Article

A Software Defect Prediction Method Based on Program Semantic Feature Mining

Wenjun Yao [1], Muhammad Shafiq [1,2,*], Xiaoxin Lin [1] and Xiang Yu [3]

[1] Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510000, China
[2] School of Computer Science, Shenyang Normal University, Shenyang 110136, China
[3] School of Electronics and Information Engineering, Taizhou University, Taizhou 318000, China
* Correspondence: mshafiq@gzhu.edu.cn

Abstract: As the size and complexity of software systems grow, knowing how to effectively judge whether there are defects in the programs has attracted extensive attention in research. However, current software defect prediction methods only extract semantic information at the syntactic level and lack features to mine defect manifestations at the semantic level of code, because defective software is incomplete or defective in semantic representation. Defective software exhibits incomplete or flawed semantic behavior. This paper proposes a software defect prediction method based on the program semantics feature mining (PSFM) method. Specifically, the semantic information is first extracted from the code grammatical structure information and code text information. Then, the defect feature is mined through the semantic information. Finally, software defects are predicted by using the mined defect features. The experimental results show that, compared with the existing software defect prediction methods, the method in this paper (PSFM method) obtained a higher F-measure value.

Keywords: software defect prediction; abstract syntax tree; tree based on convolution neural network; semantic extraction; feature mining

check for updates

Citation: Yao, W.; Shafiq, M.; Lin, X.; Yu, X. A Software Defect Prediction Method Based on Program Semantic Feature Mining. *Electronics* **2023**, *12*, 1546. https://doi.org/10.3390/electronics12071546

Academic Editor: Andrei Kelarev

Received: 28 February 2023
Revised: 23 March 2023
Accepted: 23 March 2023
Published: 25 March 2023

1. Introduction

Emerging computer technologies such as mobile Internet and cloud applications have made great progress, and have promoted the rapid development of the computer software industry. Software is applied in every aspect in people's work and life. With the increased development of software applications and the introduction of new technologies, the size of software is also increasing, as well as the complexity of software. This results in higher requirements for software quality assurance technology and security. The safety, quality, and reliability of software not only affect the user experience and production activities, but may even endanger people's lives and property. The high quality of software depends on the software defects. Software defects are common in programs, especially with the rapid increase in the complexity and scale of software systems. Therefore, it is very important to predict software defects so as to generate high-quality software.

In recent years, artificial intelligence technology has made great progress, and research applying artificial intelligence technology to software defect prediction methods has also made great progress [1]. The current software defect prediction method combined with artificial intelligence technology has achieved good defect prediction results. It uses deep learning and other methods to extract the features from the static attributes of the software or the intermediate representation of the code. The research on software defect prediction can be conducted after building a classifier [2]. However, many research works on the application of deep learning methods to software defect prediction ignore the impact of program semantics on software defects. Even though some research takes into account the influencing factors of program semantics, the extracted program semantics are all based on the syntax level, not on the understanding and analysis of the program, which is not the semantic level. Program semantics is an important part of program understanding [3]. Therefore, research for software defect prediction based on program semantic feature mining is of great significance.

In order to fully exploit the impact of program semantics on software defects, this paper proposes a software defect prediction method based on the program semantic feature mining (PSFM) method. Specifically, semantic information is first extracted from the code grammatical structure information and code text information. Then, the defect feature is mined through the semantic information. Finally, software defects are predicted by using the mined defect features. The experimental results show that the PSFM method can effectively improve the accuracy of software defect prediction.

The contributions of this paper are summarized as follows:

- First, this paper proposes a method for extracting code semantic information through syntax structure information and code text information. This method first extracts the syntax structure information and code text information, and then combines it to form semantic information. The semantic information provides the basis platform for the research on the influence of program semantics on software defects.
- This paper proposes a method for the influence of software defects on program semantics, of which the research on the influence of program semantics in software defects is completed by mining defect features in the program semantic information.

The rest of this paper is organized as follows. Section 2 introduces related work in software defect research. Section 3 introduces code semantic extraction methods that combine syntax structure information and code text information. Section 4 introduces the software defect prediction method based on semantic information. Section 5 introduces the experimental setup and analysis of the experimental results. Finally, this paper is concluded and future work is discussed.

2. Related Work

The current software defect prediction methods are mainly divided into four categories according to the software engineering catalog, namely: WPDP (Within Project Defect Prediction), CPDP (Cross-Project Defect Prediction), CVDP (Cross Version Defect Prediction), and HDP (Heterogeneous Defect Prediction). The current research is focused mainly on CPDP.

The research on code defect prediction methods mainly is divided into two methods: one is mining software defect characteristics through the static properties in code, and the other is mining software defect characteristics through machine learning [4]. Mining software defect characteristics through the static properties in code mainly focuses on the static properties in software features that are obtained manually. Radjenovic et al. [5] completed code defect prediction by analyzing the complexity of the software modules and the distribution of the software defects. Specifically, using the source code complexity features as an input to predict software defects, which mainly includes based on operator and operation number-based Halstead features and dependency-based McCabe features. Wang et al. [6] firstly formally defined the attribute characteristics of software vulnerabilities, and then proposed a program intermediate representation of the vulnerability executable path set. Finally, Wang completed the code vulnerability detection after designing a software vulnerability static detection framework based on the vulnerability executable path set. Liu et al. [7] used feature selection and machine learning algorithms to perform feature dimensionality reduction and weight optimization on vectorized raw data. Evaluating the severity of the software defect reports was completed with using classification algorithms. The experiments showed that the AUROC (area under the receiver operating characteristic curve) value of the software defect report severity prediction was improved to 0.767. Li et al. [8] proposed a CPDP-FSTr (Cross-Project Defect Prediction method based on Feature Selection and TrAdaBoost) method for cross-item defect prediction, which was based on TrAdaBoost with feature selection. The method used the KPCA (Kernel Principal Component Analysis) algorithm to delete redundant data in the source items, continuing to select the candidate source item data that were closest to the distribution of the target item. The experiments showed that compared with the FeCTrA (Feature Clustering and TrAdaBoost) and CMKEL (Multiple Kernel Ensemble Learning) methods, the CPDP-FSTr method had an improved F1-score. However, the method for mining software defect characteristics through the static properties in code cannot completely cover all of the features in the code. It may ignore other features in code defects, which cannot cope with the continuous increase in the scale of software. It may incur a problem, in that the accuracy and coverage in the defect prediction model cannot be effectively improved. Finally, the defect prediction performance is not acceptable.

Mining software defect characteristics through machine learning refers to mining the code features in the source code through machine learning methods, thereby improving the performance of software defect prediction [9]. Wang et al. [10] used a Deep Belief Network (DBN) to extract symbolic vectors from the AST of the program to learn code features, which is superior to traditional feature-based methods in effect. However, the structural features of the program are not taken into account with software defect prediction. M. Shafiq et al. [11] proposed a new framework model and a hybrid algorithm that selected an effective machine learning algorithm for the identification of malicious and anomaly traffic. The experimental results show that the proposed model with the algorithm was effective. In order to fully consider the structural features of the program code, Li et al. [9] mapped the representative nodes in AST, combined with the traditional defect features in order to train the classifier, and they achieved better results than DBN. Dam et al. [12] mapped each node in the AST, and input each AST branch into the tree-based LSTM unit to train the model. This method can fully take into account the structural features in AST, and is also very helpful for discovering specific sequence patterns. It is worth mentioning that this method uses an unsupervised method to train LSTM units. So, it does not require a large number of expensive dataset labels. J. Qiu et al. [13] aimed to provide theoretical, methodological, and technical guidance for IoT search access control mechanisms in large-scale dynamic heterogeneous environments. Z. Tian et al. [14] proposed a web attack detection system that takes advantage of analyzing URLs, which was designed to detect web attacks and could be deployed on edge devices. The experimental results showed 99.410% for accuracy, 98.91% for TPR, and 99.55% for DRN, demonstrating that the system is competitive when detecting web attacks. Pan et al. [15] proposed the Improved-CNN method, which achieved better results and explored the instability of hyperparameters in the defect prediction model. Huang et al. [16] proposed a defect prediction method based on the heavy son-based abstract syntax tree (HSAST). By marking the nodes on the AST and dividing the HSAST, the code defect prediction model was trained by CNN and RNN. The experiments showed that compared with the DFS method, HSAST improved F1 by 3% on average and AUC by 4% on average. Dong et al. [17] proposed a software defect prediction framework based on feature fusion. By parsing the program into two different program representations, AST and Token sequence, the tree convolutional neural network and text convolutional neural network were used to extract the program structure and semantic features. The experimental results show that this framework could obtain an excellent F1-score. X Cheng et al. [18] proposed a ContraFlow software defect detection framework by selecting and retaining feasible value flow (aka program dependency) paths, using a self-supervised pre-trained path embedding model with comparative learning for path-based vulnerability detection. When making real item predictions, the ContraFlow model achieved a better recall value. M. Shafiq et al. [19] built a framework model with a novel feature selection metric approach, which was based on the wrapper technique to filter the features accurately and select effective features for the selected ML algorithm by using the area under the curve (AUC) metric. The experimental results analysis showed that our proposed method was efficient and could achieve >96% results on average. Z. Tian et al. [20] proposed a secure digital evidence framework using blockchain (Block-DEF) with a loose coupling structure in which the evidence and the evidence information were maintained separately. The experimental results showed that Block-DEF is a scalable framework, it guarantees the integrity and validity of evidence, and balances privacy and traceability well. Kovačević et al. [21] introduced a multi-threaded approach, which accelerated the evaluation process by over 9.5 times, while the number of fitness evaluations using the improved Long-Term Memory Assistance (LTMA) was reduced by up to 7.3%. Xie et al. [22] proposed Semantic-Guided Pixel Contrast (SePiCo), a novel one-stage adaptation framework that highlights the semantic concepts of individual pixels to promote the learning of class-discriminative and class-balanced pixel representations across domains, eventually boosting the performance of self-training methods. The experiments show that SePiCo can make progress on both synthetic-to-real and daytime-to-nighttime adaptation scenarios. Yang, D. et al. [23] considered the LF Contextual Feature (LFCF) and LF Geometric Feature (LFGF) for occluded area perception and segmentation edge refinement. The experimental results for both the real-world and synthetic datasets proved the state-of-the-art performance of our method. Sheng, H. et al. [24] proposed a high-quality and challenging urban scene dataset, containing 1074 samples composed of real-world and synthetic light field images, as well as pixel-wise annotations for 14 semantic classes, showing that the proposed dataset presented new challenges and supported detailed comparisons among different methods. Shi, K. et al. [25] proposed a new AST path pair-based source code representation method (PathPair2Vec) and applied it to software project defect prediction. The experiments showed that PathPair2Vec could improve the F1 score. Khalilian, A. et al. [26] proposed leveraging the common components that constitute the APR techniques and tools, and developed a principled method for the application of components. Khalilian, A. et al. concluded that knowledge accumulation and characterization through literature reviews could thus be facilitated through the identified suite of components. Fu, W. et al. [27] repeated and refuted these results, stating there was too much variability in the efficacy of the Yang et al. [28] predictors, so their approach and Yang's findings were grouped across N projects. Kovačević et al. [29] described a research work on Semantic Inference, which can be regarded as an extension of Grammar Inference. The results show that the authors were able to infer the semantics only from the samples and their associated meanings for several simple languages, including the Robot language.

However, most of the machine learning methods above ignored the impact of program semantics on defects, or the extracted semantics were based on the grammatical level, not on an understanding and analysis of the program. Furthermore, program semantics is an important part of program understanding [3].

In summary, the current software defect prediction methods only extract semantic information at the syntactic level and lack features to mine defect manifestations at the semantic level of the code. The performance of the semantic information in detecting codes is extremely important. Therefore, using the method proposed in this paper (PSFM method), the influence of program semantics on defective software will be explored, and the prediction of software defects will be completed based on mining defect features from semantic information.

3. Component Design

In this section, the overall framework of the PSFM method will be described in detail. The flow of the PSFM method is as follows: (a) code semantic extraction, (b) defect feature mining, and (c) performing defect prediction. Figure 1 illustrates the overall framework of the PSFM method.

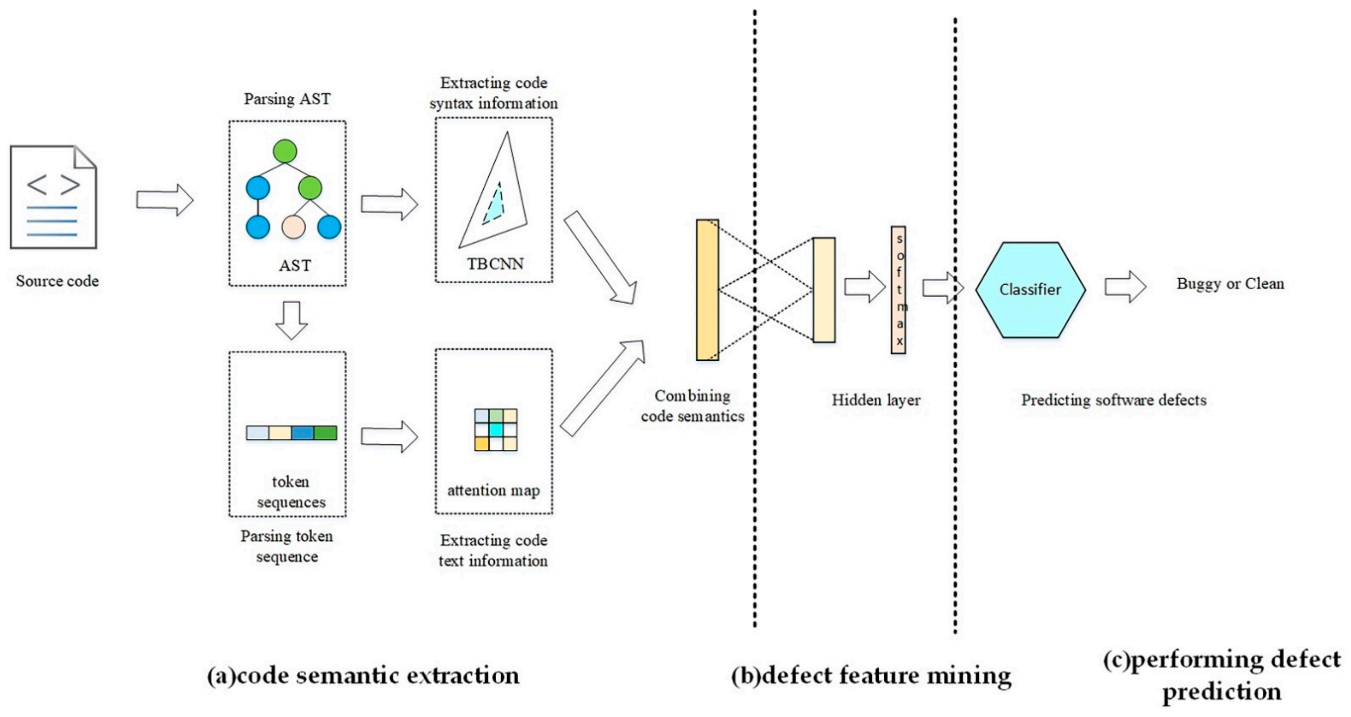

Figure 1. The overall framework of the PSFM method: (**a**) code semantic extraction, (**b**) defect feature mining, and (**c**) performing defect prediction.

3.1. Code Semantic Extraction

In this section, this paper introduces the code semantic extraction method of combining syntactic structure and code text information in detail. This method first extracts the syntax structure information and code text information, and then combines it to form the semantic information. The semantic information provides a basis for the research on the influence of program semantics on software defects. The flow of code semantic extraction includes the following: (1) parsing AST, (2) parsing token sequence, (3) extracting the code syntax information, (4) extracting code text information, and (5) combining code semantics.

3.1.1. Parsing AST

The program code contains plenty of feature information, such as syntax structure information, function information, and so on. However, it is difficult to directly extract feature information from the code. On the one hand, the amount of information is relatively large because the code content is the realization of the overall function of the software; on the other hand, the distribution of the feature information of the code is relatively sparse. It is often difficult to be directly extracted. For software defect prediction methods, not all of the feature information contained in the code is valuable for defect prediction, such as commented code and so on. Therefore, it is necessary to parse the program code into an intermediate form, which can save all the information in the code file as much as possible, and can facilitate the feature mining of the code. As an intermediate form of code representation, AST (Abstract Syntax Tree) can reflect the characteristic information of the code well. In this paper, an open source Python package called javalang was used to complete the task of code module parsing to AST. Based on the recognized Java language specification, javalang supports parsing Java code [30]. Another reason for choosing javalang was to compare the performance with a baseline method that uses javalang to parse Java codes. The package can be obtained through the javalang tool [31], which can parse Java source code into the corresponding AST.

According to the AST object parsed by the javalang tool, AST is composed of nodes, and each node contains type information and value information. The type information includes (1) declared nodes, including method definitions, class definitions, variable definitions, etc.; (2) called nodes, including method calls, etc.; and (3) other data nodes, including control flow nodes (such as loops, branches, exception capture, etc.). Value information Value refers to the information in the code content. Such as the statement "i = 1", in the node, whose type information is VariableDeclarator, and its value information is the number 1.

After obtaining all of the nodes, building a tuple {*V*, *C*} will represent the whole AST. Among the tuple, *V* represents the information of the node, including the type information and value information, and *C* represents the parent−child relationship of all child nodes. The tuple can be represented as the AST of a piece of code. The representation style of a tuple is as follows:

$$\begin{cases} V = \{node_{type}, node_{value}\} \\ C = \{child_0, child_1, \ldots, child_m\} \end{cases} \tag{1}$$

Among the representation style, $node_{type}$ indicates the type information in the node, $node_{value}$ indicates the value information in the node, and $child_m$ indicates the child node of the current node.

3.1.2. Parsing Token Sequence

According to the research on related methods [32], the programming language has the characteristics of natural language. The content of the program can be regarded as natural language in order to extract the features of the text information. The nodes on the AST retained the content information of the code to the greatest extent, that is, the value information of the nodes. Therefore, AST could be traversed and extracted to obtain the token sequence representing the code content.

Algorithm 1 describes a method for parsing token sequences. Its input is AST, and its output is a sequence of tokens.

Algorithm 1 parsing token sequence algorithm

```
Input: AST
Output: token seq
1: Initialize queue = {AST}, seq = {}
2: while queue not empty do
3:     node = queue.pop(0)
4:     seq.append(node.value)
5:     if node is not leaf node then
6:         queue.append(node.children)
7:     end if
8: end while
9: return seq
```

Specifically, AST was first put into the queue, and then the head element of the queue was dequeued. AST was traversed through the breadth-first traversal method, thereby extracting the value information of all of the nodes. The time complexity of the whole algorithm was O(n).

3.1.3. Extracting Code Syntax Information

As the program code has a standardized syntax structure, it is difficult to directly extract the features of syntax information by using the code content. AST is an intermediate form of code representation. Furthermore, the type information of the nodes on AST can preserve the syntax structure information of the code well. Therefore, the syntax information of the code can be extracted on the AST. As AST is a tree structure, the TBCNN

(Tree-based CNN) method [33] was used for information extraction for the tree structure information. The process for extracting the code syntax information is shown as follows:

$$W_{conv,i} = n_i^t * W_{conv}^t + n_i^r * W_{conv}^r + n_i^l * W_{conv}^l \tag{2}$$

where $W_{conv,i}$ is a convolution kernel for the TBCNN model. It is a linear combination of W_{conv}^t, W_{conv}^r, and W_{conv}^l matrices. These three matrices use dynamic pooling [33] to construct them.

Then, the output of the syntax information is:

$$y = \tanh(\textstyle\sum_{i=1}^{n} W_{conv,i} * x_i + b_{conv}) \tag{3}$$

where n is the number of nodes with vector representations $x_1, \ldots, x_n$. y,$b_{conv} \in R^{N_c}$ (N_c is the number of feature detectors).

Specifically, a convolution kernel window was designed. The convolution calculation was performed by sliding on AST to complete the information extraction. The input of the TBCNN module was the type information in the nodes on the AST. The output was the information following the fully connected layer. Finally, the syntax information for the code was extracted.

3.1.4. Extracting Code Text Information

Extracting code text information directly from the code content may incur the problem of not being able to extract key information, because the content of the code file is cumbersome. There was plenty of invalid information, such as comment statements and function description statements. For mapping to the code content, the token sequence had the characteristics of compact information and clear content. Therefore, the extraction information for code text could be completed by using the token sequence.

As a method for text key feature mining, the attention module had the advantage of automatically identifying key features of the text information. So, the attention module has been widely used in natural language processing. Therefore, using the attention module could complete the extraction work of text information in the token sequence. The process for the attention module to extract text information is as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4}$$

where Q is a set of query, K is keys, and V is values.

Specifically, tokens were first mapped to a high-dimensional space. Then key−value pair was created to represent the vector value of the token, which was the input for the attention model. The attention score was obtained by calculating the similarity between the query and the key. After Softmax function processing, the attention module output the vector representation of the code text information.

3.1.5. Combining Code Semantics

The code syntax information and code text information extracted from AST were high-dimensional, so they were abstract. They focused on different points. Code syntax information focused on the structural features of the program, while text information focused on the code content features. In natural language, its structure was composed of the syntax and content of the sentences. It is necessary to combine the syntax structure and the text content to extract the semantics of the sentence. Therefore, similar to extraction for natural language processing, the extraction for the semantic information of the program should combine the code syntax information and code text information.

There are many common ways to combine information, such as concatenation, cross multiplication, and so on. As the weights of code syntax information features and code text information features were unknown, this paper chose cross multiplication as the combination method. Specially, the combination method first mapped the code syntax information and code text information into the same dimension space. Then, the cross-multiplication method was used to process the code syntax information and code text information. Finally, the work for combining the code semantic was completed, where the output was the extracted code semantic.

3.2. Defect Feature Mining

The semantic information of programs cannot be directly applied to software defect prediction. Further defect feature mining is necessary to accurately predict whether the software has defects in terms of semantic information.

As the CNN network has been proven in a wide range of fields, it is very effective for hidden feature mining. Therefore, this paper used the CNN network to mine the key feature information of the software defects in semantic information. The process is shown in Figure 2.

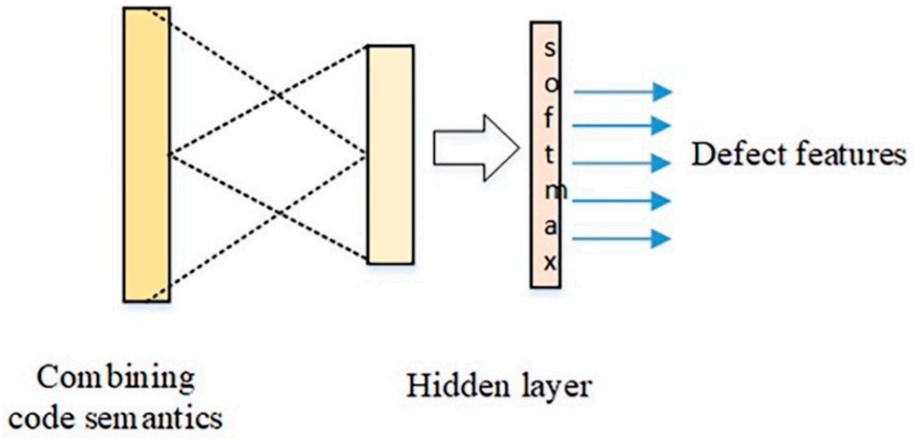

Figure 2. The process for the CNN network to mine a defect feature.

The input layer was semantic information. The hidden layer included convolutional layers, pooling layers, and fully connected layers. The convolutional layer slid in a one-dimensional direction on the semantic information through the randomly initialized convolution kernel, as the following formula shows:

$$y^l = f\left(w^l \otimes x^{l-1} + b^l\right) \tag{5}$$

where y^l is the output for the convolutional layer. w^l is the convolution kernel. x^{l-1} is the output for the last layer. b^l is the input for bias. $\otimes$ is convolution operation. $f(.)$ is activation function.

The pooling layer uses max pooling. After processing for max pooling, the pooled vectors are fully connected, as the following formula shows:

$$y^r = f\left(w^r * y^{r-1} + b^r\right) \tag{6}$$

where y^r is the output for the full connected layer. $f(.)$ is activation function. w^r is the weight matric. y^{r-1} is the output for the last layer. b^r is the input for bias.

After obtaining the numbers of epochs for the iteration, the CNN module finally output important features for defect prediction.

3.3. Performing Defect Prediction

In the field of software defect prediction, the logistic regression algorithm has become an important machine learning method to predict whether there are defects in the software [34]. Therefore, logistic regression was used as the final classifier. This paper put the mining feature obtained by the CNN network into the logistic regression classifier, then obtained the final prediction result. The prediction result was buggy or clean. Buggy means that there was defect in the software. Clean means there was no defect in the software.

4. Experiment

This section presents the related work of the experiment and evaluates the software defect prediction method based on the program semantic feature mining. The effectiveness of this paper's method is evaluated by comparing its accuracy with other state-of-the-art methods in software defect prediction. In this experiment, a device with a Windows 10 operating system, i7-9850H, 2.60 GHz processor, and 32 G memory was chosen to run the method code.

4.1. Research Question

The purpose of the software defect prediction method based on program semantic feature mining was to complete the prediction task when the target project had certain historical labeled data. This method first extracted semantic information from the source code, then mined the key features of the software defects from the semantic information, and finally used the key information to complete the prediction of software defects. In order to verify the effectiveness of the method in this paper (PSFM method), the following two research questions were proposed:

- RQ1: Compared with other deep learning methods, does the method in this paper have a better performance?
- RQ2: Can the semantic information extracted by the method in this paper be directly applied to software defect prediction?

4.2. Experiment Dataset

This paper used the Promise project in the Apache open source project as the experimental dataset. This dataset is accepted by researchers. It is widely used in the field of software defect research. It is written with java code. The projects used in this experiment contained two versions of files: the first version was the training data set and the second version was the test dataset. The dataset details of the Promise project are shown in Table 1.

Table 1. The details in the Promise project dataset.

Project	Description	Version	Total Number of Files	Ratio in Defect Files (%)
Camel	Enterprise integration framework	1.0	339	3.8
		1.2	590	36.6
		1.4	840	17.3
		1.6	926	20.3
Lucene	Text search engine library	2.0	186	48.9
		2.2	234	61.1
		2.4	329	61.4
Xalan	A library for transforming XML files	2.4	667	15.9
		2.5	754	50.3
		2.6	875	47.0
Synapse	Data transport adapters	1.0	16	10.2
		1.1	55	26.8
		1.2	86	33.6
Poi	Java library to access Microsoft format files	2.0	307	11.4
		2.5	380	65.3
		3.0	438	64.2
log4j	Record for logs	1.0	114	28.1
		1.1	100	34.0
		1.2	188	95.7

4.3. Performance Indicators

After the sample was predicted, it produced four kinds of prediction results: the buggy sample was predicted to be buggy, that is, TP (true positive); the clean sample was predicted to be clean, that is, TN (true negative); the clean sample was predicted to be buggy, namely FP (false positive); and the buggy sample was predicted to be clean, FN (false negative). In this paper, the F-measure evaluation index was selected to evaluate the generalization ability of the software defect prediction model. Before defining the F-measure evaluation index, this paper defined the precision index and the recall index.

Precision: The ratio of the number of files correctly classified as being buggy to the number of files classified as buggy.

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \tag{7}$$

Recall: The ratio of the number of files correctly classified as buggy to the number of truly buggy files.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \tag{8}$$

F-measure: The traditional F-measure is the harmonic mean of precision and recall.

$$\text{F-measure} = 2 * \text{Precision} * \text{Recall}/(\text{Precision} + \text{Recall}) \tag{9}$$

The value of the F-measure was between 0 and 1. The higher its value, the better the prediction performance of the model.

4.4. Performance Indicators

To verify the performance of the PSFM method, this paper chose several software defect prediction methods. The process for this comparison firstly chose one version of files to build a classifier for software defect prediction, then picked another version of files in the same project to complete the prediction for the software defect.

This paper used the following software defect prediction method for comparison with the method proposed by this paper (PSFM method):

2T-CNN method [17]: The method of combining the CNN and TBCNN networks. Its performance was better than only using the TBCNN network.

DP-CNN method [9]: The CNN method was combined with manually defined traditional features. Its performance was better than the CNN method alone.

Improved-CNN method [15]: The improved CNN method discarded the features that relied on manual definition. The prediction result was better than the existing CNN method.

DBN method [10]: The first attempt to use deep learning methods for software defect prediction. The effect was better than all of those relying on artificial feature definitions.

Sem method: The semantic information extracted in this paper was directly used for software defect prediction. Compared with the PSFM method, it lacked the software defect feature mining work.

The input of the model used the method of word2vec. Specifically, the type value of the token was mapped to a high-dimensional (30-dimensional in this article) matrix space through the method of word embedding. The output of the model was the software defect prediction result. The defective code was marked as buggy, and the non-defective code was marked as clean.

The results of different software defect prediction methods when conducting experiments are shown in Tables 2 and 3.

Table 2. F-measure comparison of the different methods.

Project	Train Dataset	Test Dataset	2T-CNN	DP-CNN	Improved-CNN	DBN	PSFM
Camel	1.0	1.2	0.396	0.497	0.487	0.346	**0.533**
Lucene	2.2	2.4	0.723	0.756	0.701	0.694	**0.762**
Xalan	2.5	2.6	0.643	0.696	**0.780**	0.540	0.669
Synapse	1.1	1.2	0.624	0.556	**0.655**	0.533	0.503
Poi	2.0	2.5	0.518	0.703	0.444	0.745	**0.790**
log4j	1.0	1.1	**0.618**	0.462	0.400	0.535	0.507
average F-measure	-	-	0.587	0.612	0.579	0.566	**0.627**

Table 3. F-measure comparison of the Sem method and PSFM method.

Project	Train Dataset	Test Dataset	Sem Method	PSFM Method
Camel	1.0	1.2	0.523	**0.533**
	1.4	1.6	0.475	**0.498**
Lucene	2.2	2.4	0.752	**0.762**
	2.0	2.2	0.755	**0.759**
Xalan	2.5	2.6	0.641	**0.669**
	2.4	2.5	0.613	**0.625**
Synapse	1.1	1.2	**0.512**	0.503
	1.0	1.1	0.497	**0.501**
Poi	2.0	2.5	0.741	**0.790**
	2.5	3.0	0.780	**0.782**
log4j	1.0	1.1	**0.511**	0.507
	1.1	1.2	0.496	**0.505**
	-	-	0.608	**0.619**

Table 2 lists the results of predicting the F-measure using different software defect prediction methods. The bold font indicates that the method achieved the highest F-measure value in this project. It can be seen from Table 2 that different code defect predictions had their own advantages.

Table 3 lists the results of predicting the F-measure using the Sem method and PSFM method. It can be seen from Table 3 that most of the F-measure results of the PSFM method were better than the Sem method.

4.5. Analysis

- Compared with other deep learning methods, did the method in this paper have a better performance (RQ1)?

This paper used 2T-CNN, DP-CNN, Improve-CNN, DBN, and four other software defect prediction methods for comparison with the PSFM method proposed in this paper. This comparison was made to illustrate that the software defect results predicted by the PSFM method were a more common phenomenon than the other four methods. Experiments were performed using the item projects in Table 2. Each item project was divided into the training dataset and the testing dataset. The training dataset was used to train the software defect prediction model. The test dataset was used to test the performance of the software defect prediction model. The software defect prediction result was obtained after putting the test dataset into the trained software defect prediction model. Finally, the parameter results after evaluating the software defect prediction model were achieved by comparing the prediction results with the real defects in the test data.

Table 2 lists the F-measure results of the defect prediction using all of the software defect prediction methods. Taking the Lucene item project as an example, 2T-CNN, DP-CNN, Improved-CNN, DBN, and PSFM achieved F-measure values of 0.723, 0.756, 0.701, 0.694, and 0.762, respectively. This shows that using the PSFM method was better than the other four defect prediction methods when performing defect prediction for the Lucene item projects.

When performing defect prediction on all of the items in Table 2, the average value of the F-measure predicted by the PSFM method was higher than that of the other four defect prediction methods. This shows that the defect features mined by PSFM were more universal and more conducive to the discovery of software defects. However, the F-measure value of the software defects predicted by the PSFM method were not higher than that of the other four methods in all of the projects. For example, in the Xalan item project, the F-measure value predicted by the PSFM method was lower than that of the Improved-CNN method. This may be because the features of the software defects mined on the Xalan project using the PSFM method were not sufficient enough.

In summary, the PSFM method proposed in this paper could mine the defect feature on program semantic well when predicting the software defects. Compared with the other four software defect prediction methods, PSFM had the best performance.

- Can the semantic information extracted by the method in this paper be directly applied to software defect prediction (RQ2)?

Compared with the PSFM method, the Sem method lacked the process of mining software defect features on the semantic information. The Sem method directly trained the software defect model with the extracted program semantic information, completing the defect prediction work according to the model. In order to verify the performance of the Sem method, this paper selected different software versions in the same project for software defect prediction. For example, in the Camel item project, version 1.0 was used as the training dataset, while version 1.2 was used as the test dataset. Furthermore, version 1.4 was used as the training dataset, and version 1.6 was used as the test dataset, which verified the performance of the Sem method.

In Table 3, the average value of F-measure predicted by Sem method was lower than that of the PSFM method. This shows that the performance of the PSFM method was better than that of the Sem method. However, the Sem method also had certain advantages. In the software defect prediction of different versions in all of the projects, the F-measure value predicted by the Sem method also reached 0.608. Compared with the PSFM method, the gap value was also within 0.05, which further shows that the software defect results predicted by the Sem method also had certain reference.

In summary, although the performance of the software defect prediction using the Sem method was lower than that of the PSFM method, its prediction results were also useful. This shows that the extracted semantic information could be directly applied to the software defect prediction. However, it is still recommended to use the PSFM method for software defect prediction.

5. Conclusions

This paper proposes a new software defect prediction method based on program semantic feature mining, through extracting the semantics from the code and then mining the features of software defects on the semantics. Finally, software defects are predicted by using the mined defect features. This method can effectively extract the semantic information of the code, and can further mine the impact of the defect features on semantics. However, this method uses AST to perform an intermediate representation of the code. Its semantic information may not be complete enough. In addition, the PSFM method is applicable to Java code and javalang can be used for parsing. In future work, more intermediate representations of code, such as program graphs, will be investigated to improve the performance of software defect prediction. Additional, choosing a non-Java public dataset and more parse tools can be done to verify the PSFM performance.

Author Contributions: W.Y. and M.S. were in charge of conceptualization; W.Y., M.S., X.L. and X.Y. were in charge of methodology; W.Y., M.S. and X.L. were in charge of software; W.Y. was in charge of preparing the original draft; X.L. and X.Y. were in charge of writing the review and editing; M.S. was in charge of funding acquisition; W.Y. and M.S. handled project administration; W.Y., M.S.,

X.L. and X.Y. supervised the study. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (Grant No. 62250410365) and Guangzhou Higher Education Innovation Group 202032854.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: All of the codes and datasets to reproduce our experimental results are open source, at https://github.com/AuroraHuan/SIFD-adversrial-detection, and we hope they facilitate future research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pachouly, J.; Ahirrao, S.; Kotecha, K.; Selvachandran, G.; Abraham, A. A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. *Eng. Appl. Artif. Intell.* **2022**, *111*, 104773. [CrossRef]
2. Fan, G.; Diao, X.; Yu, H.; Yang, K.; Chen, L. Software Defect Prediction via Attention-Based Recurrent Neural Network. *Sci. Program.* **2019**, *2019*, 6230953. [CrossRef]
3. Jin, Z.; Liu, F.; Li, G. Program comprehension: Present and future. *Ruan Jian Xue Bao/J. Softw.* **2019**, *30*, 110–126. (In Chinese)
4. Qiu, S.; Cai, Z.; Lu, L. Cost-sensitive Convolutional Neural Network Module for Software Defect Prediction. *J. Comput. Sci.* **2019**, *46*, 156–160. (In Chinese)
5. Radjenović, D.; Heričko, M.; Torkar, R.; Živkovič, A. Software fault prediction metrics: A systematic literature review. *Inf. Softw. Technol.* **2013**, *55*, 1397–1418. [CrossRef]
6. Wang, T.; Han, L.; Fu, C.; Zou, D.; Liu, M. Static Detection Model and Framework for Software Vulnerability. *J. Comput. Sci.* **2016**, *43*, 80–86+116. (In Chinese)
7. Liu, W.; Jiang, H. Severity Assessment of Software Defect Reports Based on Feature Selection. *J. Comput. Eng.* **2019**, *45*, 80–85. (In Chinese)
8. Li, L.; Shi, K.; Ren, Z. Cross-project defect prediction method based on feature selection and TrAdaboost. *J. Comput. Appl.* **2022**, *42*, 1554–1562. (In Chinese)
9. Li, J.; He, P.; Zhu, J.; Lyu, M.R. Software defect prediction via convolutional neural network. In Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech, 25–29 July 2017.
10. Wang, S.; Liu, T.; Tan, L. Automatically learning semantic features for defect prediction. In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, USA, 14–22 May 2016.
11. Shafiq, M.; Tian, Z.; Sun, Y.; Du, X.; Guizani, M. Selection of effective machine learning algorithm and Bot-IoT attacks traffic identification for internet of things in smart city. *Future Gener. Comput. Syst.* **2020**, *107*, 433–442. [CrossRef]
12. Dam, H.K.; Pham, T.; Ng, S.W.; Tran, T.; Grundy, J.; Ghose, A.; Kim, C.J. A deep tree-based model for software defect prediction. *arXiv* **2018**, arXiv:1802.00921.
13. Qiu, J.; Tian, Z.; Du, C.; Zuo, Q.; Su, S.; Fang, B. A Survey on Access Control in the Age of Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4682–4696. [CrossRef]
14. Tian, Z.; Luo, C.; Qiu, J.; Du, X.; Guizani, M. A Distributed Deep Learning System for Web Attack Detection on Edge Devices. *IEEE Trans. Ind. Inform.* **2019**, *16*, 1963–1971. [CrossRef]
15. Pan, C.; Lu, M.; Xu, B.; Gao, H. An Improved CNN Model for Within-Project Software Defect Prediction. *Appl. Sci.* **2019**, *9*, 2138. [CrossRef]
16. Huang, X.; Fan, G.; Yu, H.; Yang, X. Software Defect Prediction via Heavy Son-based Abstract Syntax Tree. *J. Comput. Eng.* **2021**, *47*, 230–235+248. (In Chinese)
17. Dong, Y.; Li, H.; Wei, X.; Tang, D. Software Defect Prediction based on the Features Fusion of Program Structure and Semantics. *J. Comput. Eng. Appl.* **2022**, *58*, 84–93. (In Chinese)
18. Cheng, X.; Zhang, G.; Wang, H.; Sui, Y. Path-sensitive code embedding via contrastive learning for software vulnerability detection. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual, 18–22 July 2022.
19. Shafiq, M.; Tian, Z.; Bashir, A.K.; Du, X.; Guizani, M. CorrAUC: A Malicious Bot-IoT Traffic Detection Method in IoT Network Using Machine-Learning Techniques. *IEEE Internet Things J.* **2020**, *8*, 3242–3254. [CrossRef]
20. Tian, Z.; Li, M.; Qiu, M.; Sun, Y.; Su, S. Block-DEF: A secure digital evidence framework using blockchain. *Inf. Sci.* **2019**, *491*, 151–165. [CrossRef]
21. Kovačević, Z.; Ravber, M.; Liu, S.-H.; Črepinšek, M. Automatic compiler/interpreter generation from programs for Domain-Specific Languages: Code bloat problem and performance improvement. *J. Comput. Lang.* **2022**, *70*, 101105. [CrossRef]

22. Xie, B.; Li, S.; Li, M.; Liu, C.H.; Huang, G.; Wang, G. SePiCo: Semantic-Guided Pixel Contrast for Domain Adaptive Semantic Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, 1–17. [CrossRef]
23. Yang, D.; Zhu, T.; Wang, S.; Wang, S.; Xiong, Z. LFRSNet: A robust light field semantic segmentation network combining contextual and geometric features. *Front. Environ. Sci.* **2022**, *10*, 1443. [CrossRef]
24. Sheng, H.; Cong, R.; Yang, D.; Chen, R.; Wang, S.; Cui, Z. UrbanLF: A Comprehensive Light Field Dataset for Semantic Segmentation of Urban Scenes. *IEEE Trans. Circuits Syst. Video Technol.* **2022**, *32*, 7880–7893. [CrossRef]
25. Shi, K.; Lu, Y.; Chang, J.; Wei, Z. PathPair2Vec: An AST path pair-based code representation method for defect prediction. *J. Comput. Lang.* **2020**, *59*, 100979. [CrossRef]
26. Khalilian, A.; Baraani-Dastjerdi, A.; Zamani, B. APRSuite: A suite of components and use cases based on categorical decomposition of automatic program repair techniques and tools. *J. Comput. Lang.* **2019**, *57*, 100927. [CrossRef]
27. Fu, W.; Menzies, T. Revisiting unsupervised learning for defect prediction. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, ACM, New York, NY, USA, 4–8 September 2017; pp. 72–83.
28. Yang, Y.; Zhou, Y.; Liu, J.; Zhao, Y.; Lu, H.; Xu, L.; Leung, H. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, WA, USA, 13–18 November 2016; pp. 157–168.
29. Kovačević, Ž.; Mernik, M.; Ravber, M.; Črepinšek, M. 2020: From Grammar Inference to Semantic Inference—An Evolutionary Approach. *Mathematics* **2020**, *8*, 816. [CrossRef]
30. Elbosaty, A.T.; Abdelmoez, W.; Elfakharany, E. Within-Project Defect Prediction Using Improved CNN Model via Extracting the Source Code Features. In Proceedings of the 2022 International Arab Conference on Information Technology (ACIT), Abu Dhabi, United Arab Emirates, 22–24 November 2022.
31. Available online: https://github.com/c2nes/javalang (accessed on 1 July 2014).
32. Hindle, A.; Barr, E.T.; Gabel, M.; Su, Z.; Devanbu, P. On the naturalness of software. *Commun. ACM* **2016**, *59*, 122–131. [CrossRef]
33. Mou, L.; Li, G.; Jin, Z.; Zhang, L.; Wang, T. TBCNN: A tree-based convolutional neural network for programming language processing. *arXiv* **2014**, arXiv:1409.5718.
34. He, Z.; Peters, F.; Menzies, T.; Yang, Y. Learning from open-source projects: An empirical study on defect prediction. In Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, MR, USA, 10–11 October 2013.