



Hybrid Defect Prediction Model Based on Counterfactual Feature Optimization

Wei Zheng^{1,2} · Teng Fei Chen^{1,2} · Mei Ting Hu^{1,2} · Feng Yu Yang^{1,2} · Xin Fan^{1,2} · Peng Xiao^{1,2}

Received: 22 March 2023 / Accepted: 20 June 2023 / Published online: 4 July 2023
© The Author(s) 2023

Abstract

Software defect prediction is critical to ensuring software quality. Researchers have worked on building various defect prediction models to improve the performance of defect prediction. Existing defect prediction models are mainly divided into two categories: models constructed based on artificial statistical features and models constructed based on semantic features. DP-CNN [Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural network. In: 2017 IEEE international conference on software quality, reliability and security (QRS). IEEE, 2017; 318–328.] is one of the best defect prediction models, because it combines both artificial statistical features and semantic features, so its performance is greatly improved compared to traditional defect prediction models. This paper is based on the DP-CNN model and makes the following two improvements: first, using a new Struc2vec network representation technique to mine existing information between software modules, which specializes in learning node representations from structural identity and can further extract structural features associated with defects. Let the DP-CNN model once again incorporate the newly mined structural features. Then, this paper proposes a feature selection method based on counterfactual explanations, which can determine the importance score of each feature by the feature change rate of counterfactual samples. The origin of these feature importance scores is interpretable. Under the guidance of these interpretable feature importance scores, better feature subsets can be obtained and used to optimize artificial statistical features within the DP-CNN model. Based on the above methods, this paper proposes a new hybrid defect prediction model DPS-CNN-STR. Evaluating our model on six open source projects in terms of F1 score in defect prediction. Experimental results show that DPS-CNN-STR improves the state-of-the-art method by an average of 3.3%.

Keywords Software defect prediction · Hybrid defect prediction model · Counterfactual explanations · Network representation technique

1 Introduction

As the software system grows in size, there are more and more defects in the software. The presence of defects may lead to severe economic losses or even endanger people's lives [1–3]. It has been found that nearly four-fifths of the cost is spent on defect repair throughout the development cycle [4–6], and If the above defects can be identified and changed in a timely manner at an early stage of software

development, the cost of fixing them will be significantly reduced. Therefore, related researchers have devoted themselves to building various defect prediction models [7–9], to help developers identify potential defects in the software as early as possible to reduce the losses caused by defects.

Software defect prediction can be divided into intra-project and cross-project based on data sources. Within a project, it is based on historical data from a single project, while cross-projects use historical data from multiple different projects, and most research focuses on the binary classification problem of file defect propensity [10]. In early research work, machine learning methods were widely used to construct defect prediction models. Researchers first design artificial statistical features (code metrics, process metrics) related to defects based on source code, and then obtain labels for each source file based on software history information. Finally, prediction models are constructed

✉ Teng Fei Chen
2656090080@qq.com

¹ School of Software, Nanchang Hangkong University, Nanchang 330063, People's Republic of China

² Software Testing and Evaluation Center, Nanchang Hangkong University, Nanchang 330063, People's Republic of China

based on the obtained features and labels using relevant algorithms. Most existing software defect metrics fall into two main categories: software code metrics and software process metrics [11]. Software code metrics (such as LOC [12], Halstead [13], and McCabe [14]) represent the code program' complexity, and software process metrics (such as CK [15], Martin [16], and MOOD [17]) represent the development process' complexity.

By definition, the process of defect prediction is complex, and in some cases artificial statistical features are not sufficient for defect prediction tasks. Previous research has focused on manually designing defect metrics closely related to the source program. However, manual feature extraction has the problem of low efficiency, and the defect information contained in artificial statistical features is extremely limited. With the widespread application of deep learning [18, 19], it can automatically capture highly complex nonlinear features and has strong feature extraction capabilities. Deep learning techniques are gradually being applied to defect prediction, and satisfactory results have been obtained. Wang et al. [20] proposed a model called DBN, which can learn semantic information existing in software programs end-to-end, and then construct a model with better prediction performance based on the extracted semantic information. Experimental results show that the semantic feature-based model has better prediction performance than existing artificial statistical feature-based models. Li et al. [21] use CNN's local feature extraction capability to automatically extract semantic information from software code, and combine it with artificial statistical features to construct a prediction model called DP-CNN. Experimental results show that the predictive performance of the DP-CNN model is superior to that of the DBN model, which proves that CNN has good feature extraction ability and the effectiveness of feature combination. Xia et al. proposed a deep learning method called DeepFL, which can automatically learn the most effective existing or potential features and be used for precise fault localization [22].

The researchers extracted artificial statistical and semantic features from the code, but these two types of features only reflect the internal information of each code file and lack the structural information that exists between code files, and the single source of features leads to unsatisfactory defect prediction results.

To solve this problem, network representation learning [23] is formally applied in the field of software defect prediction. The software network diagram is first constructed based on the dependencies existing between the software modules, and through representation learning, the graph information is effectively characterized and more important structural features are extracted. Typical methods are DeepWalk [24], LINE [25], Node2Vec [26], and SDNE [27], all based on the proximity similarity hypothesis. In fact, in some scenarios, two vertices that are not nearest

neighbors may also have high similarity, and because of this type of similarity, the above methods are unable to capture it. Therefore, this paper uses a new Struc2vec [28] network representation learning technique, which specifically constructs node sequences from another perspective and focuses more on the structural information of nodes, overcoming the limitations of traditional network representation learning methods. In contrast to the most advanced technologies like as DeepWalk, Node2Vec, and RoIX, Struc2vec [28] exhibits an exceptional ability to capture node structural features and demonstrates superior performance in a variety of classification tasks.

Software defect prediction models are mainly constructed by features, so the selection of features will directly affect the defect prediction results. There are two main types of classical feature selection methods, the filter method and the wrapper method [29–31]. Afzal and Torkar [32] empirically compared eight feature selection methods on five defect datasets in the PROMISE repository, and experimental results showed that the feature selection method can effectively improve the performance of software defect prediction models. Rodriguez et al. [33] conducted a comparative study on three feature selection methods based on the Filter method and two feature selection methods based on the Wrapper method on four defect datasets. Experimental results showed that Wrapper method was overall superior to Filter method. However, the evaluation indicators of feature importance scores in the Filter method (such as variance, frequency, etc.) are largely derived from the prior knowledge of the decision-maker, and this experience is subjective. Besides, the feature importance score generated by Wrapper is determined by the base model. However, the machine learning model is a black box, and the unexplainable model cannot intuitively demonstrate the source of feature importance scores for learners. Therefore, the feature importance score generated by traditional feature selection methods cannot fully explain the contribution rate of features to the model output.

Therefore, we use the generated counterfactual samples for feature selection [34–36]. The general idea is that by minimizing the change in input features to generate different model outputs, a set of different counterfactual samples can be generated for a piece of data in the same algorithm. Intuitively, the feature that changes more frequently to generate counterfactual samples is an important feature. The rate at which a feature is changed in all generated counterfactual samples is used as the importance score for that feature, so we can obtain the corresponding importance scores for all features. These important scores fundamentally explain the necessity of features for model output. We can generate the best performing subset of features guided by interpretable feature importance scores.

In summary, this paper proposes a new hybrid defect prediction model DPS-CNN-STR based on artificial statistical, semantic and structural features. Among them, the artificial statistical features are optimized by the feature selection method based on counterfactual explanations, and the structural features are extracted by Struc2vec. Our main contributions are as follows:

- As far as we know, we are one of the few to consider Struc2vec's method for extracting structural features and applying it to software defect prediction. In addition, we are the first to propose a feature selection method based on counterfactual explanations, and have achieved good results, providing a new idea for the study of feature selection.
- We present a new model called DPS-CNN-STR, whose F1 score is improved by 3.3% on average over the optimal DP-CNN model.

The paper is organized as follows: Sect. 2 presents the related work, while Sect. 3 outlines the procedure for constructing the hybrid defect prediction model. Section 4 details the experimental setup, while Sect. 5 analyses the experimental results. Section 6 provides a brief description of potential threats to the validity of the results. Finally, Sect. 7 provides a summary of the work and outlines directions for future research.

2 Related Work

In this section, we mainly introduce the following: software defect prediction technology, Counterfactual generation framework and Struc2vec's relevant background.

2.1 Software Defect Prediction

Software defect prediction techniques are primarily used to identify potential software defects in a timely manner, and to help testers perform purposeful testing activities [37]. Figure 1 shows the common file-level software defect prediction process in literature [38, 39], which mainly includes the following three steps.

1. First, mark each source file in the project according to the software history warehouse. The defect is marked as buggy and the flawless is marked as clean.
2. Then, by analyzing the software source code or historical data, features related to software defects are extracted from the source files. The most common features are artificial statistical features. The obtained features and labels are trained in various machine learning algorithms [40] (such as LR, SVM, and RF) to construct the classifier.
3. Finally, features are extracted from the files in the project to be predicted. After all the features are extracted, the prediction results of each file can be obtained by putting the features into the defect prediction model.

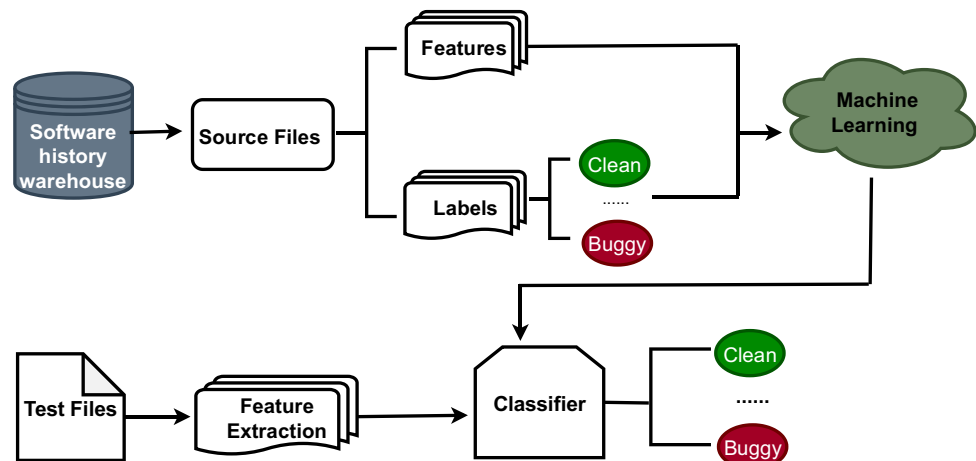
2.2 Counterfactual Generation Framework

Machine learning model is a black box, and people can't explain how it works internally. Due to the unexplainability of models, it can often lead to irreparable consequences. Therefore, it is necessary to provide explanations for machine learning models to reduce the potential threats posed by models.

The most popular explanation is the counterfactual explanation proposed by Wachter et al. [36]:

$$C = \arg \min Yloss(f(c), y) + |x - C| \quad (1)$$

Fig. 1 Software defect prediction process



In short, given the corresponding output of the input features x and model f , the output y of the model can be changed by modifying the features x , but it pursues the change of minimizing the features. Y_{loss} pushes counterfactual C to a prediction different from the original instance x , and the counterfactual sample C should be close to the original sample x .

For example, in the Adult Income dataset [41], if you want to change a low-income person to a high-income person, you can increase the value of working hours. We can imagine that this value must have a critical value. When it is equal to or greater than this value, the result of the model will change, but we only choose the critical value. People can explain the model's decision through the generated counterfactual samples.

More recently, Mothilal et al. [42] have extended Wachter's work. Mothilal et al. argue that the counterfactual samples generated should be diverse, proximal, and feasible.

To ensure the diversity of counterfactuals, Mothilal et al. used the Determinant Point Process (DPP) algorithm. The DPP algorithm ensures that the selection of subsets is diverse and that similar subsets are not easily captured simultaneously. where $\text{dist}(C_i, C_j)$ represents the distance between each of the two counterfactual samples.

$$Diversity = \frac{1}{1 + \text{dist}(C_i, C_j)} \quad (2)$$

Of the multiple counterfactual samples generated for an original piece of data, the counterfactual that is closest to the original data is likely to be the most helpful to the user in making a decision. C_i represents the i th counterfactual sample and where $\text{dist}(C_i, x)$ represents the distance between the C_i and x . Distance is used to determine the proximity of the generated counterfactuals to the original data.

$$Proximity = -\frac{1}{k} \sum_{i=0}^k \text{dist}(C_i, x) \quad (3)$$

Mothilal et al. ensured the feasibility of counterfactuals through domain knowledge between features. For example, gender (discrete features) can only be limited to men and women; the value of working time (continuity features) is limited to a normal interval.

Figure 2 shows multiple counterfactual instances generated for a single piece of original data, and the higher the ranked counterfactual, the closer it is to the original data.

The ratio of feature changes in all counterfactual samples is the feature importance score. Inspired by the above, and to solve the limitations of traditional feature selection methods, we use the generated counterfactual samples for feature selection, which can generate a better subset of features guided by feature importance scores. Details will be provided in Sect. 3.2.

2.3 Struc2vec

Traditional network representation learning algorithms have a limitation. Due to the limited sampling length of the walk, they cannot effectively model nodes with structural similarities that are far apart. However, the reason why previous algorithms perform better is that most datasets prefer the characterization of homogeneity. That is, nodes with similar distances are also similar in feature space, which is enough to cover most data sets. When constructing the graph, Struc2vec neither requires node location information nor label information. Instead, it relies solely on the concept of node degree to build the multilayer graph. An intuitive concept suggests that if two nodes have the similar degree, then the two nodes will be structurally closer together. Moreover, if all adjacent nodes of these two nodes also have the same degree, then the nodes should exhibit even greater structural similarity. In short,

	age	workclass	education	marital_status	occupation	race	gender	hours_per_week	income
Original Data	29	Private	Preschool	Never-married	Blue-Collar	White	Female	38	0
Counterfactual 1	29	Private	Preschool	Never-married	Blue-Collar	White	Female	70	1
Counterfactual 2	29	Private	Preschool	Never-married	White-Collar	White	Female	38	1
Counterfactual 3	29	Local-gov	Preschool	Never-married	Blue-Collar	White	Female	38	1
Counterfactual 4	29	Private	Some-college	Never-married	Blue-Collar	White	Female	38	1
Counterfactual 5	29	Private	9th	Never-married	Blue-Collar	White	Female	50	1

Fig. 2 Counterfactual generation instances

nodes with neighbors with similar node sets are expected to have similar potential representations, and Struc2vec specifically learns node representations from structure identification, and achieves good results.

Figure 3 nodes *m* and *n* have similar local structures, node *m* has degree 4 and node *n* has degree 3, and nodes *m* and *n* are connected to the software network with 3 and 2 triangles respectively. It can be seen that these two nodes have high structural similarity, but because there are no common nodes in their neighborhoods, traditional network representation techniques cannot learn the potential representation of nodes with similar structures, but Struc2vec solves this problem.

In our work, to address the problem of a single feature source and to improve the prediction performance, we can use Struct2vec to extract relevant structural features.

3 Proposed Method

This section describes our proposed hybrid defect prediction model in detail. Our DPS-CNN-STR model is based on artificial statistical, semantic, and structural features. It is based on the DP-CNN [21] and has made the following improvements:

1. The artificial statistical features are optimized by the feature selection method based on counterfactual explanations.
2. Using Struc2vec to learn the structural features of the software network, a new hybrid defect prediction model is jointly constructed based on the optimized DP-CNN model, combined with the newly learned structural features.

3.1 Hybrid Defect Prediction Model

To improve the performance of defect prediction models, Fig. 4 shows a new hybrid defect prediction model. First, artificial statistical features are optimized through counterfactual explanations, then semantic features are learned from the source program using CNN, and finally structural features are learned from the software network using Struc2vec. A hybrid defect prediction called DPS-CNN-STR is constructed.

First, combine the optimized artificial statistical features, semantic features of CNN end-to-end learning, and structural features of Struc2vec Unsupervised Learning, and input them into the Softmax network as a whole to obtain the prediction results. The calculation formula for output layer Softmax is as follows:

Fig. 3 Example of two nodes (*m* and *n*) with similar structure

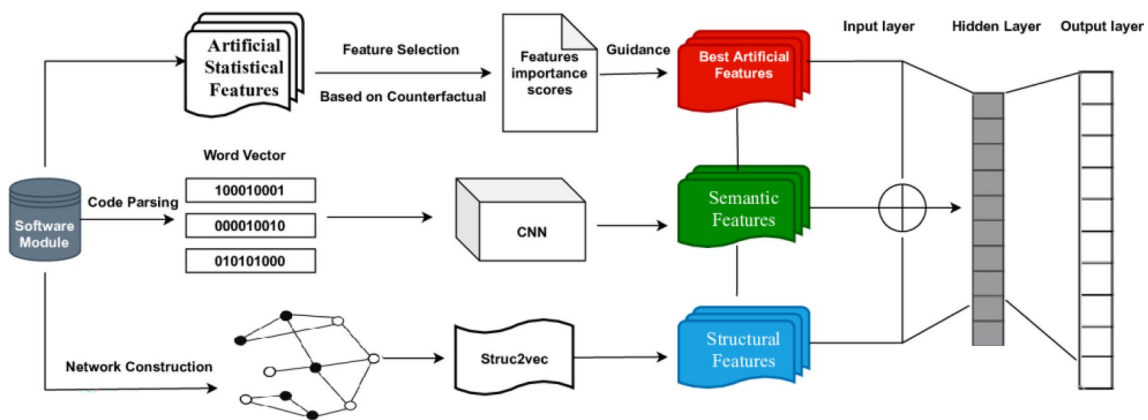
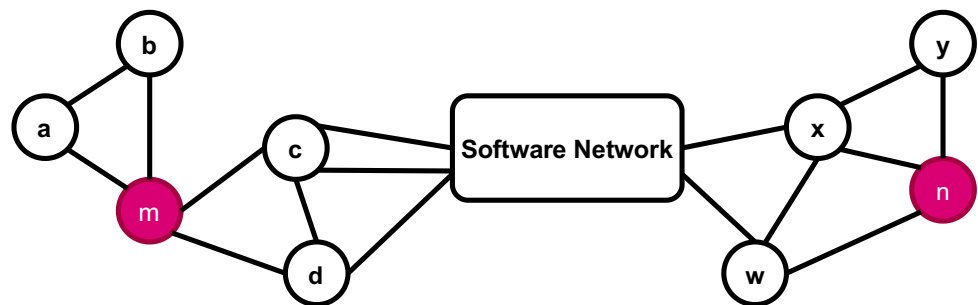


Fig. 4 Hybrid defect prediction model

$$P_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)} \quad i, j \in \{0, 1\} \tag{4}$$

where P_i represents the probability that the module is predicting as a bug. We build the neural network based on Keras, the biggest advantage of which is its simplicity and speed, and we also keep the exact same parameter settings as in the literature [21].

3.2 Feature Selection Based on Counterfactual Explanations

Figure 5 shows a feature selection method based on counterfactual explanations, consisting of the following three main steps:

1. In this paper, defect prediction is based on iterations of versions within the project, so the artificial statistical table of the old version is input into the counterfactual generation framework, and then each feature is given a corresponding importance score.
2. Features are combined one by one according to their scores of importance from the highest to the lowest. Suppose there are n features, then there are n feature subsets corresponding.
3. The old version of the artificial statistical features (the subset of the features described in step 2) and their labels are used to construct the classifier, then the new version of the artificial statistical features and their labels are used as the test set, and the corresponding optimal sub-

set of the features is selected by selecting the highest F1 score.

3.3 Structural Features Extraction

Using the existing dependencies between software modules, we construct the software network with software modules as the basic unit, and then use Struc2vec to learn the potential representation of node structure, in order to extract structural features under unsupervised learning.

First, based on the data flow relationships that exist between the modules, the software network $G=(V, E)$ is constructed, where G represents the constructed software network, $V=\{v_i|i=1,2,3,\dots, n\}$ is the set of nodes in the software network, the element v_i represents each node in the software network, $n=|V|$ is the number of nodes in the constructed software network, and k^* is the diameter. $E=\{e_y | v_i v_j=1, i, j \in [1, n]\}$ represents the set of edges. When the value is 1, it indicates that there is a relationship between node i and node j . When the value is 0, then there is no relationship between them, and the relationships that exist are as follows:

1. There is a dependency between node i and node j ;
2. There is a combination relationship between node i and node j ;
3. There is an inheritance relationship between node i and node j ;

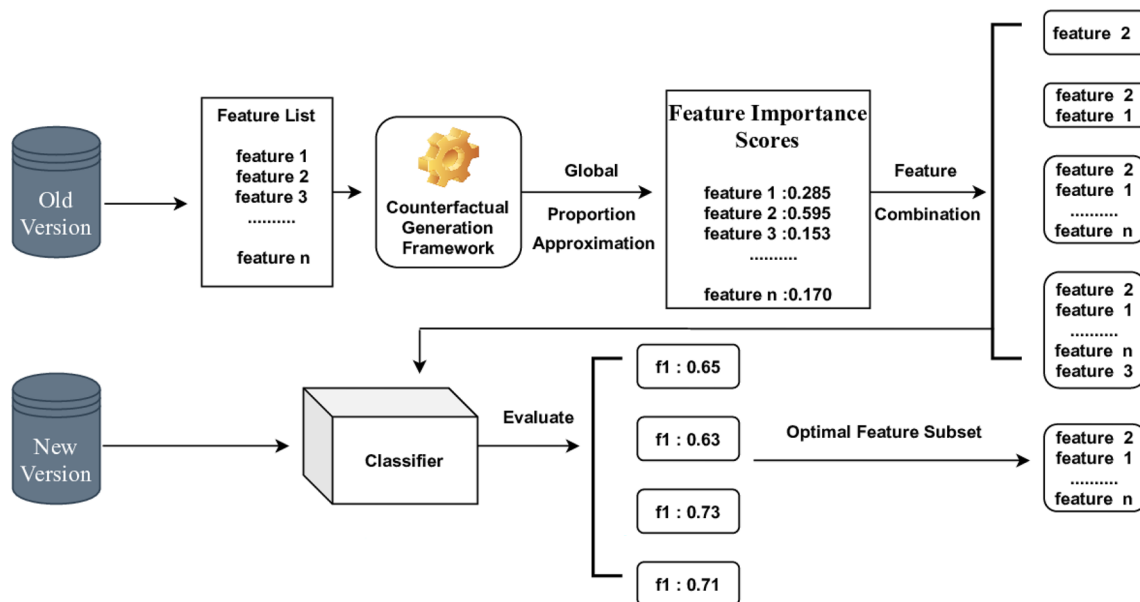


Fig. 5 Feature selection based on counterfactual explanations

Figure 6 shows software networks built by some Apache open source software projects in accordance with the above rules.

Extracting structural features from the software network using Struc2vec is divided into four main steps [28]:

(1) *Measuring structural similarity*: $R_k(u)$ represents the set of nodes whose distance from node u is k , $R_1(u)$ represents the set of directly connected nearest neighbors of u , and $s(S)$ represents the ordered degree sequences of the set S of nodes. The distance $f_k(u, v)$ between all nodes is calculated by introducing a hierarchical structure, and this distance can reflect the situation of structural similarity between nodes, defined as:

$$f_k(u, v) = f_{k-1}(u, v) + g(s(R_k(u)), s(R_k(v))), \quad k \geq 0 \text{ and } |R_k(u)|, |R_k(v)| > 0 \quad (5)$$

The $g(D1, D2) \geq 0$ is a function that measures the distance of the ordered degree sequences $D1, D2$. Since $s(R_k(u))$ and $s(R_k(v))$ have different lengths and may contain duplicate elements. To solve this problem, so a distance calculation formula called DTW is used, defined as follows:

$$d(a, b) = \frac{\max(a, b)}{\min(a, b)} - 1 \quad (6)$$

(2) *Constructing the context graph*: A multilayer weighted graph M is constructed based on the obtained node-pair distances, which is mainly intended to encode the structural similarity between nodes. The edge weight of two nodes in a certain layer k is defined as:

$$w_k(u, v) = e^{-f_k(u, v)}, \quad k = 0, \dots, k^* \quad (7)$$

The same node belonging to different layers is connected by directed edges, and the edge weight is defined as:

$$w(u_k, u_{k+1}) = \log(\Gamma_k(u) + e), \quad k = 0, \dots, k^* - 1 \quad (8)$$

$$w(u_k, u_{k-1}) = 1, \quad k = 1, \dots, k^* \quad (9)$$

$\Gamma_k(u)$ is the number of edges related to node u , and its weight is greater than the average edge weight of the complete graph in layer k , and is defined as:

$$\Gamma_k(u) = \sum_{v \in V} (w_k(u, v) > \overline{w_k}) \quad (10)$$

(3) *Generating context for nodes*: A biased random walk strategy is applied to all nodes in graph M as a way to generate the contextual representation of each node. At each sampling, if the decision is to wander to the current layer, and assuming that it is currently at layer k , the probability of going from node u to node v is:

$$P_k(u, v) = \frac{e^{-f_k(u, v)}}{Z_k(u)} \quad (11)$$

$Z_k(u)$ is the normalization factor of node u in layer k , which is obtained by the following formula:

$$Z_k(u) = \sum_{v \in V, v \neq u} e^{-f_k(u, v)} \quad (12)$$

If it is decided to switch different layers, select $k + 1$ layer or $k - 1$ layer with the following probability:

$$P_k(u_k, u_{k+1}) = \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})} \quad (13)$$

$$P_k(u_k, u_{k-1}) = 1 - P_k(u_k, u_{k+1}) \quad (14)$$

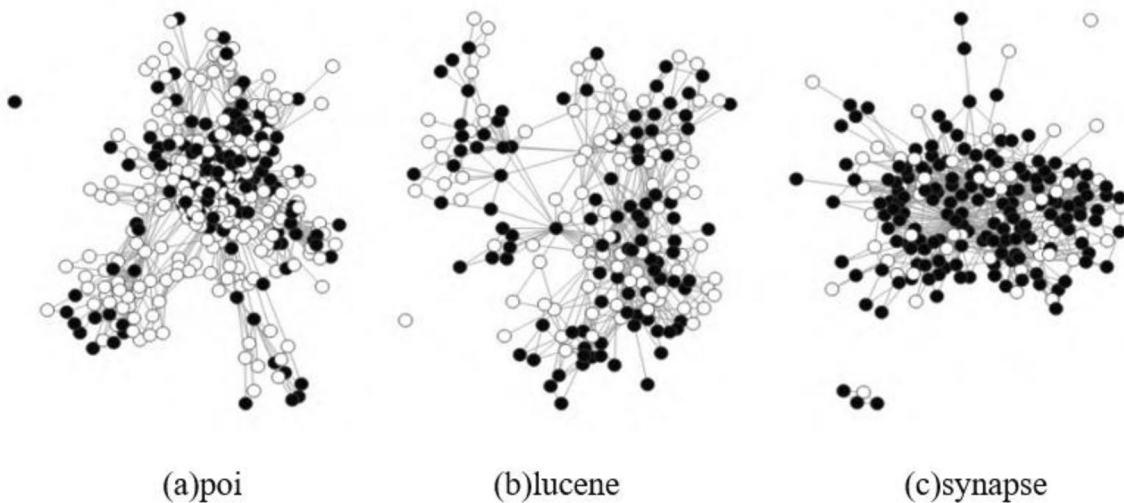


Fig. 6 The software network of poi, lucene and synapse

(4) *Learning a language model*: Finally, using Skip-Gram technique, the potential representation of each node is learned from the generated contextual representation.

3.4 Semantic Features Extraction

The source program is the main cause of software defects. Each source file program code can be parsed into a series of word sequence representations, and the word sequence representations are converted into semantic features by CNN’s efficient feature extraction capability, the process of which is shown in Fig. 7.

(1) Iterate through each file in the source project, the code in each source file is parsed into AST [43] nodes by an open-source Python package named javalang. According to the optimal approach [20], only three main node types [44, 45] are selected as word sequences for software modules. One is the node type of method invocation and class instance creation, adding their specific method names and class names to word sequences; one is the node type of declaration, such as method declaration, interface description, constructor declaration, etc., adding their values to word sequences; and one is the node type of control flow. For example, ForStatement, IfStatement, WhileStatement, etc. are added to word sequences, and some node types are shown in Table 1.

(2) Since the CNN model only receives numeric input, all words in word sequences need to be converted to numeric values. The solution is to encode words in a non-repeating manner starting with a value of 1 and increasing. During this process, make sure that a word can only correspond to one encoded value, and duplicate words refer to the previous encoded value. To distinguish between word encoding and label coding, we decided to encode labels with One-Hot encoding. As a result of the above steps, the source files are parsed into a series of numerical vectors.

(3) Besides, CNN requires input vectors to maintain a consistent length. Yet it is not possible to keep the same

Table 1 Part of the node types

No.	Node type	No.	Node type
1	PackageDeclaration	11	ForStatement
2	ClassDeclaration	12	ContinueStatement
3	InterfaceDeclaration	13	ReturnStatement
4	MethodInvocation	14	SwitchStatement
5	MemberReference	15	BlockStatement
6	ReferenceType	16	TryResource
7	MethodDeclaration	17	CatchClause
8	IfStatement	18	ForControl
9	WhileStatement	19	BasicType
10	DoStatement	20	FormalParameter

length because the code of each source file is different. To solve this problem, we first set the fixed length of the input vectors, and then if the length of the input vectors is greater than the set value, the redundant part will be discarded; otherwise, it is supplemented with 0. In short, the CNN model uses vectors as input and applies various layers such as embedding, convolutional, activation, pooling, and fully connected to extract semantic features.

4 Experimental Setup

In this section, a series of experiments are designed to evaluate the effectiveness of our proposed hybrid defect prediction model.

4.1 Datasets

This paper uses experimental data from six selected open source software projects hosted by the Apache Foundation. The selection is based on criteria such as dataset stability and other relevant factors. Due to the software defect prediction

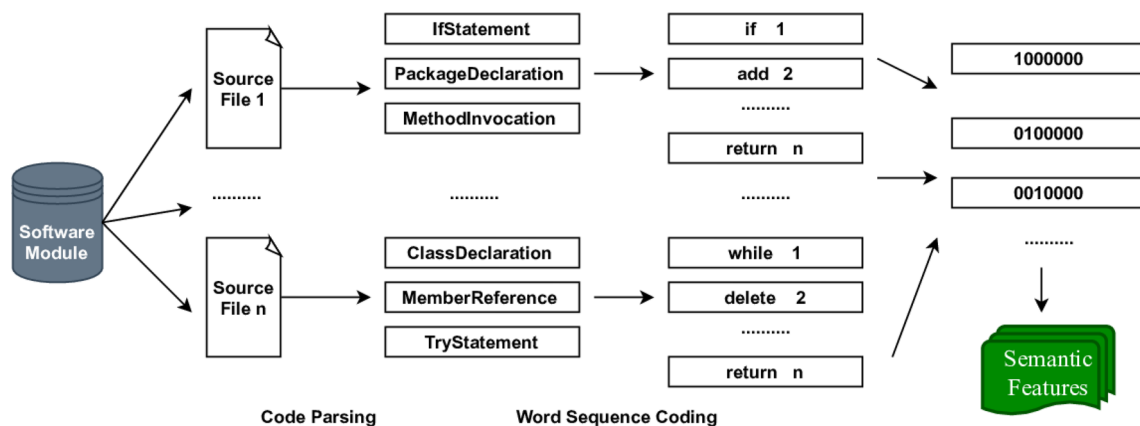


Fig. 7 The semantic features extraction process

based on version iteration in this article, each project has two successive versions, with the old version serving as the data source for model training and the new version serving as the data source for model testing. Table 2 provides detailed descriptions, versions, average file counts, and bug rates for the six projects.

First of all, the number of files in the project we selected varies from 210 to 892, and its purpose is to ensure the diversity of data. Then, we also selected projects with different defect rates to test the performance of our model, with a minimum of 15.5% and a maximum of 64.7%. The first column displays the names of six datasets, the second column gives a brief description of the projects, the third and fourth columns respectively describe the version of the projects and the average number of files, and the last column describes the percentage of defect instances.

In addition, this paper collects a dataset of 20 artificial statistical features and defect statistics for these six projects, with statistical feature data coming from the *tera-PROMISE* project. The dataset contains metrics based on software size and software complexity, and these statistical metrics are shown in Table 3.

4.2 Evaluation Measures

F1 Score [46] is a metric that combines the precision and recall of a classifier and is more suitable for evaluating model performance under imbalanced datasets than other metrics because it weighs well the classifier's performance on positive and negative classes. The precision P is calculated as follows:

$$P = \frac{\text{truepositive}}{\text{truepositive} + \text{falsepositive}} \quad (15)$$

Among them, *truepositive* represents the true number of cases, *falsepositive* represents the false positive number of cases, and *falsenegative* represents the false negative number of cases, and the recall R is calculated as follows:

$$R = \frac{\text{truepositive}}{\text{truepositive} + \text{falsenegative}} \quad (16)$$

The formula for calculating an *F1* score is as follows:

Table 3 The statistical metrics

No.	Abbreviation	No.	Abbreviation
1	WMC	11	CA
2	DIT	12	CE
3	NOC	13	IC
4	CBO	14	CBM
5	RFC	15	CAM
6	LCOM3	16	AMC
7	NPM	17	LCOM
8	DAM	18	LCOM3
9	MOA	19	Avg (CC)
10	MFA	20	MaX (CC)

$$F1 = \frac{2 * P * R}{P + R} \quad (17)$$

For datasets with balanced data distribution, precision and recall can be good measures of model performance, but for classical data imbalance datasets such as software defect prediction, where the prior probability threshold for determining the category is not equal to 0.5, it is better to use the *F1* score.

4.3 Baselines

To prove the superiority of the feature selection method based on counterfactual explanations, we compared it with the featureless selection method and the traditional optimal feature selection method—Wrapper method, where feature selection was performed on 20 artificial statistical features from the six open source projects mentioned in Sect. 4.1.

To prove that our feature selection method and structural features are conducive to improving model prediction performance, the following six models are set up for the model comparison experiment, which are described as follows:

1. *LR*: Construct traditional defect prediction models based on 20 artificial statistical features and using logistic regression algorithms.
2. *CNN*: Use CNN to learn semantic features in the source programs of software projects, and build defect predic-

Table 2 Software project related information

Project	Description	Versions	Avg files	Buggy rate (%)
poi	Can read and write Office format files	2.5, 3.0	409	64.7
lucene	Provides some basic indexing functions	2.0, 2.2	210	55.7
synapse	Open source ESB project	1.1, 1.2	239	30.5
camel	Provides many built-in components and APIs	1.4, 1.6	892	18.6
jedit	A text editor for people who write code	4.0, 4.1	284	23.8
xerces	XML parser	1.2, 1.3	441	15.5

tion models without combining traditional artificial feature indicators.

3. *DP-CNN*: Based on semantic features, a new defect prediction model is constructed by combining 20 traditional artificial feature indicators.
4. *DPS-CNN*: Based on the DP-CNN model, a feature selection method based on counterfactual explanations is used to optimize artificial statistical features.
5. *DP-CNN-STR*: Based on the DP-CNN model, combined with newly excavated structural features.
6. *DPS-CNN-STR*: Combining the above two methods based on the DP-CNN model.

4.4 Research Questions

In studying the construction of prediction models, the main concern is the performance of the model. This paper proposes a new prediction model called DPS-CNN-STR. Because this model is built on the DP-CNN model and additionally undergoes feature selection based on counterfactual explanations and combines the structural features mined by Struc2vec. Therefore, we have three questions:

- *RQ1*: Is the feature selection method based on counterfactual explanations effective?
- *RQ2*: Can Struc2vec's excavated structural features effectively improve the performance of defect prediction models?
- *RQ3*: Is the performance of the mixed defect prediction model DPS-CNN-STR superior to any other model?

4.5 Setup

In RQ1, we compare the feature selection method based on counterfactual explanations with the Wrapper method and the featureless selection method. The featureless selection method does not optimize the features and uses only logistic regression algorithms as a prediction model. The feature selection method based on counterfactual explanations and the Wrapper method require a specific classifier as the carrier, and to ensure the comparability of the experiment, both above two methods also use logistic regression as a specific classifier. Specific ideas are as follows:

- *The feature selection method based on counterfactual explanations (LRBOC)*: first, the old version of the artificial statistical feature table is input into the counterfactual generation framework, and then each feature is given a corresponding importance score, which is returned in the form of a dictionary from top to bottom, e.g. { 'feature1': 0.542, 'feature2': 0.384, ..., 'feature20': 0.112 }.

Then the features are combined one by one according to the scores of importance from the highest to the lowest. Suppose there are 20 features, then there are 20 feature subsets corresponding to the following form: [feature1], [feature1, feature2], ... [feature1, feature2, ..., feature20]. Then, the old version of the artificial statistical features (the feature subset described above) and their labels are used to construct the classifier, and finally the new version of the artificial statistical features and their labels are used as the test set, and the corresponding optimal feature subset is selected by selecting the highest F1 score.

- *The Wrapper method (WRAP)*: This paper selects the RFECV algorithm in the Wrapper method, as the RFECV method is generally consistent with the LRBOC. The RFECV method works by first using the REF (recursive feature elimination) method to derive an important ranking for each feature. Then, based on this ranking, various subsets of different numbers of features are selected for model training and cross-validation. Finally, the best performing feature subset is selected as the final feature set used in the model. There is a parameter named 'n_features_to_select' in the RFECV method, which represents the number of features in the optimal feature subset. First, the number of features in the optimal feature subset of the old version is determined by the learning curve, and afterwards the specific optimal feature subset is determined by the feature score matrix. Suppose the number of features in the optimal feature subset is 8 and the form of the feature score matrix is [1,2,2,3,1,1,2,3,1,2,1,3,1,2,2,3,1,3,2,1], we can select the 8 features corresponding to the value 1 as the optimal feature subset. Then, the best feature subset and labels of the old version are used to build the classifier, while the best feature subset and labels of the new version are used as the test set, and its F1 score is recorded.
- *Featureless selection method (LR)*: Without using any feature selection method, all features and labels of the old version are used to build the classifier, while all features and labels of the new version are used as the test set and its F1 score is recorded.

The F1 scores obtained from the above three methods are compared, and the above comparison results are used to demonstrate the superiority of the feature selection method based on counterfactual explanations.

In RQ2-3, we mainly set up five sets of comparison experiments. More details will be given in the experimental analysis in Sect. 5.2.

5 Experimental Results

In this section, we give experimental results to answer the three questions.

5.1 RQ1: Validity of Feature Selection

5.1.1 Comparative

As shown in Table 4, the performance of our proposed feature selection method LRBOC is much better than the WRAP feature selection method on six datasets.

Table 4 shows that the LRBOC F1 score is on average 2.9% higher than WRAP, and the LRBOC F1 score is on average 5.1% higher than LR. However, WRAP is only 2.2% higher on average than LR. All the above data illustrate the effectiveness of the feature selection method based on counterfactual explanations.

The advantages of the feature selection method proposed in this paper lie in the following two points:

1. Important features are frequently changed to generate counterfactual instances, resulting in higher scores. This provides objective evaluation indicators for determining feature importance scores and addresses the limitations of the Filter method.
2. In all generated counterfactual instances, learners can intuitively observe the difficulty of feature changes and determine the importance score based on the frequency of feature changes, solving the limitations of Wrapper method.

5.1.2 Extension

As described in Sect. 3.2, the feature selection method based on counterfactual explanations consists mainly of three steps. Here, we show the process of selecting the best subset of features for different projects, as shown in Fig. 8.

In Fig. 8, we can clearly see that when there are 20 features, it is equivalent to no feature selection. In the poi

Table 4 F1 scores of different feature selection methods

Project	Version	LR	WRAP	LRBOC
poi	2.5, 3.0	0.745	0.742	0.788
lucene	2.0, 2.2	0.603	0.605	0.609
synapse	1.1, 1.2	0.512	0.554	0.639
camel	1.4, 1.6	0.320	0.354	0.360
jedit	4.0, 4.1	0.578	0.610	0.626
xerces	1.2, 1.3	0.263	0.291	0.307
Avg		0.504	0.526	0.555

The bold number in each row of the table represents the largest value in the row, indicating the best effect

project, when the number of artificial statistical features is 7, the F1 score is the largest. These seven features are the top seven with the highest scores of feature importance, and they are the optimal subset of artificial statistical features for this project.

5.2 RQ2-3: Validity of Structural Features

To answer RQ2 and RQ3, we designed five sets of comparison experiments, and the results are shown in Table 5.

From Table 5, it can be seen that the F1 value of the CNN model is on average 7.1% higher than that of the LR model, proving that automated feature mining methods are more conducive to the construction of defect prediction models compared to manual feature mining methods, and also verifying the feasibility of deep learning in the field of software defect prediction. Then, by comparing the values in the third and second columns of the table, it was found that the F1 value of the DP-CNN model is on average 2.2% higher than that of the CNN model, indicating the effectiveness of feature combination. In addition, the F1 value of the DPS-CNN model is on average 0.9% higher than that of the DP-CNN model, further verifying the effectiveness of the feature selection method based on counterfactual explanations.

To better answer RQ2, this paper designed a comparative experiment between the DP-CNN-STR model and the DP-CNN model. It was found that the F1 average of the former was 2.6% higher than the latter, proving that the software structure features extracted by Struc2Vec are useful for the construction of defect prediction models.

It can be observed that the overall performance of DPS-CNN-STR model is higher than that of DP-CNN-STR. However, the DPS-CNN-STR model is not as good as the DP-CNN-STR model in two projects (lucene and jedit). After analysis, we believe that the reason for this phenomenon is that the DPS-CNN-STR model is composed of artificial statistical features, semantic features, and structural features. However, by comparing the F1 values of the CNN model with the LR model, we know that the influence of artificial statistical features on the model is far less than that of semantic features. In other words, features mined based on deep learning have a greater impact on the defect prediction model. Therefore, although the DPS-CNN-STR model has undergone artificial statistical feature optimization based on the DP-CNN-STR model, it cannot guarantee that the performance of the DPS-CNN-STR model will definitely be better than the DP-CNN-STR model.

Finally, to answer RQ3, the average F1 score of DPS-CNN-STR is 3.3% higher than that of DP-CNN, and the average F1 score of DPS-CNN-STR is also the highest of the six models. The above proves that both the feature selection method and the structural features proposed in this paper can promote the construction of the model.

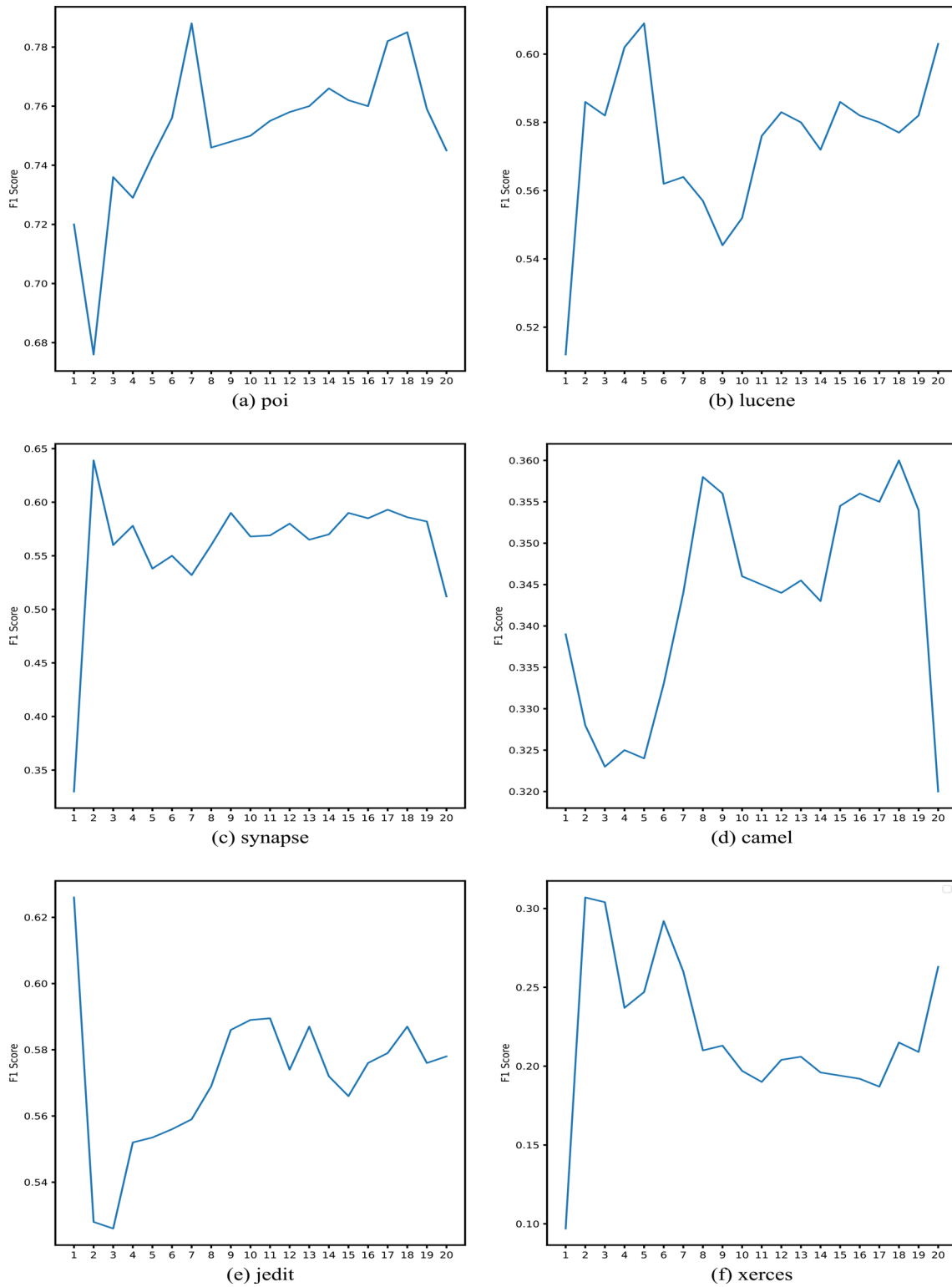


Fig. 8 The process of selecting the best feature subset

Table 5 F1 scores of six defect prediction models

Project	LR	CNN	DP-CNN	DPS-CNN	DP-CNN-STR	DPS-CNN-STR
poi	0.745	0.778	0.784	0.790	0.799	0.806
lucene	0.603	0.750	0.761	0.763	0.768	0.765
synapse	0.512	0.525	0.556	0.572	0.570	0.582
camel	0.320	0.503	0.508	0.500	0.548	0.560
jedit	0.578	0.598	0.600	0.610	0.627	0.620
xerces	0.263	0.298	0.374	0.400	0.426	0.448
Avg	0.504	0.575	0.597	0.606	0.623	0.630

The bold number in each row of the table represents the largest value in the row, indicating the best effect

Table 6 Results of LRBOC under different parameter settings

Project	Version	C=0.01	C=0.1	C=1
poi	2.5, 3.0	0.770	0.758	0.788
lucene	2.0, 2.2	0.608	0.607	0.609
synapse	1.1, 1.2	0.601	0.635	0.639
camel	1.4, 1.6	0.363	0.360	0.360
jedit	4.0, 4.1	0.619	0.621	0.626
xerces	1.2, 1.3	0.339	0.309	0.307
Avg		0.550	0.548	0.555

The bold number in each row of the table represents the largest value in the row, indicating the best effect

6 Threats to Validity

The main internal threats come from the construction of the experimental environment and the setting of parameters. Our experiments are based on the Python language environment. To reduce the uncontrollable factors during the implementation process, we have adopted sufficiently mature third-party libraries, such as calling various Python packages, to achieve the required requirements. In addition, we refer to the default values of the documentation as the parameters for defect prediction, and the parameter setting often directly affects the prediction performance.

For example, in the experiment of feature selection based on counterfactual explanations, there is a parameter called penalty term. In the process of machine learning, because we provide a lot of data for training, many dimensions will be generated during the training, some are decisive, others are irrelevant. In other words, we hope to obtain more accurate results through a large amount of data training, but at the same time, the more judgment dimensions, the worse the generalization ability of our model. This requires us to control a balance, so we introduced the penalty term. In the experiment in Sect. 5.1.1, we choose its default value of 1. However, when the values of penalty terms are different, the results are often different, as shown in Table 6.

In addition, when constructing the DPS-CNN-STR model, different internal parameter settings will also lead to different prediction performance of the model. Therefore,

it is necessary to select the appropriate parameters for the model, so as to make the prediction performance of the model better.

The main external threat stems from the universality of the project. We have only done our model on 6 projects, but these projects cannot summarize all types of software. In addition, we feel it necessary to explain why only 6 datasets were selected. Because the construction of our defect prediction model uses artificial statistical features, semantic features, and structural features, which means that not only the features that need to be manually mined, but also the corresponding source code needs to be provided. However, some source code files in some projects have syntax errors and are difficult to correct, so only 6 projects are selected as our final dataset after careful consideration.

7 Conclusion and Future Work

Researchers have developed various models for software defect prediction to reduce the losses due to defects. This paper present a new model called DPS-CNN-STR, which is built on the basis of the DP-CNN model. The improvements are mainly reflected in the following two aspects: first, we build a software network using the data flow existing between the modules and the new network characterization technique Struc2vec is used to extract the more important structural features, combining these new mined structural features on the DP-CNN model. Then, the feature importance score is determined through the generated counterfactual samples, and the importance score is used as heuristic search information to optimize the artificial statistical features in the model. Because the feature importance score of this method is interpretable, this method can produce a better feature subset than the traditional feature selection method. Finally, our experiments on six public datasets show that our proposed model DPS-CNN-STR based on multi-feature fusion can provide a new idea for the construction of software defect prediction models.

In the future, we will explore the potential of our DPS-CNN-STR for defect prediction on more projects. In addition, we will explore how to combine features more effectively under the premise of given feature importance scores, and how to extract the structural features of software network more effectively, in order to further improve the performance of defect prediction model.

Acknowledgements A preprint has previously been published. The name of the original preprint is [Software defect prediction based on counterfactual explanations]. The link location of the original preprint is at [https://assets.researchsquare.com/files/rs-1870038/v1_covered.pdf?c=1658943649].

Author Contributions All authors are contributed equally.

Funding This paper is supported by the National Natural Science Foundation of China (61867004).

Availability of Data and Materials The counterfactual generation framework is at <https://github.com/microsoft/DiCE>.

Declarations

Conflict of Interest The authors declare they have no conflicts of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Hall T, Beecham S, Bowes D, et al. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans Softw Eng.* 2011;38(6):1276–304.
- Sun X, Peng X, Zhang K, et al. How security bugs are fixed and what can be improved: an empirical study with Mozilla. *Sci China Inf Sci.* 2019;62:1–3.
- Sun X, Yang H, Xia X, et al. Enhancing developer recommendation with supplementary information via mining historical commits. *J Syst Softw.* 2017;134:355–68.
- Wang L, Sun X, Wang J, et al. Construct bug knowledge graph for bug resolution. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE, 2017; 189–191.
- Sun X, Zhou W, Li B, et al. Bug localization for version issues with defect patterns. *IEEE Access.* 2019;7:18811–20.
- Sun X, Peng X, Li B, et al. IPSETFUL: an iterative process of selecting test cases for effective fault localization by exploring concept lattice of program spectra. *Front Comp Sci.* 2016;10:812–31.
- Liu C, Yang D, Xia X, et al. A two-phase transfer learning model for cross-project defect prediction[J]. *Inf Softw Technol.* 2019;107:125–36.
- Shippey T, Bowes D, Hall T. Automatically identifying code features for software defect prediction: using AST N-grams. *Inf Softw Technol.* 2019;106:142–60.
- Li N, Shepperd M, Guo Y. A systematic review of unsupervised learning techniques for software defect prediction. *Inf Softw Technol.* 2020;122: 106287.
- Pachouly J, Ahirrao S, Kotecha K, et al. 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.
- Huda S, Alyahya S, Ali MM, et al. A framework for software defect prediction and metric selection. *IEEE Access.* 2017;6:2844–58.
- Akiyama F. An example of software system debugging. In: *Proc. of the Int'l Federation of Information Proc. Societies Congress.* New York: Springer Science and Business Media, 1971; 353–359.
- Halstead MH. *Elements of software science.* North-Holland: Elsevier; 1977. p. 32–41.
- McCabe TJ. A complexity measure. *IEEE Trans Softw Eng.* 1976;4:308–20.
- Chidamber SR, Kemerer CF. A metrics suite for object oriented design. *IEEE Trans Softw Eng.* 1994;20(6):476–93.
- Tripathi A. An analytical and comparative review of cohesion metrics. In: *Proceedings of the 2018 International Conference on Software Engineering and Information Management.* 2018: 17–25.
- Radjenović D, Heričko M, Torkar R, et al. Software fault prediction metrics: a systematic literature review. *Inf Softw Technol.* 2013;55(8):1397–418.
- Bengio Y, Goodfellow I, Courville A. *Deep learning.* Cambridge: MIT Press; 2017.
- Learning D. *Deep learning. High-dimensional fuzzy clustering,* 2020.
- Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction. In: *Proceedings of the 38th International Conference on Software Engineering.* 2016; 297–308.
- Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural network. In: *2017 IEEE international conference on software quality, reliability and security (QRS).* IEEE, 2017; 318–328.
- Li X, Li W, Zhang Y, et al. Deepfl: integrating multiple fault diagnosis dimensions for deep fault localization. In: *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis.* 2019; 169–180.
- Li B, Pi D. Network representation learning: a systematic literature review. *Neural Comput Appl.* 2020;32(21):16647–79.
- Perozzi B, Al-Rfou R, Skiena S. Deepwalk: online learning of social representations. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2014; 701–710.
- Qiu J, Dong Y, Ma H, et al. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In: *Proceedings of the eleventh ACM international conference on web search and data mining.* 2018; 459–467.
- Grover A, Leskovec J. node2vec: scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* 2016; 855–864.
- Goyal P, Raja S, Huang D, et al. Graph representation ensemble learning[C]//2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, 2020: 24–31.

28. Ribeiro LFR, Saverese PHP, Figueiredo DR. struc2vec: learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. 2017; 385–394.
29. Das H, Naik B, Behera HS. A Jaya algorithm based wrapper method for optimal feature selection in supervised classification. *J King Saud Univ Comput Inform Sci*. 2022;34(6):3851–63.
30. Wah YB, Ibrahim N, Hamid HA, et al. Feature selection methods: case of filter and wrapper approaches for maximising classification accuracy. *Pertanika J Sci Technol*, 2018; 26(1).
31. Jović A, Brkić K, Bogunović N. A review of feature selection methods with applications. In: 2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO). IEEE, 2015; 1200–1205.
32. Afzal W, Torkar R. Towards benchmarking feature subset selection methods for software fault prediction. In: Computational intelligence and quantitative software engineering. Cham: Springer; 2016. p. 33–58.
33. Rodriguez D, Ruiz R, Cuadrado-Gallego J, et al. Attribute selection in software engineering datasets for detecting fault modules. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007). IEEE, 2007; 418–423.
34. Keane MT, Smyth B. Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable AI (XAI). In: Case-based reasoning research and development: 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8–12, 2020, Proceedings 28. Springer International Publishing, 2020; 163–178.
35. Dandl S, Molnar C, Binder M, et al. Multi-objective counterfactual explanations. In: Parallel Problem Solving from Nature—PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5–9, 2020, Proceedings, Part I. Cham: Springer International Publishing, 2020; 448–469.
36. Wachter S, Mittelstadt B, Russell C. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv JL & Tech*. 2017;31:841.
37. Menzies T, Milton Z, Turhan B, et al. Defect prediction from static code features: current results, limitations, new approaches. *Autom Softw Eng*. 2010;17:375–407.
38. Jing XY, Ying S, Zhang Z W, et al. Dictionary learning based software defect prediction. In: Proceedings of the 36th international conference on software engineering. 2014; 414–423.
39. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng*. 2006;33(1):2–13.
40. Sarker IH. Machine learning: Algorithms, real-world applications and research directions. *SN Comput Sci*. 2021;2(3):160.
41. <https://archive.ics.uci.edu/ml/datasets/adult>.
42. Mothilal RK, Sharma A, Tan C. Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 conference on fairness, accountability, and transparency. 2020; 607–617.
43. Xiao Y, Keung J, Bennin KE, et al. Improving bug localization with word embedding and enhanced convolutional neural networks. *Inf Softw Technol*. 2019;105:17–29.
44. Cao S, Sun X, Bo L, et al. Bgnn4vd: constructing bidirectional graph neural-network for vulnerability detection. *Inf Softw Technol*. 2021;136: 106576.
45. Wang T, Su X, Wang Y, et al. Semantic similarity-based grading of student programs[J]. *Inf Softw Technol*. 2007;49(2):99–107.
46. Nam J. Survey on software defect prediction. Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Tech. Rep, 2014.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.