A Software Defect Prediction Method based on Multi-type Features and Feature Selection

Jiaxin Liu[a], Hao Xu[a], Lu Lu[a,c,*], Quanyi Zou[b,c], Zhanyu Yang[a]
[a] School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[b] School of Journalism and Communication, South China University of Technology, Guangzhou, China
[c] Pazhou Laboratory, Guangzhou, China
[*] Corresponding author email: lul@scut.edu.cn

Abstract—**Numerous software defect prediction methods utilize semantic information and software metrics as code features, neglecting the structural knowledge inherent in the source code. Other studies improve feature completeness by simply combining different types of defect indicators, which causes information redundancy. To address these challenges, this paper proposes a novel software defect prediction method that incorporates multi-type features and performs feature selection. Firstly, semantic and structural features are extracted by Text Convolutional Neural Network (TextCNN) and Graph Isomorphism Network (GIN) from Abstract Syntax Tree (AST) and Program Dependency Graph (PDG), respectively, which are combined with software metrics to build a multi-type feature set. Then, Recursive Feature Elimination with Cross-Validation (RFECV) integrating a novel feature importance measure is utilized to remove redundant features and generate a feature subset. Finally, a prediction model for classification is established based on the feature subset. The experiments validated the effectiveness of multi-type features and the improved RFECV. Overall our proposed method outperforms state-of-the-art techniques on nine Java open-source projects.**

Keywords—software defect prediction; feature extraction; feature selection; recursive feature elimination

I. Introduction

To enhance software quality and reduce testing costs, software defect prediction (SDP) technology has emerged as a research hotspot in software engineering and software reliability assurance in recent years. The software defect prediction model is undertaken to identify software classes or modules that are more likely to fail and enable the minimization of maintenance costs before product deployment. Furthermore, this technique can provide a reference for testers to help them quickly grasp the overall quality of the software, and allocate test resources more reasonably.

Current SDP methods [1–3], extracting features from source code, tend to emphasize software metrics and semantic features, resulting in the lack of comprehensive information, especially structural information. Some studies [4] have attempted to improve software defect prediction by expanding the feature set and extracting features from various code abstract structures. However, the increase in the number of features does not necessarily lead to better prediction performance. Expanding the feature set may result in the inclusion of redundant information, which negatively impacts the prediction performance.

DOI reference number: 10.18293/SEKE23-118

To solve these problems, this paper proposes a novel software defect prediction model named TGR (TextCNN-GIN-RFECV). TGR utilizes TextCNN [5] and GIN [6] to extract semantic and structural features from source code, respectively. The comprehensive feature set is composed of the extracted features and software metrics. Then, we propose the improved RFECV [7] to eliminate redundant features and generate a feature subset. Finally, the selected feature subset is fed into a classifier for training and prediction. In summary, this paper makes the following contributions:

- In order to enrich the feature information, we introduce a novel multi-type feature representation that combines semantic, structural, and software metric features.
- We modify the feature importance measure in RFECV by considering the real category, correlations with other features, and the classifier coefficient. And we utilize the improved RFECV to generate an appropriate feature subset and reduce redundant information.
- We reveal the impact of multi-type features and the improved RFECV on the overall performance of TGR by conducting experiments on nine java open-source projects.

II. Related Work

Software metrics are essential tools for assessing the quality and performance of software products and development processes. These metrics can significantly influence the accuracy of SDP models, including the widely used lines of code (LOC) and C&K metrics [8]. In the early stage of research, some studies use software metrics and traditional machine learning algorithms to predict and estimate whether software module contains defects, such as Logistic regression (LR) and support vector machine (SVM) [9].

Abstract Syntax Tree (AST) is widely used in software defect prediction research. Deep learning methods are often used to extract semantic features from ASTs, such as Convolutional Neural Networks (CNNs) [1], Long Short-Term Memory (LSTM) [2] networks, and Deep Belief Networks (DBNs) [3]. In addition, Code Property Graph (CPG) [10], Class Dependency Network (CDN) [4], and other code representations are also employed for structural feature extraction. Uddin et al. [11] use the Bidirectional Encoder Representations from Transformers (BERT) pre-training model to obtain the vector representation of each word in the source code, which is utilized to train bidirectional LSTM for SDP.

There are many studies trying to ensure the quality of feature sets through feature selection. Zhu et al. [12] combine the whale algorithm and the complementary simulated annealing algorithm to select features. Saifan et al. [13] assess the effectiveness of feature selection techniques, including Principal Component Analysis, Pearson's correlation, Greedy Stepwise Forward selection, and Information Gain.

III. Method

In this section, the proposed method TGR is described in detail. The entire framework of TGR is given in Fig. 1, which is divided into two stages: the feature extraction stage and the feature selection stage.

A. Feature extraction

1) Parsing source code: ASTs are facilitated by javalang[1] from source code files. The nodes and granularity specified by Wang et al. [3] is adopted to form ASTs. Each AST is traversed in pre-order to obtain a node sequence. The node sequence preserves the original semantic order of the code by pre-order traversal and is maintained at an appropriate length, avoiding the redundancy of the source code. These sequences incorporate the semantic information of the corresponding source code file in a more retrenched representation. Furthermore, a dictionary is established based on the node set to vectorize the node sequence, enabling TextCNN to process and analyze.

AST is adept at capturing semantic information but falls short in capturing structural information. Compared with semantic information, structural information has more complex representations and greater depths of features. In order to explore the abundant structural information contained in the source code, PDG is introduced into the prediction method. As an extension of Control Flow Graphs (CFGs), PDGs capture not only control dependencies but also data dependencies among the program statements. Their nodes represent a program statement, while their edges reflect the data transfer and program execution flow between nodes, which can more comprehensively exhibit the structural information of the program. This stage utilizes SourceDG [14] to generate PDGs from source code, as the source of structural features.

2) Extracting features: After obtaining the AST node vector and PDG to represent the semantic and structural information of the code, it is crucial to choose appropriate feature extraction models. Given the distinct feature depth and representation forms of semantic and structural information, they should be processed by different models. Thus, TextCNN and GIN are chosen to individually process node vectors and PDGs, and extract the semantic and structural features.

The network structure of TextCNN comprises a word embedding layer, a convolutional layer, and a pooling layer. The convolution layer employs three different sizes of convolution kernels to help the node comprehend its context in the code at different scales. Every kernel transforms the input vector to a feature map as the input of the pooling layer, and the outputs of the pooling layer are then connected. Subsequently, two fully connected layers with different output sizes are employed. The first fully connected layer generates predictions for calculating loss and performing backpropagation. When the training is completed, the output of the second fully connected layer is taken as the semantic feature of TGR.

Noted that, unlike LSTM and other neural network models, the output length of the TextCNN used in TGR is solely determined by the model parameters, and is independent of the length of the node vector. Each element of the output semantic feature is derived from the same network structure and represents the entire input sequence, which facilitates the feature selection stage of TGR.

GIN, a variant of Graph Neural Network (GNN), enhances performance on graph classification and graph embedding tasks. GIN utilizes two main operations, namely graph convolution and graph readout, to update node features and generate graph-level representations, respectively. During graph convolution, messages are passed on the edges of a PDG and update the node features to hidden representations. The hidden representations are generated through multiple message passes and subsequently passed to the graph readout part. During the graph readout, we choose summation to aggregate and summarize node features to generate a graph-level representation. The resultant graph-level representations are then passed into fully connected layers, and the outputs are summed as the structural features of TGR. Besides, the structural feature is used as input to another fully connected layer to generate predictions for calculating losses and performing backpropagate.

Since software metrics are crucial features in defect prediction models. The third type of feature in TGR, metric features, are concatenated with semantic and structural features to create a possibly abundant and comprehensive feature set after the three types of features are z-score standardized separately. The utilization of multi-type features enables the classifier to predict with the more comprehensive feature set.

B. Feature selection

Since a large number of features are extracted in the feature extraction stage, potentially redundant and irrelevant information is introduced into the prediction model. To address the redundancy, a feature selection stage is incorporated to acquire a more beneficial feature subset.

RFECV selects features considering the classifier performance. The Recursive Feature Elimination (RFE) algorithm iteratively trains a classifier and eliminates several features with low classifier coefficients until reaching the preset minimum number of features. RFECV uses RFE to get the feature ranking, then selects feature subsets of different sizes based on the ranking to perform cross-validate (CV). The CV can automatically determine the size of the feature subset, and avoid the adverse effects of the unreasonable setting of the feature subset size as a hyperparameter. Finally, the subset

[1]https://github.com/c2nes/javalang

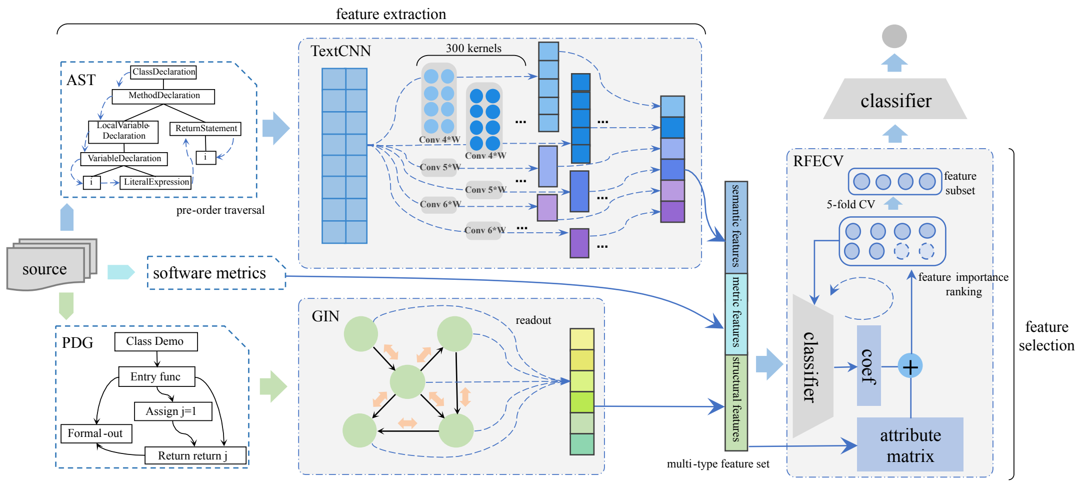

Figure 1. The entire framework of TGR method

with the highest average score is chosen as the optimal feature subset.

The feature eliminated in each recursion of RFE is determined by the coefficients of the classifier, in other words, the feature importance of RFE is measured by the classifier coefficients. These coefficients indicate the feature's impact on the classification results output by the classifier. However, the output of the classifier is not entirely correct, resulting in wrongly assigning importance to features that are not related to the real category. Moreover, feature importance is not only reflected in the coefficients, but also in factors such as correlations with other features, and correlations with real categories. Therefore, the feature importance measure in RFE is replaced by our defined comprehensive feature importance measure, which enables RFECV to select a feature subset more comprehensive, informative, and effective for classification.

In this stage, two coefficients are proposed to calculate the comprehensive feature importance with the classifier coefficients: category correlation coefficients and feature correlation coefficients. The category correlation coefficient complements the influence of the real category on feature importance. The Normalized Mutual Information (NMI) is used to measure the correlation between a feature and the real category vector after discretizing the feature. Suppose the number of features is N, x_i represents the i-th feature in the feature set, and y represents the real category vector. The formula for calculating the category correlation coefficient is given as:

$$cc_i = \frac{2 \cdot (H(x_i) - H(x_i \mid y))}{H(x_i) + H(y)}, \tag{1}$$

where $H(x_i)$ is the entropy of x_i, $H(x_i \mid y)$ is the joint entropy of x_i and y, and cc_i is the category correlation coefficient of x_i.

Feature selection often results in the removal of numerous features, causing the loss of valuable information contained in these features. With a limited number of remaining features, it is expected to select features that contain more comprehensive information. In this stage, features are categorized into three classes according to their type, namely, semantic features, structural features, and metric features. The feature correlation coefficient is calculated by the average of the correlations of a feature with all other features in the same class, which quantifies the amount of comprehensive information contained in the feature within its class. The correlation between features is measured by Pearson's coefficient. The formula for calculating the feature correlation coefficient is as follows:

$$\begin{cases} fc_i = \frac{\sum_{j=L_i, j\neq i}^{R_i} r(x_i,x_j)}{R_i - L_i}, \\ r(x_i,x_j) = \left| \frac{\sum_{k=0}^{N-1}(x_{i,k}-\bar{x_i})(x_{j,k}-\bar{x_j})}{\sqrt{\sum_{k=0}^{N-1}(x_{i,k}-\bar{x_i})^2}\sqrt{\sum_{k=0}^{N-1}(x_{j,k}-\bar{x_j})^2}} \right|, \end{cases} \tag{2}$$

where L_i and R_i are the index of the start and end of the feature class to which x_i belongs in the feature set, $r(x_i, x_j)$ is the absolute value of Pearson's coefficient of x_i and x_j, and fc_i is the feature correlation coefficient of x_i.

As shown in (3), the formula for the overall feature importance measure is a weighted sum of the classifier coefficient, category correlation coefficient, and feature correlation coefficient. The weights are denoted as w_1, w_2, and w_3. The classifier coefficient of x_i is $coef_i$.

$$score_i = w_1 \cdot |coef_i| + w_2 \cdot cc_i + w_3 \cdot fc_i \tag{3}$$

$score_i$ in (3) represents the comprehensive feature importance of x_i, which is calculated based on the three factors. The higher the $score_i$, the more important x_i is. The absolute value of classifier coefficient $|coef_i|$ in (3) reflects the importance of x_i to the classifier output, while the category correlation coefficient cc_i represents the correlation between x_i and the real categories, and the feature correlation coefficient fc_i evaluates the comprehensiveness of x_i in terms of the information it carries. By incorporating these factors, our proposed comprehensive feature importance measure provides more comprehensive and instrumental information for RFECV, which can ultimately enhance the classification performance of the classifier.

Finally, the feature subset is fed into the same type of classifier as used in RFECV to train and predict source files are defective or not. The classifier used in TGR is LR.

IV. Experiment Setup

A. Datasets

The experiments in this paper are constructed on nine Java open-source projects from the PROMISE repository [15], as presented in Table I. The older version of projects is utilized as the training set, and the newer version is the test set.

TABLE I. Dataset Characteristics

Project	Releases	Avg. Files	Avg. Defect rate(%)
ant	1.5 1.6	322	24.3
camel	1.4 1.6	919	18.1
jedit	4.0 4.1	309	25.0
log4j	1.0 1.1	244	62.0
lucene	2.0 2.2	221	53.2
poi	2.5.1 3.0	827	64.0
synapse	1.0 1.2	202	23.0
velocity	1.5 1.6.1	443	49.7
xalan	2.5 2.6	844	47.3

B. Evaluation

In order to comprehensively evaluate the experimental results, Area Under Curve (AUC) and Matthews correlation coefficient (MCC) are adopted in this paper. AUC is defined as the area enclosed by the receiver operating characteristic (ROC) curve and the coordinate axis. The horizontal axis of the ROC curve is the False Positive Rate (FPR), and the vertical axis is the True Positive Rate (TPR). The FPR and TPR can be calculated by (4). MCC is an evaluation indicator comprehensively considering the four basic evaluation indicators in the confusion matrix: TP, TN, FP, and FN, and is calculated by (5).

$$\begin{cases} FPR = \frac{FP}{FP+TN} \\ TPR = \frac{TP}{TP+FN} \end{cases} \tag{4}$$

$$MCC = \frac{TP*TN - FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{5}$$

C. Parameters Settings

The PROMISE repository not only provides the number of defects of source files but also provides 20 software metrics used in TGR, including Weighted Methods per Class (WMC), LOC, etc. The dimensionalities of semantic features and structural features are set to 50 and 30. The kernel sizes of TextCNN are 4, 5, and 6, and the number of every type of kernel is 100. The three weights of (3), w_1, w_2, and w_3, are set to 0.6, 0.2, and 0.2, respectively. The RFECV utilizes 5-fold cross-validation, removes 1 feature at each iteration, uses F-measure as the scorer of CV, and selects 20 features at least. The batch size and epoch of the experiment are respectively set to 32 and 100, and the experiments are repeated 10 times.

D. Experimental Design

In order to analyze and evaluate the proposed TGR method, three Research Questions (RQ) are posed:

RQ1: *Dose the proposed TGR method perform better than the baseline methods?*

RQ2: *How do the multi-type features, comprehensive feature importance measure, and the improved RFECV perform in TGR?*

RQ3: *Can the proposed method be effective for the prediction performance improvement of other SDP methods?*

To answer **RQ1**, four baseline methods are set to compare with TGR.

1) LR: A traditional method using software metrics as features of LR classifier.
2) SVM: A traditional method using software metrics as features of SVM classifier.
3) DP-CNN: A deep learning method using a standard CNN to extract abstract semantic features from the pre-order sequences of ASTs, and using LR as a classifier with the extracted features and software metrics as features [1].
4) BiLSTM: A deep learning method using a Bi-directional LSTM automatically learns the semantic and contextual information from the program's AST, choosing LR as the final classifier [2].

The ablation experiment is to explore the influence of a part of a model by deleting this part and testing the performance of the model. Therefore, in order to answer **RQ2**, the following methods are set up:

1) TextCNN: A method using TextCNN to extract semantic features from the pre-order sequence of ASTs, which has the same TextCNN as the one in the TGR method, and using LR as a classifier.
2) TG: A method without feature selection, using the same multi-type features, feature extraction stage, and classifier as in TGR.
3) TG+: A method using the same feature extraction stage and classifier as in TGR, and using standard RFECV (a RFECV using classifier coefficients as feature importance measure) to select features. The parameters of RFECV are consistent with those in TGR.

In order to compare the above methods accurately and fairly, the features used in these methods are limited to the features obtained from the same experiment. In detail, after feature extraction in each experiment, the extracted features are input into different classifiers after different processing, corresponding to TextCNN, TG, TG+, and TGR methods.

Finally, to verify the effect of the multi-type features and feature selection proposed in this paper on other classifiers, SVM classifiers are employed to replace all the LR classifiers in TGR, named TGR-SVM. This experiment serves to answer **RQ3**.

Besides, due to the class imbalance of the dataset, oversampling is used in all methods mentioned in this paper at the data preprocessing stage.

V. Experimental Results

RQ1: *Dose the proposed TGR method perform better than the baseline methods?*

Table II shows the average AUC values and MCC values of TGR and other baseline methods on each project. TGR's

TABLE II. AUC AND MCC FOR TGR VERSUS BASELINE METHOD

Project	**LR**		**SVM**		**DP-CNN**		**BiLSTM**		**TGR**	
	AUC	MCC	AUC	MCC	AUC	MCC	AUC	MCC	AUC	MCC
ant	0.719	0.366	0.715	0.359	0.591	0.160	0.534	0.109	**0.727**	**0.405**
camel	0.597	0.157	0.614	0.193	0.568	0.111	0.637	**0.301**	**0.642**	0.236
jedit	0.712	0.369	0.726	**0.406**	0.630	0.220	0.663	0.317	**0.727**	0.385
log4j	0.760	0.501	0.728	0.474	0.685	0.356	0.632	0.301	**0.789**	**0.558**
lucene	0.617	0.232	0.597	0.193	0.602	0.202	0.598	0.193	**0.632**	**0.259**
poi	**0.706**	**0.397**	0.664	0.319	0.636	0.260	0.604	0.207	0.658	0.303
velocity	0.625	0.238	0.575	0.144	0.627	0.240	0.599	0.194	**0.671**	**0.328**
xalan	0.541	0.081	0.644	0.287	0.617	0.234	0.630	0.261	**0.664**	**0.327**
synapse	0.638	0.263	0.638	0.278	0.567	0.129	0.554	0.184	**0.690**	**0.377**
Average	0.657	0.289	0.655	0.295	0.613	0.212	0.606	0.230	**0.689**	**0.353**

TABLE III. AUC AND MCC FOR TEXTCNN, TG, TG+

Project	**TextCNN**		**TG**		**TG+**	
	AUC	MCC	AUC	MCC	AUC	MCC
ant	0.713	0.378	0.708	0.370	0.712	0.378
camel	0.635	0.223	0.642	0.236	0.640	0.233
jedit	0.709	0.353	0.717	0.368	0.720	0.372
log4j	0.761	0.504	0.771	0.526	0.776	0.536
lucene	0.618	0.231	0.626	0.247	0.615	0.227
poi	0.644	0.275	0.649	0.287	0.647	0.283
velocity	0.648	0.284	0.663	0.313	0.657	0.302
xalan	0.642	0.284	0.661	0.321	0.656	0.312
synapse	0.672	0.341	0.675	0.349	0.678	0.352
Average	0.671	0.319	0.679	0.335	0.678	0.333

average AUC value and MCC value are 0.689 and 0.353, respectively. It is noteworthy that TGR outperforms the baseline methods in the majority of projects, and that its average AUC and MCC values are superior to those of all baseline methods. Compared with LR and SVM, which use software metrics as features, the AUC values of the TGR method achieve improvements of 4.9% and 5.2%, respectively. Compared with deep learning methods, DP-CNN, and BiLSTM, the AUC values of the TGR method achieve improvements of 12.4% and 13.7%, respectively. Especially, the biggest improvement of TGR's AUC is 36.1% on the ant project compared with BiLSTM. The experimental results demonstrate that TGR overall performs best compared with all baseline models.

RQ2: *How do the multi-type features, comprehensive feature importance measure, and the improved RFECV perform in TGR?*

Table III exhibits the average AUC and MCC values of TextCNN, TG, and TG+ on each project. To facilitate a more intuitive analysis of the impact of feature combination and feature selection stage in TGR, we present box plots depicting the AUC values of TextCNN, TG, TG+, and TGR on each project, which are displayed in Fig. 2. According to Table II and III, compared with TextCNN which utilizes only semantic features, TG, which uses multi-type features, demonstrates improvements in average AUC. Compared with TG, the proposed TGR with improved RFECV exhibits increases in average AUC. As Fig. 2 shows, it can be observed that the majority of the boxes representing TextCNN, TG, and TGR, display a gradual increase in position. These findings demonstrate that the multi-type feature and improved RFECV can improve the prediction performance of TGR.

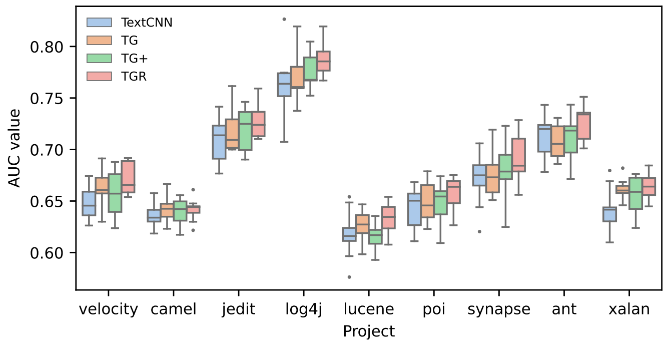

Figure 2. Comparison of TextCNN, TG, TG+, and TGR

On the other hand, the length of the box and whisker in box plots reflects the volatility of the data to some extent. According to Fig. 2, compared with TextCNN, the boxes of TG changes irregularly in length. It is worth noting that the boxes for TGR are the shortest among most projects. Hence, it is verified that the AUC values of the TGR's prediction results exhibit a more concentrated distribution on most projects. These findings indicate the improved RFECV contributes to improvement in the stability of TGR.

To explore the impact of the proposed comprehensive feature importance measure in improved RFECV, Fig. 3 shows the average improvement of the MCC value of the TG+ and TGR methods relative to TG in each Project. It is intuitive that in five projects, the average improvement of TG+'s MCC is negative. In contrast, the average improvement of TGR's MCC is positive in eight projects. These trends are also reflected in Fig. 2, where it can be observed that, apart from the log4j and synapse projects, the boxes of TG+ do not exhibit a significant elevation compared with those of TG. Moreover, the boxes of TGR are almost higher compared with TG+. And The average AUC and MCC in Table III of TG+, are lower than those in Table II of TGR. These findings demonstrate the superiority of the comprehensive feature importance in improved RFECV over the classifier coefficients alone in standard RFECV.

Based on the above analysis, TGR achieves superior and more stable prediction performance compared with TextCNN, TG, and TG+. Firstly, the introduction of multi-type features makes the feature set contain more comprehensive defect information. Then, the improved RFECV can highlight effective features, and meanwhile, eliminate invalid or interfering

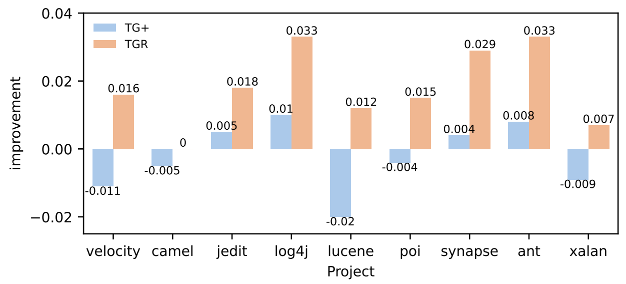

Figure 3. The average improvement of MCC relative to TG

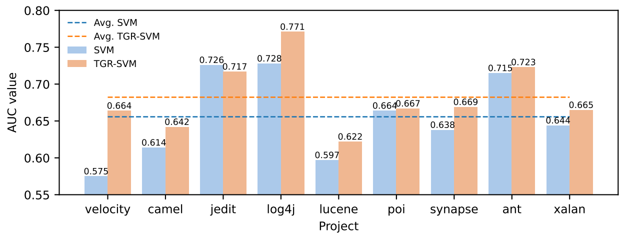

Figure 4. Comparison of TGR-SVM and SVM

features. Moreover, the comprehensive feature importance measure takes into full account the real category, correlations with other features, and classifier coefficient. Hence, TGR reduces the adverse effects of classifier misclassification and obtains the high-quality feature subset.

RQ3: *Can the proposed method be effective for the prediction performance improvement of other SDP methods?*

The average AUC values of TGR-SVM and SVM on each project are shown in Fig. 4, and the two dashed lines represent the average AUC values on all projects of the two methods. TGR-SVM exhibits superior performance over SVM, as evidenced by eight individual projects demonstrating an average AUC value higher than that of SVM. Notably, the highest improvement in the AUC value achieved by TGR-SVM is 0.089 on the velocity project. Moreover, TGR-SVM demonstrates a significant increase in its average AUC value across all projects compared with SVM. Seven AUC values of TGR-SVM exceed the average AUC value across all projects of SVM. The results demonstrate that the multi-type features and the improved RFECV in TGR are not confined to the LR classifier. Applying them to SVM classifiers also achieves increased prediction performance.

VI. Conclusion

This paper proposes TGR that comprises two main stages: feature extraction and feature selection. In the feature extraction stage, the semantic features and structural features are extracted from source codes, respectively. These features, along with software metrics, are concatenated to build a multi-type feature set. The feature selection stage employs an improved RFECV to eliminate the redundancy features, which is implemented by integrating the category correlation coefficients, the feature correlation coefficients, and the classifier coefficients as the feature importance measure. The selected feature subset is fed to the classifier to get predictions. Experimental results demonstrate that TGR outperforms the four baseline methods in prediction performance. The ablation experiments are conducted to validate the effectiveness of multi-type features and the feature selection stage in TGR. Future work will focus on exploring better methods for extracting more comprehensive defect features, and will apply TGR for cross-project software defect prediction.

VII. Acknowledgement

This work was supported by the second batch of cultivation projects of Pazhou Laboratory in 2022, No. PZL2022KF0008, and by the Zhongshan Produce and Research Fund, PR China under grant no. 210602103890051.

References

[1] J. Li, P. He, J. Zhu, and M. R. Lyu, “Software defect prediction via convolutional neural network,” in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 318–328, IEEE, 2017.

[2] J. Deng, L. Lu, and S. Qiu, “Software defect prediction via lstm,” *IET software*, vol. 14, no. 4, pp. 443–450, 2020.

[3] S. Wang, T. Liu, J. Nam, and L. Tan, “Deep semantic feature learning for software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1267–1293, 2018.

[4] C. Zhou, P. He, C. Zeng, and J. Ma, “Software defect prediction with semantic and structural information of codes based on graph neural networks,” *Information and Software Technology*, vol. 152, p. 107057, 2022.

[5] Y. Kim, “Convolutional neural networks for sentence classification,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, Association for Computational Linguistics, Oct. 2014.

[6] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?,” in *International Conference on Learning Representations*, 2019.

[7] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene selection for cancer classification using support vector machines,” *Machine Learning*, vol. 46, pp. 389–422, 2002.

[8] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, p. 476 – 493, 1994.

[9] Y. Singh, A. Kaur, and R. Malhotra, “Software fault proneness prediction using support vector machines,” in *2009 World Congress on Engineering*, vol. 1, pp. 1–3, 2009.

[10] J. Xu, J. Ai, J. Liu, and T. Shi, “Acgdp: An augmented code graph-based system for software defect prediction,” *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 850–864, 2022.

[11] M. N. Uddin, B. Li, Z. Ali, P. Kefalas, I. Khan, and I. Zada, “Software defect prediction employing bilstm and bert-based semantic feature,” *Soft Computing*, vol. 26, no. 16, pp. 7877–7891, 2022.

[12] K. Zhu, S. Ying, N. Zhang, and D. Zhu, “Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network,” *Journal of Systems and Software*, vol. 180, p. 111026, 2021.

[13] A. A. Saifan and L. Abu-wardih, “Software defect prediction based on feature subset selection and ensemble classification,” *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, vol. 14, no. 2, pp. 213–228, 2020.

[14] V. J. Marin and C. R. Rivero, “Towards a framework for generating program dependence graphs from source code,” in *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics*, pp. 30–36, 2018.

[15] Z. He, F. Peters, T. Menzies, and Y. Yang, “Learning from open-source projects: An empirical study on defect prediction,” in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 45–54, IEEE, 2013.