Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 6, June 2018, pp. 1263-1274
DOI: 10.23940/ijpe.18.06.p17.12631274

An Information Flow-based Feature Selection Method for Cross-Project Defect Prediction

Yaning Wu, Song Huang*, and Haijin Ji

Research Center of Software Engineering, Army Engineering University of PLA, Nanjing, 210001, China

Abstract

Software defect prediction (SDP) plays a significant part in identifying the most defect-prone modules before software testing and allocating limited testing resources. One of the most commonly used scenarios in SDP is classification. To guarantee the prediction accuracy, the classification models should first be trained appropriately. The training data could be obtained from historical software repositories, which may affect the performance of classification to a large extent. In order to improve the data quality, we propose a novel software feature selection method, which innovatively utilizes the information flows to perform causality analysis in the features of training datasets. More specifically, we conduct causality analysis between each feature metric and the labeled metric bug; then, based on the obtained feature ranking list, we select the top-k features to control redundancy. Finally, we choose the most suitable feature subset based on the F-measure. To demonstrate the effectiveness and practicability of the feature selection method, we select the Nearest Neighbor approach to construct a homogeneous training dataset, and utilize three commonly used classification models to implement comparison experiments. The final experimental results have verified the availability and validity of the feature selection method.

Keywords: cross-project defect prediction; data preprocessing; feature selection; information flow

(Submitted on March 12, 2018; Revised on April 17, 2018; Accepted on May 8, 2018)

1. Introduction

In recent years, software defect prediction (SDP) has attracted the attentions of a growing number of researchers in the field of software engineering [1-3]. It usually focuses on estimating the defect proneness of software modules, and helps software practitioners allocate limited testing resources to the parts that are most likely to contain defects. At present, many prevalent machine learning methods could be used for defect prediction [4-8]. One of the most commonly used methods is classification. The main task of classification is to divide software modules into two categories: defective or non-defective. Thus, the prediction accuracy of a defect prediction model (i.e. defect predictor) is quite important. A specific predictor usually consists of two components: training data and classifier [9]. The training data could be obtained from historical software repositories such as bug reports, change logs, emails of developers and so on [1], which are then used to train the classifiers. Thus, the quality of software training datasets has a great impact on the performance of defect prediction.

Based on the different sources of training data, studies on defect prediction could be divided into two groups in general: within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). WPDP means the predictors are trained from data of historical releases in the same project and used to predict defects in the new releases [9]. However, although there are many public defect datasets on-line such as PROMISE[1], Apache[2] and Eclipse[3], collection of such kind of

1 http://promisedata.org.
2 http://www.apache.org.

* Corresponding author.
E-mail address: hs0317@163.com

historical data is not always possible, especially for new projects. An attemptable method to solve this problem is to utilize public datasets as training data. This way is commonly called CPDP. In other word, CPDP refers to predicting defects in a project using predictors trained from the historical data from different projects [10]. Although CPDP can solve the problem of data deficiency, there are still some other issues to be resolved, especially concerning data quality, including data noise [11], biased datasets [12, 13], high dimensionality [7, 14, 15] and class imbalance [2, 16, 17]. In this paper, our research mainly focuses on the issue of high dimensionality, to improve the quality of training datasets in CPDP.

High dimensionality is usually caused by too many irrelevant and redundant features (also called metrics). One solution to this problem is feature selection, which keeps a small part of the features by removing the unnecessary ones [18]. A great number of feature selection methods have been developed. Meanwhile, comparisons among these available methods have also been made by researchers. Shivaji et al. [15] utilized two different classifiers (Naïve Bayes and SVM) to make an evaluation of five feature selection methods. The experimental results illustrated that all the feature selection methods made a similar effort and the prediction performance depended on the classifiers in a greater extent. Wang et al. [14] evaluated 17 ensembles of 18 feature ranking methods and concluded that the ensembles of a few ranking methods may in some extent improve the prediction performance. He et al. [19] defined a three-setup algorithm to determine a minimum subset of features. The experimental results indicated that the minimum feature subset could not only improve the performance of defect prediction, but also facilitate the procedure of general defect prediction with acceptable loss of prediction precision. Then, He et al. [9] further conducted other experiments on features and found that the top 5 metrics reached a peak in performance.

Although these feature selection methods can improve the accuracy and performance of prediction, they do not fundamentally explore the causal relationship between features and defects. Hence, we innovatively introduce the information flow algorithm (IFA), which is usually applied in network data analysis, to analyze the causality between software defects and the associated features. Information flow (or information transfer) is a logically sound measure of causality [20] and possesses the needed asymmetry for a cause-effect relation. Moreover, it provides a quantitative characterization of the otherwise statistical test [21]. It may be a valuable tool for selecting appropriate metrics in SDP.

Another significant problem of data quality in CPDP is data filtering, which means how to filter cross-company data for local tuning automatically. This issue can be resolved by instance selecting, which aims at selecting an instance subset from the cross-company classes [22, 23]. Turhan et al. [24] applied a simple nearest neighbor (NN) filtering for automatically constructing homogeneous training datasets. In specific terms, they used the Euclidean distance for measuring similarity and selecting neighbors between static code features of within-company (WC) and cross-company (CC) data. However, there are a number of algorithms available for similarity measure between the training datasets and the datasets to be tested. For example, the Euclidean, Jaccard, Cosine, and Correlation [25]. Thus, we need to identify which distance is more appropriate to establish the training datasets in CPDP automatically.

This paper mainly focuses on feature selection. We intend to apply the information flow in feature selection; in terms of data filtering, we use different distances in the NN method to explore more homogeneous datasets in an automated way. The rest of this paper is organized as follows. Section 2 and section 3 describe the approaches of our study and the detailed experimental setups, respectively. Section 4 analyzes the primary results. Finally, section 5 makes a discussion and conclusion about the whole paper and presents the agenda for future work.

2. Problems and approaches

2.1. Problems of software datasets

In cross-project defect prediction, the quality of the training datasets is significant for high prediction performance [11-13]. Many researchers have made great efforts in improving the quality of software datasets; one effective method is data preprocessing, which could be divided into feature selection and data filtering [7, 15].

Feature selection is a process that distinguishes and removes unnecessary features from software datasets, and only keeps the beneficial ones for training the classifiers. Generally, the feature selection methods can be categorized into two groups: the filters and the wrappers. The filters identify and select the most relevant features with the correlation between

3 http://eclipse.org.

features and class labels as a basis, while the wrappers use feedback from a classifier to select feature(s), which may greatly increase the computational complexity when compared with the filters [24].

Training data filtering means a process of collecting similar instances from cross-company (CC) datasets in order to construct a training dataset which is homogeneous with the validation set [24]. Turhan and Menzies [24] used NN filtering to filter cross-company data for local tuning automatically. They chose the Euclidean distance to do similarity measuring between the WC data and the CC data, and found the nearest neighbors for the WC data from the CC data. However, there are many different distances available in NN filtering; it is necessary to identify which distance can achieve the best performance of data filtering.

While both feature selection and data filtering are effective in improving the quality of the training datasets, in this paper, we design a two-stage data preprocessing method, which combine feature selection and data filtering to improve the quality of training datasets. More specifically, for stage of feature selection, we perform causality analysis to select features that have the most causal relationship with the defects; then, we proceed with the redundancy control to form the most suitable feature subset. For the stage of data filtering, we first apply different distances in NN filtering to identify which one could perform the best, and then establish a homogeneous training dataset for the classifiers.

2.2. Research questions

This paper is constructed of three parts: the first part verifies whether information flow algorithm (IFA) can work well in feature selection. The second identifies which distance in NN filtering performs best in data filtering. The last one validates whether the approach that combine feature selection and data filtering can improve the performance of CPDP. Thus, we design experiments to address the following research questions.

- RQ1. How can IFA appropriately implement feature selection to improve the performance of CPDP?
- RQ2. Which distance in NN filtering performs best in constructing homogeneous training datasets?
- RQ3. How does the defect prediction performance of the two-stage method compare with the baseline methods over different data sets?

2.3. A two-stage data preprocessing method

This section presents the two-stage method that combines feature selection and data filtering for CPDP. In particular, for the first stage, we (1) use the information flow algorithm to do the causality analysis for removing the irrelevant features; (2) rank the remained features to eliminate the redundant features on the basis of the occurrences; (3) select a most suitable feature subset. For the second stage, we (1) apply different distances in the NN filtering to find the most appropriate one; (2) construct homogeneous training datasets with the distance. Based on the two stages, a homogeneous and high quality training dataset is constructed.

2.3.1. The entire framework

Figure 1 illustrates the entire framework of the two-stage data preprocessing method.

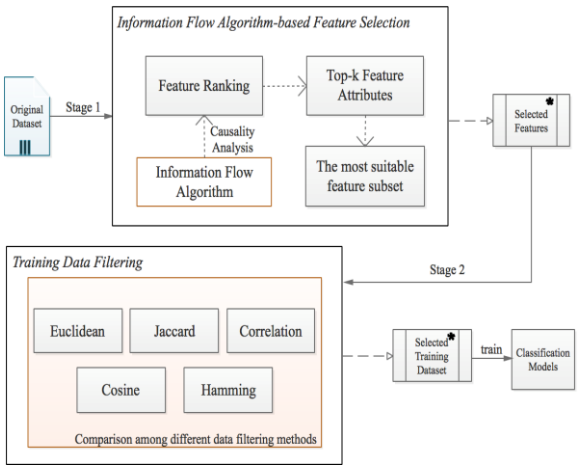

Figure 1. The entire framework of the two-stage method

In our method, feature selection is performed before data filtering. Specifically, the former performs causality analysis, feature ranking and best subset selection in sequence, while the latter applies different distances in NN filtering to measure similarity and select neighbors automatically between static code features of WC and CC data.

2.3.2. The first stage: feature selection

Figure 1 shows the overall process of the feature selection stage. First step is to remove the irrelevant features by performing the causality analysis with the use of information flow algorithm; the second step is to control feature redundancy with the top-k approach. In the last step, a most suitable feature subset is selected with the index of F-measure.

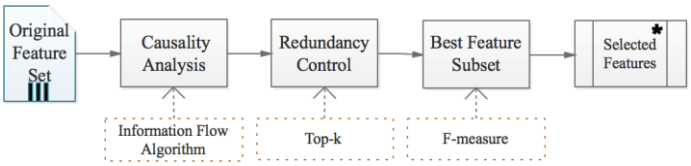

Figure 2. The framework of the first stage

In software development, a software release often includes a quantity of modules (or classes), which are completed in time sequence. Moreover, for a specific release, these modules are labeled with the same static code features. Thus, in the case of a fixed feature, we can take the value list of these modules as a pseudo-time-series (PST). Table 1 shows the details of this process.

Table 1. Brief instruction of a PST

feature / release		WMC	AMC	……	Bug
Release-1.0	module 1	a_1	b_1	……	u_1
	module 2	a_2	b_2	……	u_2
	……	……	……	……	……
	module n	a_n	b_n	……	u_n
		PST_1	PST_2	……	PST_{21}

In Table 1, *Release-1.0* has *n* software modules, and each module is labeled with 20 feature attributes and one labeled attribute *Bug* (more information about the 21 features will be described in section 3). For a fixed feature, the values of the *n* modules form a list, which is the PST. For example, the values for the feature *WMC* of each module compose a value list (a_1, a_2, ..., a_n), which is named as PST_1. Then we can get 21 PSTs.

Liang [26] put forward the information flow algorithm (IFA) to deal with the causal analysis between time series in a rigorous and quantitative way. In IFA, the causality is measured by the information flow. For series X_1 and X_2, the rate of information flowing (units: nats per unit time) from the latter to the former is

$$T_{2\to1} = \frac{C_{11}C_{12}C_{2,d1} - C_{12}^2 C_{1,d1}}{C_{11}^2 C_{22} - C_{11}C_{12}^2} \tag{1}$$

where $C_{ij} = \overline{(X_i - \overline{X_i})(X_j - \overline{X_j})}$ $(i, j = 1,2)$ is the sample covariance between X_i and X_j, and $C_{i,dj} = \overline{(X_i - \overline{X_i})(\dot{X}_j - \overline{\dot{X}_j})}$ is the sample covariance between X_i and $\dot{X}_j$, and $\dot{X}_j$ is the difference approximation of $\frac{dX_j}{dt}$ using the Euler forward scheme: $\dot{X}_{j,n} = \left\{\frac{X_{j,n+1} - X_{j,n}}{\Delta t}\right\}$, with Δt being the time step. For more detailed calculations, please refer to [26].

In the above concise formula, the information flow rate $T_{2\to1}$ could be zero or non-zero. If $T_{2\to1} = 0$, then X_2 is non-causal to X_1; if not, then it is causal. Moreover, if $T_{2\to1} > 0$, it means that X_2 functions to make X_1 more uncertain; if $T_{2\to1} < 0$, it indicates that X_2 tends to stabilize X_1. Thus, we evaluate the causality between two features by the value of $|T_{2\to1}|$. Since the formula involves only the sample covariance of the time series and their derivatives, it is quite simple to compute. In addition, the formula confirms that causation implies correlation, whereas correlation does not necessarily lead to causation.

Let $X_1 = \mathrm{PST}_{21}$ and $X_2 = \mathrm{PST}_l$ $(l = 1, 2, \ldots, 20)$. According to IFA, we can perform causality analysis between each feature attribute and the labeled attribute *Bug* through computing the values of $T_{2\to1}$ with different l, to identify which feature attribute(s) is/are more likely to cause defects.

After the process of causality analysis, we get a ranking list of the 20 feature attributes sorted by the values of $|T_{2\to1}|$ for each release. In order to control redundancy, we need to determine the most useful ones from the original features. Hence, the top-*k* features of the ranking list are selected. As recommended by Song et al. [27], we set *k* as $\lceil \sqrt{m} \times log\, m \rceil$, where *m* is the number of the original feature attributes. In this paper, the number of static code features is 20; thus, *k* is 14. In practice, we first select the top 14 features from each release, and then we choose the top 14 features, according to the occurrences in the entire 34 datasets (more details about the 34 releases will also be described in section 3). The results of this process are shown in Table 2 and 3.

Table 2. Top 14 relevant features for each release

Release (No.)	Top-14 features	Release (No.)	Top-14 features
1	1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 16, 18, 20	18	1, 2, 4, 5, 6, 8, 9, 10, 11, 12, 14, 16, 17, 18
2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15	19	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 16
3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 16, 19, 20	20	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 19, 20
4	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	21	1, 2, 3, 4, 5, 6, 9, 10, 11, 13, 14, 15, 16, 17
5	1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 18, 19, 20	22	1, 2, 3, 4, 5, 8, 9, 11, 12, 13, 15, 18, 19, 20
6	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 16, 17	23	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 20
7	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 17, 18, 20	24	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 17
8	1, 2, 3, 5, 6, 8, 9, 10, 13, 14, 16, 17, 18, 19	25	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 19
9	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 18, 19, 20	26	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 16
10	1, 2, 4, 5, 8, 9, 11, 14, 15, 16, 17, 18, 19, 20	27	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 17
11	1, 2, 3, 4, 5, 6, 9, 11, 12, 15, 16, 17, 18, 19	28	1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 15, 19, 20
12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	29	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 18, 20
13	1, 4, 5, 6, 8, 9, 11, 13, 15, 16, 17, 18, 19, 20	30	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 18
14	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 17	31	1, 2, 4, 6, 7, 8, 10, 11, 12, 15, 17, 18, 19, 20
15	1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 15, 16, 18, 19	32	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 18
16	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20	33	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 17
17	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 18, 19, 20	34	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 16, 17

Table 3. Top 14 features according to occurrences

Feature (No.)	Number of occurrences	Feature (No.)	Number of occurrences
1	**34**	**11**	**25**
5	**33**	**12**	**21**
2	**32**	**16**	**17**
4	**32**	**14**	**16**
6	**31**	18	16
9	**31**	20	15
3	**30**	13	14
8	**30**	15	14
10	**30**	17	14
7	**27**	19	14

In order to further simplify the feature subset, we use the combinations of the 14 features to find the most suitable feature subset for all the data sets. Figure 3 illustrates the procedure of this process.

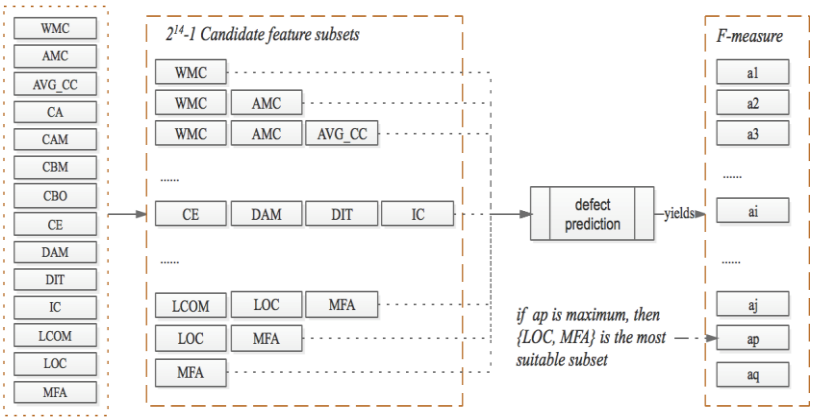

Figure 3. The process of selecting the most suitable feature subset

The most suitable feature subset is the combination, which provides the best prediction performance. For instance, for the given *Release-1.0*, we use the combinations of the 14 features as prediction metrics, i.e. $2^{14}-1$ combinations such as {*WMC*}, {*WMC*, *AMC*}, {*WMC*, *AMC*, *AVG_CC*}, …, {*LOC*, *MFA*}, {*MFA*}. For a specific combination, if it yields the best average prediction result for the entire releases, it is the most suitable feature subset. In this paper, the evaluation of prediction results is based on the *F-measure*. *F-measure* is an integration of *Recall* and *Precision*:

$$\mathrm{F}=\frac{2*\mathrm{Precision}*\mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}} \tag{2}$$

where *Recall* and *Precision* are two accuracy indicators, and

$$\mathrm{Recall}=\frac{TP}{TP+FN} \qquad \mathrm{Precision}=\frac{TP}{TP+FP} \tag{3}$$

In (3), FN (i.e. false negative) means that non-buggy classes are wrongly classified to be faulty, while FP (i.e. false positive) means buggy classes are wrongly classified to be non-faulty. TN (i.e. true negative) and TP (i.e. true positive) refer to correctly classified non-buggy and buggy classes, respectively.

In summary, the total work of the first stage can be explained by the pseudo code, as shown in Figure 4. From the figure we can find that the computation complexity of the feature selection process is $O(n^2)$.

Input:

$T_i(f_1, f_2, \ldots, f_j, \ldots, H(x))$; // $H(x)$ is the class label, i.e., *Bug*.
// $H(x) \epsilon \{X, X^c\}$, where X means defective and X^c means none-defective.
// f_j is the selected feature, i.e., WMC, AMC, etc.
// $i = 1, 2, \ldots, Num$, where Num represents the number of the original datasets.
// $j = 1, 2, \ldots, M$, where M represents the number of the feature attributes, i.e. 20.

Multiple classifiers $C_1, C_2, \ldots, C_M$.

Output:

The most suitable feature subset K.

```
    // initialization
1   constrain feature set S_i = φ, top-k feature set K = φ
    // causality analysis
2   for i = 1 to Num do
3     for each pair of features and the class label (f_j, H(x)) in T_i do
4       compute the value of IF
5     end for
6     get a ranking list L_i of features sorted by T_{f_j→H(x)}
7     S_i = L_i (1:k)
8   end for
    // redundancy control
9   for each feature f_j do
10    for i = 1 to Num do
11      compute the occurrence O_j of f_j according to S_i
12    end for
13  end for
14  get a new ranking list L' of features sorted by O_j
    // best feature subset selection
15  N = L' (1:k)
16  for each combination f_sub of features in N do
17    for j = 1 to M do
18      h_j = C_j(T_i(f_sub, H(x)))
19      compute the value of F-measure F_j according to h_j
20    end for
21    get a performance value P(f_sub) = mean(F_j)
22  end for
23  get a ranking list L'' of each combination sorted by P(f_sub)
24  select the combination of features with best performance as K
```

Figure 4. The pseudo code of the first stage

2.3.3. The second stage: data filtering

Data filtering aims at constructing a training dataset that is homogeneous with the validation set by collecting similar instances. Homogeneous data in software engineering often means the datasets have the same metric set, while heterogeneous data means that their metric sets are different [28]. We apply the k-Nearest Neighbor (k-NN) method to filter the training data, which uses a specific distance to measure similarity between the selected features of validation and CC datasets. The basic idea of the NN filter is that prediction models are built by source instances that are the nearest-neighbors of target instances [24].

The expected result of this stage is to obtain an available CC training data subset that owns similar characteristics to the local data. Please note that, in measuring similarity, we do not use the labeled attribute Bug, in accordance with a real life case that there is no defect related information available for some software modules that are completed and ready for testing. However, the static code features can be obtained with some automated tools.

There are many different distances available in the NN filter such as Euclidean, Jaccard, Cosine and so on. It is necessary to decide which distance to be used in the filter. Thus, we conduct experiments to make comparisons among several common used ones, i.e. Euclidean, Jaccard, Cosine, Correlation and Hamming. For a specific distance, we calculate distances between the local data and the CC data. Let N be the number of local dataset size, and we pick 10 (i.e. k=10) nearest neighbors from candidate CC training dataset for each local data. We get 10×N similar module feature vectors. However, the 10×N samples may not be unique because a single data sample can be a nearest neighbor of many local data. Thus, we only use the unique ones to form the final training set.

3. Experimental setup

In experimental setup, we design experiments to validate the practicality of the two-stage data preprocessing method. First, we illustrate the datasets used in the experiments. Then, based on the research questions, we put forward the experimental design.

3.1. Data collection

In our experiments, the original 34 releases are listed in Table 4, where #Classes and #DP are the number of instances and the number of defects, respectively, and %DP is the ratio of buggy classes to all classes. These 34 releases are chosen from 10 open-source projects provided by the PROMISE repository, and have already been confirmed by many literatures [9, 19].

Table 4. Details of the 34 data sets

No.	Release	#Classes	#DP	%DP	No.	Release	#Classes	#DP	%DP
1	Ant-1.3	125	20	0.160	18	Poi-1.5	237	141	0.595
2	Ant-1.4	178	40	0.225	19	Poi-2.0	314	37	0.118
3	Ant-1.5	293	32	0.109	20	Poi-2.5	385	248	0.644
4	Ant-1.6	351	92	0.262	21	Poi-3.0	442	281	0.636
5	Ant-1.7	745	166	0.223	22	Synapse-1.0	157	16	0.102
6	Camel-1.0	339	13	0.038	23	Synapse-1.1	222	60	0.270
7	Camel-1.2	608	216	0.355	24	Synapse-1.2	256	86	0.336
8	Camel-1.4	872	145	0.166	25	Synapse-1.4	196	147	0.750
9	Camel-1.6	965	188	0.195	26	Synapse-1.5	214	142	0.664
10	Ivy-1.1	111	63	0.568	27	Synapse-1.6	229	78	0.341
11	Ivy-1.4	241	16	0.066	28	Xalan-2.4	723	110	0.152
12	Ivy-2.0	352	40	0.114	29	Xalan-2.5	803	387	0.482
13	Jedit-3.2	272	90	0.331	30	Xalan-2.6	885	411	0.464
14	Jedit-4.0	306	75	0.245	31	Xerces-init	162	77	0.475
15	Lucene-2.0	195	91	0.467	32	Xerces-1.2	440	71	0.161
16	Lucene-2.2	247	144	0.583	33	Xerces-1.3	453	69	0.152
17	Lucene-2.4	340	203	0.597	34	Xerces-1.4	588	437	0.743

The 21 software attributes (20 feature attributes and 1 labelled attribute) are listed in Table 5. More details of these attributes will be found in [29].

Table 5. Descriptions of data attributes

No.	Attribute	Description	No.	Attribute	Description
1	WMC	Weighted Methods per Class	11	IC	Inheritance Coupling
2	AMC	Average Method Complexity	12	LCOM	Lack of Cohesion in Methods
3	AVG_CC	Mean values of methods in the same class	13	LCOM3	Normalized version of LCOM
4	CA	Afferent couplings	14	LOC	Lines Of Code
5	CAM	Cohesion Among Methods of class	15	MAX_CC	Maximum values of methods in the same class
6	CBM	Coupling Between Methods	16	MFA	Measure of Function Abstraction
7	CBO	Coupling Between Object classes	17	MOA	Measure of Aggregation
8	CE	Efferent couplings	18	NOC	Number of Children
9	DAM	Data Access Metric	19	NPM	Number of Public Methods
10	DIT	Depth of Inheritance Tree	20	RFC	Response for a Class
21	Bug	Number of bugs detected in the class			

3.2. Experimental design

3.2.1. Feature selection methods

In the feature selection stage, besides the proposed method, two other common methods are also implemented for comparison. As described above, our proposed method combines both causality analysis and redundancy control to choose

the most representative feature subset while removing the redundant ones. IFA is applied in implementing the causality analysis, while top-k and F-measure are used to control redundancy. In the experiments, as recommended by Song et al. [27], we set the number of pre-selected features as $\lceil \sqrt{m} \times log\, m \rceil$, where m is the number of features in the original data set.

Besides our method, we implement two other methods for comparison. The first one is the ALL method, which means no feature selection is applied. This method is used in [24]. Turhan et al. applied ALL in building localized defect predictors using static code features. The second one is the top-5 method, which means only the top 5 features are selected in accordance with the number of occurrences, as suggested by He et al. [19].

3.2.2. Data filtering methods

For the data filtering stage, we use the *k*-NN method, which measures the similarity between features of validation and CC datasets to build a homogeneous training dataset. To identify which distance performs best in NN, we make comparative experiments among Euclidean, Jaccard, Cosine, Correlation and Hamming. The descriptions of these five distances are as follows.

Given an *m*-by-*n* data matrix X, which is treated as *m* (1-by-*n*) row vectors x_1, x_2, ..., x_m, the various distances between the vector x_s and x_t are defined as follows:

- Euclidean distance

$$d_{st}^2 = (x_s - x_t)(x_s - x_t)' \tag{4}$$

- Jaccard distance

$$d_{st} = \frac{\#[(x_{sj} \neq x_{tj}) \cap ((x_{sj} \neq 0) \cup (x_{tj} \neq 0))]}{\#[(x_{sj} \neq 0) \cup (x_{tj} \neq 0)]} \tag{5}$$

- Cosine distance

$$d_{st} = 1 - \frac{x_s x_t'}{\sqrt{(x_s x_s')(x_t x_t')}} \tag{6}$$

- Correlation distance

$$d_{st} = 1 - \frac{(x_s - \overline{x_s})(x_t - \overline{x_t})'}{\sqrt{(x_s - \overline{x_s})(x_s - \overline{x_s})'}\sqrt{(x_t - \overline{x_t})(x_t - \overline{x_t})'}} \tag{7}$$

where $\overline{x_s} = \frac{1}{n}\sum_j x_{sj}$ and $\overline{x_t} = \frac{1}{n}\sum_j x_{tj}$.

- Hamming distance

$$d_{st} = \frac{\#(x_{sj} \neq x_{tj})}{n} \tag{8}$$

3.2.3. Classification models

During experiments, we implement three classification models (Naïve Bayes (NB), Logistic Regression (LR) and Random Tree (RT)) that are commonly used in software fault prediction.

NB is one of the simplest classifier based on conditional probability [30]. In the classifier, the features are assumed to be independent, and that is why the classifier is termed as 'naïve'. In practice, the Naïve Bayes classifier often performs better than more sophisticated classifiers, although the independence assumption is often violated [31]. The prediction model constructed by this classifier is a set of probabilities. The probability that a new class is buggy is estimated based on the product of the individual conditional probabilities for the feature values of the class.

LR is a type of probabilistic statistical regression model for categorical prediction by fitting data to a logistic curve [32]. It could also be utilized in a binary predictor to predict a binary response, i.e., the outcome of a categorical dependent

variable. In CPDP, the dependent variable (either buggy or non-buggy) is binary; therefore, it is suitable for solving the problem.

RT, a hypothesis space for supervised learning algorithm, is one of the simplest hypothesis spaces possible [33]. A Random Tree has two components: a schema, which is a set of features, and a body, which is a set of labeled instances.

4. Experimental results

Based on the experimental results, we study the three research questions.

4.1. RQ1: How can IFA appropriately implement features selection to improve the performance of CPDP?

To investigate the effectiveness of the IFA in feature selection, we carry out causality analysis experiments between each of the 20 feature attributes and the labeled attribute *Bug*. As recommended by Song et al.[27], we pick the top 14 most relevant features for further processing. After the procedure of redundancy control, we get the most suitable feature subset {*WMC*, *CA*, *CAM*, *CBM*, *CBO*, *CE*, *IC*}, which reaches the highest *F-measure* when compared to other combinations. Then, on the basis of the 34 releases, we conduct comparison experiments with other feature selection methods: the ALL and the Top-5 methods. Table 6 shows the results of the experiments.

Table 6. Performances of different feature selection methods (wins in bold)

Release No.	Consist ALL			TOP 5			IFA		
	NB	LR	RT	NB	LR	RT	NB	LR	RT
1	0.49	**0.48**	0.39	0.56	0.21	**0.41**	**0.60**	0.43	0.37
2	0.28	0.41	0.24	0.45	0.38	**0.32**	**0.48**	**0.43**	0.31
3	0.36	0.28	0.24	**0.40**	**0.41**	0.14	0.38	0.36	**0.24**
4	0.59	0.41	0.42	0.65	**0.51**	0.46	**0.66**	**0.52**	0.48
5	0.54	**0.43**	**0.43**	0.56	0.36	0.41	**0.56**	0.41	0.42
6	0.11	0.18	0.08	0.13	0.16	0.11	**0.17**	**0.20**	**0.17**
7	0.48	0.54	0.50	0.50	0.53	0.51	**0.52**	**0.56**	**0.51**
8	0.38	0.32	0.29	0.36	0.18	0.30	**0.41**	**0.36**	**0.32**
9	0.35	0.29	0.32	0.34	0.34	0.32	**0.37**	**0.35**	**0.34**
10	**0.65**	**0.17**	0.34	0.49	0.03	**0.57**	0.58	0.12	0.51
11	**0.30**	**0.23**	0.18	0.18	0.20	**0.23**	0.25	0.14	0.12
12	0.38	**0.46**	0.27	0.41	0.45	**0.42**	**0.42**	0.45	0.38
13	**0.61**	**0.51**	**0.60**	0.56	0.39	0.42	0.59	0.48	0.47
14	**0.60**	0.43	0.39	0.53	0.08	**0.44**	0.57	**0.49**	0.41
15	0.69	0.67	**0.62**	0.69	0.68	0.59	**0.72**	**0.71**	0.59
16	0.71	0.72	0.66	0.76	0.75	0.68	**0.76**	**0.76**	**0.68**
17	0.75	0.75	0.73	0.74	0.76	0.71	**0.77**	**0.78**	**0.73**
18	0.74	0.73	0.46	0.75	0.75	0.63	**0.75**	**0.78**	**0.66**
19	0.27	0.27	**0.30**	0.27	0.27	0.22	**0.30**	**0.30**	0.26
20	0.78	0.82	**0.68**	0.82	0.81	0.59	**0.84**	**0.83**	0.55
21	0.84	0.71	0.64	0.85	**0.84**	**0.67**	**0.86**	0.83	0.59
22	0.37	0.41	**0.35**	0.35	0.33	0.29	**0.42**	**0.41**	0.23
23	0.55	**0.55**	0.48	0.52	0.51	**0.50**	**0.56**	0.43	0.44
24	**0.62**	**0.59**	0.53	0.57	0.55	0.55	0.59	0.58	**0.58**
25	0.63	0.79	0.68	0.76	**0.81**	0.74	**0.77**	0.80	**0.76**
26	0.71	0.75	0.70	0.73	0.77	0.70	**0.77**	**0.78**	**0.73**
27	0.57	0.53	0.54	0.57	0.58	0.51	**0.61**	**0.60**	**0.60**
28	0.39	**0.39**	0.33	0.42	0.23	**0.41**	**0.46**	0.29	0.38
29	0.58	0.66	0.56	0.59	**0.65**	0.56	**0.61**	**0.67**	0.63
30	0.54	0.60	**0.66**	**0.57**	0.65	0.59	0.56	**0.68**	0.63
31	0.43	0.47	0.42	0.44	0.57	**0.43**	**0.47**	**0.62**	0.34
32	0.23	**0.25**	**0.29**	0.19	0.17	0.25	**0.24**	0.22	0.25
33	**0.43**	0.34	0.40	0.32	0.33	0.29	0.34	**0.34**	**0.40**
34	0.56	0.73	0.71	**0.66**	**0.88**	0.73	0.61	0.78	**0.75**
F	0.52	0.50	0.45	0.52	0.47	0.46	**0.55**	**0.51**	**0.47**

From Table 6, we observe that, with regard to different classification models (i.e. NB, LR and RT), our IFA method performs better than the other two methods with the highest *F-measure* in average.

4.2. RQ2: Which distance in NN filter performs best in constructing homogeneous training datasets?

In the data filtering stage, we apply NN method in similarity measuring to construct a homogeneous training dataset from CC data. To identify which distance performs better, we conduct experiments on the basis of the selected feature subset to make comparisons among the 5 distances, i.e. Euclidean, Jaccard, Cosine, Correlation and Hamming.

Figure 5 shows the standardized boxplots of the performances of different distances achieved by different classification models based on Naïve Bayes, Logistic Regression and Random Tree, respectively. From the bottom to the top of a standardized box plot: minimum, first quartile, median, third quartile, and maximum. Any data not included between the boxes is plotted as a small cross.

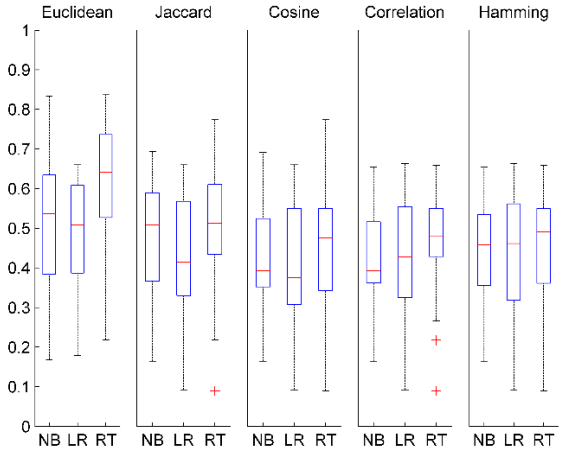

Figure 5. The standardized boxplots of the performances

Considering the median line of the boxes, we find that, for all the classification models, the Euclidean performs better than the other distances in general.

4.3. RQ3: How does the defect prediction performance of the two-stage method compare with the baseline methods over different data sets?

To demonstrate the overall effectiveness of the two-stage method, we combine feature selection and data filtering together, and perform comparison experiments among ALL + Euclidean, Top-5 + Euclidean and IFA + Euclidean. Figure 6 describes the experimental results.

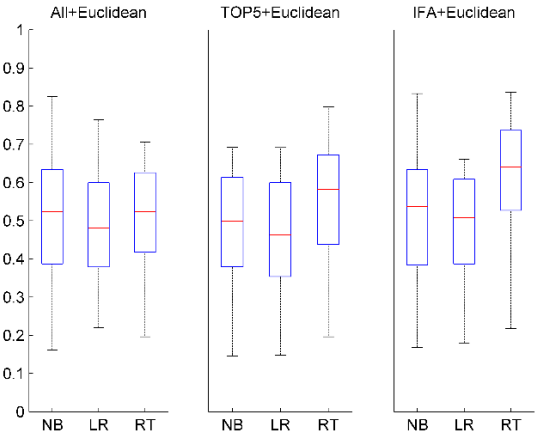

Figure 6. The standardized boxplots of the performances achieved by different classification models

From Figure 6, we find that, for a specific classification model, the combination of IFA + Euclidean performs better than the others, especially obviously viewed in RT. On the other hand, for a specific combination, RT seems to perform better than the other classification models.

5. Conclusions

In this study, we design and implement an information flow based software feature selection method, combined with training data filtering, to improve the data quality for CPDP. In feature selection, we utilize the information flow algorithm to do the causality analysis among software metrics, in order to find the most correlative ones. In data filtering, we use the NN filter to construct a homogeneous training datasets from the CC data automatically. To validate the effectiveness and practicality of our approach, we systematically conduct verification experiments. Several classic methods are included for comparison. In the studies, we will conduct more comparison experiments with other novel feature selection methods. The final experimental results demonstrate the potential of our approach in improving the quality of training data and the performance of CPDP.

In addition, interpreting these in terms of the posed questions, we conclude that:

(1) For feature selection, causality analysis is more effective than relevance analysis, and IFA is useful in implementing feature selection to improve performances for CPDP.

(2) For data filtering, we find that the Euclidean distance norm performs best in the NN filter.

(3) For classification models, we observe that the RT may yield better prediction results than the other models.

Acknowledgements

This paper was supported in part by the National Natural Science Foundation of China and the Natural Science Foundation of Jiangsu Province, China.

References

1. J. Ba and S. Wu, "SdDirM: A dynamic defect prediction model," in Ieee/asme International Conference on Mechatronics and Embedded Systems and Applications, 2012, pp. 252-256.
2. C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics): Springer-Verlag New York, Inc., 2006.
3. G. Boetticher, T. Menzies, T. Ostrand, and G. Boetticher, "\{PROMISE\} Repository of empirical software engineering data," West Virginia University Department of Computer Science, 2007.
4. S. Chamoli, G. Tenne, and S. Bhatia, "Analysing Software Metrics for Accurate Dynamic Defect Prediction Models," Indian Journal of Science & Technology, vol. 8, 2015.
5. S. S. Choi, S. H. Cha, and C. C. Tappert, "A Survey of Binary Similarity and Distance Measures," Journal of Systemics Cybernetics & Informatics, vol. 8, pp. 43--48, 2009.
6. K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," Software Practice & Experience, vol. 41, pp. 579–606, 2011.
7. C. W. J. Granger, "Investigating Causal Relations by Econometric Models and Cross-spectral Methods," Econometrica, vol. 37, pp. 424-438, 1969.
8. H. He and E. A. Garcia, "Learning from Imbalanced Data," Knowledge & Data Engineering IEEE Transactions on, vol. 21, pp. 1263-1284, 2009.
9. P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," Information & Software Technology, vol. 59, pp. 170-190, 2015.
10. Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," Automated Software Engineering, vol. 19, pp. 167-199, 2012.
11. K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in International Conference on Software Engineering, 2013, pp. 392-401.
12. Y. Hong, W. Kim, and J. Joo, "Prediction of defect distribution based on project characteristics for proactive project management," in International Conference on Predictive MODELS in Software Engineering, 2010, p. 15.
13. G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, "A Practical Differentially Private Random Decision Tree Classifier," in IEEE International Conference on Data Mining Workshops, 2009, pp. 114-121.
14. X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu, and J. Liu, "Dictionary learning based software defect prediction," 2014, pp. 414-423.
15. M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in International Conference on Predictive MODELS in Software Engineering, Promise 2010, Timisoara, Romania, September, 2010, pp. 1-10.
16. T. M. Khoshgoftaar, K. Gao, and A. NAPOLITANO, "AN EMPIRICAL STUDY OF FEATURE RANKING TECHNIQUES FOR SOFTWARE QUALITY PREDICTION," International Journal of Software Engineering & Knowledge Engineering, vol. 22, pp. 161-183, 2012.
17. S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A

Proposed Framework and Novel Findings," IEEE Transactions on Software Engineering, vol. 34, pp. 485-496, 2008.
18. X. S. Liang, "Normalizing the causality between time series," Phys. Rev. E, vol. 92, 2015.
19. X. S. Liang, "Unraveling the cause-effect relation between time series," Physical Review E Statistical Nonlinear & Soft Matter Physics, vol. 90, p. 052150, 2014.
20. W. Liu, S. Liu, Q. Gu, and J. Chen, "Empirical Studies of a Two-Stage Data Preprocessing Approach for Software Fault Prediction," IEEE Transactions on Reliability, vol. 65, pp. 1-16, 2015.
21. T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," IEEE Transactions on Software Engineering, vol. 33, pp. 2-13, 2007.
22. T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," Automated Software Engineering, vol. 17, pp. 375-407, 2010.
23. J. Nam and S. Kim, "Heterogeneous defect prediction," IEEE Transactions on Software Engineering, vol. PP, pp. 1-1, 2015.
24. F. Rahman, S. Khatri, E. T. Barr, and P. Devanbu, "Comparing static bug finders and statistical prediction," Macbeth.cs.ucdavis.edu, pp. 424-434, 2014.
25. Rahman, Foyzur, Posnett, Daryl, Herraiz, Devanbu, et al., "Sample size vs. bias in defect prediction," 2013.
26. C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano, "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance," IEEE Transactions on Systems Man & Cybernetics Part A Systems & Humans, vol. 40, pp. 185-197, 2010.
27. M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," IEEE Transactions on Software Engineering, vol. 39, pp. 1208-1215, 2013.
28. S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing Features to Improve Code Change-Based Bug Prediction," IEEE Transactions on Software Engineering, vol. 39, pp. 552-569, 2013.
29. Q. Song, J. Ni, and G. Wang, "A Fast Clustering-Based Feature Subset Selection Algorithm for High-Dimensional Data," Knowledge & Data Engineering IEEE Transactions on, vol. 25, pp. 1-14, 2013.
30. B. Turhan, T. Menzies, A. B. Bener, and J. D. Stefano, "On the relative value of cross-company and within-company data for defect prediction. Empir Softw Eng," Empirical Software Engineering, vol. 14, pp. 540-578, 2009.
31. J. Vaidya, M. Kantarcıoğlu, and C. Clifton, "Privacy-preserving Naïve Bayes classification," The VLDB Journal, vol. 17, pp. 879-898, 2008.
32. H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A Comparative Study of Ensemble Feature Selection Techniques for Software Defect Prediction," in International Conference on Machine Learning and Applications, Icmla 2010, Washington, Dc, Usa, 12-14 December, 2010, pp. 135-140.
33. S. Wang and X. Yao, "Using Class Imbalance Learning for Software Defect Prediction," IEEE Transactions on Reliability, vol. 62, pp. 434-443, 2013.
34. F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," 2014, pp. 182-191.