2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering

MICHAC: Defect Prediction via Feature Selection based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering

Zhou Xu[1], Jifeng Xuan[1], Jin Liu[1*], Xiaohui Cui[2]
[1]State Key Lab of Software Engineering, School of Computer, Wuhan University, Wuhan, China
[2]International School of Software, Wuhan University, Wuhan, China
{zhouxullx, jxuan, jinliu, xcui}@whu.edu.cn

***Abstract*—Defect prediction aims to estimate software reliability via learning from historical defect data. A defect prediction method identifies 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 will hurt the performance of defect prediction methods. Existing work employs feature selection to preprocess defect data to filter out useless features. In this paper, we propose a novel feature selection framework, MICHAC, short for defect prediction via Maximal Information Coefficient with Hierarchical Agglomerative Clustering. MICHAC consists of two major stages. First, MICHAC employs maximal information coefficient to rank candidate features to filter out irrelevant ones; second, MICHAC groups features with hierarchical agglomerative clustering and selects one feature from each resulted group to remove redundant features. We evaluate our proposed method on 11 widely-studied NASA projects and four open-source AEEEM projects using three different classifiers with four performance metrics (precision, recall, F-measure, and AUC). Comparison with five existing methods demonstrates that MICHAC is effective in selecting features in defect prediction.**

Keywords—defect prediction; feature selection; maximal information coefficient

I. INTRODUCTION

Defect prediction aims to estimate software reliability via learning from defect data. Based on the investigation of historical metrics [1], [2], defect prediction identifies the effect of design and testing process over a number of defects. Therefore, defect prediction is often used to help to reasonably allocate limited development and maintenance resources [3], [4], [5]. In defect prediction, each software module is viewed as a class label and a set of features. The class label of a module denotes whether a module is defect-prone or not; the set of features is used to build learnable models.

Many learning models have been proposed for defect prediction [58], [65]. 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 improves when irrelevant and redundant features are eliminated from the original dataset [9], [48]. It is crucial to apply feature selection to defect prediction since feature selection can filter out irrelevant and redundant features by evaluating the contributions of module features. The output of feature selection is a subset of the original feature set. This feature subset is more effective in distinguishing software modules.

However, directly applying feature selection is not suitable to defect prediction. Existing work by Chen et al. [29] and Liu et al. [30] shows that a combination of feature ranking and clustering can improve the performance of predictive models. Such work selects features for defect prediction methods via manual setting the number of feature clusters. In this paper, our proposed approach is motivated by the feature selection method by Chen et al. [29]. In contrast, we enhance feature ranking with a recently proposed relevance measure and improve the feature clustering framework via automatically determining the number of clusters without manual setting.

In this paper, we propose a novel framework, MICHAC, to support feature selection for defect prediction. MICHAC is short for defect prediction via Maximal Information Coefficient with Hierarchical Agglomerative Clustering, which enhances feature selection for defect prediction via a two-stage approach. Frist, MICHAC employs Maximal Information Coefficient (MIC) [21] to rank candidate features to remove irrelevant features; second, MICHAC groups features with Hierarchical Agglomerative Clustering (HAC) and selects one feature from each resulted group. Technically, MIC has the advantage of exploring the hidden relationship between two variables and resisting noise [21], [22]; HAC is effective to cluster features that share a similar pattern [23]. In MICHAC, we determine the optimized number of clusters according to the increment of a statistic measure, called inconsistency coefficient.

We evaluate our proposed approach, MICMAC, by answering three research questions on performance and generality. Experiments are conducted on 11 publicly available projects in the NASA dataset. Experimental results show that MICHAC can effectively select features to improve existing defect prediction methods. On most of projects under evaluation, MICMAC performs the best AUC and F-measure values. We also evaluate the generality of our method on four projects in the AEEEM dataset. This experiment shows that MICHAC can yield a competitive prediction performance against the compared methods in defect prediction in open source projects.

This paper makes the following contributions.

1. We propose a novel feature selection framework, MICHAC, which combines an effective relevance measure technique Maximal Information Coefficient (MIC) with a clustering technique Hierarchical Agglomerative Clustering

* Corresponding author.

978-1-5090-1855-0/16 $31.00 © 2016 IEEE
DOI 10.1109/SANER.2016.34

(HAC). To the best of our knowledge, this is the first time to introduce MIC as the relevance metric into the field of defect prediction. The output of MICHAC is a subset of the original feature set, which can be applied to any defect prediction method, such as Naïve Bayes and Random Forest.

2. We utilize the inconsistency coefficient as a criterion to select the optimized number of clusters when clustering features of software modules. Comparing with existing work [29], [30], we do not need to manually define the number of clusters.

3. We experimentally evaluate MICHAC and five other existing feature selection methods for defect prediction on 15 software projects.

II. Background

In this section, we describe the background of our work and two key techniques used in our proposed method.

A. Defect Prediction via Feature Selection

Defect prediction detects defect-prone modules based on historical defect data. Most of existing work can be unified as a binary-class machine learning problem, i.e., predicting whether a module is defect-prone or not by learning from known modules. A predictive model can be learnt based on a set of extracted features of software modules. A feature of modules could be metrics of software quality, such as the count of lines of source code. In defect prediction, an original feature set may hurt the performance of predictive models since many learning algorithms (e.g., Naïve Bayes [6], [7] or Random Forrest [8], [9]) are sensitive to irrelevant or redundant features.

Feature selection is a family of data preprocessing techniques, which identifies a subset of representative features to replace the original set. Classic feature selection techniques, such as information gain and chi-square statistics [2], [53], are employed to investigate the features of defect prediction. In general, a feature selection algorithm ranks features according to their relevance scores. The larger a score is, the better the attribute is to distinguish between potential classes [12]. Then the top-ranked features are selected as a subset of representative features for predicting defects.

Our work is motivated by the success of feature selection in defect prediction. Comparing with directly applying feature selection to defect prediction, we enhance feature selection in two directions. First, we employ a recently proposed relevance measure, i.e., maximal information coefficient, to improve the ability of selecting relevant features; second, we use hierarchical agglomerative clustering to remove redundant features. We describe two key techniques in our work as follows.

B. Maximal Information Coefficient

Maximal Information Coefficient (MIC) was developed as a robust measure of relevance by Reshef et al. in 2011 [21]. MIC has attracted much interest from academia because of its effectiveness as an indicator of measuring the correlation between two variables [67], [68], [69], [70]. In our work, we use MIC to detect the correlation between each module feature and the class label (i.e., defect-prone or not).

MIC is based on the theory of mutual information; hence we briefly introduce the mutual information before introducing MIC. Let X be a random variable with discrete values. The entropy of X [11] is defined as

$$H(x) = -\sum_{x\in X} p(x)\log p(x) \quad (1)$$

where $p(x)$ is the probability density function of X. Then the joint entropy $H(X,Y)$ of two random variables X and Y is defined as

$$H(X,Y) = -\sum_{y\in Y}\sum_{x\in X} p(x,y)\log p(x,y) \quad (2)$$

To quantify the reduction in uncertainty about variable X after observing variable Y, or by symmetry, the reduction in uncertainty about Y after observing X, the mutual information is introduced as follows.

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \quad (3)$$

i.e., $$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \quad (4)$$

Mutual information is a measure of independence. Based on the above formula, the value of $I(X;Y) = 0$ if variables X and Y are independent; otherwise, the value is greater than zero if they are dependent. In this case, the greater the value is, the more relevant the two variables are [31], [32].

MIC is designed based on an ideal status: if there exists a correlation between two variables X and Y, then a grid could be drawn on the scatter diagram of the two variables to make most of the data points fall into several cells of the grid. By searching for the optimal grid, MIC can calculate correlation of two variables by counting the cells.

Given a finite dataset D, let X and Y be two variables with a sample size n; in our work, X and Y denote a feature and the class label with n software modules. Suppose that the x value and y value of the two variables are divided into x bins and y bins, respectively, allowing empty bins, we call this partition as an x-by-y grid. Let $D|_G$ denote the distribution of the points in D on the cells of a grid G. For each cell of G, the probability mass of the cell is counted by dividing the proportion of points falling into the cell with the total points. Then for different x-by-y partitions, we can obtain different distributions of $D|_G$.

For a specific x-by-y partition, the maximum mutual information of $D|_G$ is defined as

$$I^*(D,x,y) = \max I(D|_G) \quad (5)$$

where $I(D|_G)$ denotes the mutual information of $D|_G$. That is, $I^*(D,x,y)$ is the maximum value of $I(D|_G)$ for all cells of the grid.

For different x-by-y partitions, we can obtain different values of $I^*(D,x,y)$. Then, under different x-by-y partitions, a characteristic matrix $M(D)$ can be constructed by choosing the $I^*(D,x,y)$ of each x-by-y partition as

$$M(D_{x,y}) = \frac{I^*(D,x,y)}{\log(\min\{x,y\})} \quad (6)$$

where normalizing by $\log(min\{x,y\})$ can make the entries of the matrix range from zero to one and guarantee that all

noiseless functions get perfect mutual information scores [21]. Furthermore, the MIC value can be defined as

$$\mathrm{MIC}(D_{x,y}) = \max_{xy<B(n)} \{M(D_{x,y})\} \quad (7)$$

where $B(n)$ is the upper bound of the grid size. In this paper, we follow [21] to set $B(n) = n^{0.6}$ as the default value. In the context of defect prediction, given the class label Y, we calculate the MIC value for each feature X. We rank all original features according to their MIC values and select a subset of these features. This will be further explained in Section III-B.

C. Hierarchical Agglomerative Clustering

We employ a Hierarchical Agglomerative Clustering (HAC) algorithm to divide features into groups and thus to reduce redundant features. In Section III-C, we will later show how to group features via HAC based on the feature values across software modules. Note that our goal is to group features rather than modules and features are characterized via their numeric values in software modules.

The clustering process of HAC is described below. First, the algorithm treats each feature as a cluster and initializes the distance of every two clusters. Then HAC merges the nearest two clusters into a new cluster and calculates the distance between the new cluster and other clusters. The merging process repeats until a pre-defined criteria reaches or all features belong to one group [33], [34]. HAC can form a feature dendrogram of the resulting cluster hierarchy, which serves as a valuable tool in visualization [35].

The distance between two features can be defined by the similarity of these features, such as the cosine similarity and the Pearson correlation coefficient. According to different distance definitions, there are several kinds of commonly used linkage methods for calculating the distance between clusters, such as the single linkage method, the complete linkage method, and the average linkage method. More detailed description can be found in [36].

In this paper, we determine the final number of clusters according to a statistic, called inconsistency coefficient during clustering (in Section III-D). Based on this statistic, our method avoids pointing out a specific number of clusters, which was manually decided in existing work [29], [30].

III. Our Proposed Approach, MICHAC

In this section, we first introduce the framework of our proposed method; then we present the detailed steps in the stages of feature ranking and feature clustering; finally, we illustrate how to determine the number of clusters in the stage of feature clustering.

A. Overview

We propose MICHAC, short for defect prediction via Maximal Information Coefficient (MIC) with Hierarchical Agglomerative Clustering (HAC). In MICHAC, we provide a novel feature selection framework, which combines feature ranking with feature clustering for defect prediction. MICHAC selects an optimized subset of module features. With the support of MICHAC, existing defect prediction methods, such as Naïve Bayes, can benefit from a high-quality training dataset that replaces the original one.

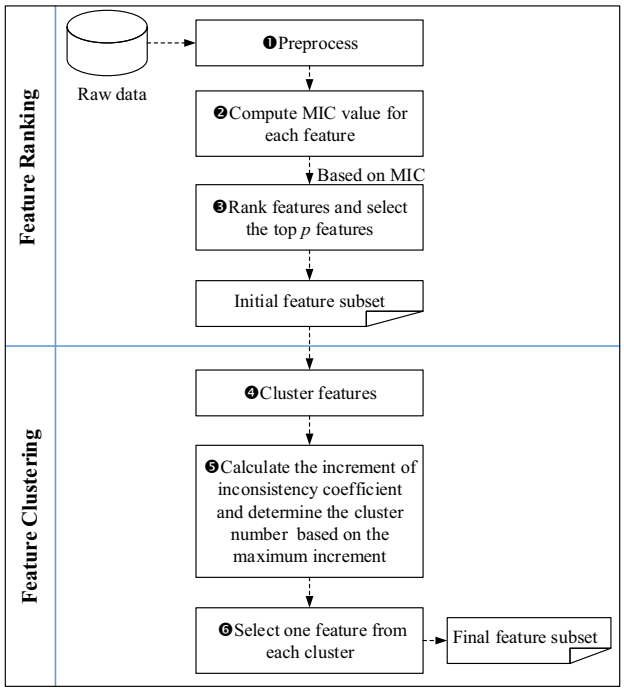

Figure 1. Overview of our proposed approach, MICHAC.

Fig. 1 illustrates the overall structure of MICHAC. This structure consists of two major stages: feature ranking and feature clustering. In the stage of feature ranking, we measure the relevance of features with respect to the class label via a new feature ranking technique based on MIC (in Section II-B); this stage filters out module features that have a low correlation with the class label. In the stage of feature clustering, we cluster features into groups based on HAC (in Section II-C); this stage eliminates redundant features via selecting one feature per cluster. As a result, we construct an optimized subset of module features, which is used to replace the original feature set in defect prediction.

B. Feature Ranking Stage Based on MIC

In the stage of feature ranking, we mainly conduct the relevance analysis between each feature with respect to the class label. That is, features that can distinguish whether a module is defect-prone or not are selected for the next stage (in Section III-C). We rank features independently, without considering any learning algorithm.

The input of feature ranking is a set of defect data, which can be used to build a predictive model in defect prediction. As shown in Fig. 1, feature ranking consists of three major steps. In Step ❶, we preprocess defect data, such as removing features with only one value and non-numeric features. In addition, we convert the class label of modules into binary label. Specifically, we label modules with one or more defects as 1, otherwise as 0. In Step ❷, we calculate the MIC values between each feature x and the class label y based on Equation 7 in Section II-B. In Step ❸, we sort all features based on their MIC values in descending order and

select the top p features which have the highest correlation with respect to the class label. These selected features form the initial subset of features for the next stage. The value of p is defined as $p = \lfloor m/\log m \rfloor$, where m is the number of the original features. The number of top features is set according to Song et al. [24].

C. Feature Clustering Stage Based on HAC

The main goal of the stage of feature clustering is to eliminate redundant features that have similar effect with other features in distinguishing modules with different labels. Note that in contrast to traditional clustering, our goal is to group features rather than instances.

Feature clustering consists of three major steps. In Step ❹, we use the HAC algorithm (in Section II-C) to cluster the features that are selected as the initial subset in the previous stage. 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 [26]. The average linkage between two clusters is defined as the average of the distance between any feature pair between 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:

$$D_{a,b} = \frac{1}{m_a m_b} \sum_{x_i \in U_a, x_j \in U_b} d_{i,j} \tag{8}$$

where $m_a$and m_b are the number of features inside clusters U_a and U_b 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.

$$d_{i,j} = 1 - c_{i,j} \tag{9}$$

For two given features, Pearson correlation coefficient measures the relevance between numeric values of both features in instances [25]. Given two module features x_i and x_j, the observation vectors of features are symbolized as n-dimension vectors, i.e., $\langle x_{i1}, x_{i2}, \dots, x_{in} \rangle$ and $\langle x_{j1}, x_{j2}, \dots, x_{jn} \rangle$, respectively, where values x_{ik} and x_{jk} denote the numeric values of the features x_i and x_j in the kth instance ($k = 1,2, \dots, n$) and n is the number of instances in the dataset. As mentioned in Section III-B, we select top p features as the initial subset of features. Then the correlation coefficient $c_{i,j}$ for features x_i and x_j is calculated by the following formula.

$$c_{ij} = \frac{\sum_{k=1}^{n}(x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)}{\sqrt{\sum_{k=1}^{n}(x_{ik} - \bar{x}_i)^2 \sum_{k=1}^{n}(x_{jk} - \bar{x}_j)^2}} \tag{10}$$

where $\bar{x}_i = \frac{1}{n}\sum_{k=1}^{n} x_{ik}$ and $\bar{x}_j = \frac{1}{n}\sum_{k=1}^{n} x_{jk} (i = 1,2, \dots, p, j = 1,2, \dots, p$, and p is the number of current features after the stage of feature ranking in Section III-B). In this step, we record all information during clustering, including the orders of merging and distances between clusters [37].

Step ❺ determines when to stop the process of clustering according to the recorded information during clustering all features. 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 relatively consistent of one link [73]. 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 one 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.

$$\mathrm{IC}(L_{curr}) = \frac{D_{L_{curr}} - \mathrm{avg}(D_{L_{neighbor}})}{\mathrm{Std}(D_{L_{neighbor}})} \tag{11}$$

where $\mathrm{Std}(D_{L_{neighbor}})$ denotes the standard deviation of all links in its neighbors. We define the increment of IC values between two links as $\Delta_{L_{curr},L_{prev}} = \mathrm{IC}(L_{curr}) - \mathrm{IC}(L_{prev})$, 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 clustering before this link [19], [37]. We will illustrate the calculation of inconsistency coefficient in Section III-D.

In Step ❻, for each cluster, we select the feature with the maximal MIC value as the representative feature. Then s features are finally selected from s clusters. These features form the final subset of features, which replaces the original training set and serves as the input of defect prediction methods.

D. Inconsistency Coefficient in HAC

We use a simple clustering dendrogram to visualize the concept above. Fig.2 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, first, f_1 and f_2 are 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 Equation 9 and Equation 10, 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 Equation 8, 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 $D_{4,3} = (d_{1,3} + d_{2,3})/2$.

The inconsistency coefficient value of the link $L_{4,3}$ (at the depth of two in the dendrogram) can be calculated as follows,

$$\mathrm{IC}(L_{4,3}) = \frac{D_{4,3} - \frac{D_{1,2}+D_{4,3}}{2}}{\mathrm{Std.}} \tag{12}$$

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

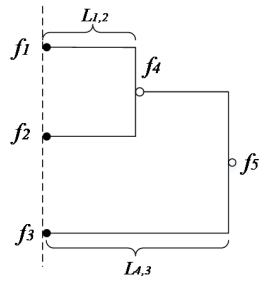

Figure 2. A simple clustering dendrogram with three features.

We determine the final number of clusters according to the increment of inconsistency coefficient. The detail of selecting the number of final clusters is 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.

IV. Experimental Setup

A. Projects in the NASA Dataset

Our experiments are mainly conducted on 11 widely-studied projects in the NASA dataset [13]. The NASA dataset was donated by Menzies and was cleaned by Shepperd et al. [10]. Software modules in these projects are characterized with static code metrics, such as LOC counts, Halstead complexity metrics, McCabe complexity metrics. These metrics closely relate to software quality. For instance, LOC counts measures the number of code lines; the Halstead complexity measure the complexity of the program according to the total number of operators and operands in the program [41]; McCabe Complexity is based on the analysis towards program flow chart and identifies the complexity by calculating the number of direct circles of the strongly connected flow chart [54]. Table I shows the details of the datasets, 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 II lists the maximal increment of inconsistency coefficient and the corresponding number of features selected by our MICHAC method, as mentioned in Section III-D. We find that the maximal increment varies among different projects. The average number and percentage of selected features are 7.5 and 24.6%, respectively.

B. Research Questions

Our evaluation answers three Research Questions (RQs), including both performance and generality.

RQ1. Does our method MICHAC, perform better than state-of-the-art feature selection methods on defect prediction?

TABLE I. 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%

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 Section IV-D). To answer this question, we compare our method against three classic feature selection methods (Chi-Square, Gain Ratio and ReliefF) and two recently proposed feature selection methods (TC [29] and FECAR [30]). Methods in comparison will be described in Section IV-C.

RQ2. Is the feature clustering strategy in MICHAC effective, comparing with the same method without feature clustering?

MICHAC mainly consists of two stages, feature ranking stage and feature clustering stage. Feature ranking, derived from the family of feature selection algorithms, filters out irrelevant features while feature clustering is added to remove redundant features. To investigate the effectiveness of feature clustering, we compare MICHAC with the same algorithm without feature clustering strategy, called MIC for short.

RQ3. Can our method be generalized to other datasets?

The NASA dataset in Section IV-A is widely-studied in defect prediction. Our evaluation in this paper is also conducted on this dataset. However, we tend to investigate whether our proposed method can be applied to other datasets. We employ another popular dataset, AEEEM, to evaluate the generality of our work.

C. Methods in Comparison

We compare our method with three classic feature selection methods in defect prediction [2], [18]: Chi-Square, Gain Ratio and ReliefF. We also compare our work with two existing feature selection methods, which are originally designed for defect prediction: TC [29] and FECAR [30].

These five methods are briefly described below. In the domain of software defect prediction, the Chi-Square (CS) statistic measures the independence between the feature f and the class label c [38], [39]. Gain Ratio (GR) is an updated version of the information gain by penalizing multi-valued attributes to counter the bias [40]. ReliefF (ReF) is

TABLE II. NUMBER OF SELECTED FEATURES BY MAXIMIZING THE INCREMENT OF INCONSISTENCY COEFFICIENT IN MICHAC ON 11 PROJECTS

Project	CM1	JM1	KC1	KC2	MC1	MC2	MW1	PC1	PC3	PC4	PC5	**AVG**
Maximal increment	2.762	1.825	1.978	2.134	2.345	2.913	1.788	2.672	2.602	1.966	2.429	**2.310**
# Selected feature	8	8	9	6	9	5	5	9	8	12	4	**7.5**
% Selected feature	21.1	36.4	40.9	37.5	23.1	12.5	13.2	23.7	21.1	31.6	10.3	**24.6**

an extension of the Relief method that can solve the multi-class learning problem [43]. Note that these three methods are different representative of feature selection methods. The CS, GR, ReF are based on statistics, entropy, and instances, respectively. The number of features in these methods is set to $\lceil log_2 m \rceil$ where m is the number of original features. This setting is suggested by Khoshgoftaar et al. [44] and Gao et al. [2]. Their work shows that various classifiers in defect prediction are appropriate to this setting.

TC is a feature selection method combining feature ranking and feature clustering proposed by Chen et al. [29]. This method uses symmetrical uncertainty to removing irrelevant features and then employs a threshold-based clustering to eliminate redundant features. Another similar framework is FECAR, proposed by Liu et al. [30]. This framework first applies the k-medoids clustering to group the features, and then selects a certain number of features per cluster to form the final feature subset. There are three variants of FECAR; we select the one using information gain, which performs best among three variants. Both of these two methods require manually setting the number of clusters. In our evaluation, we follow the original work to set the number of clusters. In both papers, the final feature numbers are set to $\lceil log_2 m \rceil$, which is derived from Khoshgoftaar et al. [44] and Gao et al. [2].

D. Evaluation Metrics

In this paper, we used F-measure and AUC as main metrics to compare the performance of classifiers.

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 ($n_{f\to f}$), and classifying a defect-free module as defective ($n_{f\to d}$) [28]. Based on the possible output, defective precision, defective recall, defective F-measure are defined as follows:

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

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

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

Defective F-measure is the harmonic mean of defective precision and defective recall. We highlight this evaluation metric in our analysis of classifier performance in this paper. In general, the greater the F-measure is, the better the prediction performance of the classifier is.

2) AUC: AUC is an abbreviation of *Area Under the ROC Curve*, which is widely used in defect prediction [7], [14], [15]. 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 [50], [51]. This curve is used to evaluate the different classifier performance; meanwhile, this curve can be used to compare the influence of different feature sets towards a classifier. Nevertheless, since a curve like ROC cannot quantitatively describe the classifier performance, it is common to calculate the relative area under the curve (so-called AUC) as a single scalar value [66]. In defect prediction, AUC is regarded as a performance metric of various classifiers over different software projects [7]. In general, a higher AUC value indicates that the classifier performance is better.

In experiments, we performed 10-fold cross validation when training classifiers on the selected features throughout this paper, to avoid any potential problem of overfitting particular training and test sets within a specific project. In 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. The final result of performance evaluation is the average of the 10 results.

E. Defect Prediction Models

In order to compare the performance of feature selection methods, we employ three representative classifiers in defect prediction, Naive Bayes (NB) [45], Random Forest (RF) [46], and Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [49]. The reason we choose these classifiers is that these classifiers fall into three different families of learning methods. NB is a probabilistic classifier [32]; RF is a decision-tree classifier [47]; and RIPPER is a rule-based classifier [49], [55].

In this paper, experiments were conducted on a workstation with an Intel Core i7-4790 CPU with 3.60 GHz. We implemented feature selection methods in Java with the Weka package and used the classifiers inside Weka with the default parameter settings [51]. We calculate MIC values via the MINE toolkit [56] and implement MICHAC in Matlab 7.0.

V. EXPERIMENTAL RESULTS

We present the experimental results to answer our three research questions on performance and generality in this section.

TABLE III. AVERAGE PERFORMANCE OF 11 NASA PROJECTS WITH THREE CLASSIFIERS ON PRECISION, RECALL, AND F-MEASURE

Model	Metric	Full	MICHAC	CS	GR	ReF	TC	FECAR
NB	P	0.407	0.427	0.440	**0.454**	0.350	0.410	0.442
	R	**0.429**	0.397	0.332	0.366	0.424	0.342	0.376
	F	0.350	**0.373**	0.359	0.360	0.355	0.346	0.367
	W/D/L	7/1/3		8/1/2	7/0/4	6/0/5	6/1/4	6/0/5
RF	P	0.540	**0.560**	0.488	0.498	0.468	0.526	0.501
	R	0.288	0.311	0.302	0.311	0.258	0.287	**0.321**
	F	0.372	**0.392**	0.366	0.377	0.326	0.367	0.385
	W/D/L	4/1/7		8/1/2	8/0/3	9/0/2	7/0/4	7/0/4
RIPPER	P	0.488	**0.550**	0.535	0.536	0.423	0.498	0.530
	R	0.268	**0.280**	0.266	0.255	0.171	0.237	0.267
	F	0.333	**0.351**	0.345	0.334	0.230	0.309	0.344
	W/D/L	7/0/4		4/0/7	7/0/4	9/0/2	8/0/3	5/0/6

A. *RQ1, Does our method MICHAC, perform better than state-of-the-art feature selection methods on defect prediction?*

As mentioned in Sections IV-C, we compare our method MICHAC with three widely-studied feature selection methods (CS, GR, and ReF) and two recently-proposed methods in defect prediction (TC and FECAR).

Table III records the average defective precision, recall and F-measure of all 11 NASA projects with five different 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, short for Win/Draw/Loss, presents the number of projects, on which MICHAC performs better than, the same as, or worse than another method, in terms of F-measure [42], [64].

Table III shows that on all three classifiers, MICHAC performs better F-measure values than all the other methods. Besides MICHAC, FECAR performs well. For NB classifier, MICHAC achieves the best average F-measure value, but fails in the best precision or recall. Regarding the average precision, MICHAC is inferior to the CS, GR and FECAR; regarding the average recall, MICHAC is inferior to Full and ReF. For RF classifier, MICHAC can achieve the best precision and F-measure values. For RIPPER classifier, MICHAC can achieve the best values in terms of all the three metrics, comparing with all other methods. Among all feature selection methods in our experiment, RF classifier with MICHAC reaches the best F-measure as well as the best precision.

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

Fig. 3 shows the box-plots of F-measure values, with six methods for three classifiers on 11 projects. For the NB classifier, the median value by MICHAC is higher than that by GR, ReF, and TC, while is very similar with that by CS and FECAR. In addition, the maximum by MICHAC is much higher than that by other methods except ReF. For RF classifier, the median value by MICHAC is much higher

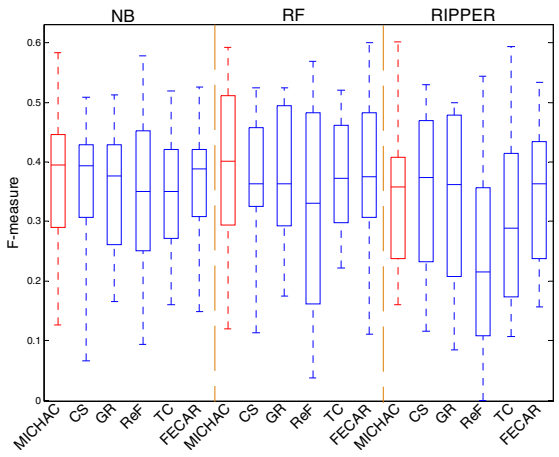

Figure 3. Box-plots for F-measure on 11 NASA projects with three classifiers.

than that by all other methods, and the maximum by MICHAC is much higher than that by other methods except FECAR. For RIPPER classifier, the median value by MICHAC is much higher than that by ReF and TC, while is a litter lower than that by CS. In addition, the maximum by MICHAC is much higher than that by all other methods except TC.

We perform the Wilcoxon signed-rank test [71] to analyze whether the performance values of MICHAC is statistically significant different with 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 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 [62]. In additional, we compute the effect size, Cliff's Delta (d) [72], to quantify the amount of difference between two methods. A positive d indicates that the performance of the prevision method has a greater effect than that of the latter method [16], [63]. For the sake of space limitation, in this paper we only list the detailed p-values and d-values in term of AUC for the NASA dataset since AUC is widely used as a performance evaluation metric in defect prediction [4], [7], [48].

Tables IV, V, and VI present the detailed AUC values of each project on three classifiers with the p-values and d-values. From these tables, we can observe that MICHAC can achieve the best AUC values on NB classifier and RIPPER classifier, but a little weaker than the Full method on RF classifier. The Full method and MICHAC in Table V shows that there exists a threat that feature selection hurts the performance for a specific defect prediction method. The Win/Draw/Loss records also indicate that MICHAC wins other methods on most projects on three classifiers in term of AUC metric. An exception is the Full method on RF classifier, where MICHAC wins and losses on 4 and 7 out

TABLE IV. AUC VALUES ON 11 NASA PROJECTS USING NAÏVE BAYES WITH THE WILCOXON TEST (P-VALUE) AND CLIFF'S DELTA (D)

Project	Full	MICHAC	CS	GR	RF	TC	FECAR
CM1	0.694	0.725	0.729	0.75	0.752	0.72	0.75
JM1	0.678	0.642	0.629	0.629	0.624	0.682	0.636
KC1	0.791	0.791	0.783	0.774	0.782	0.798	0.78
KC2	0.832	0.836	0.816	0.817	0.827	0.833	0.825
MC1	0.892	0.917	0.768	0.812	0.845	0.87	0.883
MC2	0.717	0.722	0.637	0.637	0.627	0.666	0.63
MW1	0.728	0.774	0.714	0.714	0.737	0.736	0.735
PC1	0.768	0.788	0.663	0.701	0.673	0.795	0.777
PC3	0.743	0.785	0.783	0.778	0.784	0.735	0.784
PC4	0.825	0.837	0.823	0.835	0.823	0.807	0.823
PC5	0.69	0.66	0.651	0.617	0.64	0.708	0.624
AVG	0.760	**0.771**	0.727	0.733	0.738	0.759	0.750
W/D/L	8/1/2		10/0/1	10/0/1	10/0/1	7/0/4	10/0/1
p-value	0.169		**0.006**	**0.016**	**0.021**	0.374	**0.021**
d	**0.116**		**0.355**	**0.314**	**0.240**	**0.041**	**0.190**

TABLE V. AUC VALUES ON 11 NASA PROJECTS USING RANDOM FORREST WITH THE WILCOXON TEST (P-VALUE) AND CLIFF'S DELTA (D)

Project	Full	MICHAC	CS	GR	RF	TC	FECAR
CM1	0.782	0.795	0.732	0.728	0.75	0.812	0.728
JM1	0.76	0.742	0.721	0.721	0.733	0.692	0.729
KC1	0.83	0.802	0.793	0.744	0.789	0.796	0.793
KC2	0.825	0.812	0.782	0.808	0.798	0.81	0.795
MC1	0.923	0.93	0.879	0.918	0.916	0.929	0.937
MC2	0.682	0.658	0.588	0.588	0.575	0.691	0.573
MW1	0.716	0.737	0.696	0.696	0.641	0.719	0.754
PC1	0.844	0.858	0.839	0.854	0.844	0.84	0.804
PC3	0.864	0.848	0.835	0.844	0.85	0.822	0.849
PC4	0.944	0.942	0.887	0.916	0.887	0.845	0.914
PC5	0.8	0.754	0.759	0.749	0.759	0.777	0.755
AVG	**0.815**	0.807	0.774	0.779	0.777	0.794	0.785
W/D/L	4/0/7		10/0/1	11/0/0	9/0/2	8/0/3	7/0/4
p-value	0.197		**0.004**	**0.003**	**0.008**	0.248	0.068
d	-0.074		**0.223**	**0.207**	**0.174**	**0.099**	**0.157**

TABLE VI. AUC VALUES ON 11 NASA PROJECTS USING RIPPER WITH THE WILCOXON TEST (P-VALUE) AND CLIFF'S DELTA (D)

Project	Full	MICHAC	CS	GR	RF	TC	FECAR
CM1	0.516	0.541	0.542	0.513	0.494	0.51	0.55
JM1	0.553	0.563	0.56	0.554	0.541	0.54	0.545
KC1	0.578	0.587	0.59	0.582	0.588	0.599	0.594
KC2	0.698	0.749	0.722	0.641	0.711	0.727	0.71
MC1	0.626	0.619	0.614	0.632	0.623	0.614	0.612
MC2	0.576	0.584	0.573	0.582	0.595	0.601	0.582
MW1	0.633	0.602	0.682	0.682	0.581	0.63	0.657
PC1	0.581	0.558	0.569	0.578	0.5	0.554	0.565
PC3	0.538	0.574	0.527	0.551	0.542	0.539	0.561
PC4	0.73	0.761	0.689	0.689	0.528	0.6	0.694
PC5	0.623	0.591	0.615	0.61	0.54	0.628	0.623
AVG	0.605	**0.612**	0.608	0.601	0.568	0.595	0.608
W/D/L	7/0/4		6/0/5	7/0/4	8/0/3	7/0/4	6/0/5
p-value	0.35		0.593	0.477	**0.016**	0.424	0.722
d	**0.041**		**0.041**	**0.041**	**0.397**	**0.058**	-0.025

of 11 projects, respectively. Although only nearly half of p-values are lower than 0.05, the d-values indicate that in almost all cases, MICHAC have a greater effect than other methods, except for the Full method on RF classifier (-0.074) and FECAR method on RIPPER classifier (-0.025).

To conclude the above observation, our method can yield better prediction results than the compared methods of feature selection in defect prediction.

B. *RQ2, Is the feature clustering strategy in MICHAC effective, comparing with the same method without feature clustering?*

As mentioned in Section III, our proposed method, MICHAC, can be viewed as a combination of two stages, feature ranking and feature clustering. Comparing with general feature selection, such as CS or GR, the stage of feature clustering in MICHAC can introduce further refinement of the subset of features. We compare MICHAC with the method only used MIC to illustrate the influence of redundant features in defect prediction.

Table VII shows the average defective precision, recall, F-measure, and AUC values of 11 NASA projects on three classifiers as well as the p-values and d-values with respect to AUC. We can observe that all the four metrics of MICHAC are better than those of MIC on all three classifiers. The Win/Draw/Loss records show that MICHAC wins MIC as well, especially on NB classifier; MICHAC wins MIC on all projects in terms of F-measure and AUC. The p-values indicate that there exists significant difference between the performance of MICHAC method and that of MIC method on NB classifier and RF classifier. The non-negative d-values indicate that effect size of MICHAC method is greater than that of MIC method on NB classifier and RF classifier.

Fig. 4 shows the box-plots of F-measure and AUC values, for MICHAC and MIC, with three classifiers on 11 projects. The median values of both metrics of MICHAC are higher than those of MIC, except on RIPPER classifier. Meanwhile, for F-measure, the maximum by MICHAC is higher than that by MIC method on NB classifier and RF classifier. For AUC, the maximum by MICHAC is higher than that by MIC on NB classifier and RIPPER classifier while is a litter lower on RIPPER classifier. Moreover, the minimum by MICHAC is higher than that by MIC for two metrics on NB classifier and RF classifier.

Experimental results show that MICHAC outperforms MIC. The reason is that HAC keeps the features that can

TABLE VII. COMPARISON BETWEEN MICHAC AND MIC ON 11 NASA PROJECTS

Model	NB		RF		RIPPER	
Metric	MICHAC	MIC	MICHAC	MIC	MICHAC	MIC
P	**0.427**	0.405	**0.560**	0.539	**0.550**	0.544
R	**0.397**	0.347	**0.311**	0.310	**0.280**	0.270
F	**0.373**	0.309	**0.392**	0.388	**0.351**	0.347
W/D/L	11/0/0		7/0/4		6/0/5	
AUC	**0.771**	0.715	**0.807**	0.780	**0.612**	0.608
W/D/L	11/0/0		9/0/2		5/1/5	
p-value	**0.003**		**0.021**		0.878	
d	**0.438**		**0.165**		-0.008	

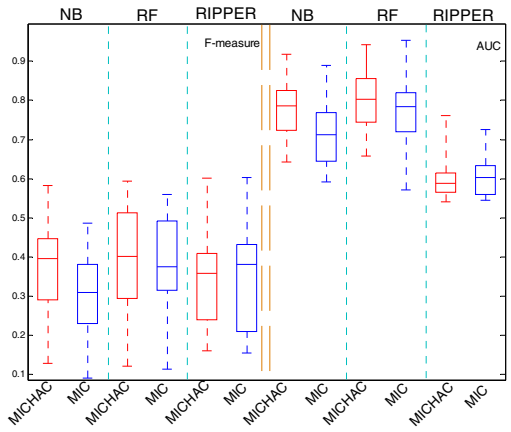

Figure 4. Box-plots for F-measure and AUC on 11 projects with three classifiers.

distinguish the class label. In details, we first remove irrelevant features, which have little beneficial effect on defect prediction. Then we employ a HAC to group features that share a similar pattern into a cluster and select one feature with the highest MIC value from each cluster to construct the final features subset. Thus, all the selected features can almost make a contribution to identify the class label.

To sum up, MICHAC seeks a better feature subset that are the most useful for defect prediction. The results confirm that redundant features can indeed hurt the prediction performance.

C. *RQ3, Can our method be generalized to other datasets?*

To further investigate the generality of our MICHAC on other software projects, we apply our method to four software projects in the AEEEM dataset. This dataset was collected by D'Ambros et al [17], aiming to perform defect prediction at the granularity of class level. Features in this dataset include the change metrics, source code metrics, entropy of source code metrics, churn of source code metrics, etc. Table VIII shows the details of these four projects.

Table IX presents the result comparison between MICHAC and five feature selection methods in defect prediction. On NB classifier and RIPPER classifier, the precision, recall, and F-measure do not perform the best; but on RF classifier, the recall and F-measure values of MICHAC are superior to those of all other methods. In addition, the AUC values by MICHAC are higher than those by all the other methods on NB classifier and RIPPER classifier. Although the p-values with respect to AUC indicate that there exists no significant difference between the performance of MICHAC and those of the compared methods on three classifiers, the Cliff's Delta d-values suggest the superiority of the effect size of MICHAC to those of the compared methods, except the two cases with the Full method on RF classifier (-0.125) and CS method on RIPPER classifier (-0.063).

As mentioned above, we find that our feature selection method can be applied to other software projects for defect prediction and can obtain a competitive prediction performance, comparing with results of other methods.

TABLE VIII. FOUR PROJECTS IN THE AEEEM DATASET

Project	# features	# modules	# defective modules	% defective modules
Eclipse-JDT	77	997	206	20.7%
Equinox	77	324	129	39.8%
Mylyn	77	1862	245	13.2%
Eclipse-PDE	77	1497	209	14.0%

TABLE IX. AVERAGE PERFORMANCE OF FOUR AEEEM PROJECTS WITH THREE CLASSIFIERS ON FOUR METRICS

Model	Metric	Full	MICHAC	CS	GR	ReF	TC	FECAR
NB	P	0.538	0.549	**0.594**	0.567	0.491	0.547	0.582
	R	0.418	0.412	**0.419**	0.343	0.316	0.343	0.414
	F	0.468	0.469	**0.491**	0.426	0.376	0.420	0.481
	W/D/L	2/0/2		2/0/2	3/0/1	3/0/1	3/0/1	1/0/3
	AUC	0.771	**0.780**	0.772	0.734	0.733	0.759	0.772
	W/D/L	3/0/1		2/0/2	3/0/1	4/0/0	3/0/1	2/0/2
RF	P	**0.673**	0.667	0.610	0.528	0.535	0.601	0.608
	R	0.421	**0.425**	0.399	0.383	0.376	0.417	0.406
	F	0.490	**0.500**	0.471	0.427	0.426	0.475	0.465
	W/D/L	4/0/0		2/0/2	4/0/0	3/0/1	2/0/2	3/0/1
	AUC	**0.839**	0.832	0.776	0.748	0.806	0.794	0.784
	W/D/L	1/0/3		4/0/0	4/0/0	3/0/1	4/0/0	4/0/0
RIPPER	P	0.569	0.574	0.579	0.537	0.532	**0.590**	0.576
	R	0.393	0.398	**0.400**	0.346	0.302	0.390	0.381
	F	0.446	0.449	**0.458**	0.392	0.350	0.448	0.447
	W/D/L	3/0/1		1/0/3	3/0/1	3/0/1	2/0/2	2/0/2
	AUC	0.662	**0.669**	0.659	0.643	0.629	0.660	0.658
	W/D/L	3/0/1		2/0/2	4/0/0	3/0/1	2/0/2	2/0/2

VI. THREATS TO VALIDITY

In this subsection, we discuss several main types of validity threats that affect our studies.

External validity. Threats to external validity occur when the results of our experiments cannot be generalized. As a preliminary result, we performed our experiments on the NASA and AEEEM datasets to explore the generality of our method. The NASA dataset has been widely used in many software defect prediction studies, thus it provides us a way to compare our method with some representative feature selection methods for defect predict. Additional studies are needed to evaluate how well our method can be generalized.

Internal validity. We list several concerns about the bias in classifier selection and the incorrect implementation process of experiments. To avoid these threats, 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.

Construct validity. In experiments, we mainly use F-measure and AUC metrics to measure the effectiveness of

the defect prediction performance using different feature selection methods on three classifiers. However, the choice of these two metrics is based on previously published empirical work and we have not provided any proof. A potential solution is to theoretically optimize the benefit of predictive results [27]. A further discussion about metrics in defect prediction is expected in the future.

VII. Related Work

A. Classic Models in Defect Prediction

Many researchers have proposed various models for defect prediction in terms of within-projects defect prediction. Zimmermann et al. [57] showed that employing network analysis based on a dependency graph can be efficient to predict the central program units, which are defect-prone on Windows Server 2003. Thwin et al. [58] employed Ward neural network and General Regression neural network models for predicting the number of software defects. They found that those two models can achieve a good performance when using the object-oriented metrics. Recently, Jing et al. [59] proposed a novel cost-sensitive dictionary-learning model for defect prediction. This method can achieve the best recall and F-measure metrics among existing methods.

For cross-project defect prediction, Zimmermann et al. [60] applied decision tree and logistic regression models for cross-project defect prediction on 12 real-world software application projects. They found that the data and the process metric are important for prediction performance. Nam et al. [61] introduced transfer learning to utilize the feature information of the source projects to help the target projects for defect prediction. They have shown that this method can achieve an acceptable prediction performance for target projects that have limited historical training dataset. Jing et al. [74] proposed a novel method employing canonical correlation analysis for heterogeneous cross-project defect prediction on 14 open software projects. This method can match the distribution of the source and the target datasets maximize the correlation between both datasets.

B. Feature Selection in Defect Prediction

A number of prior studies have investigated feature selection methods on predicting defective software modules. Gao et al. [20] 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] presented a comparative investigation to evaluate their proposed hybrid feature selection method, which first uses feature ranking to reduce the search space and then applies feature subset selection. Results indicated that removing 85 percent of features does not adversely affect prediction performance.

Prior studies have shown that feature selection can help to identify defect-prone changes [48]. In order to investigate different feature selection methods to classification-based bug prediction, Shivaji et al. [52] utilized six feature selection methods to iteratively remove irrelevant features until achieving the best performance of F-measure.

Different from our feature selection framework, all the feature selection methods of the above literatures are only aimed at eliminating irrelevant features with respect to the class label, while our framework considers removing redundancy within features.

Recently work proposes a combined framework to apply feature selection to eliminate both irrelevant and redundant features from the original dataset. Chen et al. [29] proposed a two-stage data preprocessing framework, TC, which combines feature selection and instance reduction [53]. In the feature selection phase, they proposed a new algorithm using feature selection and threshold-based clustering. Liu et al. [30] proposed a new feature selection framework, FECAR, to conduct feature clustering and feature ranking. FECAR first clusters features via k-medoids method and then select several representative features from each cluster.

In our paper, the MICHAC approach is similar to the feature selection framework in [29]. The differences between MICHAC and TC as well as FECAR are as follows. TC uses symmetrical uncertainty to conduct feature ranking and feature clustering while our framework MICHAC uses maximal information coefficient with hierarchical agglomerative clustering to conduct feature ranking and feature clustering, respectively. Similarly, FECAR employs the k-medoids clustering to detect representative features. Meanwhile, as mentioned in Section III-B, in MICHAC, the number of clusters is automatically determined according to the increment of inconsistency coefficient.

VIII. Conclusion and Future Work

In this paper, we have presented a novel feature selection method, MICHAC, to select an optimized feature subset towards improving defect prediction performance. The method involves the following two stages: in the first stage, we introduce MIC statistic to select the highly relevant features with respect to the class label; in the second stage, we leverage HAC algorithm to eliminate the redundant features. Experiments on 11 NASA projects and four additional AEEEM projects indicate that the proposed method, MICHAC, can perform competitive results and scale to open source projects for defect prediction tasks.

In the future, we plan to apply our method to other classifiers to study the effect of feature selection on classifiers for defect prediction. Meanwhile, we would like to employ complex projects that contain more features to validate the generality of our feature selection method. In addition, we plan to apply our method, MICHAC, to cross-project defect prediction to identify defects with different development backgrounds.

Acknowledgment

This work is partly supported by the grants of National Natural Science Foundation of China (No.61572374, No.U1135005, No.61502345) and the Fundamental Research Funds for the Central Universities (No.2042014kf0272, No.2014211020201).

References

[1] F. Rahman, D. Posnett, Devanbu P. Recalling the imprecision of cross-project defect prediction. Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. p.61. ACM, 2012.

[2] K. Gao, T.M. Khoshgoftaar, H. Wang, et al. Choosing software metrics for defect prediction: an investigation on feature selection techniques . Software Practice & Experience, 41(5):579-606, 2011.

[3] M. Shepperd, D. Bowes, T. Hall. Researcher Bias: The Use of Machine Learning in Software Defect Prediction. IEEE Transactions on Software Engineering, 40(6):603-616, 2014.

[4] Q. Song, Z. Jia, M. Shepperd, et al. A general software defect-proneness prediction framework. Software Engineering, IEEE Transactions on, 37(3): 356-370, 2011.

[5] X. Yang, K. Tang, X. Yao. A Learning-to-Rank Approach to Software Defect Prediction. IEEE Transactions on Reliability, 64(1): 234-246, 2015.

[6] T. Menzies, J. Greenwald, A. Frank. Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering, 33(1): 2-13, 2007.

[7] S. Lessmann, B. Baesens, C. Mues, et al. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. Software Engineering, IEEE Transactions on, 34(4): 485-496, 2008.

[8] L. Guo, Y. Ma, B. Cukic, et al. Robust prediction of fault-proneness by random forests. Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on IEEE, 417-428, 2004.

[9] H. Lu, B. Cukic, M. Culp. Software defect prediction using semi-supervised learning with dimension reduction Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. 314-317. IEEE, 2012.

[10] M. Shepperd, Q. Song, Z. Sun, et al. Data Quality: Some Comments on the NASA Software Defect Datasets. IEEE Transactions on Software Engineering, 39(9):1208-1215, 2013.

[11] C. E. Shannon. A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review, 5(1): 3-55, 2001.

[12] J. Xuan, M. Monperrus. Learning to Combine Multiple Ranking Metrics for Fault Localization. Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME 2014), Sept. 28-Oct. 3, pp. 191-200, 2014.

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

[14] X. Xia, D. Lo, S. McIntosh, et al. Cross-project build co-change prediction. Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on. 311-320. IEEE,2015.

[15] A. Panichella, R. Oliveto, A. De Lucia. Cross-project defect prediction models: L'Union fait la force. Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. 164-173. IEEE, 2014..

[16] P. He, B. Li, X. Liu, Y. Ma. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59: 170-190, 2015.

[17] M. D'Ambros, M. Lanza, R. Robbes. An extensive comparison of bug prediction approaches. Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 31-41, 2010.

[18] T. M. Khoshgoftaar, K. Gao, A. Napolitano. An empirical study of feature ranking techniques for software quality prediction. International Journal of Software Engineering and Knowledge Engineering, 22(02): 161-183, 2012.

[19] M. GhasemiGol, H. S. Yazdi, R. Monsefi. A new hierarchical clustering algorithm on fuzzy data (FHCA). International Journal of Computer and Electrical Engineering, 2(1): 1793-8163, 2010.

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

[21] D. N. Reshef, Y. A. Reshef, H. K. Finucane, et al., Detecting Novel Associations in Large Data Sets, Science, 2011.

[22] D. Reshef, Y. Reshef, M. Mitzenmacher, et al. Equitability analysis of the maximal information coefficient, with comparisons. arXiv preprint arXiv:1301.6314, 2013.

[23] W. H. E. Day, H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. Journal of Classification, 1(1):7-24, 1984.

[24] Q. Song, J. Ni, G. Wang. A Fast Clustering-Based Feature Subset Selection Algorithm for High-Dimensional Data. IEEE Transactions on Knowledge & Data Engineering, 25(1):1-14, 2013.

[25] V. Kumar, J. K. Chhabra, D. Kumar. Impact of distance measures on the performance of clustering algorithms. Intelligent Computing, Networking, and Informatics. Springer India, 183-190, 2014.

[26] H. K. Seifoddini. Single linkage versus average linkage clustering in machine cells formation applications. Computers & Industrial Engineering, 16(3): 419-426, 1989.

[27] H. Jiang, J. Xuan, Z. Ren. Approximate backbone based multilevel algorithm for next release problem. Proceedings of the 12th annual conference on Genetic and evolutionary computation. ACM, 1333-1340, 2010.

[28] S. Kim, H. Zhang, R. Wu, et al. Dealing with noise in defect prediction. Software Engineering (ICSE), 2011 33rd International Conference on. 481-490. IEEE, 2011.

[29] J. Chen, S. Liu, W. Liu, et al. A Two-Stage Data Preprocessing Approach for Software Defect prediction. Software Security and Reliability (SERE), 2014 Eighth International Conference on. 20 - 29. IEEE, 2014.

[30] S. Liu, X. Chen, W. Liu, et al. FECAR: A Feature Selection Framework for Software Defect Prediction. 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC). IEEE Computer Society, 426-435, 2014.

[31] H. Liu, J. Sun, L. Liu, et al. Feature selection with dynamic mutual information. Pattern Recognition, 42(7):1330 – 1339, 2009.

[32] K. P. Murphy. Machine learning: a probabilistic perspective. Mathematics Education Library, 58(8):27-71, 2012.

[33] C. Ding, X. He. Cluster merging and splitting in hierarchical clustering algorithms. Proceedings of the 2002 IEEE International Conference on Data Mining. IEEE Computer Society, 139, 2002.

[34] C. H. Park. A Feature Selection Method Using Hierarchical Clustering. Lecture Notes in Computer Science, 1-6, 2013.

[35] D. Ienco, R. Meo. Exploration and Reduction of the Feature Space by Hierarchical Clustering. SDM, 2008.

[36] A. K. Jain, R. C. Dubes. Algorithms for clustering data. Prentice-Hall, Inc., 1988.

[37] http://cn.mathworks.com/help/stats/hierarchical-clustering.html.

[38] C. D. Manning, P. Raghavan, H. Schütze. Introduction to information retrieval, 43(3):824-825. Citeseer, 2008.

[39] X. Jin, A. Xu, R. Bie, et al. Machine Learning Techniques and Chi-Square Feature Selection for Cancer Classification Using SAGE Gene Expression Profiles . Data Mining for Biomedical Applications, 106-115, 2006.

[40] J.R. Quinlan, C4.5: programs for machine learning. Morgan Kaufmann Pub. San Mateo, California, 1993.

[41] M. H. Halstead. Elements of Software Science (Operating and programming systems series). Elsevier Science Inc., 1977.

[42] G. I. Webb. Multiboosting: A technique for combining boosting and wagging. Machine learning, 40(2): 159-196, 2000.

[43] M. Robnik-Šikonja, I. Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. Machine learning, 53(1-2): 23-69, 2003.

[44] T. M. Khoshgoftaar, M. Golawala, J. V. Hulse. An empirical study of learning from imbalanced data using random forest. Tools with

Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on. 2: 310-317. IEEE, 2007.

[45] E. Arisholm, L. C. Briand, E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate defect prediction models . Journal of Systems & Software, 83(1):2–17, 2010.

[46] T. K. Ho. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis & Machine Intelligence, 20(8):832 – 844, 1998.

[47] C. Vens, F. Costa. Random Forest Based Feature Induction Data Mining (ICDM), 2011 IEEE 11th International Conference on. 744 - 753. IEEE, 2011.

[48] S. Shivaji, E. J. Whitehead, R. Akella, et al. Reducing features to improve code change-based bug prediction. Software Engineering, IEEE Transactions on, 39(4): 552-569, 2013.

[49] W. W. Cohen. Fast effective rule induction. Proceedings of the twelfth international conference on machine learning. 115-123, 1995.

[50] T. Fawcett. An introduction to ROC analysis. Pattern Recognition Letters, 27(8):861–874, 2006.

[51] I. H. Witten, E. Frank. Data Mining: Practical Machine Learning Tools and Techniques. Biomedical Engineering Online, 5:51(1), 2005.

[52] S. Shivaji, J. E. J. Whitehead, R. Akella, et al. Reducing features to improve bug prediction. Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 600-604, 2009.

[53] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, X. Wu. Towards effective bug triage with software data reduction techniques. IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 1, pp. 264-280, Jan. 2015.

[54] T. J. McCabe. A complexity measure. Software Engineering, IEEE Transactions on, 308-320, 1976.

[55] M. E. R. Ruiz. Combining machine learning and hierarchical structures for text categorizatio.The University of Iowa, 2001.

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

[57] T. Zimmermann, N. Nagappan. Predicting defects using network analysis on dependency graphs. Proceedings of the 30th international conference on Software engineering. 531-540. ACM, 2008.

[58] M. M. T. Thwin, T. S. Quah. Application of neural networks for software quality prediction using object-oriented metrics. Journal of systems and software, 76(2): 147-156, 2005.

[59] X. Y. Jing, S. Ying, Z. W. Zhang, et al. Dictionary learning based software defect prediction. Proceedings of the 36th International Conference on Software Engineering. 414-423. ACM, 2014.

[60] T. Zimmermann, N. Nagappan, H. Gall, et al. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 91-100. ACM, 2009.

[61] J. Nam, S. J. Pan, S. Kim. Transfer defect learning. Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 382-391, 2013.

[62] A.P. Field. Discovering statistics using SPSS for Windows: Advanced techniques for the beginner. Discovering Statistics Using SPSS for Windows: Advanced Techniques for Beginners. Sage Publications, Inc., 2000.

[63] P. He, B. Li, Y. Ma. Towards Cross-Project Defect Prediction with Imbalanced Feature Sets. arXiv preprint arXiv:1411.4228, 2014.

[64] W. Liu, S. Liu, Q. Gu, et al. FECS: A Cluster Based Feature Selection Method for Software Fault Prediction with Noises. Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. 2: 276-281. IEEE, 2015.

[65] B. Turhan, A. Bener. Analysis of Naive Bayes' assumptions on software fault data: An empirical study. Data & Knowledge Engineering, 68(2): 278-290, 2009.

[66] K. Zuva, T. Zuva. Evaluation of Information Retrieval Systems. International Journal of Computer Science & Information Technology (IJCSIT), 4(3), 2012.

[67] Y. Zhang, W. Zhang, Y. Xie. Improved heuristic equivalent search algorithm based on maximal information coefficient for Bayesian network structure learning. Neurocomputing, 117: 186-195, 2013.

[68] E. Martinez-Gomez, M. T. Richards, D. S. P. Richards. Distance correlation methods for discovering associations in large astrophysical databases. The Astrophysical Journal, 781(1): 39, 2014.

[69] H. M. Liu, N. Rao, D. Yang, et al. A novel method for identifying SNP disease association based on maximal information coefficient. Genetics and molecular research: GMR, 13(4): 10863, 2014.

[70] C. Lin, T. Miller, D. Dligach, et al. Maximal information coefficient for feature selection for clinical document classification. ICML Workshop on Machine Learning for Clinical Data. Edingburgh, UK. 2012.

[71] F. Wilcoxon. Individual comparisons by ranking methods. Biometrics bulletin, 80-83, 1945.

[72] G. Macbeth, E. Razumiejczyk, R. D. Ledesma. Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. Universitas Psychologica, 10(2): 545-555, 2011.

[73] D. Cordes, V. Haughton, J. D. Carew, et al. Hierarchical clustering to measure connectivity in fMRI resting-state data. Magnetic resonance imaging, 20(4): 305-317, 2002.

[74] X. Jing, F. Wu, X. Dong, et al. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 496-507. ACM, 2015