of a two-stage training scheme, where the first learns feature representations robust to sample mislabelling and the second builds a classifier robust to class imbalance based on the learned representations in the first stage. We apply RobustTrainer to two popular Software Engineering tasks, i.e., Bug Report Classification and Software Defect Prediction. Evaluation results show that RobustTrainer effectively tackles the mislabelling and class imbalance issues and produces significantly better deep predictive models compared to the other six comparison approaches.

CCS CONCEPTS

• **Software and its engineering → Software development techniques**.

KEYWORDS

Predictive Models, Mislabelling, Imbalanced Data, Deep Learning

1 INTRODUCTION

With the success of Deep Learning (DL), various DL-based predictive models have been developed in Software Engineering (SE) tasks to improve the efficiency of development processes and software quality. Common applications include defect prediction [51], bug report management [20], API issue classification [34] and code smell detection [38]. In general, a deep predictive model is usually learned from a dataset of a particular task and then used to provide outcomes for the instances in this task, e.g., to decide whether a software module is defective or not. As a recent survey [62] shows, the number of publications in top-tier venues on deep predictive models in software engineering has stayed at a high level for the past few years.

Despite this success, there are still challenges to overcome in developing deep predictive models. One major challenge is to collect high-quality datasets for learning the deep predictive models. Like traditional DL models, the performance of deep predictive models also depends heavily on the quality of datasets learned [23, 37, 54, 62]. However, in contrast to the standard DL tasks where a good training dataset is available (e.g., ImageNet [11]), collecting large-scale, high-quality datasets can be challenging for SE tasks.

The difficulties of collecting datasets for learning deep predictive models in SE tasks are mainly two folds. First, it is difficult to obtain a large pool of samples having precise labels due to the source of SE data. Most large-scale data collection techniques rely on web data such as GitHub and Stack Overflow that are labelled by user tags [24, 61]. They unavoidably result in mislabelled samples that are incorrectly labelled in the collected datasets [21, 50, 60]. For example, Herzig et al. [21] manually inspected over 7,000 issue reports collected from the online issue tracking systems and found that more than 40% issues are inaccurately labelled. Manually cleaning every sample can reliably improve the label quality, but it is expensive and time-consuming, especially for large-scale datasets. Second, it is difficult to collect balanced datasets in which each class is represented equally due to the nature of SE data. In most SE tasks, the data naturally exhibit an imbalanced class distribution [23], where a small portion of classes have massive samples, but the others are associated with only a few samples. For example, in the defect prediction scenario, the defective cases are less likely to happen than the non-defective cases; thus, the collected datasets typically contain much more non-defective modules than the defective ones [46, 47]. Furthermore, due to the nature of SE tasks, collecting more samples for the minority classes (e.g., defects) inevitably introduce significantly more samples of the majority classes (e.g., non-defects). Motivated by the above two challenges

*Corresponding author.

of constructing high-quality training datasets, it is essential to develop a robust learning algorithm for deep predictive models to accommodate the mislabelled samples and the imbalanced classes.

Currently, there are approaches being proposed to addressing the dataset quality issue. However, all these approaches just study one of the two problems of sample mislabelling and class imbalance, thus limiting their effectiveness in real-world applications. Particularly, the SE community mainly focuses on addressing the mislabelling issue regarding the defect prediction task and proposed methods to detect mislabelled samples by examining the neighbour information of the samples [15, 27, 48]. These techniques are designed specific to the task and cannot be directly applied to other domains of samples (e.g., text samples). More recently, there have been several approaches proposed by the AI community [16, 30, 45, 64] to identifying mislabelled samples based on the training dynamics of deep predictive models. In particular, these approaches assume that the samples with small training losses are more likely to have clean labels. However, this assumption fails to generalise to class-imbalanced datasets, because the models trained on imbalanced datasets would be skewed towards the majority class, making both clean and mislabelled samples of the minority class have large training losses [23, 25]. To address the class imbalance problem, existing approaches mainly focus on re-sampling the training dataset to achieve a more balanced class distribution [7, 17, 44], or designing robust loss functions by considering the class distribution of the training dataset [5, 10]. However, most of these approaches ignore the impact of the mislabelled samples, and thereby their performance would degrade drastically in the presence of the mislabelled samples [57]. Therefore, for deep learning from SE dataset where mislabelling and class imbalance coexist, a more general and practical learning approach is urgently needed.

In this paper, we propose RobustTrainer, a robust learning approach to effectively train deep predictive models on datasets that are not only class imbalanced but also in the presence of mislabelled samples. The key insight of RobustTrainer is that the mislabelled samples and the class imbalance have different impacts on deep predictive models. That is, the mislabelled samples mainly damage the feature representations learned in the deep predictive models while less affecting the classifier part of the models [65], but, in contrast, the class imbalance mainly influences the performance of the classifier part in the deep predictive models while has less impact on the learned feature representations of the models [25]. Based on this insight, we propose a two-stage learning framework for RobustTrainer consisting of (1) a **Representation Learning** stage that focuses on learning feature representations robust to the mislabelled samples, and (2) a **Classifier Learning** stage that aims to build a balanced classifier upon the features learned in the representation learning stage. For the two issues of mislabelling and class imbalance, RobustTrainer differs from existing approaches that leverage a single element in the deep predictive model, e.g., the loss values or the loss functions, to resolve one issue but would be adversely affected by the other. In each stage, it aims to solve one issue by exploiting a unique element that will not be affected by the other issue. More importantly, the second stage builds on the results of the first stage, thus ensuring that the final predictive model solves both issues together.

To evaluate the performance of RobustTrainer, we conduct an empirical study based on two SE tasks, i.e., Bug Report Classification (BRC) and Software Defect Prediction (SDP), where the deep predictive models are popularly used [62]. As the datasets used are collected from the real world, they all have mislabelling and class imbalance issues. Our experimental results demonstrate that RobustTrainer is able to effectively and efficiently learn deep predictive models against both the mislabelling and the class imbalance. Specifically, the models learned by RobustTrainer significantly outperform the ones learned by all the six comparison approaches in terms of all the four performance metrics of F-measure, G-measure, MCC, and AUC. We further investigate the contribution of each component in RobustTrainer, and the results demonstrate that all the components make contributions.

To sum up, this paper makes the following major contributions:

- **Problem.** We propose a research problem that sample mislabelling and class imbalance are coexisting in SE datasets and need to be treated together.
- **Approach.** We present RobustTrainer, a novel learning approach to learning robust deep predictive models against both the mislabelling and the class imbalance.
- **Evaluation.** We extensively evaluate RobustTrainer using real-world datasets of two popular SE tasks. Experimental results demonstrate that RobustTrainer effectively and efficiently learns deep predictive models and outperforms the other approaches in comparison.
- **Artifact.** We have released our code as well as all the experimental data to fuel future studies. [1]

2 PRELIMINARY

2.1 Deep Predictive Model

We target deep predictive models that perform the classification tasks, such as the ones in the defect prediction task used to decide whether a software module is defective or not. Formally, let us consider a task with C classes. Let $\mathcal{X}$ denote the space of instances in the task, and $\mathcal{Y}$ denote the space of the C classes. A predictive model $\mathcal{F}$ for this task is a function $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$ that takes a instance $x \in \mathcal{X}$ as input and outputs a predicted label $\hat{y} \in \mathcal{Y}$ for x. In particular, we mainly consider a Deep Neural Network (DNN) model as the function $\mathcal{F}$ in this work.

Learning Deep Predictive Model. To learn a deep predictive model $\mathcal{F}$ for a particular task, the first step is to collect a training dataset $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$ from the joint distribution over $\mathcal{X} \times \mathcal{Y}$ of the task, where $x_i \in \mathcal{X}$ is an instance and $y_i \in \mathcal{Y}$ is the class label assigned to it. For example, to construct a training dataset $\mathcal{D}_{\text{train}}$ for the defect prediction task, one can collect the software metrics of software modules as the instances and label every instance based on whether its corresponding module is defective or not. Then, based on the training dataset $\mathcal{D}_{\text{train}}$, a model $\mathcal{F}$ is learned via minimising the empirical risk $\mathcal{R}_{\mathcal{L}}(\mathcal{F})$ under a loss function $\mathcal{L}$,

$$\mathcal{R}_{\mathcal{L}}(f) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \mathcal{L}(\mathcal{F}(x), y) \tag{1}$$

[1] Our code and experimental data are available at https://github.com/RobustTrainer/RobustTrainer

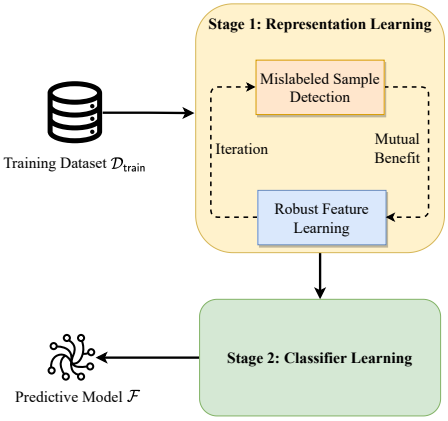


Figure 1: The pipeline of RobustTrainer.

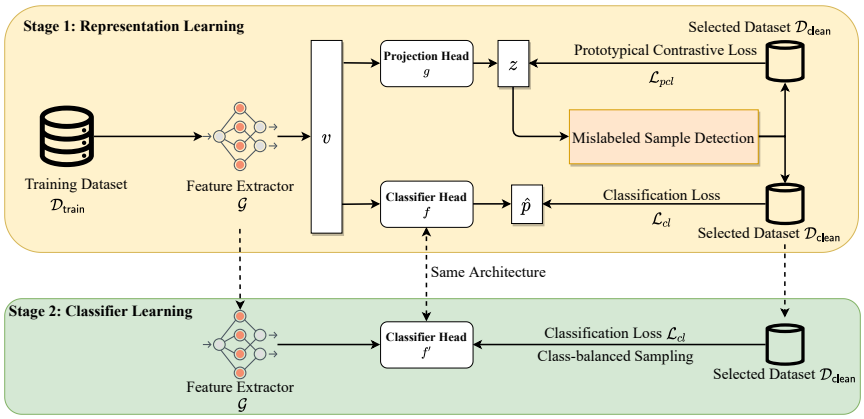


Figure 2: The model architecture of RobustTrainer.

In general, the cross-entropy (CE) loss is usually adopted as the loss function $\mathcal{L}$ for classification. Note that there also are other manners to learning $\mathcal{F}$ like unsupervised learning [22]. In this work, we mainly consider supervised learning described in Equation 1 due to its effectiveness and representativeness.

2.2 Challenges for Dataset Construction

From the discussion above, we can see that an important requisite for learning a deep predictive model $\mathcal{F}$ is a training dataset $\mathcal{D}_{\text{train}}$. However, it is non-trivial to construct a high-quality dataset $\mathcal{D}_{\text{train}}$ for $\mathcal{F}$ in SE tasks. As we discussed in Section 1, there are two inherent problems for SE data that can damage the usability of the datasets, i.e., the mislabelled samples and the class imbalance.

Mislabelled Samples. Most data collection techniques rely on web data (e.g., GitHub and Stack Overflow) to construct the training datasets [24, 61]. However, web data can be easily incorrectly labelled, leading to mislabelled samples in the collected training datasets [21, 50, 60]. More specially, in this work, we refer to these samples whose labels are corrupted as the mislabelled samples, e.g., issue reports that describe defects but are not classified as such.

Definition 1 (**Mislabelled Sample.**). *A sample x_i is a mislabelled sample when its assigned label y_i does not match the ground-truth label y_i^*, i.e., $y_i \neq y_i^*$.*

Class Imbalance. Most datasets of SE tasks typically show an imbalanced class distribution [23], where a small portion of classes have massive samples but the others are associated with only a few samples. For example, the datasets of the defect prediction task typically contain much more non-defective samples than defective ones due to the nature of software programs [46, 47]. Formally, we define a class imbalanced dataset as follows:

Definition 2 (**Class Imbalance.**). *A training dataset $\mathcal{D}_{\text{train}}$ is class imbalanced when one class contains significantly fewer samples than the other classes.*

2.3 Problem Statement

The training dataset $\mathcal{D}_{\text{train}}$ that suffers from the mislabelled samples and the class imbalance can bias the learning process and damage the effectiveness of the deep predictive model $\mathcal{F}$ [23, 37, 54, 62].

Our problem is thus defined as follows. Given a training dataset $\mathcal{D}_{\text{train}}$ in which the mislabelled samples and the class imbalance coexist, how can we effectively and efficiently learn a robust deep predictive model $\mathcal{F}$ against both the mislabelled samples and the class imbalance?

3 THE ROBUSTTRAINER FRAMEWORK

In this paper, we propose a novel learning approach, named RobustTrainer, to robustly learn deep predictive models in the presence of both sample mislabelling and class imbalance issues. The novelty of RobustTrainer lies in a two-stage learning process, in which each stage tackles one of two issues. The **Representation Learning** stage focuses on learning feature representations robust to the mislabelled samples, and the **Classifier Learning** stage aims to build a balanced classifier upon the representations learned in the representation learning stage. This is inspired by the observation that *the mislabelled samples mainly damage the learned representations in the deep predictive model while the class imbalance mainly affects the classifier part in the deep predictive model* [25, 65]. Therefore, by employing the two-stage learning process of deep predictive models, RobustTrainer can effectively alleviate the negative impacts induced by each of the two issues of mislabelling and class imbalance; and by building the second stage on top of the results of the first stage, RobustTrainer can obtain better deep predictive models after the learning process.

Overview. Figure 1 presents the overall pipeline of RobustTrainer. Given a training dataset $\mathcal{D}_{\text{train}}$, the **Representation Learning** stage first learns feature representations that are robust to the mislabelled samples. Specifically, it performs two steps iteratively: (1) mislabelled sample detection, which aims to select clean samples out of mislabelled samples based on the feature representations of the samples, and (2) robust feature learning, which aims to learn robust feature presentations by training the model with the selected clean samples. Such an iterative method fulfils a positive cycle in which better clean samples will result in better feature representations, and better feature representations will identify better clean samples. As such, the representation learning stage can progressively reduce the negative impacts of the mislabelled samples, and

thus, produce robust feature representations for the deep predictive models.

After the representation learning stage, with the learned feature representations, the **Classifier Learning** stage further trains a classifier to mitigate the impacts of class imbalance. Specifically, we adopt class-balanced sampling [25] (one of the imbalanced learning approaches) in the classifier learning stage to train the classifier with balanced decision boundaries. Once trained, a predictive model $\mathcal{F}$ that is robust to both the mislabelled samples and the class imbalance is obtained.

Model Architecture. Figure 2 depicts the overall architecture of the deep predictive model $\mathcal{F}$ in RobustTrainer. To realise the idea of RobustTrainer, we decompose the architecture of a deep predictive model $\mathcal{F}$ into two components: (1) a feature extractor $\mathcal{G}$ that maps a sample x_i to a high-dimensional representation v_i, and (2) a classifier head f that receives v_i as an input and outputs class predictions $\hat{p}(x_i)$ for x_i. That is, we view the model $\mathcal{F}$ as $\mathcal{F}(x_i) = f(\mathcal{G}(x_i))$. In particular, we treat the function of the network layers before the logit layer in $\mathcal{F}$ as the feature extractor $\mathcal{G}$, and use the final full-connected layer in $\mathcal{F}$ as the classifier head f. In addition, we further introduce a projection head g that maps v_i into a low-dimensional representation z_i to help us identify the mislabelled samples, in the representation learning stage. Note that we will discard this projection head g after the representation learning stage finishes.

4 REPRESENTATION LEARNING

In the representation learning stage, we aim to learn feature representations that are robust to mislabelled samples. Intuitively, better feature representations can be obtained as the training dataset becomes less mislabelled. With this intuition, RobustTrainer proposes to first select clean samples out of the mislabelled samples, and then use the selected samples to learn feature representations. In the following subsections, we describe the details of the representation learning stage, including how to identify mislabelled samples (mislabelled sample detection, Section 4.1), how to update the networks for learning robust representations (robust feature learning, Section 4.2), and the overall training scheme of the representation learning stage (Section 4.3).

4.1 Mislabelled Sample Detection

The goal of the mislabelled sample detection is to construct a clean subset $\mathcal{D}_{\text{clean}} \subset \mathcal{D}_{\text{train}}$ that has as less mislabelled samples as possible. As such, the feature representations learned from $\mathcal{D}_{\text{clean}}$ would be less impacted by the mislabelled samples.

To achieve this goal, the key challenge is to ensure that the clean samples can be accurately filtered out from mislabelled ones. One popular criterion to identify the mislabelled samples is the small-loss trick [16, 45, 64], i.e., the samples with larger training losses would be more likely to be mislabelled samples. However, it is known that a deep predictive model learned on the class imbalanced dataset would favour the majority classes (i.e., classes have more samples) but hurt the minority classes (i.e., classes have few samples) [23, 25]. Therefore, it is not trustworthy to use the small-loss trick to detect mislabelled samples because both clean and mislabelled samples of minority classes would have large losses when learning on the class imbalanced dataset.

Instead of relying on training losses for identifying mislabelled samples, this work distinguishes clean from mislabelled samples by exploring the feature representations of samples that are proven robust to class imbalance [25]. The intuition is that samples from the same class should be intrinsically similar (having similar feature representations), while samples mislabelled into the class are generally not [3]. Therefore, we can decide whether a sample $(x_i, y_i) \in \mathcal{D}_{\text{train}}$ is mislabelled or not by measuring the similarity between the sample x_i and the samples belonging to the class y_i. If the sample x_i has a high similarity with the samples of class y_i, then it indicates that x_i is likely to be correctly labelled. Furthermore, it is worth noting that representations learned with mislabelled samples are discriminative enough to distinguish clean samples from mislabelled ones, regardless of their negative impact on the classifiers [37]. Below, we elaborate on each step of mislabelled sample detection with more technical details.

4.1.1 Extracting Feature Representation. The first step of mislabelled sample detection is to generate a feature representation for each sample $(x_i, y_i) \in \mathcal{D}_{\text{train}}$. A naive approach would directly use the representations v produce by the feature extractor $\mathcal{G}$. However, the representations v are typically high-dimensional, leading to high computational complexity, especially for the large-scale dataset. To address this problem, we follow prior work [31, 32] to adopt a projection head g to map the representations v into a low-dimensional space. That is, we generate the feature representation z_i for a sample x_i by $z_i = g(v_i) = g(\mathcal{G}(x_i))$. In practice, similar to previous work [31, 32], we implement the projection head g using a multiple layer perceptron (MLP) with one hidden layer.

4.1.2 Measuring Sample Similarity. After obtaining feature representations for the samples in the dataset $\mathcal{D}_{\text{train}}$, we determine whether a sample (x_i, y_i) is mislabelled or not by measuring the similarity between its representation z_i and other representations of samples that belong to the class y_i. To this end, given two low-dimensional representations z_i and z_j, we first measure their similarity by calculating the cosine distance between them, i.e., $d(z_i, z_j) = z_i^T z_j / (\|z_i\| \|z_j\|)$. The rationale for considering the cosine distance here is that the cosine distance can better measure the representation similarity than other distance measurements (e.g., the Euclidean distance) [18]. Then, the similarity between the sample x_i and samples of the class y_i can be accordingly computed as $\sum_{(x_j, y_j) \in \mathcal{D}_{\text{train}}, y_j = y_i} d(z_i, z_j)$, where z_i and z_j are the representations of samples x_i and x_j, respectively. However, comparing the sample (x_i, y_i) to each sample of the class y_i is time expensive, which is difficult to scale to large-scale training dataset.

To enable the efficient detection of the mislabelled samples, we instead use a set of class prototypes to represent each class category, and decide whether the label y_i of the sample x_i is mislabelled or not by comparing its feature z_i with the class prototypes of class y_i. To acquire the class prototypes for a class c, we perform k-means on the representations of all samples belonging to c, and then select the centroids of the k clusters as the class prototypes of the class c. Note that we use $\mathcal{Z}_c = \{z_{cl}\}_{l=1}^{k}$ to denote the set of class prototypes of the class c in the following. Furthermore, according to [29], we

set the value of k to be equal to $\lfloor\sqrt{\rho/2}\rfloor$ in this work, where ρ is the average number of instances per class in the training dataset $\mathcal{D}_{\text{train}}$. With the class prototypes of each class, we can measure the similarity score between an sample x_i and its label y_i by:

$$\text{score}\,(x_i, y_i) = \frac{1}{k} \sum_{z_{y_{il}} \in \mathcal{Z}_{y_i}} \frac{z_i^T z_{y_{il}}}{\|z_i\| \|z_{y_{il}}\|} \tag{2}$$

Intuitively, a higher value of score (x_i, y_i) suggests that the feature representation z_i of x_i has a larger similarity with the class prototypes of class y_i, and hence, indicates that y_i is much more likely to be a clean label.

4.1.3 Selecting Clean Samples. Based on the similarity scores (Equation 2) of the training samples, we next identify mislabelled samples from the training dataset $\mathcal{D}_{\text{train}}$ for constructing a clean subset $\mathcal{D}_{\text{clean}}$. One possible approach to detect the mislabelled samples is to sort all samples of class c in increasing order with their similarity scores (Equation 2). Then, selecting the first m_c samples with low scores as the mislabelled samples. However, it is non-trivial to find the optimal threshold m_c to select the mislabelled samples because this requires estimating the rate of mislabelled samples for class c. Inspired by previous work [30, 57], we avoid setting a threshold m_c to identify mislabelled samples by employing a two-component Gaussian Mixture Model (GMM) [41]. Intuitively, in one class, clean samples would have higher similarity scores than mislabelled ones, and thus a mixture of two Gaussian components can be observed on the similarity scores [41]. Therefore, we employ a two-component GMM to model the distribution of the samples' similarity scores in a class and treat the samples in the Gaussian component with a large mean value as the clean ones.

More specifically, we iteratively apply the GMM for each class c to select the clean samples for this class. Let $\mathcal{D}_c = \{x_i | y_i = c\}$ denote the samples belonging to the class c. We fit a two-component GMM on the representations of samples in $\mathcal{D}_c$ to maximise the log-likelihood value by

$$\max \sum_{i=1}^{|\mathcal{D}_c|} \log\left(\sum_{j=1}^{2} \phi_j \mathbb{P}\left(s_i | u_j, \sigma_j\right)\right) \tag{3}$$

where $s_i = \text{score}\,(x_i, c)$ is the similarity score of the example $x_i \in \mathcal{D}_c$ computed in Equation 2, u_j and σ_j are the mean and variance of the j-th component, and ϕ_j denotes the weight of the j-th component and $\sum_{j=1}^{2} \phi_j = 1$. Once the GMM is trained, we can use the model to determine whether an sample $x_i \in \mathcal{D}_c$ is clean or not. Without loss of generality, let us assume the means u_1 and u_2 of the two components in the GMM satisfy $u_1 > u_2$. Then, we flag a sample $x_i \in \mathcal{D}_c$ as the clean one if and only if $\mathbb{P}\,(s_i | u_1, \sigma_1) > \mathbb{P}\,(s_i | u_2, \sigma_2)$, where $\mathbb{P}\left(s_i | u_j, \sigma_j\right)$ is the probability that the sample x_i belong to the j-th component in the GMM. That is to say, a sample $x_i \in \mathcal{D}_c$ is determined as a clean sample when it belongs to a Gaussian component with a larger mean, indicating that x_i has a high similarity to the class prototypes of its class label. As such, we can obtain a clean subset $\mathcal{S}_c = \{x_i | \mathbb{P}\,(s_i | u_1, \sigma_1) > \mathbb{P}\,(s_i | u_2, \sigma_2) \wedge x_i \in \mathcal{D}_c\}$ for the class c, and hence ultimately, the clean dataset $\mathcal{D}_{\text{clean}}$ is obtained by $\mathcal{D}_{\text{clean}} = \bigcup_{c=1}^{C} \mathcal{S}_c$.

Remark. Compared to existing mislabel cleaning methods in the SE community [15, 27, 48] which are specific to the defect prediction task, the mislabelled sample detection of RobustTrainer is more general because the detection leverages the feature representations which can be extracted from all types of samples instead of the raw samples themselves. Therefore, RobustTrainer is available for SE tasks beyond the defect prediction task. We demonstrate the generalisation of RobustTrainer in Section 6.

4.2 Robust Feature Learning

Given the clean dataset $\mathcal{D}_{\text{clean}}$ produced by the mislabelled sample detection, the robust feature learning aims to learn feature representations using the supervision information of the samples in $\mathcal{D}_{\text{clean}}$. Using dataset $\mathcal{D}_{\text{clean}}$, the learning of feature representations would rarely be affected by the mislabelled samples, thus improving the quality of the learned representations. In particular, we design the following two objectives/losses for learning the representations.

• **Prototypical Contrastive Loss.** The prototypical contrastive loss enforces each sample to be pulled towards the prototypes of its corresponding class and pushed away from prototypes of all other classes. In this way, we can learn a feature space with the property of intra-class compactness and inter-class separability, which is beneficial for both classification and mislabelled sample detection. Mathematically, the prototypical contrastive loss of a training sample $(x_i, y_i) \in \mathcal{D}_{\text{clean}}$ is calculated as:

$$\mathcal{L}_{pcl}\,(x_i, y_i) = -\frac{1}{k} \sum_{z_{y_{il}} \in \mathcal{Z}_{y_i}} \log \frac{\exp\left(z_i \cdot z_{y_i l}\right)}{\sum_{c=1, c \neq y_i}^{C} \sum_{z_{cj} \in \mathcal{Z}_c} \exp\left(z_i \cdot z_{cj}\right)} \tag{4}$$

where k is the number of prototypes per class, z_{cj} denotes the representation of the j-th prototype of class c, which is obtained as described in Section 4.1.2. Note that we apply the prototypical contrastive loss on the representation z produced by the projection head g but not the representation v generated by the feature extractor $\mathcal{G}$. This is inspired by the practice of SimCLR [8], which shows that more information can be formed and maintained in v by leveraging the projection head g.

• **Classification Loss.** Following the common training strategy for the classification task, we also employ the cross-entropy loss as the classification loss to stabilise the convergence and achieve better representations in RobustTrainer. Formally, the classification loss of a training sample $(x_i, y_i) \in \mathcal{D}_{\text{clean}}$ is computed by:

$$\mathcal{L}_{cl}\,(x_i, y_i) = -log\left(\hat{p}_{y_i}\,(x_i)\right) \tag{5}$$

where $\hat{p}_{y_i}\,(x_i)$ is the class probability of y_i.

By combining the above objectives, the overall training loss is:

$$\mathcal{L}_{total} = w_1 \mathcal{L}_{cl} + w_2 \mathcal{L}_{pcl} \tag{6}$$

where hyper-parameters w_1 and w_2 are trade-off parameters. For simplicity, we set $w_1 = w_2 = 1$ in this work and leave the study of weight optimisation for future work.

4.3 Iterative Representation Learning

Algorithm 1 shows the high-level pseudo-code of the representation learning stage, aiming at learning robust feature representations by alternatively performing the mislabelled sample detection (Section 4.1) and the robust feature learning (Section 4.2) at each epoch. The inputs to the algorithm are a training dataset $\mathcal{D}_{\text{train}}$ which

Algorithm 1: Iterative Representation Learning

input: training Dataset $\widetilde{\mathcal{D}}_{\text{train}} = \{x_i, \widetilde{y}_i\}_{i=1}^{N}$; number of maximum epochs T; number of warm-up epochs T_0; batch size B, structure of $\mathcal{G}$, f, and g

1 **for** $t \leftarrow 1$ **to** T **do**
2 **if** $t < T_0$ **then**
3 $\mathcal{D}_{\text{clean}} \leftarrow \widetilde{\mathcal{D}}_{\text{train}}$;
4 **else**
5 $\mathcal{D}_{\text{clean}} \leftarrow \emptyset$;
6 **for** $c \leftarrow 1$ **to** C **do**
7 Select k class prototypes $\{z_{cl}\}_{l=1}^{k}$;
8 Compute similarity scores for each $x_i \in \mathcal{D}_c$ by Eq. 2;
9 Fit GMM and select the clean subset $\mathcal{S}_c$;
10 $\mathcal{D}_{\text{clean}} \leftarrow \mathcal{D}_{\text{clean}} \bigcup \mathcal{S}_c$;
11 **end**
12 **end**
13 **for** $n \leftarrow 1$ **to** $\frac{|\mathcal{D}_{\text{clean}}|}{B}$ **do**
14 Fetch n-th mini-batch $\mathcal{B}$ from $\mathcal{D}_{\text{clean}}$;
15 Calculate the loss $\mathcal{L}_{total}$ by Eq. 6;
16 Update $\mathcal{G}$, f, and g to minimize $\mathcal{L}_{total}$;
17 **end**
18 **end**
19 **return** *the feature extractor $\mathcal{G}$ and the clean subset $\mathcal{D}_{clean}$*

may be mislabelled and may have class imbalance, the number of maximum (warm-up) epochs T (T_0), the batch size B, and the networks required to be trained including the feature extractor $\mathcal{G}$, the classifier head f, and the projection head g.

At the beginning, we warm-up the networks for T_0 epochs by training on the dataset $\mathcal{D}_{\text{train}}$ using the loss defined in Equation 6. The rationale is that the feature representations are not available when the networks have not been trained, and hence, we can only treat all samples in $\mathcal{D}_{\text{train}}$ as clean samples (Line 3). Note that the mislabelled samples can be ignored at the early learning phase [2, 33]. Therefore, such a warm-up training on $\mathcal{D}_{\text{train}}$ would only slightly affect the quality of learned representations. After that, at each training epoch, we first select clean samples out of mislabelled samples in $\mathcal{D}_{\text{train}}$ to construct a clean subset $\mathcal{D}_{\text{clean}}$ (Line 5- 11), and then update the networks by performing mini-batch gradient descent on the dataset $\mathcal{D}_{\text{clean}}$ with the training loss defined in Equation 6 (Line 13- 17). As such, these two steps benefit each other, resulting in both better clean subset and better feature representations. Finally, after the learning stage is finished, this algorithm returns the trained feature extractor $\mathcal{G}$ as well as the clean subset $\mathcal{D}_{\text{clean}}$ selected in the last epoch for training classifiers in the next classifier learning stage (Line 19).

5 CLASSIFIER LEARNING

With the pre-trained robust representations from the representation learning stage, the classifier learning stage aims to learn a classifier robust to the class imbalance. To achieve it, there are two major challenges to be addressed. The first one is how to construct the new classifier? To overcome this challenge, we apply a new classifier head f' on the top of the pre-trained feature extractor $\mathcal{G}$ produced by the representation learning stage for constructing the classifier. That is, the new classifier in the classifier learning stage is $f' \circ \mathcal{G}$. Specifically, we use the same structure of the original classifier head f for f', in order to maintain the architecture of the target predictive model $\mathcal{F}$. Furthermore, we keep the feature extractor $\mathcal{G}$ fixed during the classifier learning stage; Thus, the new classifier head f' can benefit from the robust representations learned from the representation learning stage. The second challenge is how to mitigate the influence of class imbalance in training the new classifier f'. To overcome this challenge, we adopt the class-balanced sampling [25] to generate the mini-batch $\mathcal{B}$, and employ the standard cross-entropy loss to optimise the parameters of the new classifier head f'. In addition, we only use the clean subset $\mathcal{D}_{\text{clean}}$ generated on the representation learning stage to train the new head f' to further reduce the threats of mislabelled samples. Note that other methods on handling the class imbalance, for example, using class imbalance robust loss functions [5, 35, 36], can also be applied to train the new classifier head f'. In this work, we mainly consider the class-balanced sampling due to its simplicity and effectiveness [25], and leave the exploration of other methods for future work.

Remark. Directly training deep predictive models via prior imbalanced learning approaches (e.g., SMOTE [7]) fails to obtain desirable classifiers when the training dataset suffers from both the mislabelled samples and the class imbalance. In contrast, RobustTrainer builds the classifier learning stage on top of the results of the representation learning stage. Hence, the adopted class-balanced sampling method would be less impacted by the mislabelled samples and thus generate more robust classifiers. We discuss this in more detail in Section 6.

6 EVALUATION DESIGN

Our evaluation aims to address the following research questions:

- **RQ1: Effectiveness of RobustTrainer.** How effective are the predictive models trained via RobustTrainer?
- **RQ2: Impact analysis of individual components.**
  - **RQ2a: Impact of mislabelled sample detection.** How does to the mislabelled sample detection impact the effectiveness of RobustTrainer?
  - **RQ2b: Impact of robust feature learning.** How does the robust feature learning impact the effectiveness of RobustTrainer?
  - **RQ2c: Impact of classifier learning.** How does the classifier learning stage impact the effectiveness of RobustTrainer?
- **RQ3: Efficiency of RobustTrainer.** How does RobustTrainer perform in terms of efficiency?

6.1 Datasets and Models

In our study, we evaluate RobustTrainer's ability to train models for two SE tasks, i.e., Bug Report Classification (BRC) and Software Defect Prediction (SDP), where the deep predictive models are popularly used [62].

Datasets. To ensure high practicality and realism of our evaluation, we opt to use the issue datasets provided by Herzig et al. [21] for the BRC task, and the defect datasets provided by Yatish et al. [63] for the SDP task. We select these two datasets because (1) the datasets are collected from the real world so that the mislabelled samples and the class imbalance are realistic, and (2) the datasets are representative benchmarks for each subject task.

• **BRC Dataset.** The BRC dataset [21] contains 5,591 issue reports collected from 3 open-source projects [2]. For this dataset, Herzig et al. manually inspected each issue and observed that more than 40% issues are incorrectly labelled [21]. In our study, we treat the raw issue types as the sample labels and the manually inspected issue types as the ground-truth sample labels. Moreover, Herzig et al. classified the issues into six types, i.e., BUG, RFE, DOC, IMPR, REFAC, and OTHER, but the BRC task only focuses on separating bug and non-bug issues. Thus, we follow prior work [39, 43] to regard issues of BUG as bug issues and others as non-bug issues.

• **SDP Dataset.** The SDP dataset [63] consists of 32 releases that span 9 open-source software systems. Each module has 65 software metrics including 54 code metrics, 5 process metrics, and 6 human metrics. Yatish et al. [63] compared the modules labelled by the heuristic approaches and the software development teams and found that more than half of defective modules are mislabelled by the heuristic approaches. According to this, we adopt the labels produced by the heuristic approaches as the sample labels and the manually generated labels as the ground-truth sample labels.

Table 1 shows the statistical information of the studied datasets. In this table, Columns "Dataset", "Project", and "Studied Release" list the studied datasets, their corresponding projects, and release versions, respectively. Column "# sample" presents the total number of samples. Column "Mislabel Rate" presents the ratio of mislabelled samples. Column "Minority Rate" presents the ratio of samples belonging to the minority class (i.e., the bug issue in the BRC and the defective module in the SDP). Table 1 shows that on the one hand, these datasets contain substantial mislabelled samples where the mislabel rate ranges 2%-29%; on the other hand, they suffer from the class imbalance, where the samples of the minority class are significantly smaller than those of the majority class.

Deep Predictive Models. For the BRC task, we consider the attention-based Bi-directional Long Short-Term Memory (Bi-LSTM) network as the predictive model [67]. In particular, we use the summary of an issue as the input of the model. We set the size of Bi-LSTM to be 256, and the number of Bi-LSTM layers to be 1. For the SDP task, we adopt a feedforward neural network (FNN) as the predictive model [6]. This FNN consists of an input layer that takes the 65 software metrics as input, two hidden layers respectively with 500 neurons and 1000 neurons, and an output layer with two neurons that performs classification. Note that, since our goal is to investigate whether RobustTrainer can effectively learn deep predictive models on noisy and imbalanced training datasets, we do not tune the parameters of these two models in our experiments.

[2]The original dataset contains 7,401 issues from 5 projects [21]. However, only issues from Jackrabbit, Lucene, and HttpClient are used in prior work [39, 43]. We follow the prior practice and use the issues from the 3 projects in our study.

Table 1: The statistical summary of the studied dataset.

Dataset	Project	# samples	Mislabel Rate	Minority Rate	Studied Release
BRC	Jackrabbit	2402	14%	39%	-
	Lucene	2443	15%	28%	-
	HttpClient	746	21%	40%	-
SDP	ActiveMQ	1884-3420	5%-14%	6%-15%	5.0.0,5.1.0,5.2.0,5.3.0,5.8.0
	Camel	1515-8846	2%-28%	2%-18%	1.4.0,2.9.0,2.10.0,2.11.0
	Derby	1963-2705	12%-29%	14%-33%	10.2.1.6,10.3.1.4,10.5.1.1
	Groovy	757-884	8%-13%	3%-8%	1.5.7,1.6.0.Beta1,1.6.0.Beta2
	HBase	1059-1834	20%-26%	20%-26%	0.94.0,0.95.0,0.95.2
	Hive	1416-2662	8%-18%	8%-19%	0.9.0,0.10.0,0.12.0
	Jruby	731-1614	9%-22%	5%-18%	1.1,1.4,1.5,1.7
	Lucene	805-2806	12%-23%	3%-24%	2.3.0,2.9.0,3.0.0,3.1.0
	Wicket	1672-2578	6%-18%	4%-7%	1.3.0.beta1,1.3.0.beta2,1.5.3

[a] Symbol '-' means not available at those cases

6.2 Compared Approaches

Existing Approaches. To the best of our knowledge, as of this writing, there are no approaches that can simultaneously tackle the mislabelling and class imbalance in SE tasks. Note that although there are a few approaches that address both these issues in the image classification domain, they cannot be applied to SE tasks due to the image-specific data augmentations [26, 57] or the specific network architectures [56]. Therefore, to demonstrate the effectiveness of RobustTrainer, in RQ1, we compare RobustTrainer with the following state-of-the-art techniques that target the mislabelled samples or the class imbalance:

• **SCE** (Standard Cross-entropy) represents the standard training scheme which simply trains models with the cross-entropy (CE) loss. SCE is regarded as the baseline in our study.

• **Co-teaching** [16] focuses on the mislabelled samples. It simultaneously maintains two models and each model selects small-loss samples to teach the peer model during the training process. After training, co-teaching predicts a sample based on the sum of the outputs from the two models.

• **JoCoR** [55] is similar to Co-teaching focusing on mislabelling. The main difference is that JoCoR trains the two models with a joint loss to maximise the agreement between them and meanwhile uses this joint loss to select the clean samples. JoCoR predicts a sample based on the ensemble of the two trained models.

• **LDAM** [5] addresses the class imbalance and proposes the label-distribution-aware margin (LDAM) loss, which encourages the minority classes to have larger margins (i.e., the distances to decision boundaries), to learn deep predictive models.

• **Decoupling** [25] addresses the class imbalance by decoupling the learning process of deep predictive models, which first adopts SCE to learn a feature extractor and then uses the class-balanced sampling to build a classifier upon the learned feature extractor.

• **SMOTE** [7] is for class imbalance, which re-balances datasets by creating new samples for minority classes using interpolation between near neighbours. In our study, it is applied only on SDP because the interpolation strategy is specific to numerical samples.

Variants of RobustTrainer. To answer RQ2, we propose the following three variants of RobustTrainer to analyse the impacts of each component in RobustTrainer.

• **RobustTrainer**$_{noM}$ removes the mislabelled sample detection from the representation learning stage. That is, it directly uses the

original training datasets in the whole learning process. This variant aims to investigate the contribution of the mislabelled sample selection in RobustTrainer.

• **RobustTrainer**$_{noR}$ removes the robust feature learning from the representation stage. That is, it replaces the default loss function (Equation 6) with the standard CE loss; in addition, the mislabelled samples are detected based on the representations produced from the feature extractor $\mathcal{G}$. This variant aims to investigate the contribution of the robust feature learning in RobustTrainer.

• **RobustTrainer**$_{noC}$ removes the classifier learning stage from RobustTrainer. That is, it directly uses the classifier head f from the representation learning stage for the predictive model $\mathcal{F}$. This variant aims to investigate the contribution the classifier learning stage in RobustTrainer.

6.3 Measurements

To measure the performance of the deep predictive models, we use four metrics, i.e., F-measure, G-measure, Matthews Correlation Coefficient (MCC), and Area Under the Curve (AUC), which have been used and shown to be important in related work [12, 19]. In general, there are four types of predictions for a model: (1) True Positive (TP), where the model correctly predicts the positive class (bug report or defect module); (2) True Negative (TN), where the model correctly predicts the negative class (non-bug report or non-defect module); (3) False Positive (FP), where the model incorrectly predicts the positive class; and (4) False Negative (FN), where the model incorrectly predict the negative class. Accordingly, based on these four possible prediction results, the performance metrics F-measure, G-measure, MCC and AUC are defined as follows.

• **F-measure**. F-measure is the harmonic mean of precision and recall. It ranges from 0 to 1, and a higher F-measure indicates more accurate predictions. F-measure is calculated as $F-measure = \frac{2 \cdot precision \cdot recall}{precision+recall}$, where precision is defined as $precision = \frac{TP}{TP+FP}$ and recall (True Positive Rate, TPR) is defined as $recall = \frac{TP}{TP+FN}$.

• **G-measure**. G-measure is the harmonic mean of TPR and True Negative Rate (TNR). It ranges from 0 to 1, and a larger G-measure means better model performance. G-measure is computed as $G-measure = \frac{2 \cdot TPR \cdot TNR}{TPR+TNR}$, where TNR is defined as $TNR = \frac{TN}{TN+FN}$.

• **MCC**. MCC is calculated based on true and false positives and negatives. It ranges from −1 to 1, where 1 means perfect prediction, 0 means random prediction, and −1 means all predictions failed. MCC is calculated as: $MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.

• **AUC**. AUC is the area under the Receiver Operating Characteristic (ROC) curve that plots TPR against false positive rate $FPR = \frac{FP}{FP+TN}$, while varying the threshold that is used to determine whether a sample is predicted as positive class or negative class. It ranges from 0 to 1, where 0.5 means a random prediction, and 1 means a perfect prediction.

Note that, among these four performance metrics, G-measure, MCC, and AUC are all robust against the class imbalance, but F-measure is sensitive toward the class imbalance.

Statistical Testing. In our study, we also investigate whether the studied approaches show a statistically significant performance difference. To do this, we use the Wilcoxon signed-rank test [58] with Bonferroni correction [1] at the significant level of 0.05 to investigate statistical significance between RobustTrainer and the compared approaches, and further adopt the Cliff's delta effect size [9] to quantify the magnitude of the difference. Following the existing work [13], a Cliff's delta less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474, and larger than 0.474 is considered as a negligible, small, medium, and large effect size, respectively.

Table 2: Effectiveness of studied approaches on the BRC task.

Method	Performance Metrics				Statistical Testing			
	F-Meas.	G-Meas.	MCC	AUC	F-Meas.	G-Meas.	MCC	AUC
SCE	0.66	0.70	0.40	0.70	1.00 (L)***	1.00 (L)***	1.00 (L)***	1.00 (L)***
Co-teaching	0.70	0.75	0.49	0.75	1.00 (L)***	1.00 (L)***	1.00 (L)***	1.00 (L)***
JoCoR	0.71	0.74	0.50	0.75	1.00 (L)***	1.00 (L)***	1.00 (L)***	1.00 (L)***
LDAM	0.65	0.70	0.41	0.70	1.00 (L)***	1.00 (L)***	1.00 (L)***	1.00 (L)***
Decoupling	0.67	0.71	0.42	0.71	1.00 (L)***	1.00 (L)***	1.00 (L)***	1.00 (L)***
RobustTrainer	**0.73**	**0.78**	**0.55**	**0.78**	-	-	-	-

[a] $^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$
[b] Symbol '-' means not available at those cases
[c] SMOTE [7] is omitted since its interpolation strategy fails to apply to this task.

6.4 Implementation

Dataset Partition. We use the following two setups to construct training and testing datasets for the subject tasks. For the BRC task, we follow previous work [39, 43] to combine all issues of the 3 projects as a new dataset. Then, we perform the 10-fold cross-validation with 10 repetitions on this new dataset. For the SDP task, we follow related work [19] to conduct Cross-Project Defect Prediction (CPDP) on the SDP dataset. For each project, we also repeat the experiment 10 times to avoid randomness. Furthermore, we use the ground-truth labels for samples in the testing dataset to evaluate the learned models for all experiments.

Hyperparameters. The settings of RobustTrainer are the following: we globally use the widely used learning rate of 0.001, batch size of 256, and training epoch of 200 in the representation learning stage, and learning rate of 1e−5, batch size of 256, and training epoch of 10 in the classifier learning stage. Regarding the compared approaches, we use the settings same as RobustTrainer for a fair comparison. In addition, the Adam optimiser [28] is employed in all experiments.

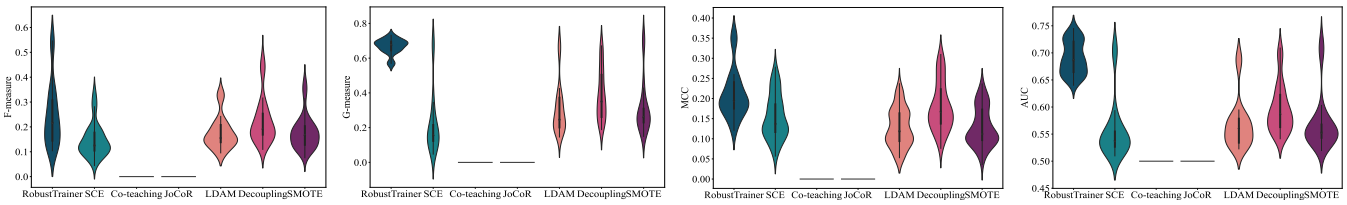
Environment. All experiments are conducted on a workstation with Intel Core i7-8700K CPU, 64G memory, and one GeForce GTX 1080Ti GPU, running Ubuntu 16.04 LTS. We build our experiments on Pytorch V1.6.0 [42].

7 RESULTS AND ANALYSIS

7.1 RQ1: Effectiveness of RobustTrainer

This RQ compares the effectiveness of RobustTrainer with the six compared approaches listed in Section 6.2 on the BRC task and the SDP task.

Results on the BRC task. Table 2 presents the effectiveness comparison results among all the studied approaches on the BRC task. In this table, Column "Method" lists all the studied approaches. Column "Performance Metrics" presents the achieved average values in terms of F-measure, G-measure, MCC, and AUC. Column "Statistical Testing" reports the results of the statistical testing when comparing RobustTrainer with each comparison approach. As



(a) F-measure **(b) G-measure** **(c) MCC** **(d) AUC**

Figure 3: Effectiveness of studied approaches on the SDP task.

shown in Table 2, RobustTrainer achieves the overall best performance among all the studied approaches, with an F-measure of 0.73, a G-measure of 0.78, a MCC of 0.55, and an AUC of 0.78, on average. More specifically, the average improvements of RobustTrainer over all the comparison approaches are remarkable, ranging from 2.8% (JoCoR) to 12.3% (LDAM) on F-measure, from 4.0% (Co-teaching) to 11.4% (LDAM) on G-measure, from 10.0% (JoCoR) to 37.5% (SCE) on MCC, and from 4.0% (JoCoR) to 11.4% (LDAM) on AUC, respectively. In addition, the results of statistical testing further suggest that the improvements in terms of all metrics achieved by RobustTrainer are all statistically significant, in which all cases show a p-value less than 0.001 and a large Cliff's delta effect size of 1.00. All these results demonstrate the effectiveness of RobustTrainer in learning deep predictive models.

From Table 2 we also observe that LDAM and Decoupling can only achieve comparable performance with the SCE baseline. The reason is that the BRC dataset is less imbalanced, where the imbalance ratio of the number of non-bug issues to the number of bug issues is on average 1.8. Therefore, the class imbalance issue, which is the target of the LDAM and Decoupling approaches, has less impact on the deep predictive models in the BRC task.

In contrast to these existing approaches that focus on only the mislabelled samples or the class imbalance, RobustTrainer addresses these two problems simultaneously, and thus, can generate better predictive models.

Results on the SDP task. Figure 3 shows the effectiveness of all the studied approaches on the CPDP task. The four sub-figures present the results in terms of F-measure, G-measure, MCC, and AUC, respectively. In each sub-figure, the x-axis presents all the studied approaches, while the y-axis presents the corresponding metric distributions using violin plots. From Figure 3, we can observe that RobustTrainer still substantially outperforms all the compared techniques in all metrics. Taking the best comparison approach Decoupling as an example, RobustTrainer outperforms Decoupling by 10.4% in F-measure, 71.7% in G-measure, 17.2% in MCC, and 15.8% in AUC, respectively. Furthermore, we also conduct statistical testing to compare the performance of RobustTrainer and the comparison approaches; The results suggest that RobustTrainer can also achieve significantly better performances than all the comparison approaches on the CPDP task. These experimental results again confirm the effectiveness of RobustTrainer.

Moreover, from Figure 3, we notice that the Co-teaching and JoCoR perform extremely poorly on the CPDP task, i.e., these two approaches both have an AUC value of 0.5, which indicates that the models are equal to the random guessing. The reason is that the class imbalance problem in the SDP dataset is much heavy, where the imbalance ratios are at least 14.0, i.e., the majority class (non-defect) have at least 14 times more samples than the minority class (defect). Therefore, the models trained on the SDP dataset would be skewed towards the majority class, and thus, the samples of the majority class tend to have small losses. As a result, the small-loss selection strategy used by Co-teaching and JoCoR bias toward selecting the samples of the majority class as the clean samples for training, further enhancing the class imbalance problem. On the contrary, RobustTrainer avoids the biased sample selection by adopting the feature representations instead of the loss values so that an effective predictive model can be learned by RobustTrainer. This observation further demonstrates the strength of RobustTrainer.

Table 3: Effectiveness of RobustTrainer and its variations on BRC.

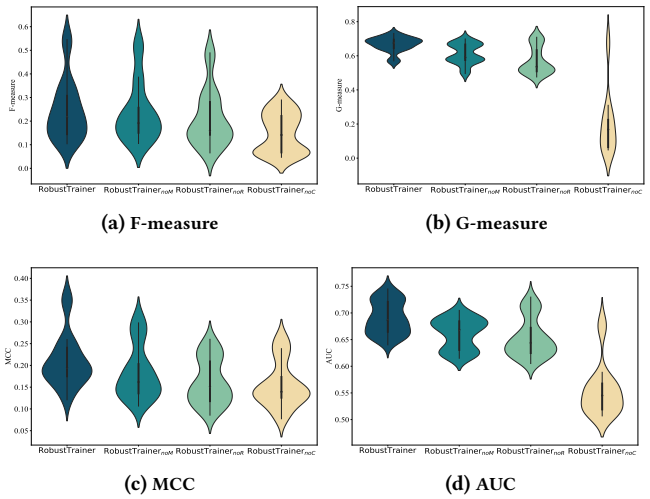
Method	F-measure	G-measure	MCC	AUC
RobustTrainer$_{noM}$	0.61	0.66	0.44	0.70
RobustTrainer$_{noR}$	0.62	0.67	0.42	0.70
RobustTrainer$_{noC}$	0.72	0.73	0.51	0.76
RobustTrainer	**0.73**	**0.78**	**0.55**	**0.78**

7.2 RQ2: Impact Analysis

In this RQ, we conduct a series of ablation studies to further analyse the impact of each component in RobustTrainer. Table 3 and Figure 4 show the comparison of effectiveness between the default RobustTrainer and its variants on BRC and SDP, respectively. The following observations are made based on the comparison results:

RQ2a: Impact of mislabelled sample detection. The results demonstrate the contribution of the mislabelled sample detection to the overall effectiveness of RobustTrainer. For example, without the mislabelled sample detection, the AUC values of RobustTrainer are decreased by approximately 11.4% on the BRC dataset and by approximately 4.6% on the SDP dataset. It confirms the importance of the mislabelled sample detection in RobustTrainer for learning effective deep predictive models.

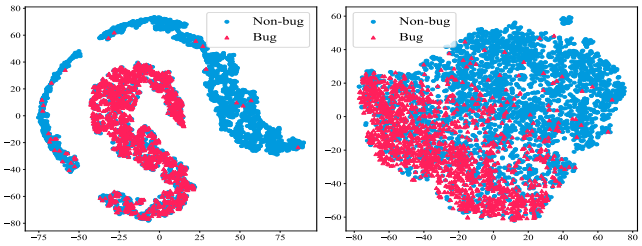
RQ2b: Impact of robust feature learning. The results demonstrate that the robust feature learning loss in Equation 6 positively contributes to the effectiveness of RobustTrainer. Fom example, in terms of AUC, RobustTrainer outperforms RobustTrainer$_{noR}$ by on average 11.4% and 5.7% on the BRC and the SDP, respectively. The reason is that the proposed loss (Equation 6) can help to learn discriminative features, which are useful for the mislabelled sample detection and classifier learning. To verify this, we employ t-distributed stochastic neighbour embedding (t-SNE) [52] to visualise the learned representations of these two methods on the BRC



(a) F-measure (b) G-measure

(c) MCC (d) AUC

Figure 4: Effectiveness of RobustTrainer and variations on SDP.



(a) RobustTrainer (b) RobustTrainer$_{noR}$

Figure 5: Visualisation of learned feature representations by RobustTrainer and RobustTrainer$_{noR}$ on the BRC task.

dataset in Figure 5. From this figure we can see that the representations learned by RobustTrainer are more class-discriminative compared to the ones learned by RobustTrainer$_{noR}$. This observation further demonstrates the usefulness of the proposed loss function in RobustTrainer.

RQ2c: Impact of classifier learning. The results demonstrate that the classifier learning stage is essential in RobustTrainer, especially for datasets that suffer heavily from class imbalance. For example, the average improvement of RobustTrainer over RobustTrainer$_{noC}$ in terms of AUC on the BRC dataset (average imbalance ratio 1.8) is 2.63%, compared to the average improvement improvement on the SDP dataset (average imbalance ratio 14.0) is 24.59%. This finding confirms the importance of the classifier learning stage in RobustTrainer to learn predictive models with balanced decision boundaries.

7.3 RQ3: Efficiency of RobustTrainer

In this RQ, we further analyse the efficiency of RobustTrainer in learning a deep predictive model. Table 4 presents the time cost of training a model for each studied approach on the BRC and the SDP datasets, respectively. As shown in this table, RobustTrainer requires more time to learn a deep predictive model compared to the other studied techniques, because RobustTrainer needs to evaluate the class prototypes and select the clean samples at each epoch. However, RobustTrainer needs to address the mislabelled samples and the class imbalance simultaneously, while the existing methods only handle one of these two problems. Therefore, we believe such a time cost of RobustTrainer is reasonable for achieving better deep predictive models. Furthermore, considering that the training process is offline, the training time of RobustTrainer (i.e., only several minutes) is acceptable in practice. To sum up, RobustTrainer can efficiently learn robust deep predictive models with an acceptable time cost.

Table 4: Time cost of studied approaches.

Method	BRC		SDP	
	Per-epoch	Total	Per-epoch	Total
SCE	0.2s	40.0s	0.3s	71.3s
Co-teaching	0.4s	78.8s	0.5s	90.6s
JoCoR	0.4s	80.8s	0.5s	95.1s
LDAM	0.2s	43.1s	0.4s	80.6s
Decoupling Stage 1	0.2s	40.0s	0.3s	71.3s
Decoupling Stage 2	0.2s	1.9s	0.6s	6.1s
SMOTE	-	-	0.4s	75.8s
RobustTrainer Stage 1	0.9s	189.7s	1.2s	222.0s
RobustTrainer Stage 2	0.1s	1.1s	0.3s	3.7s

[a] Symbol '-' means not available at those cases

8 DISCUSSION

Extensions of RobustTrainer. RobustTrainer has demonstrated remarkable effectiveness in learning deep predictive models against both the mislabelled samples and the class imbalance. Nevertheless, we believe it can be further improved in the following aspects in the future. First, RobustTrainer uses the samples selected by the mislabelled sample detection component to learn the deep predictive models. It is also possible to utilise the mislabelled samples, by using pseudo-labelling methods such as semi-supervised learning [53] to correct the mislabels. We will investigate this possibility in the future. Second, the experiments in this work evaluated the performance of RobustTrainer in the domains of text samples (the BRC task) and numerical samples (the SDP task), but RobustTrainer can be generalised, e.g., to the domain of graph samples such as AST graphs or control-flow graphs. The generalisability of RobustTrainer comes from that its learning process only relies on the model outputs such as the feature representations and predicted class probabilities, which can be extracted from all types of inputs. For future work, we plan to evaluate the performance of RobustTrainer in various domains of samples. Finally, we mainly focus on the deep predictive models that perform classification tasks. While such classification models are the most widely used ones in SE tasks [62], there are also other deep predictive models that perform regression tasks. In the future, we plan to extend RobustTrainer to learn the deep predictive models performing regression tasks.

Threats to Validity The main threat to **internal** validity lies in the correctness of implementation of RobustTrainer and the compared approaches. To mitigate this threat, we manually check our sour code and build them on state-of-the-art libraries (e.g., Pytorch [42]). Regarding the compared approaches, we directly adopt their open-source implementations. The main threat to **external** validity lies in the selection of the studied datasets. To reduce this threat, we perform experiments on two datasets that are collected from the real world (having realistic mislabelled samples and class

distributions) and are widely used in previous work. In addition, we also perform our experiments under various tasks and settings to strengthen the generality of the study. However, future study is still needed to examine the performance of RobustTrainer on other datasets and tasks. The main threat to **construct** validity lies in the performance metrics used in our study. To reduce this threat, we consider in total four metrics, namely AUC, F-measure, G-measure, MCC, which have been widely used in related work [12, 19].

9 RELATED WORK

Deep Predictive Model. With the increased popularity of deep learning, many deep predictive models have been developed to solve a range of software engineering (SE) problems, which contribute to improving the efficiency of the development process and software quality. Common ones include defect prediction [51], bug report management [20], API issue classification [34], log analysis [66], and code smell detection [38]. Please refer to a recent survey for more details about deep predictive models [62]. In this work, we do not aim to design a new deep predictive model for a SE task but try to propose a training framework for learning robust models against both the mislabelled samples and the class imbalance.

Impact of Dataset Quality. A good deep predictive model heavily relies on the dataset it learns from. There have been numerous studies investigating the impact of mislabelling [13, 21, 27, 50, 59] and class imbalance [4, 15, 49] on the deep predictive models. In this work, we take a step forward and propose RobustTrainer that can effectively and efficiently address both the mislabelled samples and the class imbalance in the dataset.

Learning with Mislabelled Samples. The SE community mainly focuses on the mislabelling issue regarding the defect prediction task [15, 27, 48]. For example, Kim et al. [27] propose CLNI to eliminate mislabelled samples by exploring the neighbour information of each sample. However, these methods cannot generalise to other domains of samples (e.g., text samples) beyond the numeric samples.

More recently, learning with mislabelled samples has been widely studied in the AI community, which aims to train a model on the datasets containing mislabelled samples directly. Existing techniques of learning with mislabelled samples can be roughly classified into two families, namely, label correction [14, 40] and sample selection [16, 30, 45, 64]. The former method requires the estimation of mislabelling transition matrix to correct the sample labels, which is hard to estimate for high number of classes and in high mislabelling scenarios. The latter method tries to filter out the mislabelled samples from the clean ones based on the small-loss criterion, where the low-loss samples are assumed to have clean labels. However, all of these approaches assume a balanced class distribution; Thus, they easily produce less favourable performance when the dataset suffer from the class imbalance issue (as discussed in Section 7.1). In contrast, our proposed approach RobustTrainer can effectively handle the mislabelled samples in the presence of class imbalance.

Moreover, we notice that there is some recent work [26, 56, 57] attempting to handle the class imbalance and the mislabelled samples simultaneously, but these approaches are not applicable to the SE tasks because they rely on the image-specific data augmentations [26, 57] or the specific network architectures [56].

Learning with Class Imbalance. For datasets of class imbalance, researchers also have proposed several approaches to address the class imbalance issue. In general, existing imbalanced learning methods can be grouped into two categories: (1) data distribution re-balancing [7, 17, 44] which aims to re-sample the dataset to achieve a more balanced data distribution; and (2) class-balanced loss [5, 10] which focused on designing specific loss functions that better facilitate learning with imbalanced data. However, none of the existing imbalanced learning methods considers the potentially mislabelled samples in the training datasets, limiting their effectiveness in real-world scenarios. In contrast, RobustTrainer considers the mislabelled samples and the class imbalance simultaneously and is thus able to effectively learn a predictive model that is robust to both issues.

10 CONCLUSION

In this paper, we raise a problem, common in approaches that use deep prediction models to accomplish SE tasks, that mislabelled samples and class imbalance in the SE datasets can threaten the validity of the learned models. To address this problem, we propose a novel learning framework RobustTrainer consisting of a two-stage learning that is able to robustly learn deep predictive models against both mislabelled samples and class imbalance issues. Experimental results on two popular SE tasks demonstrate the effectiveness of RobustTrainer, where the deep prediction models learned by RobustTrainer significantly outperform the models learned by the other six comparison methods in all four evaluation metrics.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. This research is supported by the National Natural Science Foundation of China under Grants No.62032010 and No. 61972193, the Leading-edge Technology Program of Jiangsu Natural Science Foundation under Grant No. BK20202001, and the program B for Outstanding PhD candidate of Nanjing University.

REFERENCES

[1] Hervé Abdi et al. 2007. Bonferroni and Šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics* 3 (2007), 103–107.

[2] Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. 2021. Understanding and Improving Early Stopping for Learning with Noisy Labels. arXiv:2106.15853 https://arxiv.org/abs/2106.15853

[3] Charles Bouveyron and Stéphane Girard. 2009. Robust supervised classification with mixture models: Learning from data with uncertain labels. *Pattern Recognit.* 42, 11 (2009), 2649–2658. https://doi.org/10.1016/j.patcog.2009.03.027

[4] George G. Cabral, Leandro L. Minku, Emad Shihab, and Suhaib Mujahid. 2019. Class imbalance evolution and verification latency in just-in-time software defect prediction. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, Montreal, QC, Canada, 666–676. https://doi.org/10.1109/ICSE.2019.00076

[5] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Aréchiga, and Tengyu Ma. 2019. Learning Imbalanced Datasets with Label-Distribution-Aware Margin Loss. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). Curran Associates, Inc., Vancouver, BC, Canada, 1565–1576. https://proceedings.neurips.cc/paper/2019/hash/621461af90cadfdaf0e8d4cc25129f91-Abstract.html

[6] Qimeng Cao, Qing Sun, Qinghua Cao, and Huobin Tan. 2015. Software defect prediction via transfer learning based neural network. In *2015 First International*

Conference on Reliability Systems Engineering (ICRSE). IEEE, Beijing, China, 1–10. https://doi.org/10.1109/ICRSE.2015.7366475

[7] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.* 16 (2002), 321–357. https://doi.org/10.1613/jair.953

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, Virtual Event, 1597–1607. http://proceedings.mlr.press/v119/chen20j.html

[9] Norman Cliff. 2014. *Ordinal methods for behavioral data analysis*. Psychology Press, London, England, United Kingdom.

[10] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, Long Beach, CA, USA, 9268–9277. https://doi.org/10.1109/CVPR.2019.00949

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, Miami, Florida, USA, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[12] Davide Falessi, Aalok Ahluwalia, and Massimiliano Di Penta. 2022. The Impact of Dormant Defects on Defect Prediction: A Study of 19 Apache Projects. *ACM Trans. Softw. Eng. Methodol.* 31, 1 (2022), 4:1–4:26. https://doi.org/10.1145/3467895

[13] Yuanrui Fan, Xin Xia, Daniel Alencar da Costa, David Lo, Ahmed E. Hassan, and Shanping Li. 2021. The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction. *IEEE Trans. Software Eng.* 47, 8 (2021), 1559–1586. https://doi.org/10.1109/TSE.2019.2929761

[14] Jacob Goldberger and Ehud Ben-Reuven. 2017. Training deep neural-networks using a noise adaptation layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, Toulon, France. https://openreview.net/forum?id=H12GRgcxg

[15] Lina Gong, Shujuan Jiang, Rongcun Wang, and Li Jiang. 2019. Empirical Evaluation of the Impact of Class Overlap on Software Defect Prediction. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, San Diego, CA, USA, 698–709. https://doi.org/10.1109/ASE.2019.00071

[16] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). Curran Associates, Inc., Montréal, Canada, 8536–8546. https://proceedings.neurips.cc/paper/2018/hash/a19744e268754fb0148b017647355b7b-Abstract.html

[17] Hui Han, Wenyuan Wang, and Binghuan Mao. 2005. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In *Advances in Intelligent Computing, International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 3644)*, De-Shuang Huang, Xiao-Ping (Steven) Zhang, and Guang-Bin Huang (Eds.). Springer, Hefei, China, 878–887. https://doi.org/10.1007/11538059_91

[18] Jiangfan Han, Ping Luo, and Xiaogang Wang. 2019. Deep Self-Learning From Noisy Labels. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, Seoul, Korea (South), 5137–5146. https://doi.org/10.1109/ICCV.2019.00524

[19] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2018. A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches. *IEEE Trans. Software Eng.* 44, 9 (2018), 811–833. https://doi.org/10.1109/TSE.2017.2724538

[20] Steffen Herbold, Alexander Trautsch, and Fabian Trautsch. 2020. On the Feasibility of Automated Issue Type Prediction. arXiv:2003.05357 https://arxiv.org/abs/2003.05357

[21] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, San Francisco, CA, USA, 392–401. https://doi.org/10.1109/ICSE.2013.6606585

[22] Geoffrey Hinton and Terrence J Sejnowski. 1999. *Unsupervised learning: foundations of neural computation*. MIT press, Cambridge, Massachusetts, USA.

[23] Xiao-Yuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. 2017. An Improved SDA Based Defect Prediction Framework for Both Within-Project and Cross-Project Class-Imbalance Problems. *IEEE Trans. Software Eng.* 43, 4 (2017), 321–339. https://doi.org/10.1109/TSE.2016.2597849

[24] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE, Cleveland, OH, USA, 406–409. https://doi.org/10.1109/ICSME.2019.00070

[25] Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. 2020. Decoupling Representation and Classifier for Long-Tailed Recognition. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, Addis Ababa, Ethiopia. https://openreview.net/forum?id=r1gRTCVFvB

[26] Shyamgopal Karthik, Jérôme Revaud, and Chidlovskii Boris. 2021. Learning From Long-Tailed Data With Noisy Labels. arXiv:2108.11096 https://arxiv.org/abs/2108.11096

[27] Sunghun Kim, Hongyu Zhang, Rongxin Wu, and Liang Gong. 2011. Dealing with noise in defect prediction. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic (Eds.). ACM, Waikiki, Honolulu , HI, USA, 481–490. https://doi.org/10.1145/1985793.1985859

[28] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). San Diego, CA, USA. http://arxiv.org/abs/1412.6980

[29] Trupti M Kodinariya and Prashant R Makwana. 2013. Review on determining number of Cluster in K-Means Clustering. *International Journal* 1, 6 (2013), 90–95.

[30] Junnan Li, Richard Socher, and Steven C. H. Hoi. 2020. DivideMix: Learning with Noisy Labels as Semi-supervised Learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, Addis Ababa, Ethiopia. https://openreview.net/forum?id=HJgExaVtwr

[31] Junnan Li, Caiming Xiong, and Steven C. H. Hoi. 2021. Learning from Noisy Data with Robust Representation Learning. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, Montreal, QC, Canada, 9465–9474. https://doi.org/10.1109/ICCV48922.2021.00935

[32] Junnan Li, Caiming Xiong, and Steven C. H. Hoi. 2021. MoPro: Webly Supervised Learning with Momentum Prototypes. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, Virtual Event, Austria. https://openreview.net/forum?id=0-EYBhgw80y

[33] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. 2020. Gradient Descent with Early Stopping is Provably Robust to Label Noise for Overparameterized Neural Networks. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy] (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, Online [Palermo, Sicily, Italy], 4313–4324. http://proceedings.mlr.press/v108/li20j.html

[34] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. 2019. Pattern-based mining of opinions in Q&A websites. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, Montreal, QC, Canada, 548–559. https://doi.org/10.1109/ICSE.2019.00066

[35] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, Venice, Italy, 2999–3007. https://doi.org/10.1109/ICCV.2017.324

[36] Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. 2021. Long-tail learning via logit adjustment. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, Virtual Event, Austria. https://openreview.net/forum?id=37nvvqkCo5

[37] Diego Ortego, Eric Arazo, Paul Albert, Noel E. O'Connor, and Kevin McGuinness. 2021. Multi-Objective Interpolation Training for Robustness To Label Noise. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, virtual, 6606–6615. https://openaccess.thecvf.com/content/CVPR2021/html/Ortego_Multi-Objective_Interpolation_Training_for_Robustness_To_Label_Noise_CVPR_2021_paper.html

[38] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2021. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? *IEEE Trans. Software Eng.* 47, 1 (2021), 108–129. https://doi.org/10.1109/TSE.2018.2883603

[39] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. 2017. Automated Classification of Software Issue Reports Using Machine Learning Techniques: An Empirical Study. *Innov. Syst. Softw. Eng.* 13, 4 (2017), 279–297.

[40] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. 2017. Making Deep Neural Networks Robust to Label Noise: A Loss Correction Approach. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer

Society, Honolulu, HI, USA, 2233–2241. https://doi.org/10.1109/CVPR.2017.240
[41] Haim H. Permuter, Joseph M. Francos, and Ian Jermyn. 2006. A study of Gaussian mixture models of color and texture features for image classification and segmentation. *Pattern Recognit.* 39, 4 (2006), 695–706. https://doi.org/10.1016/j.patcog.2005.10.028
[42] Pytorch. 2022. Pytorch. https://pytorch.org/. [online, accessed 07-May-2022].
[43] Hanmin Qin and Xin Sun. 2018. Classifying Bug Reports into Bugs and Non-bugs Using LSTM. In *Proceedings of the Tenth Asia-Pacific Symposium on Internetware, Internetware 2018, Beijing, China, September 16-16, 2018*. ACM, Beijing, China, 20:1–20:4. https://doi.org/10.1145/3275219.3275239
[44] Li Shen, Zhouchen Lin, and Qingming Huang. 2016. Relay Backpropagation for Effective Learning of Deep Convolutional Neural Networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII (Lecture Notes in Computer Science, Vol. 9911)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer, Amsterdam, The Netherlands, 467–482. https://doi.org/10.1007/978-3-319-46478-7_29
[45] Yanyao Shen and Sujay Sanghavi. 2019. Learning with Bad Training Data via Iterative Trimmed Loss Minimization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 5739–5748. http://proceedings.mlr.press/v97/shen19e.html
[46] Qinbao Song, Yuchen Guo, and Martin J. Shepperd. 2019. A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. *IEEE Trans. Software Eng.* 45, 12 (2019), 1253–1269. https://doi.org/10.1109/TSE.2018.2836442
[47] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online Defect Prediction for Imbalanced Data. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, Florence, Italy, 99–108. https://doi.org/10.1109/ICSE.2015.139
[48] Wei Tang and Taghi M. Khoshgoftaar. 2004. Noise identification with the k-means algorithm. In *16th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, Boca Raton, FL, USA, 373–378. https://doi.org/10.1109/ICTAI.2004.93
[49] Chakkrit Tantithamthavorn, Ahmed E. Hassan, and Kenichi Matsumoto. 2020. The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models. *IEEE Trans. Software Eng.* 46, 11 (2020), 1200–1219. https://doi.org/10.1109/TSE.2018.2876537
[50] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, Akinori Ihara, and Ken-ichi Matsumoto. 2015. The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, Florence, Italy, 812–823. https://doi.org/10.1109/ICSE.2015.93
[51] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, Austin, TX, USA, 321–332. https://doi.org/10.1145/2884781.2884857
[52] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html
[53] Jesper E. van Engelen and Holger H. Hoos. 2020. A survey on semi-supervised learning. *Mach. Learn.* 109, 2 (2020), 373–440. https://doi.org/10.1007/s10994-019-05855-6
[54] Shuo Wang and Xin Yao. 2013. Using Class Imbalance Learning for Software Defect Prediction. *IEEE Trans. Reliab.* 62, 2 (2013), 434–443. https://doi.org/10.1109/TR.2013.2259203
[55] Hongxin Wei, Lei Feng, Xiangyu Chen, and Bo An. 2020. Combating Noisy Labels by Agreement: A Joint Training Method with Co-Regularization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, Seattle, WA, USA, 13723–13732. https://doi.org/10.1109/CVPR42600.2020.01374
[56] Tong Wei, Jiang-Xin Shi, Yu-Feng Li, and Min-Ling Zhang. 2021. Prototypical Classifier for Robust Class-Imbalanced Learning. arXiv:2110.11553 https://arxiv.org/abs/2110.11553
[57] Tong Wei, Jiang-Xin Shi, Wei-Wei Tu, and Yu-Feng Li. 2021. Robust Long-Tailed Learning under Label Noise. arXiv:2108.11569 https://arxiv.org/abs/2108.11569
[58] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968
[59] Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo. 2022. Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction. *IEEE Trans. Software Eng.* 48, 7 (2022), 2541–2556. https://doi.org/10.1109/TSE.2021.3063727
[60] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. 2015. Learning from massive noisy labeled data for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, Boston, MA, USA, 2691–2699. https://doi.org/10.1109/CVPR.2015.7298885
[61] Bowen Xu, Thong Hoang, Abhishek Sharma, Chengran Yang, Xin Xia, and David Lo. 2021. Post2Vec: Learning Distributed Representations of Stack Overflow Posts. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3093761
[62] Yanming Yang, Xin Xia, David Lo, Tingting Bi, John Grundy, and Xiaohu Yang. 2022. Predictive Models in Software Engineering: Challenges and Opportunities. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 56 (apr 2022), 72 pages. https://doi.org/10.1145/3503509
[63] Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. 2019. Mining software defects: should we consider affected releases?. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, Montreal, QC, Canada, 654–665. https://doi.org/10.1109/ICSE.2019.00075
[64] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor W. Tsang, and Masashi Sugiyama. 2019. How does Disagreement Help Generalization against Label Corruption?. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 7164–7173. http://proceedings.mlr.press/v97/yu19b.html
[65] Hui Zhang and Quanming Yao. 2020. Decoupling Representation and Classifier for Noisy Label Learning. arXiv:2011.08145 https://arxiv.org/abs/2011.08145
[66] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, Tallinn, Estonia, 807–817. https://doi.org/10.1145/3338906.3338931
[67] Yuxiang Zhu, Minxue Pan, Yu Pei, and Tian Zhang. 2019. A Bug or a Suggestion? An Automatic Way to Label Issues. arXiv:1909.00934 http://arxiv.org/abs/1909.00934