An Integrated Semi-supervised Software Defect Prediction Model

Fanqi Meng[1, 2], Wenying Cheng[1], Jingdong Wang[1]*

[1] School of Computer Science, Northeast Electric Power University, China
[2] School of Computer Science, Guangdong Atv Academy For Performing Arts, China
mengfanqi@neepu.edu.cn, 2202000697@neepu.edu.cn, wangjingdong@neepu.edu.cn

Abstract

A novel semi-supervised software defect prediction model FFeSSTri (Filtered Feature Selecting, Sample and Tri-training) is proposed to address the problem that class imbalance and too many irrelevant or redundant features in labelled samples lower the accuracy of semi-supervised software defect prediction. Its innovation lies in that the construction of FFeSSTri integrates an oversampling technique, a new feature selection method, and a Tri-training algorithm, thus it can effectively improve the accuracy. Firstly, the oversampling technique is applied to expand the class of inadequate samples, thus it solves the unbalanced classification of the labelled samples. Secondly, a new filtered feature selection method based on relevance and redundancy is proposed, which can exclude those irrelevant or redundant features from labelled samples. Finally, the Tri-training algorithm is used to learn the labelled training samples to build the defect prediction model FFeSSTri. The experiments conducted on the NASA software defect prediction dataset show that FFeSSTri outperforms the existing four supervised learning methods and one semi-supervised learning method in terms of F-Measure values and AUC values.

Keywords: Software defect prediction, Semi-supervised learning, Feature selection, Unbalanced classification, Oversampling techniques

1 Introduction

Software defects are generated by programmers during the coding process due to improper management, inadequately understanding of software requirements or lack of development experience [1]. Defective software may produce unexpected results or behaviours after deployment, which can seriously cause substantial economic losses to the company and even threaten people's lives. In addition, the later in the development lifecycle of a software project a defect is detected, the more expensive it is to fix it [2], and the cost of detection and fixing increases significantly, especially after the software is released. As a result, software developers use software quality assurance tools such as software testing or a code review to identify as many defects as possible in software before deployment, but focusing on all program modules would be labour-intensive, so the head of the software quality assurance department wants to be able to identify potentially defective program modules in advance and subsequently allocate sufficient testing resources to them [3].

Software defect prediction techniques can be used to predict whether a software module contains defects based on historical data information about the software, using methods such as machine learning so that sufficient testing resources can be devoted to modules that may contain defects [4]. Currently, many researchers have built up many software defect prediction models with superior performance through supervised machine learning. However, due to various practical reasons, it is often not possible to obtain sufficient marked template information. In such cases where there are limited marked modules and sufficient unmarked modules, software defect prediction models built using supervised machine learning often fail to achieve good prediction results [5].

Academics have started experimenting with semi-supervised methods to predict software defects to address these problems. Some researchers have used semi-supervised machine learning to build software defect prediction models to alleviate the problem of the lack of labelled samples. However, few studies have considered the impact of class imbalance and feature redundancy and irrelevance problems on semi-supervised software models. In software testing, 80% of defects are found in 20% of the code, meaning that the majority of software defects are concentrated in a small number of software modules. As a result, software defect history data is characterised by significant "class imbalance" [5], which can lead to poor learning and inaccurate predictions. In addition, as modern software systems grow in size and complexity, the number of features (software metrics) extracted from software modules becomes much larger than before, and these features may be redundant or irrelevant. This paper, therefore, presents a novel semi-supervised software defect prediction model, FFeSSTri. In this paper, the FFeSSTri algorithm is validated on the NASA dataset, and with a smaller number of marker modules, FFeSSTri can achieve better prediction results than the classical machine learning algorithm and the semi-supervised software defect model Tri_SSDPM.

The rest of the paper is organized as follows: Section 2 reviews related work; Section 3 describes an integrated semi-supervised software defect model, including the general framework, data pre-processing, filtered feature selection based on relevance and redundancy and the Tri-training algorithm; Section 4 tests the validity of the model by

*Corresponding Author: Wenying Cheng; E-mail: 2202000697@neepu.edu.cn
DOI: 10.53106/160792642023112406013

describing the experimental procedure; Section 5 summarizes the work of the paper and describes the focus of the next steps.

2 Related Work

The number of labelled samples in software defect prediction is minimal compared to the number of unlabelled samples. Typical supervised learning methods are challenging to build effective prediction models, and it is costly to obtain many labelled samples.

Several researchers have built software defect prediction models using semi-supervised machine learning in recent years. Liao et al. [6] proposed the sampling-based semi-supervised support vector machine prediction model S4VM+. The model uses a semi-supervised support vector machine method to construct a prediction model based on a small amount of labelled sample data using information from unlabelled data, solving the problem of difficult access to labelled data in defect prediction studies. Jiang et al. [7] proposed a semi-supervised software defect prediction method ROCUS, which uses semi-supervised learning to solve the problem of a small number of labelled samples and under-sampling methods to solve the data imbalance problem, and was empirically demonstrated on eight NASA data. He et al. [8] proposed a semi-supervised random forest algorithm extRF based on self-training, which employs information transformation to improve the accuracy of software defect prediction and can be applied to historical datasets with less defect information and validated on Eclipse. Lu et al. [9] proposed a semi-supervised dimensionality reduction learning method for software defect prediction, which is a self-training variation of the algorithm FTcF, which predicts unlabelled samples in each iteration by repeatedly training using labelled samples, and embeds a preprocessing strategy in the dimensional complexity approach for reducing software metrics. Experimental results show that the prediction effect of semi-supervised learning is significantly better than the random forest algorithm after dimensionality reduction. Ma et al. [10] improved the Tri-training algorithm. Meng et al. [11] proposed a semi-supervised software defect prediction model Tri_SSDPM, which solves the problems of low marker samples and class imbalance in the early stage of software development by using oversampling techniques and semi-supervised learning methods. The problem of low marker samples and class imbalance at the early stage of software development was solved using oversampling techniques and semi-supervised learning. It can be found that the semi-supervised software defect model can make full use of unlabeled samples to improve the performance of classifier prediction in the case of limited labelled samples. However, the impact of feature redundancy and irrelevance on semi-supervised software defect prediction has been seldom considered in the above studies.

Feature selection can eliminate redundant features and irrelevant features to obtain a good subset of features to achieve the effect of dimensionality reduction, making the model more generalizable and effectively improving the classification effect of the classification model [12-14]. The prediction performance of a few feature selection methods was better than that of all feature selection methods. Liu et al. [15] proposed a feature selection method based on cluster analysis based on a filtering approach. In a defect prediction framework, Song et al. [16] considered using wraparound feature selection methods. The results showed that the performance of software defect prediction was better than using all features. Ni et al. [17] proposed the MOFES method, which uses a multi-objective Pareto-based algorithm for feature selection. Gao et al. [18] conducted an experimental comparison of multiple filtered feature selection methods on different subset evaluation strategies.

From the above research, it can be found that the feature selection methods commonly used in software defect prediction are mainly based on the correlation between features and label columns, with little consideration of the similarity between features, resulting in a subset of selected features that are prone to redundancy problems. Therefore, this paper proposes a filtered feature selection method based on relevance and redundancy. The feature selection algorithm considers both the relevance of features and label columns as well as the similarity between features, which is more general than wrapped and embedded feature selection methods, eliminates the training step of classifiers, has low algorithm complexity, and is thus suitable for large-scale data sets and can quickly remove a large number of irrelevant and redundant features. It can quickly remove a large number of irrelevant and redundant features. Finally, a novel semi-supervised software defect prediction model, FFeSSTri, is proposed by applying the feature selection method to a semi-supervised software defect model, which is innovative in that it integrates an oversampling technique, a new feature selection method and a Tri-training algorithm, and is therefore effective in improving the accuracy of defect prediction.

3 Construction of FFeSSTri

3.1 Overall Framework

The overall architecture of the FFeSSTri model is divided into two main phases: the feature selection phase and the model construction phase, and the overall framework is shown in Figure 1. In the feature selection phase, the original labelled feature data is first feature normalised, then the labelled dataset is expanded and sampled using the SOMTE oversampling method to generate a new training dataset, and finally a filtered feature selection method based on relevance and redundancy is proposed to feature select the labelled sample set to obtain the optimal feature subset. In the model building stage, the pre-processed test dataset is feature selected according to the obtained optimal feature subset, a software defect prediction model is constructed using the Tri-training algorithm, and the test module after feature selection is fed into the trained classifier to predict whether the module has defects.

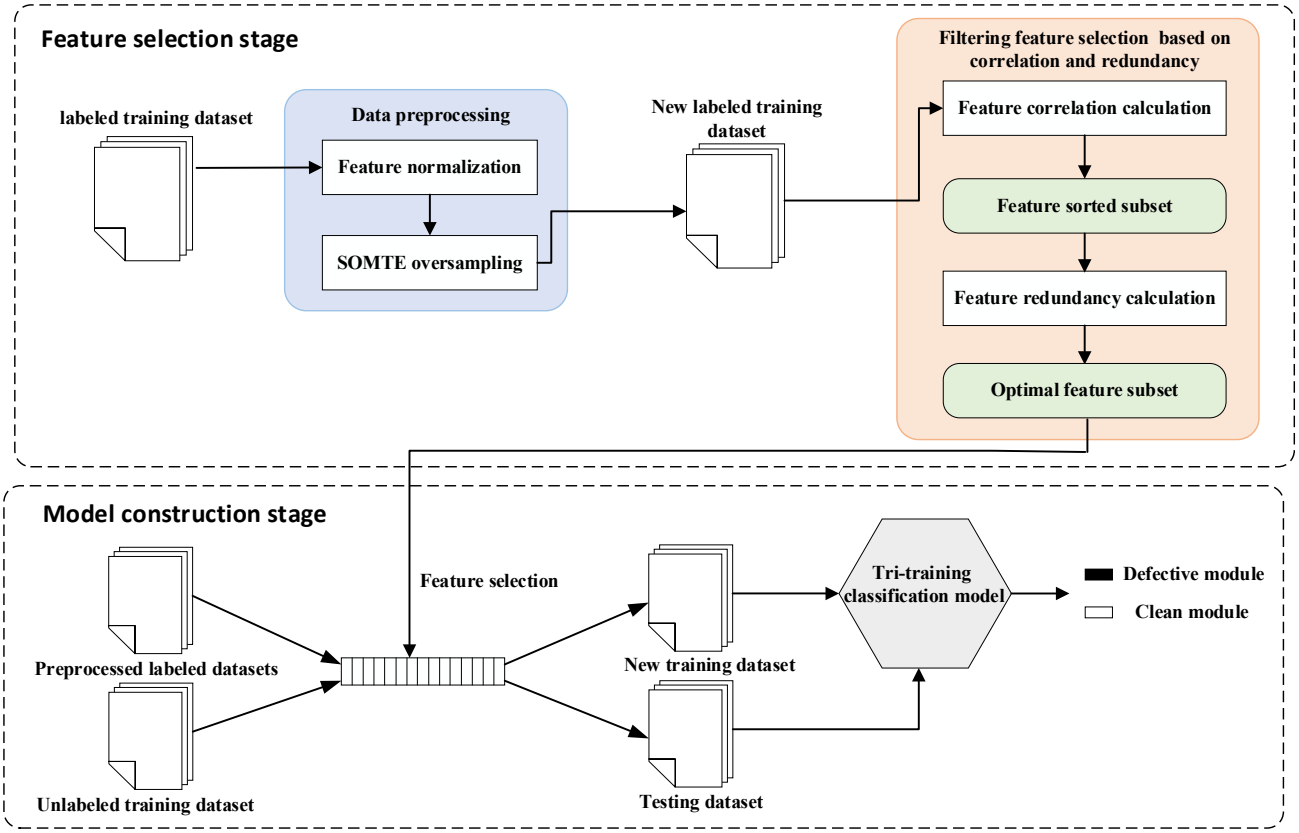

Figure 1. The general architecture of the FFeSSTri model

3. 2 Data Preprocessing

3.2.1 Feature Normalization

Each piece of data in the dataset used in this paper is a set of 21 metric attributes and a label value extracted from a software module. The attributes include the McCabe [19] attribute, the Halstead [20] attribute, the number of lines of code, etc. These attributes objectively characterise the quality features associated with the quality of the software, and the label value indicates whether the module is defective. Table 1 shows the 21 metric attributes used in this paper. These 21 feature measures are used as independent variables in this experiment, and the dependent variable is a binary variable (0 or 1) to indicate whether the code is defective or non-defective. As some feature values are too large or too small, they can affect the classification results of the final model. The data with a relatively large skewness can first be transformed using the $\log 1p$ function to compress the feature data to a certain interval to make it more obedient to the Gaussian distribution, which may lead to a good result for our subsequent classification results.

3.2.2 SOMTE Oversampling

In practice, there are usually fewer defective instances than non-defective ones, known as the class imbalance problem in software defect prediction [21]. If a random division of the dataset or an under-sampling preprocessing approach is used directly, the training dataset will likely contain very little or even no defective data, and it is not easy to train a better prediction model using such data the training set. The basic idea is to generate more samples with fewer labels according to the pattern of samples with fewer labels, thus making the data more balanced and solving insufficient initial samples. A typical oversampling type is the SMOTE (Synthetic Minority Oversampling Technique) algorithm proposed by Chawla [22]. The steps of this algorithm are as follows.

(1) For each sample x in the minority class, calculate its Euclidean distance to all samples in the minority class sample set $S_{\sin}$, to obtain its k-nearest neighbours.

(2) Set a sampling ratio according to the sample imbalance ratio to determine the sampling multiplier N. For each minority class sample x, select some samples at random from its k nearest neighbours, assuming the selected nearest neighbours are x_n.

(3) For each randomly selected nearest neighbor x_n, construct a new sample with the original sample respectively according to the following formula.

$$x_{new} = x + \text{rand}(0,1) * \left| x_n - x \right| . \quad (1)$$

Table 1. Feature metrics

Feature	Description
McCabe's line count of code	It counts the lines of code in module.
McCabe "cyclomatic complexity"	It indicates complexity of the module on basis of number of linearly independent paths.
McCabe "essential complexity"	It indicates the extent to which a flowgraph can be reduced.
McCabe "design complexity"	It indicates cyclomatic complexity of its reduced flowgraph.
Halstead total operators +operands	It gives the count of operators and operands used in the module.
Halstead "volume"	It measures the product of length and log of vocabulary on base.
Halstead "program length"	It indicates the length of the program.
Halstead "difficulty"	It is related to the difficulty of the program to write or understand. Also computed as reciprocal of length.
Halstead "intelligence"	It determines amount of intelligence presented in the module.
Halstead "effort"	It translates into actual coding time.
Halstead	It is a base Halstead measure.
Halstead's time estimator	It evaluates the testing time of C/C++codes.
Halstead's line count	It indicates the numbers of lines in the code.
Halstead's count of lines of comments	It indicates the number of lines of comments.
Halstead's count of blank	It indicates the number of lines of comments.
IOCodeAndComment	It gives the lines of code and comment in the module.
Unique operators	It counts the total number of distinct operators in the module.
Unique operands	It counts the total number of operators in the module.
Total operators	It counts the total number of operands in the module.
Branch count of the flow graph	It gives the count of branches in the flow graph.

3.3 Filtered Feature Selection based on Relevance and Redundancy

3.3.1 Feature Correlation Analysis

The main objective of feature relevance analysis is to calculate the correlation between features and categories and select a subset of features with a high correlation with the categories. Typical feature selection methods include information gain rate, mutual information, and chi-square value. The performance of different feature selection methods varies and can even be said to vary greatly, which stems from their different methodological principles and the different sequences of feature rankings obtained. A typical subset of features from the three commonly used feature selection methods can be selected to enable better prediction of defects. Experiments have shown [23] that the feature subset selected after combining the three methods for feature selection is more accurate and avoids the low performance of the prediction model brought about by one or several feature selection methods incorrectly selecting the feature subset. Therefore, at this stage, three commonly used feature selection methods for calculating relevance are mainly selected for feature ranking.

(1) Information gain rate

The information gain rate method performs feature selection based on how much information a feature brings to the classification, and the more information a feature brings, the more critical it is. Its calculation formula is shown in equation (2).

$$\text{GainRatio}(A) = \frac{\text{Gain}(A)}{\text{Split}\,E(A)}. \tag{2}$$

Where $Gain(A)$ is the information gain of feature after dividing the dataset S, $SplitE(A)$ represents the splitting information of the feature. It can be considered that the feature with the most significant information gain rate is the feature that is most relevant to the category attributes.

The information gain rate introduces splitting information compared to the information gain, and the more points the feature takes, the larger the splitting information value is, which counteracts the effect of the number of points the feature takes on the number of information the feature brings to the classification system.

(2) Mutual information

Mutual information [24] is knowledge belonging to the information theory and represents the relationship between information quantities. It measures the correlation between two variables using the understanding of information entropy theory. Assuming that X and Y are two random variables, the formula for calculating the value of mutual information between X and Y is shown in equation (3).

$$MI(X,Y) = H(X) - H(X \mid Y), \tag{3}$$

where $H(X)$ and $H(X|Y)$ denote the information entropy and conditional entropy, respectively, and a larger mutual information value $MI(f, C)$ between the feature variable f and the class label C indicates that the feature is more relevant to the class and has better differentiation ability.

(3) Chi-square value

A chi-square value is a non-parametric statistical value. The chi-square test is a widely used hypothesis test that is non-parametric and is used to verify whether the category distribution is correlated with the feature taking values. The null hypothesis of this test is that the category distribution is assumed to be uncorrelated with the feature values. The

method calculates the difference between the theoretical inferred value and the actual observed value when the null hypothesis holds, i.e., the chi-square value. Its calculation formula is shown in equation (4).

$$\chi^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{\left(o_{i,j} - E_{i,j}\right)^2}{E_{i,j}}. \quad (4)$$

Where m is the number of different feature values, and n is the number of categories. $o_{i,j}$ is the actual number of samples with feature value i in the sample with category j, and $E_{i,j}$ is the theoretical number of samples with feature value i in the sample with category j. The larger the cardinality value, the more significant the difference between the actual and theoretical values, and the less likely the null hypothesis will hold, i.e., the more significant the correlation between the feature values and the categories.

3.3.2 Feature Redundancy Analysis

In the filtered feature selection method based on relevance and redundancy, the main goal of feature redundancy analysis is to calculate the correlation between features and features. A feature is considered redundant if it is more correlated with another feature. For a non-linear variable like feature metric in software defect prediction, symmetric uncertainty is chosen at this stage to measure feature-to-feature correlation.

Symmetric uncertainty is based on the theory of information entropy. It measures the correlation between two features by measuring the difference in distribution between two features X and Y, making the amount of information shared by two features comparable. Its calculation formula is shown in equation (5).

$$SU(X,Y) = 2\times\frac{IG(X,Y)}{H(X)+H(Y)}. \quad (5)$$

Where $H(X)$ denotes the information entropy of feature X, $H(Y)$ denotes the information entropy of feature Y, and $IG(X,Y)$ means the information gain. The value of $SU(X,Y)$ is in the range of [0,1], and the more significant this value is, the larger the correlation between feature X and feature Y is.

3.3.3 Algorithm Description

In order to improve the classification performance of software defect prediction models, this paper proposes a filtered feature selection method based on relevance and redundancy, which is more general than wrapped and embedded feature selection methods, eliminates the training step of classifiers, has low algorithm complexity, and is thus suitable for large-scale datasets and can quickly remove a large number of irrelevant features. The feature selection method selects the final optimal subset of features after ranking the relevance of features to class labels and removing redundancy between features. The algorithm flow is shown in Algorithm 1. In the feature relevance analysis stage, three different methods, Information Gain Rate (IGR), Mutual Information (MI) and chi-square (CS), are first used to calculate the relevance magnitude between features and classes to obtain three sets of feature sub ghted according to the ranked order. Then each feature will get three weighted values. The sum of the three weighted values for each feature is called the weighted sum. Finally, the features are re-ranked according to the weighted sum of the features, and the top N_1 features are retained to obtain the feature subset S optimized by relevance in this phase. In the feature redundancy analysis phase, the feature subset obtained from the previous phase of relevance analysis is used as the original feature set in this phase. Then, the correlation between two features is calculated using the symmetric uncertainty (SU) method, and the features with a high correlation are clustered into one class. Finally, based on the feature relevance ranking results, the features in each class with high feature-category relevance are selected for retention, the remaining features in the same class are considered redundant features for removal and the top N_2 features are retained to obtain the optimal feature subset S_2. Studies have shown [25] that the proportion of features retained in the final feature selection should preferably be between 20% and 40%. Therefore, the number of features N_2 = *Total number of datasets* * 40. In the previous stage, N_1= 2*N_2 features are retained. The innovation of this algorithm is that it combines the feature ranking sequences obtained by the three methods in the correlation analysis stage and assigns a weight to each feature, which can effectively improve the generalization ability of the prediction model and effectively avoid the instability of a single feature selection method. At the same time, the similarity between features is considered, which can effectively eliminate redundant features.

Algorithm 1. Filtered feature selection algorithm based on relevance and redundancy

Input: Defective dataset $S = \{f_1, f_2, ..., f_m, C\}$ is the category label

Output: Optimal feature subset S_2

1. Sample the dataset S to generate k sets of data.
2. **for** k re-cross-test:
3. find feature correlations using three algorithms, *IGR*, *MI* and *CS*.
4. sort the features in descending order according to the correlation size to obtain three feature subsets S_{IGR}, S_{MI}, S_{CS}.
5. weight the features proportionally according to the order size, and the three weights of feature f_i = (i = 1, 2, ..., m) are W_{fi}^{IGR}, W_{fi}^{MI}, W_{fi}^{CS};
6. calculate the weighted sum of each feature in the three subsets $sum_{fi} = W_{fi}^{IGR} + W_{fi}^{MI} + W_{fi}^{CS}$.
7. sort the features in descending order according to sum_{fi} and keep the first N_1 features to get the feature subset S_1.
8. calculate the correlation between features in S_1 among the features with high correlation, remove the ones with low correlation and keep N_2 features to get the optimal feature subset S_2.
9. **end for**

Algorithm 2. Pseudo code of simplified Tri-training

Input: Training set $L = \{x_i, y_I\}_{i=1}^{l}, U = \{x_i\}_{i=1}^{u}$

Three classifiers $\{h_1, h_2, h_3\}$

Output: Ensemble h using majority vote;

```
for i = 1, 2, 3 do
    Train h_i on L;
end for
while any of {h_1, h_2, h_3} changes do
    for i = 1, 2, 3 do
        L_i = Ø;
        for x ∈ U do
            if h_j(x) = h_k(x)(j, k ≠ i) then
                L_i = L_i ∪ (x, h_j(x));
            end if
        end for
    end for
    for i = 1, 2, 3 do
        Train h_i on L ∪ L_i;
    end for
end while
```

3.4 Tri-training

Tri-training is a compelling semi-supervised integrated learning method that uses a small amount of labelled data to build a good classifier, which is an excellent solution to the problem of insufficient labelled sample data. Prediction is then performed on the unlabelled samples. Suppose two classifiers predict an unlabelled sample as defect-free, and a third classifier predicts a defective sample. In that case, the third classifier should learn this sample as a defect-free sample, and the labelling confidence estimation problem is easily handled by the three classifiers informing each other. The three classifiers are trained for several iterations until the results stabilize, and finally, the three classifiers are integrated for learning using the maximum vote method. However, in some cases, the prediction of the majority-vote classifier may be wrong, which then introduces noise to the minority-vote classifier, but Zhou et al. [26] have shown that under certain conditions, the increase in classification noise rate can be compensated by the number of newly labelled samples. Algorithm 2 shows a simplified pseudo-code of the Tri-training method, more details of which can be found in the literature [26].

4 Experiment and Analysis

In order to verify whether the model in this paper can improve the defect prediction accuracy, this paper analyzes the NASA MDP dataset. It selects three classical machine learning algorithms, NaiveBayes, Decision Tree and RandomForest, an Adaboost integrated learning algorithm, and the semi-supervised method Tri_SSDPM proposed in the literature [11] compared with the FFeSSTri proposed in this paper to calculate the prediction, recall, accuracy, comprehensive evaluation index F-value and AUC-value, respectively. Decision Tree, NaiveBayes, RandomForest and Adaboost learning algorithms were run with the parameter's default values in sklearn.

This experiment was set up as follows, setting the labelling rate R of the experimental dataset to 0.3, that is, 30% of the dataset was randomly selected as the training set of labelled samples, and the remaining 70% of unlabelled samples were used as the auxiliary dataset and test set. With reference to the literature [25], the number of features selected was set to 40% of the original number of features. In order to eliminate chance from the experimental results, the experiment was repeated 20 times for a specific tagging rate R for the method in this paper and the chosen comparison method, and the average of the 20 times was taken as the final result of the experiment.

4.1 Experimental Data

In this paper, the NASA MDP defect dataset in PROMISE library [27] is selected for experiment, including CM1, JM1, KC1 and KC2. Because the defect data in NASA have a vital authenticity that enhances the credibility of the defect datasets, these datasets are widely used in the field of software defect prediction. The properties in the dataset are all software code metrics, including the McCabe [19] loop complexity metric and the Halstead [20] scientific metric. The McCabe Loop Complexity Metric analyses the complexity of the internal structure of a program, assuming that the more loops and selections there are, the more complex the program and the more likely it is to be defective; the Halstead Scientific Metric measures the propensity of software modules to be defective based on operators and operands. The basic information on the dataset is shown in Table 2.

Table 2. Basic information of NASA MDP datasets

Project	Module number	Non- Defects	Defects	Defects rate /%
CM1	498	449	49	9.8
JM1	7782	6020	1762	22.6
KC1	2109	1783	326	15.5
KC2	522	415	107	20.5

4.2 Experimental Evaluation Index

Software defect prediction is a binary problem and for each sample there are two markers, one for the true class of the sample and the other for the predicted class, so we can use the data in the confusion matrix [28] to evaluate the performance of the prediction algorithm. As shown in Figure 2, the possible results are as follows: samples whose true marker is positive are also predicted to be positive, denoted as TP (true positive); samples whose true marker is positive but are predicted to be negative, denoted as FN (false negative); samples whose true marker is negative but are predicted to be positive, denoted as FP (false positive); samples whose the true marker is negative and is also predicted to be negative, denoted as TN (true negative).

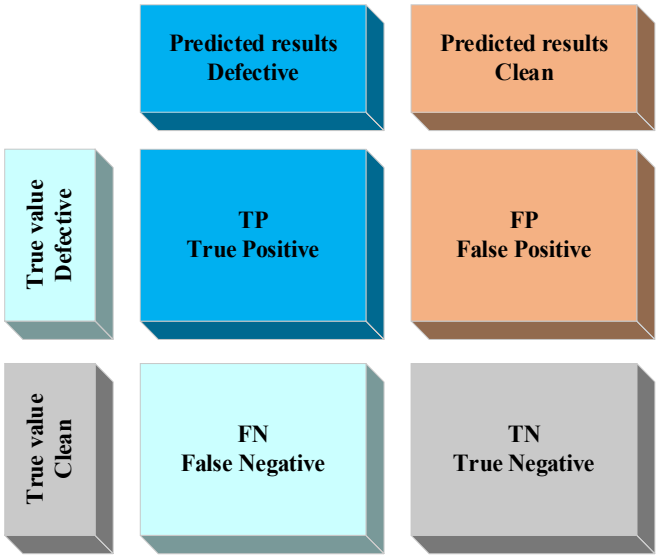

Figure 2. Confusion matrix

In order to effectively evaluate the performance of the classifier, the experimental results were analysed using five evaluation metrics: accuracy, precision, recall, F-measure value and AUC value, which is an intuitive way to evaluate the performance of the classifier and is a measure of how good the defect prediction model is. The evaluation metrics can be expressed as follows.

Accuracy rate is the proportion of the total modules that the model predicts correctly and the accuracy rate is calculated as follows.

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}. \tag{6}$$

Precision rate is the proportion of instances where the true category is defective out of all instances that are predicted to be defective. The formula for calculating this is as follows.

$$Precision = \frac{TP}{TP + FP}. \tag{7}$$

Recall is the proportion of instances correctly predicted to be defective out of all instances whose true category is defective. A good prediction model should have a high recall rate, finding as many defective modules as possible. The recall rate is calculated as follows.

$$Recall = \frac{TP}{TP + FN}. \tag{8}$$

F-measure is a weighted summed average of the accuracy and recall rates, combining the results of the recall and accuracy rates, and is used to evaluate the overall performance of the model.

$$F = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{9}$$

The horizontal coordinate of the ROC curve is the FPR (false positive rate), i.e., the proportion of samples that were actually in the negative category that were misclassified as positive to the total number of samples that were actually in the negative category. The vertical coordinate is the TPR (true positive rate), the proportion of correctly classified positive class samples to the total number of positive class samples.

4.3 Analysis of Results

In this paper, experimental analysis is performed on four sub-datasets CM1, JM1, KC1 and KC2 of the NASA MDP dataset, and the experimental results are as follows:

On the CM1 dataset, As can be seen in Table 3, the FFeSSTri method obtained optimal values for all five indicators compared to the other five methods.

On the KC1 dataset, As can be seen in Table 4, FFeSSTri could not always get the best value compared to the other five methods. But FFeSSTri obtains the best values of Recall, Precision, F-Measure and AUC on the KC1 dataset. Therefore, overall, the prediction accuracy of FFeSSTri is higher than that of the other methods.

On the KC2 dataset, As can be seen in Table 5, the FFeSSTri method could not always obtain optimal values on Precision, but its F-Measure and AUC values were higher than those of the comparison methods. Therefore, the FFeSSTri method still outperforms the other five methods in terms of prediction performance.

On the JM1 dataset, As can be seen in Table 6, the FFeSSTri method obtained optimal values for all five indicators compared to the other five methods.

Table 3. Values of each indicator for each method in dataset CM1

Classifier	Precision	Recall	Accuracy	F-Measure	AUC
NaiveBayes	0.582	0.207	0.693	0.268	0.578
RandomForest	0.181	0.285	0.867	0.222	0.610
DecisionTree	0.182	0.286	0.836	0.222	0.604
Adaboost	0.250	0.214	0.838	0.231	0.579
Tri_SSDPM	0.892	0.803	0.86	0.843	0.839
FFeSSTri	0.916	0.82	0.884	0.865	0.855

Table 4. Values of each indicator for each method in dataset KC1

Classifier	Precision	Recall	Accuracy	F-Measure	AUC
NaiveBayes	0.659	0.331	0.745	0.439	0.632
RandomForest	0.194	0.541	0.856	0.285	0.717
DecisionTree	0.344	0.365	0.806	0.351	0.725
Adaboost	0.247	0.476	0.832	0.325	0.675
Tri_SSDPM	0.842	0.814	0.826	0.828	0.820
FFeSSTri	0.866	0.821	0.846	0.843	0.831

Table 5. Values of each indicator for each method in dataset KC2

Classifier	Precision	Recall	Accuracy	F-Measure	AUC
NaïveBayes	0.786	0.476	0.784	0.590	0.711
RandomForest	0.500	0.608	0.823	0.549	0.738
DecisionTree	0.391	0.529	0.792	0.45	0.702
Adaboost	0.625	0.701	0.801	0.661	0.761
Tri_SSDPM	0.866	0.808	0.843	0.835	0.823
FFeSSTri	0.865	0.833	0.855	0.849	0.846

Table 6. Values of each indicator for each method in dataset JM1

Classifier	Precision	Recall	Accuracy	F-Measure	AUC
NaiveBayes	0.253	0.473	0.796	0.330	0.654
RandomForest	0.203	0.502	0.808	0.289	0.668
DecisionTree	0.065	0.614	0.808	0.118	0.713
Adaboost	0.218	0.535	0.810	0.310	0.682
Tri_SSDPM	0.83	0.801	0.811	0.816	0.810
FFeSSTri	0.838	0.817	0.823	0.827	0.823

Table 7. Average of each classifier over the four data sets

Classifier	Precision	Recall	Accuracy	F-Measure	AUC
NaiveBayes	0.57	0.371	0.754	0.406	0.643
RandomForest	0.269	0.484	0.838	0.336	0.683
DecisionTree	0.245	0.448	0.771	0.285	0.686
Adaboost	0.335	0.482	0.820	0.382	0.674
Tri_SSDPM	0.857	0.806	0.835	0.83	0.823
FFeSSTri	0.871	0.823	0.852	0.846	0.834

The results of each classifier under the four datasets were averaged as shown in Table 7. In order to visualize the classification effect of each classification model, the average value of each metric for each classification model on the four datasets is represented as a bar chart. As can be seen from Figure 3, the FFeSSTri method is higher than the other methods in all evaluation metrics, and combined with Table 7, it can be concluded that the F-Measure and AUC averages are at least 0.44 (0.846-0.406) and 0.148 (0.834-0.686) higher than the classical machine learning algorithms, respectively, and are the same as the F-Measure and AUC averages in this research area. The proposed Tri_SSDPM improved by at least 0.016 (0.846-0.83) and 0.011 (0.834-0.823), respectively.

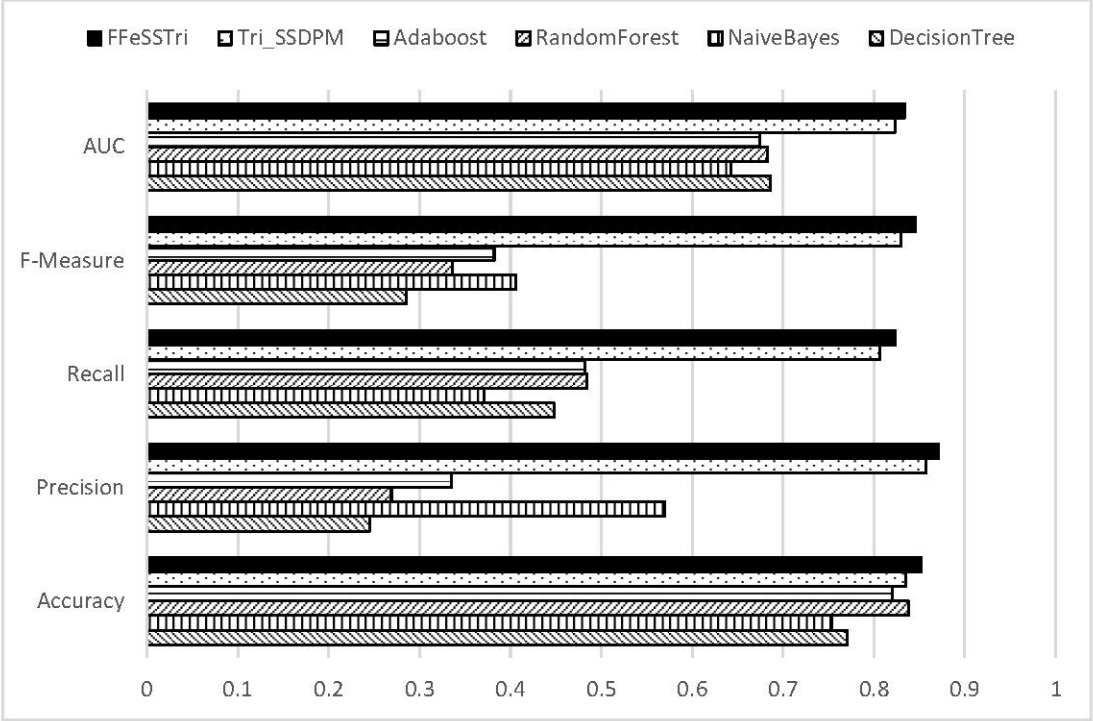

Figure 3. Average of each classifier over the four datasets

The FFeSSTri model achieves higher accuracy than other methods because the DecisionTree, NaiveBayes, and RandomForest algorithms do not consider under-labelled samples, classification imbalance and feature redundancy and irrelevance. The integrated learning method Adaboost, which builds and combines multiple classifiers for training and classification, usually outperforms the performance of a single classifier in generalisation. However, it does not fully use the potential information in unlabelled samples and does not address the problems of classification imbalance and feature irrelevance and redundancy, resulting in poor classification results. The Tri_SSDPM model effectively mitigates the problem of insufficient marker samples and class imbalance by combining the SMOTE oversampling technique with the Tri-training algorithm, but does not take into account the impact of feature redundancy and irrelevance issues on the semi-supervised software defect model. However, the FFeSSTri model proposed in this paper solves the problems of insufficient labeling of training samples, class imbalance, as well as feature redundancy and irrelevance in features, so it achieves higher prediction accuracy in the experimental results.

In summary, a combination of oversampling techniques, filtered feature selection based on correlation and redundancy, and the use of unlabelled samples can improve the prediction accuracy of the model to some extent. FFeSSTri is an effective semi-supervised software defect prediction model and has better prediction accuracy than several other methods.

5 Conclusion

Semi-supervised software defect prediction overcomes the problem of lacking labelled samples but still suffers from class imbalance and excessive irrelevant or redundant features. To this end, this paper proposes an integrated semi-supervised software defect model FFeSSTri. Its innovation lies in that the construction of FFeSSTri integrates an oversampling technique, a new feature selection method, and a Tri-training algorithm, thus it can effectively improve the accuracy. Firstly, the oversampling technique is applied to expand the class of inadequate samples, thus it solves the unbalanced classification of the labelled samples. Secondly, a new filtered feature selection method based on relevance and redundancy is proposed, which can exclude those irrelevant or redundant features from labelled samples. Finally, the Tri-training algorithm is used to learn the labelled training

samples to build the defect prediction model FFeSSTri. The experimental results show that the model obtains better prediction results on NASA datasets.

Considering that the number of features selected affects the prediction results of the software defect prediction model. Therefore, the following work focuses on exploring the effect of the number of features selected on the software defect prediction model and selecting the optimal number of features to improve the software defect prediction.

Acknowledgement

This article is supported by the Science and Technology Development Plan Project of Jilin Province, China (No. 20230101242JC), and the Science and Technology Research Project of the Jilin Provincial Department of Education (JJKH20230133KJ).

References

[1] X. Chen, Q. Gu, W. S. Liu, S. L. Liu, C, Ni, Survey of static software defect prediction, *Ruan Jian Xue Bao/ Journal of Software*, Vol. 27, No. 1, pp. 1-25, January, 2016.

[2] Q. Wang, S.-J. Wu, M.-S. Li, Software defect prediction, *Ruan Jian Xue Bao/Journal of Software*, Vol. 19, No. 7, pp. 1565-1580, July, 2008.

[3] Z. Chen, X. Ju, H. Wang, X. Chen, Hybrid Multiple Deep Learning Models to Boost Blocking Bug Prediction, *Journal of Internet Technology*, Vol. 23, No. 5, pp. 1099-1107, September, 2022.

[4] A. Alsaeedi, M. Z. Khan, Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study, *Ruan Jian Xue Bao/ Journal of Software Engineering and Applications*, Vol. 12, No. 5, pp. 85-100, May, 2019.

[5] X. Zhang, L. M. Wang, Semi-supervised Ensemble Learning Approach for Software Defect Prediction, *Journal of Chinese Computer Systems*, Vol. 39, No. 10, pp. 2138-2145, October, 2018.

[6] S. P. Liao, L. Xu, M. Yan, Software defect prediction using semi-supervised support vector machine with sampling, *Computer Engineering and Applications*, Vol. 53, No. 14, pp. 161-166, July, 2017.

[7] Y. Jiang, M. Li, Z. Zhou, Software defect detection with ROCUS, *Journal of Computer Science & Technology*, Vol. 26, No. 2, pp. 328-342, March, 2011.

[8] Q. He, B. Shen, Y. Chen, Software Defect Prediction Using Semi-Supervised Learning with Change Burst Information, *2016 IEEE 40th Annual Computer Software and Applications Conference*, Atlanta, GA, USA, 2016, pp. 113-122.

[9] H. H. Lu, B. Cukic, M. Culp, Software defect prediction using semi-supervised learning with dimension reduction, *Proc. of the Automated Software Engineering*, Essen, Germany, 2012, pp. 314-317.

[10] Y. Ma, W. Pan, S. Zhu, H. Yin, J. Luo, An Improved Semi-supervised Learning Method for Software Defect Prediction, *Journal of Intelligent & Fuzzy Systems*, Vol. 27, No. 5, pp. 2473-2480, 2014.

[11] F. Meng, W. Cheng, J. Wang, Semi-supervised Software Defect Prediction Model Based on Tri-training, *KSII Transactions on Internet and Information Systems*, Vol. 15, No. 11, pp. 4028-4042, November, 2021.

[12] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, D. Chen, FECAR: A Feature Selection Framework for Software Defect Prediction, *2014 IEEE 38th Annual Computer Software and Applications Conference*, Vasteras, Sweden, 2014, pp. 426-435.

[13] H. D. Tran, L. T. M. Hanh, N. T. Binh, Combining feature selection, feature learning and ensemble learning for software fault prediction, *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, Da Nang, Vietnam, 2019, pp. 1-8.

[14] S. Ghosh, A. Rana, V. Kansal, A Nonlinear Manifold Detection based Model for Software Defect Prediction, *Procedia Computer Science*, Vol. 132, pp. 581-594, 2018.

[15] W. S. Liu, X. Chen, Q. Gu, S. Liu, D. Chen, A cluster-analysis-based feature-selection method for software defect prediction, *SCIENTIA SINICA Informationis*, Vol. 46, No. 9, pp. 1298-1320, September, 2016.

[16] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A General Software Defect-Proneness Prediction Framework, *IEEE Transactions on Software Engineering*, Vol. 37, No. 3, pp. 356-370, May-June, 2011.

[17] C. Ni, X. Chen, F. Wu, Y. Shen, Q. Gu, An empirical study on pareto based multi-objective feature selection for software defect prediction, *Journal of Systems and Software*, Vol. 152, pp. 215-238, June, 2019.

[18] K. Gao, T. M. Khoshgoftaar, H. Wang, N. Seliya, Choosing software metrics for defect prediction: an investigation on feature selection techniques, *Software: Practice and Experience*, Vol. 41, No. 5, pp. 579-606, April, 2011.

[19] T. J. Mccabe, A Complexity Measure, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, pp. 308-320, December, 1976.

[20] R. Bohrer, Review Work: Elements of Software Science. Operating and Programming Systems Series by Maurice H. Halstead, *American Scientist*, Vol. 66, No. 1, p. 100, January-February, 1978.

[21] S. Feng, J. Keung, X. Yu, Y. Xiao, M. Zhang, Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction, *Information and Software Technology*, Vol. 139, Article No. 106662, November, 2021.

[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique, *Journal of artificial intelligence research*, Vol. 16, pp. 321-357, June, 2002.

[23] L. Jiang, S. J. Jiang, Q. Yu, Feature selection method based on sorting integration in software defect prediction, *Journal of Chinese Computer Systems*, Vol. 39, No. 7, pp. 1410-1414, July, 2018.

[24] T. M. Cover, J. A, Thomas, Elements of information theory, *John Wiley & Sons, Inc.*, 2006.

[25] H. L. Lei, X. F. Gao, H. Liu, Mixed feature selection method based on machine learning, *Electronic*

measurement technology, Vol. 41, No. 16, pp. 42-46, December, 2018.
[26] Z. H. Zhou, M. Li, Tri-training: exploiting unlabeled data using three classifiers, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 11, pp. 1529-1541, November, 2005.
[27] T. Menzies, R. Krishna, D, Pryor, The promise repository of empirical software engineering data, *North Carolina State University, Department of Computer Science*, 2016.
[28] L. N. Gong, S. J. Jiang, L. Jiang, Research progress of software defect prediction, *Ruan Jian Xue Bao/Journal of Software*, Vol. 30, No. 10, pp. 3090-3114, October, 2019.

Biographies

Fanqi Meng received the B.E. degree in computer science and technology from Northwest Agriculture and Forest University, Yangling, in 2003 and the M.E. degree in computer application technology from Northeast Electric Power University, Jilin, in 2010 and the Ph.D. degree in computer application technology from Harbin Institute of Technology, Harbin, in 2018.He has been working at the Northeast Electric Power University since 2003. He is currently an Associate Professor in the School of Computer Science. His research interests include software safety, natural language processing, fault diagnosis of electric power equipment and other aspects, involve software engineering, artificial intelligence, data mining and other fields.

Wenying Cheng received the B.S. degree from Binzhou Medical College in 2020. He is currently pursuing a master's degree in the School of Computer Science, Northeastern Electric Power University. His main research interests are software defect prediction and software defect localization.

Jingdong Wang received the B.E. degree and M.E. degree in computer science and technology from Northeast Electric Power University, Jilin and the Ph.D. degree in information science from University of Science and technology of China, in 2017. He has been working at the Northeast Electric Power University since 2008. From 2008 to 2011, he was a Teaching Assistant. From 2011 to 2016, he was a Lecturer. Since 2017, he has been an Associate Professor with the School of Computer Science. His research interests include public security, natural language processing, text mining, knowledge graph and other aspects, involve software engineering, artificial intelligence, emotional analysis and other fields.