PAPER

A Hybrid Feature Selection Method for Software Fault Prediction

Yiheng JIAN†, ***Student Member***, **Xiao YU**††a), **Zhou XU**††b), ***and* Ziyi MA**†††, ***Nonmembers***

SUMMARY Fault prediction aims to identify whether a software module is defect-prone or not according to metrics that are mined from software projects. These metric values, also known as features, may involve irrelevance and redundancy, which hurt the performance of fault prediction models. In order to filter out irrelevant and redundant features, a Hybrid Feature Selection (abbreviated as HFS) method for software fault prediction is proposed. The proposed HFS method consists of two major stages. First, HFS groups features with hierarchical agglomerative clustering; second, HFS selects the most valuable features from each cluster to remove irrelevant and redundant ones based on two wrapper based strategies. The empirical evaluation was conducted on 11 widely-studied NASA projects, using three different classifiers with four performance metrics (precision, recall, F-measure, and AUC). Comparison with six filter-based feature selection methods demonstrates that HFS achieves higher average F-measure and AUC values. Compared with two classic wrapper feature selection methods, HFS can obtain a competitive prediction performance in terms of average AUC while significantly reducing the computation cost of the wrapper process.

key words: *fault prediction, feature selection, hierarchical agglomerative clustering*

1. Introduction

Fault prediction aims to predict whether or not a particular software module is defective via learning from historical defect data [1], [2]. Therefore, fault prediction is often used to help to reasonably allocate limited development and maintenance resources [3]–[5]. Many learning models have been proposed for fault prediction. The performance of these models is still vulnerable to irrelevant and redundant module features that undermine the prediction effect. Previous results have shown that the performance of predictive models improve when irrelevant and redundant features are eliminated from the original dataset [6], [7]. It is crucial to apply feature selection to fault prediction since feature selection can filter out irrelevant and redundant features by evaluating the contributions of module features.

There are two main types of feature selection methods: the filter method and the wrapper method. The former selects a feature subset based on some evaluation criteria, such as information gain and chi-square statistics. The process does not involve any classification algorithms. Therefore, the computational complexity is less but the prediction error is higher. Contrary to the filter method, the wrapper method employs the prediction accuracy as an evaluation criterion and searches the best feature subset through an exhaustive search on all feature subsets. The wrapper method is optimized directly for a given classifier, therefore it improves the accuracy but also greatly increases the computational complexity [24].

Recent research efforts on feature selection for fault prediction mainly focus on the filter methods. For example, Chen et al. [8] and Liu et al. [9] proposed two filter-based feature selection method using both feature clustering and feature ranking. In [10], we proposed a filter-based feature selection method, i.e., MICHAC, which employs Maximal Information Coefficient (MIC) [17] to rank candidate features to remove irrelevant ones, and groups features with Hierarchical Agglomerative Clustering (HAC) and selects one feature from each resulted group. Since MICHAC is a filter-based feature selection method, and the training process doss not involve any classification algorithms, the prediction error is higher than the wrapper method. Therefore, we want to employ the wrapper-based feature selection strategy to further improve the performance of MICHAC.

In this paper, we extend MICHAC and propose a hybrid feature selection (HFS) method for fault prediction, which combines feature clustering with wrapper-based feature selection strategies. In the stage of feature clustering, HFS groups features according to the correlation between every pair of features using HAC, such that redundant features can be grouped into the same cluster. While MICHAC selects one feature from each cluster, HFS employs Maximal Information Coefficient (MIC) to calculate the relevance between features and labels, and selects the most valuable features from each cluster based on two wrapper-based strategies.

HFS is generally superior to MICHAC through the two wrapper-based strategies, because it determines the final feature subset according to the prediction accuracy of fault prediction model. In addition, adding the most relevant feature or removing the most irrelevant feature from each cluster at each iteration not only can filter out irrelevant and redundant features, but also may reduce the number of wrapper evaluations (proportional to the computation cost).

We evaluate our proposed method, HFS, by answer-

Manuscript received February 1, 2019.
Manuscript revised May 25, 2019.
Manuscript publicized July 9, 2019.

†The author is with School of Information and Electronics, Beijing Institute of Technology, Beijing 102488, China.
††The authors are with School of Computer Science, Wuhan University, Wuhan, Hubei 430000, China.
†††The author is with School of Computer Science, Huazhong University of Science and Technology, Wuhan, Hubei 430000, China.
a) E-mail: xiaoyu_whu@yahoo.com (Corresponding authors)
b) E-mail: zhouxullx@whu.edu.cn (Corresponding authors)

DOI: 10.1587/transinf.2019EDP7033

ing two research questions on effectiveness and efficiency. Experiments are conducted on 11 widely-studied NASA projects. Experimental results show that HFS can effectively and efficiently select features to improve existing fault prediction methods. On most of projects under evaluation, HFS performs the best AUC and F-measure values compared with six state-of-the-art filter-based feature selection methods. HFS_BE with the model built with RF achieves the highest average F-measure value (0.399), and 1.79% – 43.48% higher than the compared methods. In addition, HFS_BE with the model built with RF achieves the highest average AUC value (0.820), and 1.61% – 44.37% higher than the compared methods. We also compare our method with two classic wrapper-based feature selection methods. This experiment shows that HFS can yield a competitive prediction performance against the compared methods in fault prediction, while significantly reducing the computation cost of the wrapper process.

This paper makes the following contributions: We propose a hybrid feature selection method, HFS, which combines feature clustering with wrapper-based feature selection. To the best of our knowledge, this is the first time to combine feature clustering with wrapper-based feature selection to filter out irrelevant and redundant features in the field of fault prediction. In this way, HFS can not only construct a more optimized feature subset, but also may significantly decrease the computation cost of classic wrapper-based feature selection methods.

This paper is based on our conference paper [10]. The extensions of the conference paper which are made in this paper are as follows:

1. We modify the strategy of selecting features from each cluster. MICHAC selects one feature from each cluster, while HFS selects the most valuable features from each cluster based on two wrapper-based strategies.
2. We provide a comprehensive evaluation and comparison of HFS against two classic wrapper-based feature selection methods, and address new research question about the computation cost when conducting feature selection.

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes our proposed HFS method. Section 4 and Sect. 5 show the experiment setup and experiment results, respectively. Section 6 discusses the threats to validity. Finally, Sect. 7 addresses the conclusion and points out the future work.

2. Related Work

A number of prior studies have investigated feature selection methods on predicting defective software modules. For example, Gao et al. [11] studied four different filter-based feature selection methods with five different classifiers on a large telecommunication system and found that the Kolmogorov-Smirnov method performed the best. Gao et al. [2] proposed a hybrid feature selection method, which first uses feature ranking to reduce the search space and then applies feature subset selection. The results indicated that removing 85 percent of features does not adversely affect prediction performance. Shivaji et al. [12] utilized six feature selection methods to iteratively remove irrelevant features until achieving the best performance of F-measure. He et al. [13] performed an empirical study to investigate the feasibility of the fault prediction model built with a simplified feature set, and found that the model built with the simplified feature set provides acceptable prediction results.

Recent works have proposed some combined frameworks to apply feature selection to eliminate both irrelevant and redundant software features from the original dataset. Chen et al. [8] proposed a two-stage data preprocessing framework, TC, which combines feature selection and instance reduction. In the feature selection phase, they proposed a new algorithm using feature selection and threshold-based clustering. Liu et al. [9] proposed a new feature selection framework, FECAR, to conduct feature clustering and feature ranking. FECAR first clusters features via k-medoids method and then selects several representative features from each cluster. In our work, HFS has a similar feature selection framework to FECAR. But the differences between HFS and FECAR are significant. FECAR employs the k-medoids clustering to group features into groups, while HFS employs HAC and can automatically determine the number of clusters. Meanwhile, FECAR selects a certain number of relevant features from each cluster, while HFS selects the most valuable features based on two proposed wrapper-based strategies.

3. The HFS Method

We propose HFS as a novel feature selection method, which combines feature clustering with wrapper-based feature selection. In the stage of feature clustering, HFS groups features according to the correlation between every two features using Hierarchical Agglomerative Clustering (HAC), such that redundant features can be grouped into the same cluster. In the stage of wrapper-based feature selection, HFS employs Maximal Information Coefficient (MIC) to calculate the relevance between features and labels, and selects the most valuable features from each cluster based on two wrapper-based strategies. As a result, HFS constructs an optimized subset of module features to replace the original feature set in fault prediction.

3.1 Feature Clustering Stage

The main goal of the feature clustering stage is to group redundant features that have similar effect with other features in distinguishing modules with different labels. We use the HAC algorithm to cluster the features. HAC is an iterative process which merges current clusters continuously. It is possible that a current cluster only contains one feature, e.g., each feature is treated as one cluster at the beginning of the iteration. In HAC, features are merged into clusters according to the distances between current clusters. We employ the average linkage method to define the distance of two current clusters [14]. The average linkage between two clusters is defined as the average of the distance between any pair of features from two clusters. Suppose that U_a and U_b are two current clusters during the clustering process. The distance of the two clusters $D_{a,b}$ can be calculated by the following formula:

$$D_{a,b} = \frac{1}{m_a m_b} \sum_{x_i \in U_a, x_j \in U_b} d_{i,j} \quad (1)$$

where m_a and m_b are the number of features inside clusters U_a and U_b, respectively, and $d_{i,j}$ is the distance between two features x_i and x_j. We define the distance $d_{i,j}$ of two features x_i and x_j with Pearson correlation coefficient $c_{i,j}$ as follows:

$$d_{i,j} = 1 - c_{i,j} \quad (2)$$

For two given features, Pearson correlation coefficient measures the relevance between numeric values of both features in instances [15]. Given two module features x_i and x_j, the observation vectors of features are symbolized as n-dimension vectors, i.e., $< x_{i1}, x_{i2}, \cdots, x_{in} >$ and $< x_{j1}, x_{j2}, \cdots, x_{jn} >$, respectively, where values x_{ik} and x_{jk} denote the numeric values of the features x_i and x_j in the kth instance ($k = 1, 2, \cdots, n$) and n is the number of instances in the dataset. Then the correlation coefficient $c_{i,j}$ for features x_i and x_j is calculated by the following formula:

$$c_{ij} = \frac{\sum_{k=1}^{n}(x_{ik} - \overline{x_i})(x_{jk} - \overline{x_j})}{\sqrt{\sum_{k=1}^{n}(x_{ik} - \overline{x_i})^2 \sum_{k=1}^{n}(x_{jk} - \overline{x_j})^2}} \quad (3)$$

where $\overline{x_i} = \frac{1}{n}\sum_{k=1}^{n} x_{ik}$ and $\overline{x_j} = \frac{1}{n}\sum_{k=1}^{n} x_{jk}$ ($i = 1, 2, \cdots, p$, $j = 1, 2, \cdots, p$, and p is the number of features). In this step, we record all information during clustering, including the orders of merging and distances between clusters.

In our work, we choose the number of final clusters by maximizing the increment of inconsistency coefficient during the clustering process of HAC. Inconsistency Coefficient (IC) is used to quantitatively express the relative consistence of a link [16]. A link denotes an action of merging two current clusters. The value of inconsistency coefficient can be calculated by comparing the distance of the current link and the average distance of its neighbors. The neighbors of a specific link denote all children links that lead to this link as well as the link itself. For each link, we count its IC value of a link L_{curr} to measure the change of clustering as follows:

$$IC(L_{curr}) = \frac{D_{L_{curr}} - avg(D_{L_{neighbor}})}{Std(D_{L_{neighbor}})} \quad (4)$$

where $Std(D_{L_{neighbor}})$ denotes the standard deviation of all links in its neighbors. We define the increment of IC values between two links as $\Delta_{L_{curr},L_{prev}} = IC(L_{curr}) - IC(L_{prev})$, where L_{curr} and L_{prev} denote a current link and its previous link, respectively. Then we find the link with the maximal increment value and stop the process of HAC before this link [16]. We use a simple clustering dendrogram to visualize the above concept. Figure 1 shows a clustering dendrogram including three module features f_1, f_2, and f_3. At first, features f_1, f_2, and f_3 are treated as three initial clusters. As shown in Fig. 2, f_1 and f_2 are first linked to a cluster, labeled f_4; then f_4 and f_3 are linked to a cluster, labeled f_5. The link $L_{1,2}$ between f_1 and f_2 and the link $L_{4,3}$ between f_4 and f_3 are shown in the dendrogram. According to Eq. (2) and Eq. (3), the distance between f_1 and f_2 is $D_{1,2} = d_{1,2} = 1 - c_{1,2}$, where $c_{1,2}$ denotes the Pearson correlation coefficient of f_1 and f_2. $d_{1,2}$ denotes the distance of two features f_1 and f_2, and $D_{1,2}$ denotes the distance of two clusters (initialized based on f_1 and f_2). Meanwhile, according to Eq. (1), the distance between f_4 and f_3 is the average distance between any feature pair between cluster f_4 and cluster f_3, namely $D_{4,3} = (d_{1,3} + d_{2,3})/2$.

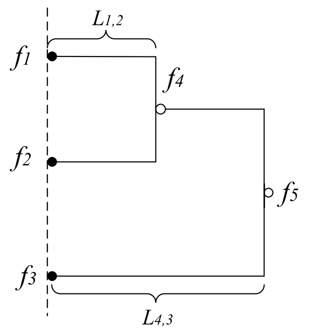

Fig. 1 A simple clustering dendrogram with three features.

The inconsistency coefficient value of the link L_4 (at the depth of two in the dendrogram) can be calculated as follows:

$$IC(L_{4,3}) = \frac{D_{4,3} - \frac{D_{1,2}+D_{4,3}}{2}}{Std} \quad (5)$$

where Std denotes the standard deviation of $D_{1,2}$ and $D_{4,3}$.

We determine the final number of clusters according to the increment of inconsistency coefficient. The details of selecting the number of final clusters are as follows. In the process of merging two clusters, i.e., building a link, a higher increment of inconsistency coefficient indicates that the link of merging previous clusters will lead to a better clustering result. Then we determine the optimized number S of clusters according to the maximal increment of inconsistency coefficient. Specifically, for all the links, if the increment of the inconsistency coefficient of one link is the maximal, we take the cluster number of the pervious one of this link as the final cluster number.

3.2 Feature Selection Stage

In the stage of feature selection, we select several features from the clusters to represent the whole feature subset based on two wrapper-based strategies. Feature selection stage consists of two major steps.

In the first step, we mainly conduct the relevance analysis between each feature and the class label. We calculate

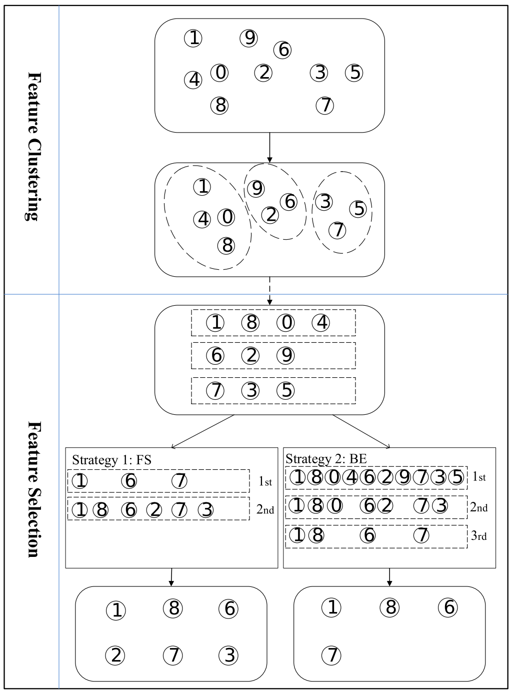

Fig. 2 An example of our proposed approach, HFS

the MIC values between each feature x and the class label y. Maximal Information Coefficient (MIC) was developed as a robust indicator of measuring the correlation between two variables by Reshef et al. in 2011 [17]. In the second step, we select the most valuable features from each cluster based on the wrapper-based strategies, which are explained immediately below.

1. Forward Selection (FS): we start with having no feature in the fault prediction model. In each iteration, we keep adding the most relevant feature from each cluster till an addition of new ones does not improve the performance of the model.
2. Backward Elimination (BE): we start with all the features and remove the most irrelevant feature from each cluster at each iteration until no improvement of the fault prediction model is observed on removal of features.

We use a simple example to explain these two wrapper-based strategies. As shown in Fig. 2, there are ten features (f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, and f_9) in the original feature set. We group these features into three clusters, such that the first cluster contains f_0, f_1, f_4, and f_8, the second cluster contains f_2, f_6, and f_9, and the third cluster contains f_3, f_5, and f_7. Then, we employ MIC to calculate the relevance between features and labels, and rank these features. For example, in the first cluster, f_1 has the highest relevance and f_4 has the lowest relevance.

According to the FS strategy, we start with having no feature in the fault prediction model. In the first iteration, we add the most relevant features from each cluster, i.e., f_1, f_6, and f_7, into the current feature subset. Then, we train the fault prediction model based on this current feature subset and take the F-measure as the quality criterion for evaluating this feature subset. In principle, any other measures can be used. In the second iteration, we select the most relevant features from each cluster, i.e., f_8, f_2, and f_3, and add them to the current feature subset, such that the new feature subset is $\{f_1, f_2, f_3, f_6, f_7, f_8\}$. Next, we train the fault prediction model based on this new feature subset, and then evaluate it. Suppose that the new feature subset has a higher F-measure value than the current feature subset generated in the first iteration, then it replaces the current feature subset and the algorithm continues. In the third iteration, we construct the new feature subset $\{f_0, f_1, f_2, f_3, f_5, f_6, f_7, f_8, f_9\}$ and evaluate it in the same way. Suppose that the new feature subset has a lower F-measure value than the current feature subset generated in the second iteration, then the algorithm terminates, and the final feature subset is the current feature subset, i.e., $\{f_1, f_2, f_3, f_6, f_7, f_8\}$.

According to the BE strategy, in contrast, we start with having all the features in the fault prediction model, such that the current feature subset is $\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9\}$. We train the fault prediction model based on this feature subset and take the F-measure as the quality criterion for evaluating it. In the first iteration, we remove the most irrelevant features from each cluster, i.e., f_4, f_9 and f_5, such that the new feature subset is $\{f_0, f_1, f_2, f_3, f_6, f_7, f_8\}$. Then, we train the fault prediction model based on this new feature subset, and then evaluate it. Suppose that the new feature subset has a higher F-measure value than the current feature subset generated in the first iteration, then it replaces the current feature subset and the algorithm continues. In the second iteration, we construct the new feature subset $\{f_1, f_6, f_7, f_8\}$ and evaluate it in the same way. Suppose that the new feature subset has a higher F-measure value than the current feature subset generated in the second iteration, then it replaces the current feature subset and the algorithm continues. In the third iteration, we construct the new feature subset f_1 and evaluate it in the same way. Suppose that the new feature subset has a lower F-measure value than the current feature subset generated in the third iteration, then the algorithm terminates, and the final feature subset is the current feature subset, i.e., $\{f_1, f_6, f_7, f_8\}$.

4. Experimental Setup

4.1 The NASA Dataset

Our experiments are mainly conducted on 11 widely-studied projects in the NASA dataset [18]. The NASA dataset was donated by Menzies and was cleaned by Shepperd et al. [19]. Software modules in these projects are characterized with static code metrics, such as LOC counts, Halstead complexity metrics, McCabe complexity metrics. Table 1 shows the details of the 11 projects in the NASA dataset, where # features, # modules, # defective modules and % defective modules denote the number of features, the number of modules, and the number of defective modules, and the percentage of defective modules, respectively.

Table 1 Projects in the NASA dataset used in our experiment

Project	# features	# modules	# defective modules	% defective modules
CM1	37	344	42	12.2%
JM1	21	9593	1759	18.3%
KC1	21	2096	325	15.5%
KC2	21	522	107	20.5%
MC1	38	9277	68	0.7%
MC2	39	127	44	34.6%
MW1	37	253	27	10.7%
PC1	37	759	61	8.0%
PC3	37	1125	140	12.4%
PC4	37	1399	178	12.7%
PC5	38	1711	471	27.5%

4.2 Research Questions

Since our feature selection method aims to improve the performance of existing feature selection methods, but also greatly reduce the computational complexity, our evaluation focuses on answering two Research Questions (RQs), including both effectiveness and efficiency.

RQ1. Does our method, HFS, perform better than the state-of-the-art filtered-based feature selection methods on fault prediction?

This question validates the important criterion of defect prediction: the performance improvement in terms of defective precision, recall, F-measure and AUC (as defined in Sect. 4.3). To answer this question, we compare our method against three classic feature selection methods, i.e., Chi-Square, Gain Ratio, and ReliefF.

In the domain of software fault prediction, the Chi-Square (CS) statistic measures the independence between the feature f and the class label c [20]. Gain Ratio (GR) is an updated version of the information gain by penalizing multi-valued attributes to counter the bias [21]. ReliefF (ReF) is an extension of the Relief method that can solve the multi-class learning problem [22]. The number of features in these methods is set to $[\log_2 m]$ where m is the number of original features. This setting is suggested by Khoshgoftaar et al. [23] and Gao et al. [2]. Their works show that various classifiers in fault prediction are appropriate to this setting.

We also compare our work with three existing feature selection methods, which are originally designed for defect prediction: TC, FECAR [9], and MICHAC [10], as to be described in the related work.

RQ2. Are the wrapper-based feature selection strategies effective and efficient, comparing with two classic sequential wrapper feature selection methods?

HFS mainly consists of two stages, feature clustering stage and wrapper-based feature selection stage. To investigate the effectiveness and efficiency of the two wrapper-based strategies, we compare HFS with two classic sequential wrapper feature selection methods, i.e., Sequential Forward Selection (SFS) and Sequential Backward Elimination (SBE) [24].

SFS starts with an empty feature subset, and each time adds one feature that results in the highest prediction accuracy, until no improvement of the defect prediction model is observed. SBE starts with all the features and remove them one by one, at each round removing the one that decreases the prediction error most, until a removal of new ones increases the prediction error.

4.3 Evaluation Metrics

In this empirical study, we use F-measure and AUC as main metrics to compare the performance of classifiers, because the defect datasets are imbalanced.

(1) F-measure: There are four possible outcomes from a binary predictive model on a test set: classifying a defective module as defective ($n_{d\to d}$), classifying a defective module as defect-free ($n_{d\to f}$), classifying a defect-free module as defect-free ($n_{f\to t}$), and classifying a defect-free module as defective ($n_{f\to d}$). Based on the possible output, the defective precision, defective recall, defective F-measure are defined as follows:

$$\text{defective precision, } P(d) = \frac{n_{d\to d}}{n_{d\to d} + n_{f\to d}} \tag{6}$$

$$\text{defective recall, } R(d) = \frac{n_{d\to d}}{n_{d\to d} + n_{d\to f}} \tag{7}$$

$$\text{defective F-measure, } F(d) = \frac{2 \times P(d) \times R(d)}{P(d) + R(d)} \tag{8}$$

Note that defective F-measure is the harmonic mean of defective precision and defective recall. In general, the greater the F-measure is, the better the prediction performance of the classifier is.

(2) AUC: AUC (Area Under the ROC Curve) is widely used in fault prediction. An ROC is a curve plotted on a two dimensional plane with the true positive rate as y-axis and the false positive rate as x-axis, so this curve is used to visualize the performance of binary classifiers. The curve of ROC illustrates the trade-off between true positive and false positive. This curve is used to evaluate the different classifier performance.

In the experiments, we perform 10-fold cross validation, since it is used in many defect prediction studies. In the 10-fold cross validation, a dataset is divided into 10 folds at random. Nine of the ten folds take turns to be used as the training set while the other fold is used as the test set. The training data are used to build a classifier; then the built classifier is evaluated on the test data. This cross-validation is repeated ten times so that each partition is used exactly once as the test data. The above procedure is repeated 20 times in total to avoid sample bias. The final results of all methods are estimated by averaging the results over 20 runs of the 10-fold cross-validation.

4.4 Defect Prediction Models

In order to compare the performance of feature selection methods, we employ three representative classifiers in fault prediction, Naive Bayes (NB), Random Forest (RF), and

Table 2 Performance of 11 NASA projects with the three classifiers on precision, recall, and F-measure.

Model	Metric	HFS_FS	HFS_BE	Full	CS	GR	RF	TC	FECAR	MICHAC
NB	AverageP	0.429	0.434	0.407	0.44	**0.454**	0.35	0.41	0.442	0.427
	AverageR	0.405	0.412	**0.429**	0.332	0.366	0.424	0.342	0.376	0.397
	AverageF	0.382	**0.385**	0.35	0.359	0.36	0.355	0.346	0.367	0.373
	MedianF	0.352	0.385	0.331	0.393	0.376	0.351	0.351	0.388	**0.395**
	W/D/L(1)		4/0/7	8/0/3	9/0/2	7/0/4	6/0/5	8/0/3	7/0/4	7/0/4
	W/D/L(2)	7/0/4		9/0/2	9/0/2	8/0/2	7/0/4	8/0/2	7/0/4	7/0/4
RF	AverageP	0.559	**0.562**	0.54	0.488	0.498	0.468	0.526	0.501	0.56
	AverageR	0.315	0.307	0.288	0.302	0.311	0.258	0.287	**0.321**	0.311
	AverageF	0.397	**0.399**	0.372	0.366	0.377	0.326	0.367	0.385	0.392
	MedianF	0.395	**0.419**	0.417	0.363	0.363	0.331	0.372	0.375	0.401
	W/D/L(1)		4/0/7	4/0/7	7/0/4	7/0/4	8/0/3	7/0/4	6/0/5	5/0/6
	W/D/L(2)	7/0/4		6/0/5	9/0/2	9/0/2	8/0/3	8/0/3	6/0/5	6/0/5
RIPPER	AverageP	**0.567**	0.558	0.488	0.535	0.536	0.423	0.498	0.53	0.55
	AverageR	**0.294**	0.282	0.268	0.266	0.255	0.171	0.237	0.267	0.28
	AverageF	0.373	**0.374**	0.333	0.345	0.334	0.23	0.309	0.344	0.351
	MedianF	0.372	**0.425**	0.4	0.374	0.362	0.216	0.289	0.364	0.358
	W/D/L(1)		5/0/6	8/0/3	5/0/6	6/0/5	9/0/2	9/0/2	7/0/4	7/0/4
	W/D/L(2)	6/0/5		8/0/3	5/0/6	5/0/6	9/0/2	8/0/3	7/0/4	7/0/4

Repeated Incremental Pruning to Produce Error Reduction (RIPPER). The reason we choose these classifiers is that they fall into three different families of learning methods. NB is a probabilistic classifier; RF is a decision-tree classifier; and RIPPER is a rule-based classifier.

In this study, experiments were conducted on a workstation with an Intel Core i7-4790 CPU with 3.60 GHz. We implemented the feature selection methods in Java with the Weka package, and used the classifiers inside Weka with the default parameter settings. We calculate MIC values via the MINE toolkit [25] and implement HFS in Matlab 7.0.

4.5 Statistical Analysis

We perform the Wilcoxon signed-rank test [26] to analyze whether the performance values of HFS are statistically significant different from those of the compared methods on three classifiers over all projects. The Wilcoxon signed-rank test is a non-parameter method of statistically significant test. For the performance values of any two methods compared, the null hypothesis is that there exists no significant difference between the two methods. If the p-value that results from Wilcoxon test is less than 0.05, the null hypothesis is rejected. That is, the difference between the two methods is identified as statistically significant. The significant test is implemented in IBM SPSS Statistics. In additional, we calculate the effect size (i.e., Cliff's δ) to measure the differences between median results achieved by the two methods over all datasets. By convention, the magnitude of the difference is considered negligible ($0 <$ Cliff's $\delta < 0.x147$), small ($0.147 <$ Cliff's $\delta < 0.33$), medium ($0.33 <$ Cliff's $\delta < 0.474$), or large (Cliff's $\delta > 0.474$) [27].

5. Experimental Results

In this section, we present the experimental results as an attempt to answer the two research questions raised in Sect. 4.2.

5.1 Does Our Method, HFS, Perform Better than the State-of-Art Filtered-Based Feature Selection Methods on Fault Prediciton?

To answer RQ1, we compare our method HFS with three widely-studied feature selection methods (CS, GR, and ReF) and three recently-proposed methods in fault prediction (TC, FECAR and MICHAC).

Table 2 records the average defective precision, recall and the average and median F-measure of all 11 NASA projects with all feature selection methods on three classifiers, NB, RF and RIPPER. The column "Full" denotes the training set without involving any feature selection method; P, R, and F denote the defective precision, recall, and F-measure, respectively; W/D/L(1), short for Win/Draw/Loss, presents the number of projects, on which HFS_FS performs better than, the same as, or worse than another method, in terms of F-measure. In the same way, W/D/L(2) presents the comparison results between HFS_BE and other methods. According to the results of W/D/L, we can know the percentage of datasets on which HFS performs better than the existing methods.

Table 2 shows that on all three classifiers, HFS_FS and HFS_BE obtain better F-measure values than all filter-based methods. For the NB classifier, HFS_BE achieves the best average F-measure value, but fails in the best precision or recall. Regarding the average precision, HFS_FS and HFS_BE are inferior to the CS, GR and FECAR; regarding the average recall, HFS_FS and HFS_BE are inferior to Full and ReF. For the RF classifier, HFS_BE can achieve the best precision and F-measure values, but fail in the best recall. Regarding the average precision, HFS_FS is inferior to MICHAC; regarding the average recall, HFS_FS is inferior to FECAR. For the RIPPER classifier, HFS_FS achieves the best values in terms of precision and recall. Meanwhile, HFS_BE achieves the best F-measure value. Overall, HFS_BE with the model built with RIPPER achieves

Table 3 AUC values on the 11 NASA projects using naive Bayes with the Wilcoxon test (p-value) and Cliff's δ value.

Project	HFS_FS	HFS_BE	Full	CS	GR	RF	TC	FECAR	MICHAC
CM1	0.734	**0.746**	0.694	0.729	0.75	0.752	0.72	0.75	0.725
JM1	**0.685**	0.674	0.678	0.629	0.629	0.624	0.682	0.636	0.642
KC1	**0.797**	0.782	0.791	0.783	0.774	0.782	0.798	0.78	0.791
KC2	0.81	**0.838**	0.832	0.816	0.817	0.827	0.833	0.825	0.836
MC1	**0.968**	0.923	0.892	0.768	0.812	0.845	0.87	0.883	0.917
MC2	0.8	0.784	0.717	0.637	0.637	0.627	0.666	0.63	0.722
MW1	0.916	**0.92**	0.728	0.714	0.714	0.737	0.736	0.735	0.774
PC1	0.771	**0.795**	0.768	0.663	0.701	0.673	**0.795**	0.777	0.788
PC3	0.775	0.782	0.743	0.783	0.778	0.784	0.735	0.784	0.785
PC4	**0.848**	0.834	0.825	0.823	0.835	0.823	0.807	0.823	0.837
PC5	0.608	0.613	0.69	0.651	0.617	0.64	**0.708**	0.624	0.66
AVG	**0.792**	0.79	0.76	0.727	0.733	0.738	0.759	0.75	0.771
W/D/L(1)		5/0/6	9/0/2	8/0/3	7/0/4	7/0/4	7/0/4	6/0/5	7/0/4
p-value(1)		0.79	0.075	0.05	0.05	0.091	0.213	0.109	0.374
Cliff's δ(1)		0.02	0.376	0.746	0.655	0.591	0.391	0.453	0.236
W/D/L(2)	6/0/5		8/0/3	8/0/3	8/0/3	7/1/3	7/1/3	8/0/3	7/0/4
p-value(2)	0.79		0.1	0.033	0.021	0.047	0.139	0.033	0.248
Cliff's δ(2)	0.02		0.372	0.759	0.662	0.594	0.387	0.45	0.225

Table 4 AUC values on the 11 NASA projects using random Forest with the Wilcoxon test (p-value) and Cliff's δ value.

Project	HFS_FS	HFS_BE	Full	CS	GR	RF	TC	FECAR	MICHAC
CM1	0.798	**0.803**	0.782	0.732	0.728	0.75	0.812	0.728	0.795
JM1	0.776	**0.779**	0.76	0.721	0.721	0.733	0.692	0.729	0.742
KC1	0.793	**0.814**	0.83	0.793	0.744	0.789	0.796	0.793	0.802
KC2	0.829	**0.833**	0.825	0.782	0.808	0.798	0.81	0.795	0.812
MC1	**0.927**	0.925	0.923	0.879	0.918	0.916	0.929	0.937	0.93
MC2	0.694	**0.689**	0.682	0.588	0.588	0.575	0.691	0.573	0.658
MW1	0.727	0.729	0.716	0.696	0.696	0.641	0.719	0.754	**0.737**
PC1	0.868	0.866	0.844	0.839	0.854	0.844	0.84	0.804	0.858
PC3	0.867	**0.874**	0.864	0.835	0.844	0.85	0.822	0.849	0.848
PC4	0.94	0.947	0.944	0.887	0.916	0.887	0.845	0.914	0.942
PC5	0.758	0.762	0.8	0.759	0.749	0.759	0.777	0.755	0.754
AVG	0.816	**0.82**	0.815	0.774	0.779	0.777	0.794	0.785	0.807
W/D/L(1)		4/0/7	5/0/6	10/0/1	10/0/1	9/0/2	7/0/4	07/01/04	8/0/3
p-value(1)		0.055	0.422	0.007	0.003	0.004	0.109	0.073	0.109
Cliff's δ(1)		0.051	0.008	0.507	0.415	0.434	0.295	0.352	0.11
W/D/L(2)	7/0/4		6/0/5	10/0/1	10/0/1	10/0/1	7/0/4	8/0/3	8/0/3
p-value(2)	0.055		0.182	0.003	0.003	0.003	0.05	0.018	0.016
Cliff's δ(2)	0.051		0.058	0.553	0.458	0.476	0.346	0.396	0.158

the highest median F-measure value (0.425), and 7.59% – 96.76% higher than the compared methods; HFS_BE with the model built with RF achieves the highest average F-measure value (0.399), and 1.79% – 43.48% higher than the compared methods.

The Win/Draw/Loss values show that, on the three classifiers, HFS_FS outperforms the others on over half of projects in terms of F-measure, except for the Full and MICHAC methods on RF classifier, and the CS method on RIPPER classifier. Meanwhile, HFS_BE outperforms others on over half of projects in terms of F-measure, except for the CS and GR method on RIPPER classifier.

For the sake of space limitation, in this paper we only list the detailed p-values and Cliff's δ values in term of AUC for the NASA dataset since AUC is widely used as a performance evaluation metric in fault prediction. Tables 3, 4, and 5 present the detailed AUC values of each project on the three classifiers with the p-values and Cliff's δ values.

From these tables, we can observe that HFS_FS and HFS_BE achieve the best AUC values on all three classifiers. HFS_BE with the model built with RF achieves the highest average AUC value (0.820), and 1.61%–44.37% higher than the compared methods. The Win/Draw/Loss records also indicate that HFS_FS and HFS_BE win the other filter-based methods on most projects on the three classifiers. Although only nearly half of p-values are lower than 0.05, the non-negative Cliff's δ values indicate that in all cases, HFS_FS and HFS_BE have a small or medium effect than the other filter-based methods.

Since HFS shares most of the procedure and the formulas with MICHAC, they would select the similiar features. Since different features are selected on each run of 10-fold cross-validation on the same dataset, it is difficult to list the detailed selected features on each dataset. Therefore, we only count the selected features by HFS and MICHAC, and find that cyclomatic density, normalized cyclomatic complexity, and halstead effort are selected by HFS most times, but not by MICHAC. The three features are related to code

Table 5 AUC values on the 11 NASA projects using RIPPER with the Wilcoxon test (p-value) and Cliff's δ value.

Project	HFS_FS	HFS_BE	Full	CS	GR	RF	TC	FECAR	MICHAC
CM1	0.548	**0.554**	0.516	0.542	0.513	0.494	0.51	0.55	0.541
JM1	0.564	0.561	0.553	0.56	0.554	0.541	0.54	0.545	0.563
KC1	**0.597**	0.593	0.578	0.59	0.582	0.588	0.599	0.594	0.587
KC2	0.746	0.745	0.698	0.722	0.641	0.711	0.727	0.71	0.749
MC1	0.622	0.625	0.626	0.614	**0.632**	0.623	0.614	0.612	0.619
MC2	0.585	0.589	0.576	0.573	0.582	0.595	**0.601**	0.582	0.584
MW1	0.617	0.623	0.633	**0.682**	**0.682**	0.581	0.63	0.657	0.602
PC1	0.553	0.556	**0.581**	0.569	0.578	0.5	0.554	0.565	0.558
PC3	**0.568**	0.565	0.538	0.527	0.551	0.542	0.539	0.561	0.574
PC4	0.757	0.754	0.73	0.689	0.689	0.528	0.6	0.694	**0.761**
PC5	0.601	0.606	0.623	0.615	0.61	0.54	**0.628**	0.623	0.591
AVG	0.614	**0.616**	0.605	0.608	0.601	0.568	0.595	0.608	0.612
W/D/L(1)		5/0/6	7/0/4	8/0/3	7/0/4	10/0/1	6/0/5	7/0/4	7/0/4
p-value(1)		0.261	0.213	0.374	0.306	0.01	0.248	0.477	0.306
Cliff's δ(1)		-0.017	0.14	0.1	0.204	0.696	0.297	0.091	0.036
W/D/L(2)	6/0/5		7/0/4	8/0/3	7/0/4	10/0/1	7/0/4	7/0/4	6/0/5
p-value(2)	0.261		0.182	0.213	0.247	0.008	0.182	0.374	0.286
Cliff's δ(2)	0.017		0.158	0.118	0.224	0.72	0.318	0.11	0.053

Table 6 T-Test comparison of HFS_FS and HFS_BE

	F-measure	AUC	Total
HFS_FS	5	4	9
HFS_BE	7	6	13
Neither	21	23	45

complexity, Nam et al. pointed out that higher complexity causes more defect-proneness [29]. Therefore, the selected features by HFS can make the classification model predict the defective modules more accurately. This may be the reason that HFS outperforms MICHAC.

Then, we directly compare HFS_FS and HFS_BE to obtain a more precise comparison. We revert to computing a standard two sample t-statistic [28] to compare the means of these two approaches presented by a performance metric. Table 6 compares only HFS_FS and HFS_BE, with a two-sample t-statistic calculated for each classifier and data set. Therefore, each column totals to 33 (3 classifiers × 11 datasets), and in total, 66 pairwise comparisons between HFS_FS and HFS_BE were performed, each with a 95% confidence level. The first row represents the numebr of times that HFS_FS significantly outperforms HFS_BE, the second row denotes the number of times that HFS_BE signicantly outperforms HFS_FS, and the final row represents the cases with no signicant difference between HFS_FS and HFS_BE. Overall, HFS_FS is comparable to HFS_BE, particularly in terms of the F-measure and AUC performance measure.

As a conclusion of the above observation, our method can yield better prediction results than the filter-based feature selection methods in fault prediction.

5.2 Are the Wrapper-Based Feature Selection Strategies Effective and Efficient, Comparing with Two Classic Sequential Wrapper Feature Selection Methods?

To answer RQ2, we compare our method with two classic wrapper-based feature selection methods, i.e., SFS and SBE. For the sake of space limitation, Table 6 only presents the detailed AUC values of each method on each project, since AUC is widely used as a performance evaluation metric in fault prediciton. For the NB classifier, HFS_FS achieves the best average AUC value. However, the average AUC value by HFS_BE is lower than that by SFS, while it is the same as that by SBE. For the RF classifier, HFS_FS and HFS_BE obtain better average AUC value than SFS and SBE. For the RIPPER classifier, HFS_FS and SFS achieve the best AUC value. In addition, we count the selected faatures by HFS, SFS and SBE, and find that there is not significant difference of the selected features by these methods.

Table 8 presents the runtime of each method on each project. We can also observe that HFS_FS and HFS_BE require less runtime than SFS and SBE. The runtime of SFS and SBE is almost four times than those of HFS_FS and HFS_BE. The reason is that adding the most relevant feature or removing the most irrelevant feature from each cluster at each iteration can significantly reduce the number of wrapper evaluations.

To sum up, our methods can obtain competitive prediction performance against the two classical wrapper-based feature selection methods in fault prediction, while significantly reducing the computation cost of the wrapper process.

5.3 Discussion

Overall, HFS achieves higher average F-measure and AUC values, compared with the six filter-based feature selection methods. But HFS takes average 42-49 seconds to find the features (see Table 8), while the six filter-based feature selection methods only take less than 10 seconds to find the features (We do not list the detailed runtime of each filter-based feature selection method due to the space limitation). Compared with two classic wrapper feature selection methods, HFS can obtain a competitive prediction performance in terms of average AUC while significantly reducing the

Table 7 AUC values on 11 NASA projects using three classifiers

Project	NB				RF				RIPPER			
	HFS_FS	HFS_BE	SFS	SBE	HFS_FS	HFS_BE	SFS	SBE	HFS_FS	HFS_BE	SFS	SBE
CM1	0.734	**0.746**	0.737	0.744	0.798	**0.803**	0.794	0.802	0.548	**0.554**	0.549	0.552
JM1	**0.685**	0.674	0.676	0.671	0.776	**0.779**	0.742	0.753	0.564	0.561	**0.572**	0.565
KC1	**0.797**	0.782	0.785	0.779	0.793	**0.814**	0.805	0.798	**0.597**	0.593	0.589	0.594
KC2	0.81	**0.838**	0.817	0.824	0.829	**0.833**	0.831	0.822	0.746	0.745	**0.747**	0.751
MC1	**0.968**	0.923	0.942	0.938	**0.927**	0.925	0.916	0.91	0.622	0.625	**0.628**	0.618
MC2	0.8	0.784	0.795	**0.803**	0.694	**0.689**	0.658	0.661	0.585	0.589	0.582	**0.595**
MW1	0.916	**0.92**	0.908	0.911	0.727	0.729	0.723	**0.734**	0.617	0.623	**0.628**	0.619
PC1	0.771	**0.795**	0.788	0.784	0.868	0.866	**0.872**	0.861	0.553	0.556	**0.572**	0.553
PC3	0.775	0.782	0.785	**0.787**	0.867	**0.874**	0.828	0.833	**0.568**	0.565	0.561	0.558
PC4	**0.848**	0.834	0.846	0.839	0.94	0.947	0.944	**0.952**	**0.757**	0.754	0.75	0.752
PC5	0.608	0.613	0.611	**0.623**	0.758	**0.762**	0.757	0.759	0.601	**0.606**	0.594	0.599
AVG	**0.792**	0.79	0.79	0.791	0.816	**0.82**	0.806	0.808	0.614	**0.616**	**0.616**	0.614

Table 8 Runtime of each method on 11 NASA projects using three classifiers

Project	NB				RF				RIPPER			
	HFS_FS	HFS_BE	SFS	SBE	HFS_FS	HFS_BE	SFS	SBE	HFS_FS	HFS_BE	SFS	SBE
CM1	24	22	76	77	24	22	80	77	21	23	80	77
JM1	95	93	332	317	80	105	331	294	95	94	31	316
KC1	30	22	95	104	29	30	101	96	34	27	94	103
KC2	19	13	61	62	19	15	62	59	17	18	62	64
MC1	103	78	312	348	92	98	335	319	91	92	346	349
MC2	23	20	76	75	20	24	80	74	23	20	77	74
MW1	22	15	68	71	18	18	71	74	21	19	67	70
PC1	46	39	142	154	47	41	150	138	50	43	154	152
PC3	51	52	176	165	45	41	176	153	60	47	170	174
PC4	62	63	198	203	53	67	215	207	64	59	189	205
PC5	58	46	186	185	52	49	193	188	56	52	181	189
AVG	48.5	42.1	156.5	160.1	43.5	46.3	163.1	152.6	48.4	44.9	157.4	161.2

computation cost of the wrapper process. That is, HFS can make a tradeoff between the performance improvement and the computation cost. Therefore, HFS is recommended as an effective and efficient feature selection method for defect prediction.

6. Threats to Validity

As a preliminary result, we have performed the experiments on the NASA datasets to explore the generality of our method. Although the datasets have been widely used in many software defect prediction studies, we cannot claim that HFS outperforms the existing methods on other project datasets. We choose three state-of-the-art classifiers, which represent three categories: NB as a probabilistic model, RF a decision-tree model, RIPPER as a rule-based model. For the implementation, we use WEKA and the MINE tools to avoid the potential faults during the implementation process of the experiment. In our experiments, we mainly use F-measure and AUC metrics to measure the effectiveness of the defect prediction performance using different feature selection methods on the three classifiers. However, the choice of these two metrics is based on previously published empirical work and we have not provided any proof.

7. Conclusion and Future Work

In this paper, we have presented a novel feature selection method, HFS, to select an optimized feature subset towards improving fault prediction performance. HFS leverages the HAC algorithm to group the redundant features into the same cluster, and employs two wrapper-based strategies to select the valuable features from each cluster. Experiments on the 11 NASA projects indicate that the proposed method, HFS, can effectively and efficiently select features to improve the performance of fault prediction models. In the future, we would like to employ complex projects that contain more features to validate the generality of our feature selection method.

References

[1] H. Ji, S. Huang, X. Lv, Y. Wu, and Y. Feng, "Empirical studies of a kernel density estimation based naive bayes method for software defect prediction," IEICE Transactions on Information & Systems, vol.E102-D, no.1, pp.75–84, 2019.

[2] 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, no.5, pp.579–606, 2011.

[3] Y. Ma, S. Zhu, Y. Chen, and J. Li, "Kernel cca based transfer learning for software defect prediction," IEICE Transactions on Information & Systems, vol.E100-D, no.8, pp.1903–1906, 2017.

[4] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp.297–308, IEEE, 2016.

[5] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," 2017

IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), pp.78–89, IEEE, 2017.

[6] H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semi-supervised learning with dimension reduction," Proc. 27th IEEE/ACM International Conference on Automated Software Engineering, pp.314–317, 2012.

[7] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp.309–320, IEEE, 2016.

[8] J. Chen, S. Liu, W. Liu, X. Chen, Q. Gu, and D. Chen, "A two-stage data preprocessing approach for software fault prediction," 2014 Eighth International Conference on Software Security and Reliability (SERE), pp.20–29, IEEE, 2014.

[9] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "Fecar: A feature selection framework for software defect prediction," 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC), pp.426–435, IEEE, 2014.

[10] Z. Xu, J. Xuan, J. Liu, and X. Cui, "Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp.370–381, IEEE, 2016.

[11] K. Gao, T.M. Khoshgoftaar, and H. Wang, "An empirical investigation of filter attribute selection techniques for software quality classification," IEEE International Conference on Information Reuse & Integration 2009, IRI'09, pp.272–277, IEEE, 2009.

[12] S. Shivaji, E.J. Whitehead Jr, R. Akella, and S. Kim, "Reducing features to improve bug prediction," 2009 IEEE/ACM International Conference on Automated Software Engineering, pp.600–604, IEEE, 2009.

[13] 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, no.C, pp.170–190, 2015.

[14] H.K. Seifoddini, "Single linkage versus average linkage clustering in machine cells formation applications," Computers & Industrial Engineering, vol.16, no.3, pp.419–426, 1989.

[15] V. Kumar, J.K. Chhabra, and D. Kumar, "Impact of distance measures on the performance of clustering algorithms," Intelligent Computing, Networking, and Informatics, vol.243, pp.183–190, 2014.

[16] D. Cordes, V. Haughton, J.D. Carew, K. Arfanakis, and K. Maravilla, "Hierarchical clustering to measure connectivity in fmri resting-state data," Magnetic Resonance Imaging, vol.20, no.4, pp.305–317, 2002.

[17] D.N. Reshef, Y.A. Reshef, H.K. Finucane, S.R. Grossman, G. McVean, P.J. Turnbaugh, E.S. Lander, M. Michael, and P.C. Sabeti, "Detecting novel associations in large data sets," Science, vol.334, no.6062, pp.1518–1524, 2011.

[18] http://openscience.us/repo/

[19] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: some comments on the nasa software defect datasets, ieee trans softw eng," IEEE Transactions on Software Engineering, vol.39, no.9, pp.1208–1215, 2013.

[20] X. Jin, A. Xu, R. Bie, and P. Guo, "Machine learning techniques and chi-square feature selection for cancer classification using sage gene expression profiles," International Workshop on Data Mining for Biomedical Applications, pp.106–115, Springer, 2006.

[21] J.R. Quinlan, C4. 5: programs for machine learning, Elsevier, 2014.

[22] M. Robnik-Šikonja and I. Kononenko, "Theoretical and empirical analysis of relieff and rrelieff," Machine Learning, vol.53, no.1-2, pp.23–69, 2003.

[23] T.M. Khoshgoftaar, M. Golawala, and J. Van Hulse, "An empirical study of learning from imbalanced data using random forest," 19th IEEE international conference on Tools with Artificial Intelligence 2007, ICTAI 2007, pp.310–317, IEEE, 2007.

[24] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," Computers &amp; Electrical Engineering, vol.40, no.1, pp.16–28, 2014.

[25] http://www.exploredata.net/

[26] F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bulletin, vol.1, no.6, pp.80–83, 1945.

[27] V.B. Kampenes, T. Dybå, J.E. Hannay, and D.I. Sjøberg, "A systematic review of effect size in software engineering experiments," Information and Software Technology, vol.49, no.11-12, pp.1073–1086, 2007.

[28] B.J. Winer, D.R. Brown, and K.M. Michels, Statistical Principles in Experimental Design, McGraw-Hill, New York, 1971.

[29] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.452–463, 2015.

Yiheng Jian is currently pursing the Bachelor degree with the School of Information and Electronic, Beijing Institution of Technology, China. His research interests include software engineering and convex optimization.

Xiao Yu is currently pursuing the PhD degree with the School of Computer Science, Wuhan University, China. His research interests include software engineering and data mining.

Zhou Xu is currently pursuing the PhD degree with the School of Computer Science, Wuhan University, China. His research interests include software defect prediction, feature engineering, and data mining.

Ziyi Ma is currently pursuing the Master degree with the School of Computer Science and Technology, Huazhong University of Science and Technology, China. His research interests include data mining and machine learning.