Data Selection for Cross-Project Defect Prediction with Local and Global Features of Source Code

Xuan deng
School of Computer Science and Information Engineering
Hubei University
Wuhan, China
xuan_deng@qq.com

Peng He*
School of Computer Science and Information Engineering
Hubei University
Wuhan, China
penghe@hubu.edu.cn

Chun Ying Zhou
School of Computer Science and Information Engineering
Hubei University
Wuhan, China
zcy9838@qq.com

***Abstract*[1] :** *An open challenge for cross-project defect prediction (CPDP) is how to select the most appropriate training data for target project to build quality predictor. To our knowledge, existing methods are mostly dominated by traditional hand-crafted features, which do not fully encode the global structure between codes nor the semantics of code tokens. This work is to propose an improved method which is capable of automatically learning features for representing source code, and uses these feataures for training data selection. First, we propose a framework ALGoF to automatically learn the local semantic and global structural features of code files. Then, we analyze the feasibility of the learned features for data selection. Besides, we also validate the effectiveness of ALGoF by comparing with the traditional method. The experiments have been conducted on six defect datasets available at the PROMISE repository. The results show that ALGoF method helps to guide the training data selection for CPDP, and achieves a 48.31% improvement rate of F-measure. Meanwhile, our method has statistically significant advantages over the traditional method, especially when using both the local semantic and global structural features as the representation of code files. The maximum improvement of F-measure can reach 42.6%.*

Keyword : cross-project defect prediction; semantic feature; structural feature; software quality; representation learning.

I. INTRODUCTION

The main purpose of cross-project defect prediction (CPDP) is to predict defect-prone files in a project based on the defect data collected from other projects.. Peter et al. [1] proposed that a major issue in CPDP is how to find the appropriate training data set (TDS) for the target project. That is, the selection of high-quality cross-project training data is a key breakthrough.Nowadays,there is a growing collection of defect datasets on the Internet.Thus, the construction of an appropriate TDS is a more serious challenge for CPDP.

To address this issue, researchers in this field have attempted to characterize code files by using traditional hand-crafted features (e.g., CK, Halstead, MOOD, and McCabe's CC metrics), and guide the TDS selection based on these metric values[3-4,6]. Unfortunately, these metrics only contain statistical information of programs and require human design. As we known, programs have well-defined syntax and rich semantics hidden in the Abstract Syntax Trees (ASTs), which have been successfully extracted and used for defect prediction [7]. In addition, researchers also validated that the globally structural information extracted by network representation learning can lead to more accurate defect prediction [9-11].In other word, both the local semantic and global structural information of source code files may affect the selection of TDS in CPDP.

Thus,we propose a new framework called ALGoF to Automatically learn the Local semantic (fine-grained) and Global structural (coarse-grained) Features of code files for data selection in CPDP and seek empirical evidence that they can achieve acceptable performance compared with the traditional method. Our contributions are summarized as follows:

- We leverage representation learning technique to automatically learn the local and global features of the program from source code files, and use to guide the training data selection in CPDP.
- The results on six projects show that the proposed ALGoF method can improve CPDP, compared to using traditional hand-crafted features. The combination of global and local features has greater impact on the improvement of prediction performance.

The rest of this paper is organized as follows. Section 2 is a review of related work. Sections 3 describes the proposed approach.Section 4 is the detailed experimental setups, and Section 5 shows and discusses the experimental results. Finally, Section 6 concludes the work and presents the agenda for future work.

II. RELATED WORK

A. Data Selection for CPDP

In software engineering, CPDP has drawn wide attention and many studies are carried out to explore the strategies of training data selection. Peters et al. [1] proposed a filter guided by the structure of content-rich source project data and achieved great performance.To obtain a comprehensive evaluation, Bin et al. [8] even conducted a thorough experiment to compare nine relevancy filters on 33 datasets.Hosseini et al. [6] further showed that the selection of training data can lead to

[1]DOI reference number: 10.18293/SEKE2022-086

better performance in CPDP, and concluded that search-based methods combined with feature selection was a promising way.

From the above rich results, there is a commonality that is the defect data used are represented by traditional hand-crafted software metrics.

B. Representation Learning in Software Engineering

Representation learning has been widely applied to feature learning.In software engineering, some algorithms have been adopted to the Abstract Syntax Trees (ASTs) representation of source codes. For example, Wang et al. [7] leveraged Deep Belief Network to automatically learn semantic features from token vectors extracted from programs'ASTs and further validated that the learned semantic features significantly improved defect prediction.Besides, some studies have demonstrated the effectiveness of structural features in improving defect prediction. For example, Qu et al. [10] used network embedding technique, node2vec, to automatically learn to encode dependency network structure into low-dimensional vector spaces to improve software defect prediction. Zeng et al. [11] also recently analyzed the influence of network structure features of code on defect prediction.

Existing studies have verified the usefulness of semantic and structural features on defect prediction, but do not involve the analysis of the impact on data selection in CPDP.

III. APPROACH

This section introduces the entire framework of ALGoF method in detail, mainly comprised of three parts: automatically learning the semantic and structural features of code files, training data selection and defect prediction (Figure 1). First, we extract the dependencies between the classes from the source code files to construct a class dependency network. Then, we perform network embedding learning on the CDN to generate the global structural features of classes. Meanwhile, we leverage a Convolutional Neural Network (CNN) to automatically learn semantic features using token vectors extracted from the class files' abstract syntax trees (AST). Second, combine the structural and semantic features of the class obtained in the previous step, and use them as the representation of the class file. After that, the similarity scores between source file instances and each target file instance are recorded and used to guide how to select appropriate source file instances for cross-project defect prediction. Finally, we use the resulting source file instances to train the predictor and test on the target instances.

A. Generation of local semantic features

1) Parsing AST

As previous study [9] has shown, AST can represent the semantic information of source code file with the most appropriate granularity.

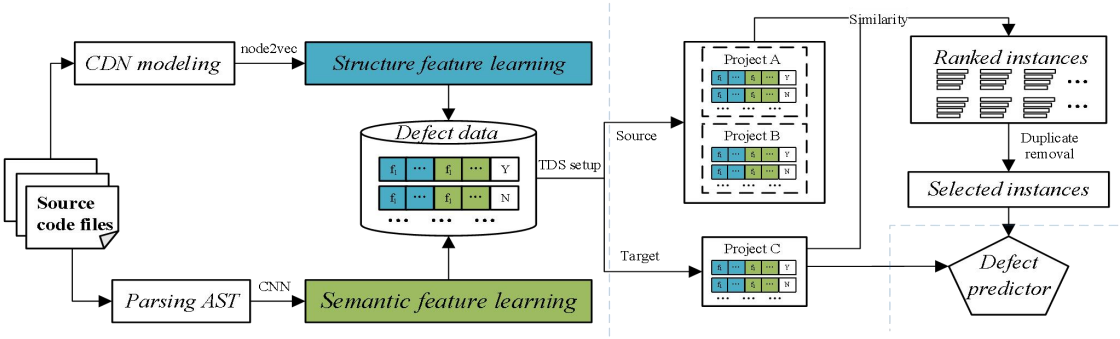

Figure 1. The entire framework of ALGoF method

We first parse the source code files into ASTs by calling an open-source python package javalang. Given a path of the source code, the token sequences of all files will be output.As treated in [7], we only select three types of nodes on ASTs as tokens: (1) nodes of method invocations and class instance creations; (2) declaration nodes, i.e., method/type/enum declarations; (3) control flow nodes, such as while, if, and throw. For more details, please refer to our previous work [11].

Then,we convert the extracted token sequences into the numerical token vectors.We append 0 to each integer vectors to make each files' lengths consistent with the longest vector.Note that, to filter out infrequent tokens, we only encode tokens occurring three or more times, the others denote as 0.

2) Building CNN

After encoding and preprocessing token vectors,we exclude the input and output layers. We train the CNN model with four layers: an embedding layer (turn integer token vectors into real-valued vectors of fixed size), a convolutional layer, a max-pooling layer, and a fully connected layer.

Given a project P, assume tha it contains n source code files, all of which have been converted to integer token vectors $x \in \mathcal{R}^l$, where l is the length of the longest token sequence.Through the embedding layer, each file becomes a real-value matrix $X_{l\times d}$. As the input of convolutional layer, a filter $\mathcal{L} \in \mathbb{R}^{h\times d}$ is applied to a region of h tokens to produce a new feature.

Then, the max-pooling operation is performed on the mapped features and the maximum value $\widehat{F} = max\{f\}$ is taken as the feature corresponding to that particular filter $\mathcal{L}$. Usually, multiple filters with different region sizes are used to get multiple features. Finally, a fully connected layer further generated the local features.

B. Generation of Global structural Features

Before applying network embedding to represent global structural features of source codes, it is necessary to build a Class Dependency Network. As did in [11], we use DependencyFinder API to parse the compiled source files (.zip or .jar extensions) and extract their relationships using a tool developed by ourselves. With the CDN, we further perform embedding learning using the *node2vec* method. For more details on *node2vec*, please refer to the literature [2].

C. Training Data Selection

The general training data selection process consists of three steps: candidate TDS setup, ranking and remove duplicate.For more details, please refer to our previous work [4].In order to characterize each code file, the two types of features obtained above are connected and marked with defect label, so as to generate the defect data set required for subsequent tasks.

An instance is characterized as a feature vector, and each target instance as an input. A similarity index (e.g., cosine similarity) is applied to construct a model for ranking source instances based on the given target instances. For instance, the cosine similarity between a source instance I_p and an input target instance I_q is computed via their vector representations:

$$Sim(I_p, I_q) = \frac{\overrightarrow{I_p} \cdot \overrightarrow{I_q}}{\|I_p\| \times \|I_q\|} = \frac{\sum_{i-1}^{n} (f_{pi} \times f_{qi})}{\sqrt{\sum_{i=1}^{n} f_{pi}^2} \times \sqrt{\sum_{i=1}^{n} f_{qi}^2}} \quad (1)$$

Where $\overrightarrow{I_p}$ and $\overrightarrow{I_q}$ are the feature vectors for instances I_p and I_q respectively. f_i represents the i^{th} feature value of the features.

For each target instance, the top-k (k=10) source instances are ranked by the Sim values and returned. Hence, the finally selected TD is composed by integrating the set of top-k source instances of each target instance (i.e., the duplicate instances are removed to maintain uniqueness).

IV. Experiment Setup

A. Dataset

In this paper, 6 defect datasets from the PROMISE repository[2] were selected for validation. Detailed information on the datasets is listed in Table 1, where *#files* and defect rate are the number of files and the percentage of defective files, respectively.

B. Experimental Design

To make a comparison between the traditional hand-crafted features and automatically learn features in our paper, four scenarios will be considered in our experiments.

(i)THC represents predictor based on the traditional hand-crafted features.

(ii)ALoF represents predictor based only on the local semantic features.

(iii)AGoF represents predictor based only on the global structural features.

(iv)ALGoF represents predictor based on both the local and global features.

C. Classifiers and Evaluation Measures

This paper utilizes logistic regression (LR), which is widely used in the defect prediction, as the classifier. We use the default parameter settings for LR specified in Weka[3] unless otherwise specified.

To evaluate the defect prediction model's performance, we use widely adopted F-measure, which is the harmonic mean of precision and recall.

TABLE I. Details of the datasets

Project	Releases	#files	defect rate(%)
Camel	1.4	892	17.1
Lucene	2.0	186	48.9
Poi	2.5	379	65.1
Synapse	1.1	222	27.0
Xalan	2.6	875	47.0
Xerces	1.3	446	15.0

V. Experimental Results

RQ1: *Does data selection based on ALGoF and its variants of CPDP work well?*

We first take the case where the initial source TDS without any selection is considered as a baseline, labeled as iTDS. On contrary, we label the case of TDS selection using the features learned in this paper as sTDS. Then we perform cross-project predictions in both cases mentioned above.

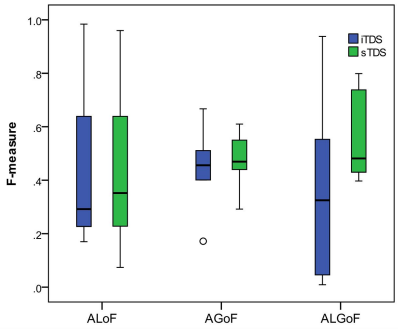

Figure 2. A comparison on F-measure of CPDP under the case of iTDS and sTDS.

TABLE II. The improvement of F-measure of CPDP under the case of iTDS and sTDS.

Model	iTDS	sTDS	Δ（%）
ALoF	0.292	0.352	20.55%
AGoF	0.456	0.470	3.07%
ALGoF	0.325	0.482	48.31%

Figure 2 shows that, on average across the six datasets, for CPDP with iTDS, the median F-measure are 0.292,0.456 and 0.325 respectively,while sTDS are 0.352,0.470 and 0.482 Clearly, the results verify the necessity of data selection for CPDP. The improvement rate is the most obvious in the ALGoF scenario, reaching 48.31%, followed by that of ALoF with 20.55%. The results also show that AGoF performs better than ALoF whether data selection is used or not. However, ALGoF outperforms ALoF and AGoF when only considering

[2] http://promise.site.uottawa.ca/SERepository/datasets-page.html

[3] http://www.cs.waikato.ac.nz/ml/weka/

the training data selection, indicated by the largest F-measure value in bold.

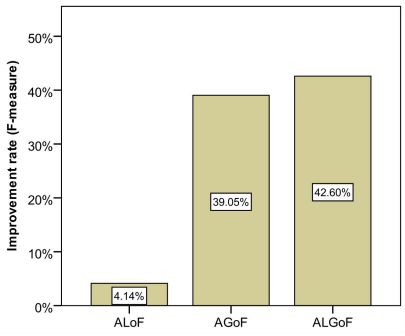

Figure 3. The improvement rate of F-measure values compared with THC scenario (with data selection).

TABLE III. THE WILCOXON SIGNED-RANK TEST AND CLIFF'S DELTA ($0.33 \leqslant |\delta| < 0.474$ MEANS THE MEDIUM EFFECTIVENESS LEVEL, AND $|\delta| \geqslant 0.474$ MEANS THE LARGE EFFECTIVENESS LEVEL [7]).

	Sig. p-value (0.05)	**cliff's delta (δ)**
ALoF_iTDS-ALoF_sTDS	0.934	-0.028
AGoF_iTDS-AGoF_sTDS	0.745	0.000
ALGoF_iTDS-ALGoF_sTDS	0.116	-0.484
ALoF_sTD-ALGoF_sTDS	0.219	-0.389
AGoF_sTD-ALGoF_sTDS	0.345	-0.083

Additionally, statistical tests assist in understanding whether a statistically significant difference between two results exists. We further utilize the Wilcoxon signed-rank test and Cliff's effect size (δ) to check whether the difference among the prediction models is significant. In Table 3, the results highlight that there are no significant differences between CPDP with iTDS and CPDP with sTDS in our experiment, indicated by all the *p*-values >0.05. However, the effectiveness level between ALGoF_iTDS and ALGoF_sTDS is large, indicated by $|\delta| = 0.484$. Besides, the effectiveness level between ALoF_sTDS and ALGoF_sTDS is medium.

TABLE IV. COMPARISON OF WILCOXON SIGNED-RANK TEST AND CLIFF'S EFFECT SIZE OF THE AUTOMATIC EXTRACTION FEATURES WITH THC.

	Sig. p-value (0.05)	**cliff's δ**
ALoF_sTDS - THC_sTDS	0.719	0.111
AGoF_sTDS - THC_sTDS	0.035	0.417
ALGoF_sTDS - THC_sTDS	0.035	0.667

In short, for the data selection task in CPDP, the features automatically learned from the source code are helpful to guide the task, so as to improve the prediction performance. In addition, the global structural feature works better than the local semantic feature, but the combination of the two is optimal. Nevertheless, data quality is more important than data quantity.

RQ2: *For CPDP data selection, which is better: automatically learned features or traditional hand-crafted features?*

In this part, we mainly compare the ALGoF method proposed in this paper with THC. As seen in Figure 3, the improvement rates of F-measure of ALoF, AGoF and ALGoF are 4.14%, 39.05% and 42.6% respectively. That is, for the data selection problem of CPDP, the use of automatically learned features to represent the source code files is better than the use of traditional hand-crafted features. In addition, Table 4 also further shows that there are significant differences between ALGoF (AGoF) and THC, indicated by the small *p*-value of 0.035 (<0.05) and the large δ value of 0.667 (>0.474). Meanwhile, according to the *p*-value of 0.719 and the δ value of 0.111, the local semantic features seem to have comparable effects to traditional source code features.

In summary, the method proposed in this paper can pick higher quality training sets for CPDP than using traditional hand-crafted features. Especially when both local semantic features and global structural features are considered.

VI. CONCLUSION

This study is to propose an improved method which is capable of automatically learning features for representing source code, and uses these feataures for training data selection. The results indicate that features automatically learned from the source code (e.g., local semantic feature and global structural feature) are helpful to guide the training data selection for CPDP.Meanwhile, compared with the case of no data selection processing, the F-measure improvement rate of ALGoF is 48.31%. In addition, the results also show that our method is significantly better than the traditional method, especially when using both the local semantic and global structural features as the representation of code files. Notedly, about 42.6% defective instances can be additionally predicted by our method.

In the future, we would like to extend our automatically feature generation approach to C/C++ projects for CPDP. In addition, it would be promising to leverage our approach to guide heterogeneous defect prediction.

REFERENCES

[1] F Peters, Menzies T , Marcus A . Better cross company defect prediction[C]// in Proceedings of the 10th MSR, 2013:409–418.

[2] A. Grover and J. Leskovec, node2vec: scalable feature learning for networks[C], in Proc. of ACM SIGKDD Inte. Conf. on Know. Dis.& Data Min., San Francisco, CA, USA, August 2016.

[3] Ryu D , Jang J I , Baik J , et al. A Hybrid Instance Selection Using Nearest-Neighbor for Cross-Project Defect Prediction[J]. Journal of Computer Science & Technology, 2015, 30(005):969-980.

[4] He P , He Y , Yu L , et al. An Improved Method for Cross-Project Defect Prediction by Simplifying Training Data[J]. Mathematical Problems in Engineering, 2018, (PT.6):2650415.1-2650415.18.

[5] Hosseini S., Turhan B., Mäntylä M.: A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. Inf. Softw. Technol. 2018,95, 296–312.

[6] Hosseini S , Turhan B . A comparison of similarity based instance selection methods for cross project defect prediction[C]. 36th ACM/ SIGAPP Symposium on Applied Computing, 2021:1455-1464.

[7] Wang S , Liu T , Nam J , et al. Deep Semantic Feature Learning for Software Defect Prediction[J]. IEEE Transactions on Software Engineering, 2020,46(12):1267-1293.

[8] Y. Bin, K. Zhou, H. Lu, Y. Zhou, B. Xu, Training data selection for cross-project defection prediction: Which approach is better?[C]. Int. Sym. on Emp. Soft. Eng. & Meas. ,2017:354–363.

[9] A. V. Phan, M. L. Nguyen, and L. T. Bui, Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction, 2018, https://arxiv.org/abs/1802.04986.

[10] Qu Y , Liu T , Chi J , et al. node2defect: using network embedding to improve software defect prediction[C]. The 33rd ACM/IEEE Inte. Conf. on Automated Software Engineering, 2018:844-849.

[11] Zeng C , Zhou C Yi , Lv S K , et al. GCN2defect：Graph Convolutional Networks for SmoteTomek-based Software Defect Prediction[C]. The 32nd Inter. Sym. on Software Reliability Engineering (ISSRE 2021)