Blocking Bugs Identification via Binary Relevance and Logistic Regression Analysis

Zhihua Chen, Xiaolin Ju*, Guilong Lu, and Xiang Chen
School of Information Science and Technology, Nantong University, Nantong, China
zhihuachen19@gmail.com, ju.xl@ntu.edu.cn, lululu_dream@163.com, xchencs@ntu.edu.cn
*corresponding author

Abstract— Blocking bugs, a type of bugs that prevents other bugs from being fixed, significantly increase the fixed time of both themself and the blocked bugs. Thus, these blocking bugs bring a considerable negative impact on software evolution. Therefore, the timely identification of blocking bugs is essential for software maintenance. This paper proposes an approach based on Binary Relevance(BR) and Logistic Regression(LR) analysis, called BR-LR, to predict bugs' blocking and blocked labels. We first filter and build a dataset consisting of two sets with a specific type of blocking relationship based on the ideas of BR. Then, we extract several fields from the bug reports and train the model by applying the logistic regression analysis with the constructed dataset in the first step, resulting in two prediction models for bug blocked and blocking labels. Finally, our approach combines the two prediction results to identify whether the bug is blocking or blocked. We also conduct empirical studies on seven open-source projects to verify the effectiveness of our approach. The final experimental results show that our model performs better from a partially correct perspective and can accurately predict bug labels than benchmarks. Specifically, the average accuracy of our model is 54.86%, and the average F1-measure is 50.61%.

Keywords— Blocking bugs, Prediction, Binary Relevance, Logistic Regression.

I. Introduction

Software bugs are prevalent at every stage of the software life cycle. Numerous software bugs bring high costs. Studies have shown that the cost of fixing software bugs accounts for 50% to 80% of the cost of software system development and maintenance [1]. Many bug tracking systems for managing software bugs and bug reports have been deployed to maintain the software better. Studies indicate that Mining Software Repositories (MSR) is a growing area of Software Engineering (SE) research [2]. Thus, exploring the features and relationships of bugs is beneficial for optimal software maintenance by mining bug reports.

There are multiple relationships between bugs in complex software ecosystems, one of which is blocking software bugs. Blocking bugs are software bugs that prevent other bugs from being fixed until they have been fixed. Due to the existence of blocking bugs, the maintenance and repair process of the downstream software system cannot be carried out [3]. This blocking relationship makes it di icult for downstream developers to fix these blocked bugs even if they have the ability and resources. To address the harmful effects of blocking bugs, many developers have conducted empirical research on them, seeking to predict the existence of blocking bugs earlier. Garcia and Shihab were the first to study blocking bugs. Their experiments found that the repair time of blocking bugs is 15-40 days longer than that of non-blocking bugs. In other words, the repair time of blocking bugs is about 2-3 times that of non-blocking bugs [3]. Their further research showed that fixing blocking bugs requires more lines of code than fixing non-blocking bugs [4].

Therefore, an automated forecast of whether a bug is a blocking bug can help reduce the negative impact of this phenomenon. Various techniques aroused for identifying whether one bug will block another. Garcia et al. applied random forest strategies to build a classifier for classifying blocking bugs [3]. Later, Xia et al. constructed the EL-Blocker approach for the identification and classification of blocking bugs [5]. ELBlocker adopts the ensemble learning approach to build classifiers on multiple disjoint datasets and combine them to automatically determine an appropriate imbalanced decision boundary to judge whether a bug is a blocking bug. These studies show that machine learning methods can effectively predict and classify blocking bugs.

Previous work focused on predicting whether blocking relationships existed between pairs of software bugs. There is no further determination of whether a bug is a blocking bug or a blocked bug. To better predict the relationship of bugs, we use two kinds of relationships as two labels of bugs. Predicting bug blocking and blocked relationships is achieved by predicting bug labels. However, the traditional single-label classification technology cannot achieve the accuracy and comprehensiveness of prediction. Therefore, we need a multi-label classification approach to help us predict different labels of bugs. There are many new machine learning techniques and approaches that have been applied to multi-label classification problems [6]–[11]. The two most common solutions are problem conversion and multi-label technology. In the work of Feng et al., the effectiveness of multi-label and question transformation techniques in software bug classification is empirically investigated [12]. After conducting comparative experiments in multiple projects, Binary Relevance, one of the approaches for problem transformation, achieves excellent results in single-label and multi-label classification. Inspired by this, we adopt the idea of Binary Relevance to achieve multi-label prediction of bugs. The Binary

2767-6684/22/$31.00 ©2022 IEEE
DOI 10.1109/DSA56465.2022.00052

Relevance approach constructs independent classification prediction models for the blocking and blocked labels of the bugs and then combines the prediction results of all classifiers. The final output of BR contains the prediction results of the two labels of the bug.

In our experiments, the Logistic Regression method is adopted as the implementation method of the BR method. Logistic regression is a widely used tool in social statistical analysis and data mining to explain the internal relationship between a set of independent variables and dependent variables. Basili et al. first applied it to bug prediction of object-oriented programs [13]. Furthermore, Briand et al. also empirically studied the Accuracy of bug prediction of the Logistic Regression model and MARS model across versions based on object-oriented metrics [14], [15]. In the work of Armah et al., the Logistic Regression method was used to improve the VVRLR algorithm for predicting software bugs and achieved good results [16]. Inspired by these works, we choose the Logistic Regression method as the base method to implement the BR method for label prediction of bugs and construct an approach named BR-LR. Specifically, we apply the approach of Logistic Regression to build different prediction models for the blocked and blocking labels of bugs. First, we filter out two types of specific bugs as a training set based on the bug's 'Blocks' and 'Depends on' fields. Subsequently, we collected several fields from the bug report to construct training features, including comments, cc lists, etc. Then, we train on two specific bug sets and build models for blocking label prediction and blocked label prediction by the Logistic Regression method. Finally, we employ multiple metrics to evaluate the effectiveness of our model.

We conduct an empirical study on seven large-scale open-source projects to evaluate whether we can effectively predict attribute labels for bugs. Meanwhile, six evaluation metrics from two aspects are used to measure the predictive performance of our approach. The main contributions of our research are summarized as follows:

- We combine Binary Relevance and Logistic Regression method to predict bug blocking and blocked labels.
- We use the number of 'Blocks' and 'Depends on' to investigate whether the severity of blocking and being blocked affects the model's predictive performance.
- We conducted empirical research on seven open-source projects. Our model was trained using bug reports from these projects and achieved good prediction results and model performance.

The rest of this paper is organized as follows. Section II describes the dataset categories we adopted, Binary Relevance method and Logistic Regression method. Section III describes the steps and details of our approach. Section IV shows empirical settings. Section V presents the results of our analysis of several research questions. Section VI shows effective threat analysis. Section VII documents some research related to our work. Finally, Section VIII summarizes our work and gives an outlook on future work.

II. Background and Preliminary

A. Binary Relevance Method

Our research desires to predict whether bugs will block other bugs and whether other bugs will block them. This goal is a multi-label prediction problem in which the customarily used solutions are problem transformation approaches [9] and multi-label techniques [7], [17]. These approaches can assign multiple labels to bugs. Among them, problem transformation approaches include Label Powerset (LP) and Binary Relevance (BR). These two problem transformation approaches and two multi-label techniques are evaluated in accuracy and efficiency [12]. It is confirmed that the Binary Relevance approach has satisfactory accuracy, and Binary Relevance is more effective than Label Powerset when the algorithm is the same. Inspired by this, we apply Binary Relevance's approach to achieve multi-label prediction of bugs in our experiments.

Binary Relevance (BR) processes each label separately, building an independent classifier for it. Then, the predictions produced by the individual classifiers are combined as the final output. In our experiments, we transformed the multi-label classification of bugs into two binary classification problems: does it contain blocking labels, and does it contain blocked labels. We build training datasets and prediction models separately for these two problems. Finally, we combine the results of the two classifiers as the bug label prediction result.

B. Typologies of Blocking and Blocked Bugs

There are various blocking relationships among many software bugs. However, overly complex relations are not conducive to our application in relational research and training in subsequent experiments. Therefore, we select a subset of specific types of bugs from many bugs as our training dataset. The Fig. 1 illustrates examples of six types of typologies among blocking relationships between bugs within Eclipse.

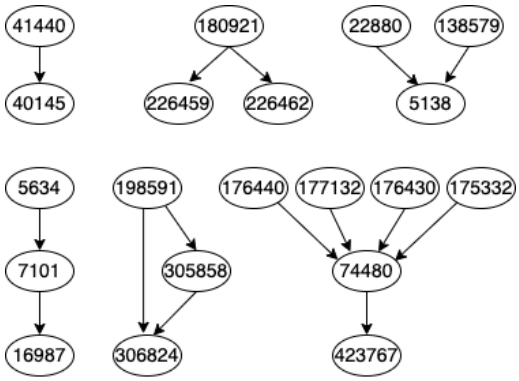

Fig. 1: Typical typologies of blocking bugs within Eclipse.

As shown in Fig. 1, each circle represents a bug, and the number in the circle represents the bug number in the Eclipse Bug Manager. Arrows indicate that one bug is blocking another. For example, the first group indicates that Bug 41440 is blocking Bug 40145. The figure shows a variety of existing relationship types for blocking bugs.

In the bug report, the 'Blocks' and 'Depends on' fields are used to record the bugs blocked by this bug and the bugs blocking this bug. We denote the number of bugs in 'Blocks' and 'Depends on' as B and D. We will divide specific training sets according to the size of B and D (greater than a certain value M and N, respectively). In the third paragraph, we describe the specific partitioning of the training datasets.

III. Our Approach

Our work aims to provide a more efficient way to distinguish between blocking labels, blocked labels, and regular labels of bugs. We instantiate learning tasks with Binary Relevance and Logistic Regression models. We filter specific types of software bugs as the data set for model training, which can better strengthen the characteristics of the two types of bugs. Then, various feature fields significantly related to the prediction target are selected via correlation analysis. Subsequently, we build prediction models separately for these two labels. Finally, we combine the prediction results of the two models as our final prediction result for the bug. As shown in Fig. 2, our approach mainly includes the following stages: Dataset Construction, Word Embedding, Correlation test and BR-LR analysis.

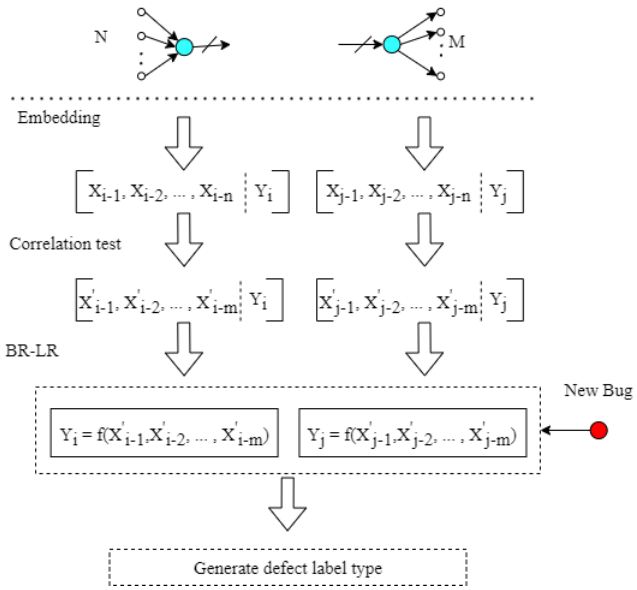

Fig. 2: The framework of our approach.

Next, we will introduce the framework of our approach in the following subsections.

Algorithm 1 Dataset Classification

Input: B(Blocks),D(Depends on)
Output: category

```
if len(B)==0 and len(D)>=N then
    category = 'depend-only bug'
else if len(B)>=M and len(D)==0 then
    category = 'blocking-only bug'
else if len(B)==0 and len(D)==0 then
    category = 'normal bug'
else
    category = 'other bugs'
return category
```

A. Dataset Construction

It is essential to clean and divide the dataset to improve prediction accuracy, construct the model faster, achieve lower consumption, and better interpret the model. Therefore, we filter out bugs that satisfy the two classification criteria to construct training data set to achieve the above goals.

Algorithm 1 presents the partitioning criteria by which we filter for the separate training set. We crawl numerous bug reports from various software bug repositories and store them in the database. In particular, two fields in the bug report, 'Blocks' and 'Depends on', were utilized to divide the former dataset. "Blocks" records the bugs that are blocked by this bug, and "Depends on" records the bugs that block this bug.

Where N and M are the numbers of blocking and blocked bugs. In our further research, we will explore whether the size of N and M affects the learning of features and the accuracy of the model prediction.

B. Word Embedding

Garcia et al. summarize the critical factors for identifying blocking bugs, such as the comment text, comment size, the number of developers in the CC list of the bug report, and the reporter's experience [3]. Inspired by Garcia's work, we obtain multiple features, including these factors, from bug reports for subsequent research and model training. All these features can be roughly divided into text features and numerical features. The main content of our work is to discover the correct vector representation for text features.

As we know, LSA, Word2Vec, GloVe, and One-Hot Encoding are several commonly used word vector representation approaches. For instance, Nellie et al. used LSA, Word2Vec, and GloVe at the same time in their experiments and compared the results of these three approaches [18]. Experiments show that the language used will affect the effect of the three approaches to a certain extent, but Word2Vec can provide a better representation of word vectors. Another encoding approach, One-Hot

Encoding, encodes text information through random algorithms. This leads to the loss of relationship information between words. As the text content expands, the word vector matrix it creates will become more extensive and sparse, which will lead to matrix dimension disaster. In contrast, Word2Vec maps each word into a shorter word vector, which can well solve the problems caused by One-Hot Encoding. So in our work, we adopt Word2Vec to convert text features into word vector representations.

C. Correlation Testing

There are many bug-related features in bug reports. According to our research content, these characteristics can be roughly divided into three categories. Related features. It helps predict whether the bug is a blocking bug, which can improve the effectiveness of the regression model and improve the accuracy of the prediction results. Irrelevant features. They do not contribute to the prediction of blocking bugs and cannot improve the model's performance. Redundant features. They do not improve the performance of the regression model or can be derived from other features. For our experiments, we want to obtain relevant features from all features for model training and prediction while excluding redundant irrelevant and redundant features. The existence of irrelevant features and redundant features often leads to the problem of dimension disaster, especially when there are text features; word vectorization of text features often produces a large number of dimensions. Feature screening can solve such problems to a certain extent, simplify the model's input, effectively reduce the time required in the model training process, and improve the accuracy of the subsequent prediction of the model. Therefore, we combined the work of Garcia et al. to analyze fields in bug reports that are more helpful for blocking bug predictions and perform correlation analysis to screen out more helpful fields for prediction work.

D. BR-LR Analysis

Based on the ideas of Binary Relevance and the approach of Logistic Regression, we classify the class label to which the bug belongs. Our predictive models were trained separately with the two labels of the bug in the previously constructed dataset. We use the class labels of bugs as the dependent variable in the Logistic Regression Analysis. Then, we use the significantly correlated features that we have done correlation analysis in the bug report as independent variables, establish a mathematical model of the quantitative relationship between multiple variables, and conduct analysis and statistics through the obtained data set. Finally, a prediction model that can complete the classification of bug categories is constructed.

A bug might have a blocking label, a blocked label, or just a regular bug in our experiments. To facilitate the construction and analysis of the model, we divide it into two parts, predicting whether the bug will block other bugs and whether other bugs will block this bug. We transform the multi-label problem of predicting blocking relations and blocked relations of bugs into two binary classification problems. Therefore, we adopt the Logistic Regression method as the implementation approach of our prediction model.

We employ the Logistic Regression method to construct our predictive model. The Logistic Regression method is a supervised learning approach and is very effective in binary classification tasks. The Logistic Regression method has the following advantages: 1. The Logistic Regression method has a low computational cost and is easy to understand and implement. 2. The calculation amount is small, and more bug types can be predicted in a shorter time. Therefore, we adopt the Logistic Regression method as the implementation approach of our prediction model.

IV. Empirical Setup

The primary goal of our work is to predict whether a bug contains blocking labels, blocked labels, or does not contain either kind of label by applying Binary Relevance and Logistic Regression analysis. Therefore, we apply it to bugs corresponding to seven open-source projects to evaluate the effectiveness and efficiency of the proposed model. In our empirical study, we wish to investigate the following research questions:

RQ1: How well does the BR-LR model perform?

RQ2: Does M and N in the training set affect the model's predictions?

RQ3: Can the trained regression model achieve better prediction results in other software projects?

A. Experimental Subjects

We obtain a total of 436,047 bug reports from seven open-source projects (such as Eclipse[1], Mozilla[2], NetBeans[3], Chromium[4], OpenOffice[5], RedHat[6], Gentoo[7]). These projects are mature and long-standing open-source projects with numerous bug reports. Six of these projects (i.e., Eclipse, Mozilla, NetBeans, OpenOffice, RedHat, and Gentoo) use Bugzilla as their issue tracking system. In Bugzilla, there is a 'Blocks' field in the bug report, which displays bugs blocked by this bug. A 'Depends on' field shows the bugs that blocked this bug. Therefore, we use these two fields in bug reports to determine whether there is a blocking relationship between bug pairs. In its bug reports, Google's issue tracking system, Chromium, also has a 'Blocks' field and a 'Depends on' field. These fields have the same semantics as the fields in Bugzilla. We use

[1]http://bugs.eclipse.org/bugs
[2]https://bugzilla.mozilla.org
[3]https://netbeans.apache.org
[4]https://bugs.chromium.org
[5]https://bz.apache.org/ooo
[6]https://bugzilla.redhat.com
[7]https://bugs.gentoo.org/

these fields to determine the blocking relationship between bug pairs.

We extracted multiple fields from each bug report, including summary, description, reviews, CC list, etc., as factors we used to predict the mislabels. TABLE I summarizes the characteristics of the seven open-source projects used in our empirical study. There are enough bug reports in these open-source projects containing many types of bugs that compose the training set.

TABLE I: The statistic of three categories of bugs in seven subjects.

Projects	Depend-only	Block-only	Normal+other
Eclipse	6735	6128	52337
Mozilla	5832	5348	58491
NetBeans	5189	5206	48921
Chromium	8757	6039	52231
OpenOffice	6958	5541	75628
RedHat	3892	3148	27744
Gentoo	2569	3746	45607
Total	39932	35156	360959

B. Performance Measures

Our approach will employ two trained models to classify bugs and predict whether a bug contains blocking labels or blocked labels. We use an encoding of the form (1,0) to record predictions about bugs. The first digit represents whether the bug blocks other bugs, the second digit represents whether other bugs block this bug, 1 means yes, 0 means no. The example means that the bug is a bug that blocks other bugs without being blocked by other bugs. We will use the following metrics to evaluate our model's predictions and model performance.

1) Partially correct evaluation methods are not considered: Exact Match Ratio(MR): The Exact Match Ratio means that the prediction is correct for each bug only when the predicted value is precisely the same as the actual value. It can be calculated as follows.

$$MR = \frac{1}{m}\sum_{i=1}^{m} I\left(y^{(i)} == \hat{y}^{(i)}\right). \tag{1}$$

Where m represents the number of bugs, $I()$ is the indicator function. $y^{(i)}$ represents the true label set of the bug, and $\hat{y}^{(i)}$ represents the predicted label set of the bug. When the $y^{(i)}$ is completely equivalent to the $\hat{y}^{(i)}$, the 1 is taken, otherwise it is the 0. It can be seen that the larger the MR value, the higher the classification Accuracy. The same symbols in the following formulas represent the same meaning.

Zero-One Loss(L_{0-1}): The Zero-One loss is calculated as the proportion of bugs that are completely mispredicted to the total number of bugs. It can be calculated as follows.

$$L_{0-1} = \frac{1}{m}\sum_{i=1}^{m} I\left(y^{(i)} \neq \hat{y}^{(i)}\right). \tag{2}$$

2) Consider a partially correct assessment method: It can be seen from the above two evaluation indicators that whether it is the Exact Match Ratio or the Zero-One loss, both of them do not consider the partially correct situation when calculating the result, which is inaccurate for the evaluation of the model.

Therefore, it is advisable to take into account the results that are partially predicted correctly [19]. To realize this idea, the literature proposed the calculation of Accuracy, Precision, Recall, and F1-measure in the multi-label classification scenario [20].

Accuracy: Accuracy is calculated as the average Accuracy of all bugs identification. The Accuracy rate is the proportion of the predicted correct labels for each bug in the total predicted correct or true correct labels. It can be calculated as follows.

$$Accuracy = \frac{1}{m}\sum_{i=1}^{m} \frac{\left|y^{(i)} \cap \hat{y}^{(i)}\right|}{\left|y^{(i)} \cup \hat{y}^{(i)}\right|} \tag{3}$$

Precision: Precision is calculated as the average Precision of all bugs. For each bug, the Precision rate is the ratio of correctly predicted labels to the total number of predicted correct labels. It can be calculated as follows.

$$Precision = \frac{1}{m}\sum_{i=1}^{m} \frac{\left|y^{(i)} \cap \hat{y}^{(i)}\right|}{\left|\hat{y}^{(i)}\right|} \tag{4}$$

Recall: Recall is calculated as the average Recall of all bugs. The Recall rate is the ratio of the number of correct labels predicted to the total number of correct labels for each bug. It can be calculated as follows.

$$Recall = \frac{1}{m}\sum_{i=1}^{m} \frac{\left|y^{(i)} \cap \hat{y}^{(i)}\right|}{\left|y^{(i)}\right|} \tag{5}$$

F1-measure: The F1-measure is calculated as the average F1-measure of all bugs. The F1-measure is the weighted harmonic mean of Precision and Recall for each bug. It can be calculated as follows.

$$F1_{measure} = \frac{1}{m}\sum_{i=1}^{m} \frac{2\left|y^{(i)} \cap \hat{y}^{(i)}\right|}{\left|y^{(i)}\right| + \left|\hat{y}^{(i)}\right|} \tag{6}$$

The higher the Accuracy, the more accurately our model can give labels that bugs have. Again, the higher the values of Precision and Recall, the better the performance of our model, but these two metrics cannot be optimal at the same time. Combining the above two metrics, a higher F1-measure means that the model is more effective in predicting bug labels.

C. Implementation Details

We construct different training sets by picking out bugs that meet our requirements from numerous bugs. Subsequently, we extracted multiple fields, including summary, description, comment, etc., from the bug reports for model training. These fields are transformed into more suitable

arrangements for machine learning by pre-processing. Different training sets will separately train models for different label predictions during the training process. The models are trained iteratively to optimize the training loss. During the testing process, input new bug reports into multiple trained models. The corresponding models will give the prediction results on this label (whether the bug contains a blocking label or a blocked label). We will combine these predictions and compute specific calculations to calculate the overall performance metric of the model on these labels to evaluate the performance of our prediction model.

We tested our proposed method on a computer running Windows 10, 64-bit, Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz and NVIDIA GeForce GTX 1650.

V. Result Analysis

A. Result Analysis for RQ1

We utilize Binary Relevance and Logistic Regression method to build a predictive model named BR-LR, which can automatically predict whether bugs contain blocking labels or blocked labels. We hope that the new prediction model constructed can help developers achieve good results in prediction work. To this end, we performed repeated ten-fold cross-validation on seven open-source projects and then calculated the model's average performances on each evaluation metric.

TABLE II: The value of the BR-LR result on each indicator.

Projects	MR	L_{0-1}	Acc	Pre	Rec	F1
Eclipse	0.335	0.665	0.536	0.448	0.447	0.447
Mozilla	0.316	0.684	0.589	0.496	0.514	0.505
NetBeans	0.379	0.621	0.518	0.579	0.462	0.514
Chromium	0.349	0.651	0.497	0.413	0.533	0.465
OpenOffice	0.298	0.702	0.593	0.549	0.526	0.537
RedHat	0.326	0.674	0.558	0.595	0.466	0.523
Gentoo	0.289	0.711	0.549	0.622	0.496	0.552

TABLE II shows the experimental results of the BR-LR model we built in different open-source projects, using multiple fields in the bug reports (for example, summary, comment, cc list, etc.) to predict the label of the bug (whether it contains blocking labels, whether it contains blocked labels).

From TABLE II, it can be found that the performance of the BR-LR model on the two indicators of MR (Exact Match Ratio) and L_{0-1} is relatively general. This is a perfectly correct case for evaluating the predicted outcome. Better values are obtained only when both labels are successfully predicted. On average, the BR-LR model had an MR of 0.327, which means that bugs that were accurately predicted on both labels accounted for 32.7% of all bugs. From another evaluation point of view, considering the partial correctness, it can be seen from the two indicators of Accuracy and F1-measure that the BR-LR model has better performance. The Accuracy and F1-measure of this model are 0.549 and 0.506, respectively. By looking at each label individually, the model can get good predictions.

Summary for RQ1: From a complete correctness perspective, our model has an MR of 32.7%, which means that our model can predict all labels for correct bugs completely. From the perspective of partial correctness, our model also has better performance. On average, our model has an F1-measure of 50.6% and can more accurately predict each label of bugs.

B. Result Analysis for RQ2

TABLE III: The number of bugs of various types under different M and N.

Projects	N=1	N=2	N=3	M=1	M=2	M=3
Eclipse	6735	3284	1117	6128	2966	1026
Mozilla	5832	2816	967	5348	2574	870
NetBeans	5189	2495	839	5206	2603	1042
Chromium	8757	4279	1752	6039	3020	1309
OpenOffice	6958	3279	1292	5541	2891	1171
RedHat	3892	2146	879	3148	1374	532
Gentoo	2569	1285	494	3746	2003	890

We want to investigate whether the number of bugs (M, N) that block a bug simultaneously or are blocked by a bug at the same time affects the performance of the model. In our assumption, larger M and N means that the bug has more and more accurate features to represent blocking and blocked bugs. More and more detailed features help model learning and training in machine learning, enabling it to get more accurate predictions.We reconstructed the training set according to the size of M and N (original datasets M and N are 1 by default), as shown in TABLE III.

TABLE III Shows the size of the dataset when the number of bugs blocking the bug and the number of bugs blocked by the bug are greater than or equal to 1, 2, and 3. The more the size of M and N increased, the less the number of bugs contained in the dataset. We will test and compare the training sets constructed by M and N of different sizes in this problem. In this way, it can be judged whether the size of M and N will affect the prediction result. If there is an impact, what value can be taken to make the indicators of the model reach the optimal value?

As shown in Fig. 3, the F1-measure value of the predictive model trained on the constructed datasets when our M and N take different values. As can be seen from the figure, when M and N are 1 and 2, there is not much difference in the model's prediction performance. When the value of M and N is 3, the F1-measure value of the model decreases significantly. We performed paired Wilcoxon Tests on data from multiple trials for each group. When M and N are 1 and 2, the p-value of the significance test results is more significant than 0.05. When compared with the value of 3, the p-value of the two is much less

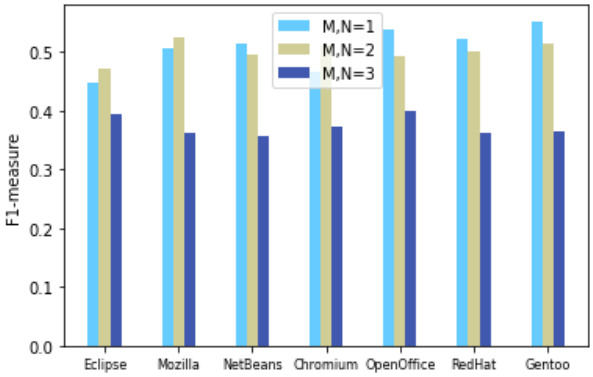

Fig. 3: The value of F1-measure of each model when M and N take different values.

than 0.05. It can also be seen from the significance test results that the results are better when M and N are 1 and 2, and there is no significant difference. Moreover, the two are better than the case where M and N are 3.

Summary for RQ2: Our model predicts the best results when M and N are 1 and 2, respectively. However, when M and N become more extensive, the model's prediction performance decreases significantly.

C. Result Analysis for RQ3

We train a set of predictive models on datasets built by each open-source project and perform the model evaluation in this open-source project. However, there are often significant differences between different open-source projects. Can a model be trained on one project to get better predictions on other projects? At the same time, some projects do not have enough bugs to build a dataset on which the model can be trained. If the training set is too small, it is impossible to get a model that can accurately predict whether a bug contains blocking labels or blocked labels. How do these projects achieve anticipation of blocking bugs? To address this type of problem, we propose a new research question: whether the model we build can achieve cross-project prediction. We conduct an empirical study on the model's cross-project prediction performance.

We analyze and judge whether the model can achieve cross-project prediction by comparing the prediction performance of a model in this project with that in other projects. We use several evaluation metrics when evaluating our models. Since F1-measure is the balance between Precision and Recall, it can better represent the pros and cons of a model. Therefore, we evaluate the cross-project forecasting ability of the forecasting model by comparing the F1-measure of the forecasting model. We have performed multiple ten-fold cross-validation on each project to ensure the stability and accuracy of the experimental results.

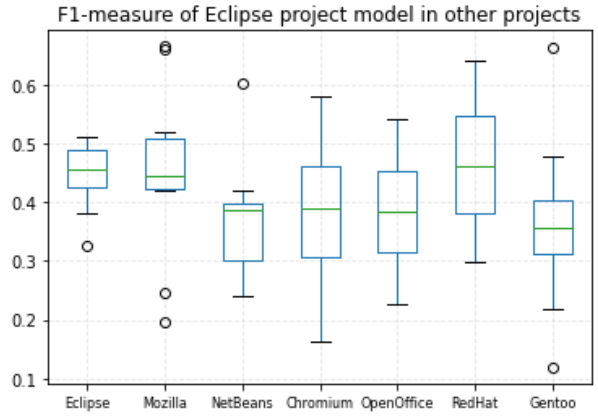

Fig. 4: F1-measure of Eclipse project model in other projects.

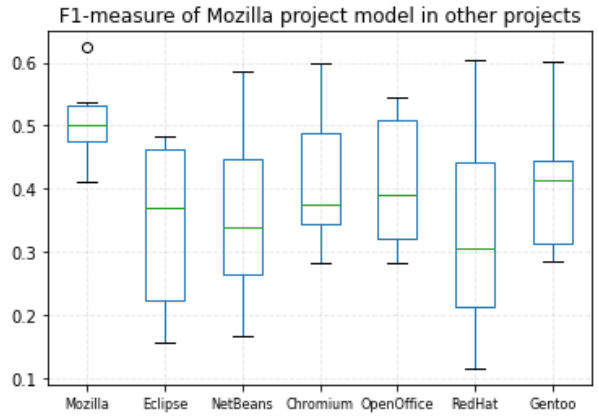

Fig. 5: F1-measure of Mozilla project model in other projects.

These figures (Fig. 4 to Fig. 10) demonstrate the boxplots of F1-measure when each prediction model on the seven open-source projects we built is used for the prediction of the other six projects. We note that each model achieves a high F1-measure on the related project, usually between 0.5 and 0.6. Moreover, relatively stable results can be obtained in multiple cross-validations. The boxplot shows that the difference between the upper and lower edges is insignificant, with no abnormal points. When the model is applied to other projects, the F1-measure of the model will decrease to a certain extent, and the overall boxplot is lower than the original project.

Moreover, the prediction effect is not as stable as in the original project, shown in the boxplot. The gap between the upper and lower edges is significant, and several projects have abnormal points. For example, the model trained by the Eclipse project is applied to the Mozilla project. In the experiment, the overall F1-measure of Mozilla is lower than that of Eclipse, and many outliers

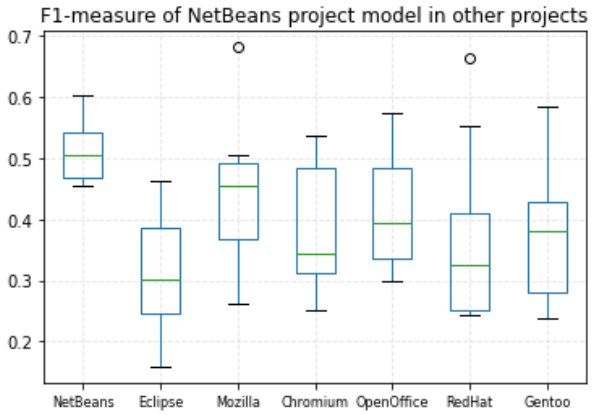

Fig. 6: F1-measure of NetBeans project model in other projects.

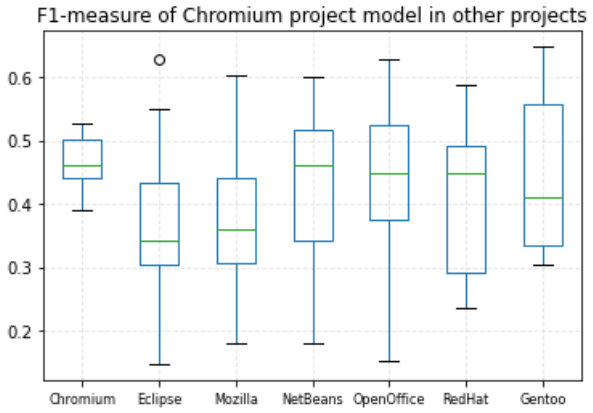

Fig. 7: F1-measure of Chromium project model in other projects.

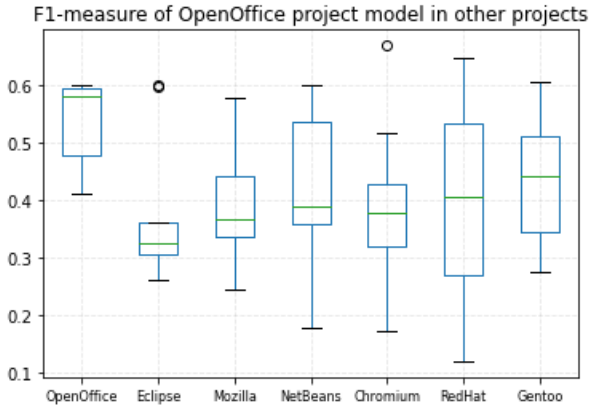

Fig. 8: F1-measure of OpenOffice project model in other projects.

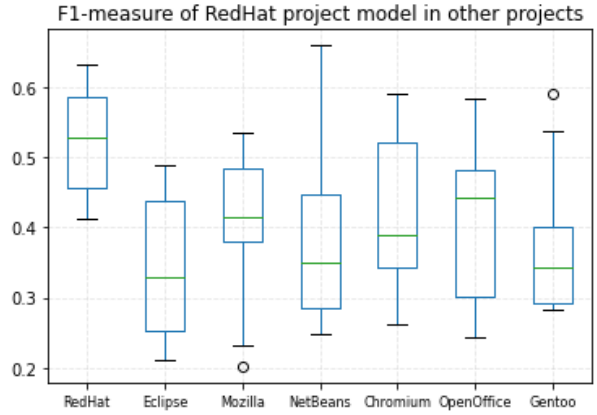

Fig. 9: F1-measure of RedHat project model in other projects.

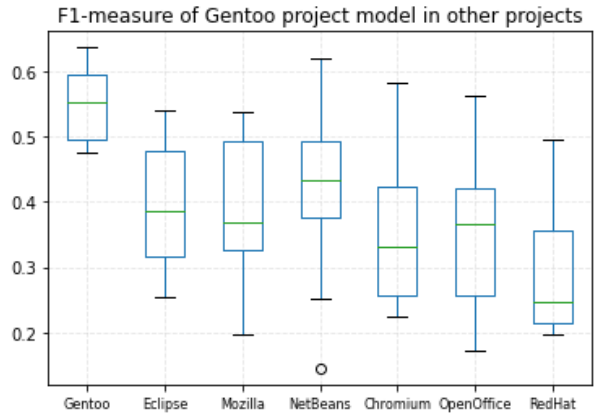

Fig. 10: F1-measure of Gentoo project model in other projects.

are generated. It shows that when the prediction model of the Eclipse project is applied to Mozilla, the prediction effect becomes worse and more unstable.

Summary for RQ3: The trained models have the best and most stable prediction performance on their projects, respectively. When the model is applied to other projects, it can also predict blocking labels and blocked labels of bugs to a certain extent accuracy. However, their effectiveness will be weaker than the prediction model specially trained by the project, and the prediction effect is prone to fluctuation.

VI. Threats to Validity

This section will discuss potential threats to our research's validity, including internal validity, and external validity.

Internal Validity. Blocking and blocked bugs tend to occupy only a tiny part of the software life cycle compared to other software bugs. This phenomenon is called the class imbalance problem in classification tasks. Class imbalance

problems often lead to biased predictions for minority classes. We expanded the training dataset using bug reports from seven open-source projects to address these issues and reduce bias. In addition, we also conducted 10-fold cross-validations to reduce the randomness and unreliability of a single experiment.

External Validity. After conducting cross-item experimental comparisons, it is concluded that the model cannot predict projects outside the training set well. This means that already trained models are challenging to scale. However, we conduct empirical studies on several different open-source projects to verify the scalability of our model by training on projects. Acceptable prediction results are obtained on seven open-source projects, which confirms the generality of our experiments. In the future, we will consider using more projects and more directions to research the validity of our experiments. This further enhances the external validity.

VII. Related Work

A. Blocking Bug Research

Bug detection and repair are an essential part of the software life cycle. Many researchers conducted a broad study on software bugs. Garcia and Shihab sponsored an earlier work about blocking bugs, a specific type of software bug which prevent other bugs from being fixed until they have been fixed. Blocking bugs yielded a series of maintenance difficulties. Their empirical study on six open-source projects discovered that fixing blocking bugs takes about two to three times as long as fixing non-blocking bugs [3]. Subsequently, in more empirical studies, sets of characteristics of blocking bugs, such as number, distribution, repair resources, etc., were found [4], [21]. For example, in the latest work, Gar et al. found that fixing blocking bugs involves 1.2 to 4.7 times more lines of code than fixing non-blocking bugs [4].

The researchers also applied bug reports to design automated approaches to do the predictive work to address the problems caused by blocking bugs. Garcia et al. constructed a predictive model based on a decision tree to predict whether a bug is a blocking bug. They analyzed various fields in the bug report. They found that the most critical factors for blocking bug predictions consisted of comment content (including comment content and comment size), the number of developers in the cc list, and the experience of the reporter [3], [4]. Taking it a step further, Xia et al. employ ensemble learning to build an approach called ELBlocker [5]. The approach divides the dataset into multiple disjoint subsets, builds separate classifiers for these subsets, and then uses ensemble learning to combine these sub-classifiers. They conducted empirical research across six open-source projects, and the results showed the reliability of ELBlocker. ELBlocker also solves the class imbalance problem of blocking bugs to a certain extent. Following the work of Xia et al., Cheng et al. also adopted ensemble learning technology to build an approach named XGBlocker, which includes two stages of feature extraction and model construction [22]. They introduced the XGBoost advanced algorithm and extracted enhancements from bug reports to enable the prediction of a bug blocking relationships. Wu et al. propose hbrPredictor, which combines interactive machine learning and active learning for the high-impact bug report (HBR) prediction [23]. Illuminated by Wu et al., work [23], Chen et al. predicted whether there is a blocking relationship between pairs of bugs through a hybrid deep learning model approach. The reliability of the model is verified by comparison with two baseline approaches [24]. Similarly, Brown et al. introduce a framework that leverages a deep learning model, DeepLaBB, to learn semantic features using token vectors automatically and then applies the features to build and train a model that predicts whether a bug is considered blocking or non-blocking [25]. More and more deep learning techniques are applied to blocking bug detection.

Different from the above work, there is also some work on specific blocking relationship research. For example, Ren et al. studied bugs that blocked at least two bugs. They call these bugs critical blocking bugs (CBBs) [21]. CBBs are well-studied with normal blocking bugs and other types of bugs in terms of importance, fix time, fix size, developer experience in fixing CBBs, and how well CBBs block multiple bugs. Their experimental results better illustrate the threat and importance of CBBs.

Meanwhile, Ding et al. conducted an empirical study on the destructibility of blocking bugss [26]. Their work expects to predict whether the blocking relationship between bugs can be cut off, thereby alleviating the repair pressure caused by blocking bugs. They built a discontinuity prediction model on two large open-source projects and achieved 78.5% and 71.7% Precision.

B. Novelty of Our Study

Illuminated by previous works, we reinforce the features of blocking bugs and blocked bugs through screening datasets. We incorporate the Binary Relevance approach to transforming the blocking bug identification task into two binary classification problems: does a bug contain blocking labels, and does it contain blocked labels. At the same time, we use multiple fields in the bug report to train a regression prediction model and make predictions on the two labels of the bug (blocking labels, blocked labels).

VIII. Conclusion

We adopt the idea of multi-label problem conversion to realize the prediction of the two labels of 'Blocks' and 'Depends on' of bugs and build a prediction model named BR-LR. Specifically, we segment a specific dataset by the number of bugs in the 'Blocks' and 'Depends on' fields in the bug reports. These two types of datasets are considered to have enhanced features that can better enable label

prediction. Then, we build prediction models for the two datasets to predict this type of label. We extract multiple fields from bug reports and analyze the correlation of these fields. Furthermore, we apply the filtered fields to a Logistic Regression method to construct a predictive model. We conduct empirical research on 436,047 bug reports across seven open-source projects. From a partially correct perspective, the results show that our model has good predictive performance and can better predict both blocked and blocked labels for bugs.

We also conduct empirical studies on the number of bugs (M and N) contained in the fields (Blocks, Depends on) of the partitioned datasets and the cross-item predictive ability of each predictive model. The results show that the constructed datasets can achieve a better training effect when M and N are 1 and 2, respectively. However, the predictive performance of the model decreases as the number increases. Meanwhile, when cross-item label prediction is performed using a trained model built from different items, the effect is weaker than when applied to this item.

In the future, we will consider acquiring and constructing datasets from a broader range of bug sources as our research objects. Moreover, we will further consider the approach of multi-label prediction, increase the selection of approaches, and compare experimental results.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61502497 and 61673384; the Guangxi Key Laboratory of Trusted Software Research Plan under Grant KX201530 and Grant KX201532; and in part by the National Natural Science Foundation of Guangxi under Grant 2018GXNSFDA138003.

References

[1] J. S. Collofello and S. N. Woodfield, "Evaluating the effectiveness of reliability-assurance techniques," Journal of Systems and Software, vol. 9, no. 3, pp. 191–195, Mar 1989.

[2] M. Vidoni, "A systematic process for mining software repositories: Results from a systematic literature review," Information and Software Technology, vol. 144, p. 106791, 2022.

[3] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in Proceedings of the 11th working conference on mining software repositories, 2014, pp. 72–81.

[4] H. Valdivia-Garcia, E. Shihab, and M. Nagappan, "Characterizing and predicting blocking bugs in open source projects," Journal of Systems and Software, vol. 143, pp. 44–58, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300530

[5] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning," Information and Software Technology, vol. 61, pp. 93–106, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914002602

[6] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier Chains for Multi-label Classification," in Machine Learning and Knowledge Discovery in Databases. Berlin, Germany: Springer, Sep 2009, pp. 254–269.

[7] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," Pattern recognition, vol. 40, no. 7, pp. 2038–2048, 2007.

[8] E. Spyromitros, G. Tsoumakas, and I. Vlahavas, "An Empirical Study of Lazy Multilabel Classification Algorithms," in Artificial Intelligence: Theories, Models and Applications. Berlin, Germany: Springer, Oct 2008, pp. 401–406.

[9] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 8, pp. 1819–1837, 2014.

[10] Y. Feng and Z. Chen, "Multi-label software behavior learning," in 2012 34th International Conference on Software Engineering (ICSE). IEEE, Jun 2012, pp. 1305–1308.

[11] X. Xia, Y. Feng, D. Lo, Z. Chen, and X. Wang, "Towards more accurate multi-label software behavior learning," in 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). IEEE, 2014, pp. 134–143.

[12] Y. Feng, J. Jones, Z. Chen, and C. Fang, "An Empirical Study on Software Failure Classification with Multi-label and Problem-Transformation Techniques," in 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). IEEE, Apr 2018, pp. 320–330.

[13] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751–761, Oct 1996.

[14] L. C. Briand, J. Wüst, J. W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," Journal of Systems and Software, vol. 51, no. 3, pp. 245–273, May 2000.

[15] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," IEEE Trans. Software Eng., vol. 28, no. 7, pp. 706–720, Aug 2002.

[16] S. M. Angolo, G. K. Armah, G. Luo, and K. Qin, "Applying Variant Variable Regularized Logistic Regression for Modeling Software Defect Predictor," May 2016, [Online; accessed 9. Mar. 2022]. [Online]. Available: http://repository.seku.ac.ke/handle/123456789/2566

[17] M.-L. Zhang and Z.-H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," IEEE transactions on Knowledge and Data Engineering, vol. 18, no. 10, pp. 1338–1351, 2006.

[18] M. Naili, A. H. Chaibi, and H. H. Ben Ghezala, "Comparative study of word embedding methods in topic segmentation," Procedia Computer Science, vol. 112, pp. 340–349, 2017, knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8 September 2017, Marseille, France. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917313480

[19] M. S. Sorower, "A Literature Survey on Algorithms for Multi-label Learning," 2010, [Online; accessed 26. Jan. 2022]. [Online]. Available: https://www.semanticscholar.org/paper/A-Literature-Survey-on-Algorithms-for-Multi-label-Sorower/6b5691db1e3a79af5e3c136d2dd322016a687a0b

[20] S. Godbole and S. Sarawagi, Discriminative Methods for Multi-Labeled Classification. Heidelberg, Germany: Springer Verlag, Aug 2004, vol. vol. 3056.

[21] H. Ren, Y. Li, and L. Chen, "An empirical study on critical blocking bugs," in Proceedings of the 28th International Conference on Program Comprehension, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 72–82. [Online]. Available: https://doi.org/10.1145/3387904.3389267

[22] X. Cheng, N. Liu, L. Guo, Z. Xu, and T. Zhang, "Blocking Bug Prediction Based on XGBoost with Enhanced Features," in 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, Jul 2020, pp. 902–911.

[23] X. Wu, W. Zheng, X. Chen, Y. Zhao, T. Yu, and D. Mu, "Improving high-impact bug report prediction with combination of interactive machine learning and active learning," Information and Software Technology, vol. 133, p. 106530, 2021.

[24] Z. Chen, X. Ju, Y. Shen, and X. Chen, "Improving blocking bug pair prediction via hybrid deep learning," in 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2021, pp. 727–732.
[25] S. A. Brown, B. A. Weyori, A. F. Adekoya, P. K. Kudjo, S. Mensah, and S. Abedu, "Deeplabb: A deep learning framework for blocking bugs," in 2021 International Conference on Cyber Security and Internet of Things (ICSIoT). IEEE, 2021, pp. 22–25.
[26] H. Ding, W. Ma, L. Chen, Y. Zhou, and B. Xu, "Predicting the breakability of blocking bug pairs," in 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 1. IEEE, 2018, pp. 219–228.