Published in IET Software
Received on 7th August 2011
Revised on 1st May 2012
doi: 10.1049/iet-sen.2011.0138

ISSN 1751-8806

Artificial neural network-based metric selection for software fault-prone prediction model

C. Jin[1] *S.-W. Jin*[2] *J.-M. Ye*[1]

[1]*Department of Computer Science, Central China Normal University, Wuhan 430079, People's Republic of China*
[2]*Faculté des Sciences et Technologies, Université Claude Bernard Lyon 1, Bâtiment Gabriel Lippmann; 14, rue Enrico Fermi 69622 Villeurbanne Cedex, France*
E-mail: jincong@mail.ccnu.edu.cn

Abstract: The identification of a module's fault-proneness is very important for minimising cost and improving the effectiveness of the software development process. How to obtain the relation between software metrics and a module's fault-proneness has been the focus of much research. One technical challenge to obtain this relation is that there is relevance between software metrics. To overcome this problem, the authors propose a reduction dimensionality phase, which can be generally implemented in any software fault-prone prediction model. In this study, the authors present applications of artificial neural network (ANN) and support vector machine in software fault-prone prediction using metrics. A new evaluation function for computing the contribution of each metric is also proposed in order to adapt to the characteristics of software data. The vital characteristic of this approach is the automatic determination of ANN architecture during metrics selection. Four software datasets are used for evaluating the performance of the proposed model. The experimental results show that the proposed model can establish the relation between software metrics and modules' fault-proneness. Moreover, it is also very simple because its implementation requires neither extra cost nor expert's knowledge. The proposed model has good performance, and can provide software project managers with trustworthy indicators of fault prone components.

1 Introduction

Software systems are being utilised to solve or model increasingly sophisticated and complex problems in a variety of application domains. There has been a tremendous growth in the demand for software quality during recent years. As a consequence, issues related to classifying fault-prone software modules are becoming increasingly crucial. Studies have shown that the majority of faults are often found in only a few software modules [1, 2]. Such fault software modules may cause software failures, increase development and maintenance costs and decrease customer satisfaction. Accordingly, effective prediction of fault-prone software modules can enable software developers to focus quality assurance activities and allocate effort and resources more efficiently. This in turn can lead to a substantial improvement in software quality.

Therefore the possibility of predicting the potential faultiness of software modules at an early stage can be of great help for planning activities. Unfortunately, software fault-proneness cannot be directly measured. However, it can be estimated based on software metrics. A number of studies provide empirical evidence that relations exist between software metrics and fault-proneness [3, 4]. Identification of fault-prone software modules is commonly achieved through binary classification models that predict whether or not a module is fault-prone using various software metrics.

Classification has been one of the most important data mining tasks. The performance of the prediction models depends on an appropriate selection of the most relevant software metrics because some irrelevant and/or redundant metrics generally exist in the software datasets that not only make prediction harder, but also degrade generalisation performance of the software fault-prone prediction model. Therefore the reduction dimensionality of software metrics space is an essential task [5–7].

Determining an optimal metrics subset is a very complex task. The final metrics subset is expected to retain high prediction performance. The goal is to select a relevant subset of $\boldsymbol{n}$ metrics from a set of $\boldsymbol{D}$ metrics ($\boldsymbol{n} < \boldsymbol{D}$) in a given dataset [8]. $\boldsymbol{D}$ is comprised of all metrics in a given dataset, and it may include noisy, redundant and misleading metrics. Therefore an exhaustive search performed over the entire solution space, which usually takes a long time, often cannot be applied in practice. To resolve this metrics selection problem, we aimed at retaining only a subset of $\boldsymbol{n}$ relevant metrics. Irrelevant metrics are not only useless for prediction, but could also potentially reduce the prediction performance. By removing irrelevant metrics, prediction performance can be increased.

For the identification of relevant metrics and the removal of irrelevant metrics three different methods can be employed, namely the filter, wrapper and hybrid models. The filter model relies on general characteristics of the data to evaluate and select metrics subsets without involving

prediction algorithm. The wrapper model first implements an optimising algorithm that adds or removes metrics to produce various subset metrics, and then employs a prediction algorithm to evaluate this subset of metrics. The hybrid model attempts to take advantage of the filter and wrapper models by exploiting their complementary strengths [9].

In this paper, we propose a new software fault-prone prediction model, called SFPM based on the concept of the filter model and sequential search strategy. This paper is organised as follows. Section 2 describes some related work about metrics selection. Section 3 introduces the methods used in this study, namely artificial neural network (ANN) and support vector machine (SVM). SFPM is discussed elaborately in Section 4. Section 5 presents the experimental studies including the experimental datasets, parameters initialisation, experimental results and the comparison with other existing prediction approaches. Finally, conclusions are given in Section 6.

2 Related work

The filter model only depends on the inherent characteristics of the dataset, for example, the distance or correlation between the data is used to measure the important degree of the metrics. Owing to its computing efficiency and obtained metrics subset with good robustness for prediction algorithm, the filter model is suitable for online learning [10]. The filter model has received a lot of attention. As SFPM uses the filter model for reducing dimensionality of the metrics space, we briefly describe some to reduce the dimensionality technique based on the filter model.

Sequential forward selection (SFS) and sequential backward selection (SBS) are the two most common greedy methods. SFS begins with an empty set and sequentially adds metrics until some termination criterion is met. SFS suffers from a nesting effect. Since SFS do not examine all possible metrics subsets, they are not guaranteed to produce an optimal result [11]. In SFS, metrics discarded cannot be re-selected, and selected metrics cannot be removed later. To avoid the nesting effect, 'plus-*l*-take-away-*r*' was proposed [12], where SFS is applied l times forward and then r back-tracks steps of SBS. The challenge with the 'plus-*l*-take-away-*r*' is predicting the best (l, r) values to obtain good results. SBS is the backward version of SFS, and it starts with the full metrics set and sequentially removes metrics until some termination criterion is met.

A number of studies have been proposed by many researchers. Sindhwani *et al.* [13] proposed three feature selection algorithms for multilayer ANN and multiclass SVM, using mutual information between class labels and classifier outputs as an objective function. Sivagaminathan and Ramakrishnan [9] reported a feature selection approach, which evaluated the error of an ANN using mutual information. In some studies [14, 15], the goodness of a feature is computed directly as the value of a loss function. The cross entropy with a penalty function is used in these approaches as a loss function. In [15], the penalty function encourages small weights to converge to zero or prevents the weights to converge to large values. On the other hand, in [14], the penalty function forces a network to keep the derivative of its neurons' transfer functions from becoming low. The aim of such a restriction is to reduce output sensitivity to the input changes. However, these approaches can not be used directly in software fault-prone prediction models. The reason is that these approaches are good for experimental datasets with a large number of features, but there are few metrics in software experiment datasets.

Currently, the most common approach to reduce dimensionality is principal component analysis (PCA) [16] in software engineering [17–21]. For example, Kanmani *et al.* [17], Hochman *et al.* [18] and Khoshgoftaar *et al.* [22] have used PCA to reduce dimensionality. However, a disadvantage of PCA is that, in contrast to the original input variables, the derived dimensions may not have an intuitive interpretation. Adriano *et al.* [23] proposed a GA-based metrics selection method for software effort estimation. The proposed method improves the accuracy of software development effort, and reduces model complexity. However, this method not only failed to improve the estimation function for computing contribution of each metric, but also involves the automatic generation of neural network architecture. Karim *et al.* [24] used a correlation-based metrics selection technique (CMS) to reduce dimensionality. However, the CMS' performance has not been studied carefully.

In this paper, we proposed a novel model for predicting software fault-proneness, that is, SFPM. In the proposed model, ANN is used for solving the metrics selection and SVM [25] for prediction. The proposed metrics selection technique differs from previous works [9, 14, 15], which determine the ANN architecture automatically, that is, to determine automatically the number of hidden neurons in the ANN. It is well-known that the performance of any ANN is greatly dependent on its architecture [26]. Thus, determining hidden neurons' number automatically provides a novel approach in building prediction approaches with good performance. Another difference is that a new evaluation function for computing the contribution of each metric is used in this paper to improve the performance of the software fault-prone prediction model.

3 ANN and SVM

3.1 Artificial neural network

We used the ANN to reduce dimensionality of the metrics space.

ANN-based classifiers can incorporate both statistical and structural information and achieve better performance than simple minimum distance classifiers [27]. An ANN possesses the following characteristics [28]:

1. It has a rapid training process, simple structure and good extendability.
2. It learns from experience, hence, has no need for any a priori modelling assumptions.
3. It is capable of inferring complex non-linear input–output transformations.

Therefore based on the advantages of ANN, multi-layered ANN, usually employing the back propagation (BP) algorithm, is also widely used in software engineering [28]. In this paper, an effective reduced dimensionality approach using ANN according to the characteristics of software metrics is proposed.

3.2 Support vector machine

Recently, SVM has been introduced as an effective model in both machine learning and data mining communities for solving both classification and regression problems [13, 24].

doi: 10.1049/iet-sen.2011.0138

It is therefore motivating to investigate the capability of the SVM in software fault-prone prediction.

Assume there are $\boldsymbol{n}$ samples and $\boldsymbol{m}$ metrics, where $\{x_1, \ldots, x_m\}$ and $\{1, 2, \ldots, n\}$ are sample and metric sets, respectively. $Y = \{y_1, \ldots, y_t\}$ refers to class labels categories. Given a training set (x_i, y_i), $i = 1, 2, \ldots, m$ where $x \in R^n$ and $y \in \{+1, -1\}$. The SVM finds an optimal separating hyperplane, $d(x) = w^{\mathrm{T}}\phi(x) + b$, by solving the following optimisation problem

$$\underset{w,b,\xi}{\mathrm{Min}} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

Subject to

$$\begin{aligned} &y_i \cdot (w^{\mathrm{T}} \cdot \phi(x_i) + b) \geq 1 - \xi_i,\ i = 1, \ldots, m; \\ &\xi_i \geq 0,\ i = 1, \ldots, m \end{aligned} \tag{1}$$

where ϕ is a mapping function that maps the training patterns from the input space into a higher dimensional metrics space, and C is a penalty parameter on the training error. ξ_i is the non-negative slack variable. Under this mapping, the solution obtained by applying SVM has the form

$$\begin{aligned} f(x) &= \mathrm{sign}(w^{\mathrm{T}}\phi(x) + b) \\ &= \mathrm{sign}\left(\sum_{i=1}^{m}\alpha_i y_i(\phi^{\mathrm{T}}(x_i)\phi(x)) + b\right) \end{aligned} \tag{2}$$

In a binary classification problem, we use f using the training set. Let us represent the module into either fault-prone with $y = 1$ or non-fault-prone with $y = -1$.

In (2), the required scalar product $\phi^{\mathrm{T}}(x_i)\phi(x_j)$ is calculated directly by computing the kernel function $K(x_i, x_j) = \phi^{\mathrm{T}}(x_i)\phi(x_j)$ for given training pattern in an input space. The radial basis function is a common kernel function, as follows

$$K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2) \tag{3}$$

where $\gamma > 0$ is the parameter controlling the width of the kernel.

4 Proposed methodology

In this paper, we propose a new adaptive reduced dimensionality approach of metrics space. The ANN is designed to reduce dimensionality of the metrics space according to the SBS stratagem, and the SVM to predict software fault-proneness. Fig. 1 displays the main processes to implement the proposed scheme.

Reduction dimensionality and SVM predicted process steps are described in detail, respectively.

4.1 Reduction dimensionality

Selection of input variables is an essential problem in the design of ANN. Each additional input unit adds another dimension and contributes to what is known as the curse of dimensionality [27], a phenomenon in which performance degrades as the number of inputs increases. A reduction dimensionality technique can be used to reduce the number of input variables by keeping only the most important ones. In this paper, ANN is used for reduction dimensionality.

In order to achieve good prediction accuracy (PA), the reduction dimensionality technique is implemented automatically using ANN. In other words, the number of hidden neurons of the ANN are determined during the reduction dimensionality process.

We will certainly explain this technique as follows:

Step 1: Initialisation. Choose a three layered feed-forward perceptron ANN architecture and initialise its different parameters. The number of neurons in the hidden and output layers is set to one, respectively. Initially, the number of input neurons is set to the number of metrics set $\boldsymbol{X}$. Randomly initialise all connection weights of the ANN within a small range.

Step 2: Train the ANN via on the X for a predefined number of training epochs υ using the BP learning algorithm. Initially, the element number of X is $|X|$. By removing metrics, the element number of $|X|$ reduces continually. In other words, the element number of $|X|$ changes dynamically.

Step 3: Check the termination criterion to stop the training. If this criterion is satisfied, the current metrics set is selected as the final metrics set. Otherwise continue. In this

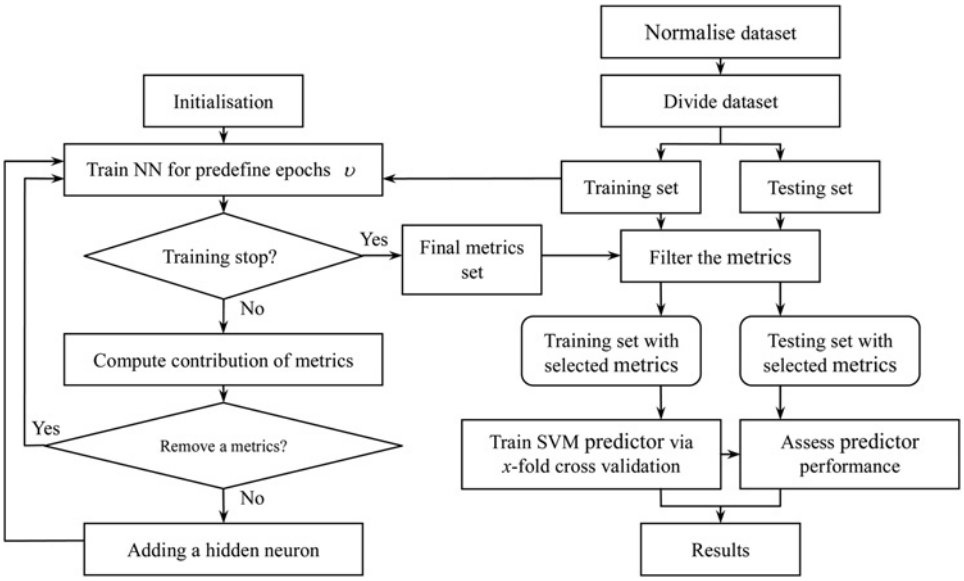

Fig. 1 *Block diagram of dimensionality reduction approach*

work, the error of the ANN is used in the termination criterion. The error, E, is calculated as

$$E = \frac{1}{2}\sum_{j=1}^{|X|}(o(\boldsymbol{j}) - t(\boldsymbol{j}))^2 \quad (4)$$

where $o(\boldsymbol{j})$ and $t(\boldsymbol{j})$ are the actual and target responses of the output neuron for metric $\boldsymbol{j}$, respectively. If the training error is decreasing, that is, $E(t+1) - E(t) < \delta$, then stop training. δ is a predefined parameter.

Step 4: Compute the contribution of metrics. Battiti [29] proposed an evaluation function for computing the contribution of each metric as follows

$$J(j) = I(Y; j) - \beta \sum_{s \in S} I(s; j) \quad (5)$$

where, $\boldsymbol{S}$ is selected metric subset, β is a parameter and determined empirically and $I(Y; j)$ is mutual information of Y and j. Battiti suggested a value between 0.5 and 1 for β. However, this $J(j)$ does not consider the effect of the size of the selected metric subset. The effect of the first term $I(Y; j)$ would increase when the size of the selected metric subset decreased. In order to avoid this situation, Amiri *et al.* [30] suggested an evaluation function as follows

$$J(j) = I(Y; j) - \frac{\beta}{|S|}\sum_{s \in S} I(s; j) \quad (6)$$

Liu *et al.* [31] proposed another evaluation function as follows

$$J(j) = \frac{S_b(Y; S, j)}{|S| + S_w(S, j)} \quad (7)$$

where, S_b and S_w represent inter-distance and intra-distance, respectively.

Equations (6) and (7) are good for datasets with large number of metrics. They can improve the robustness of the classifiers. However, (6) and (7) are not suitable for datasets with few metrics. Lots of metrics would be removed even if they are important for classifiers when we use (6) or (7) in the software experiment datasets. It is not reasonable.

We suggest an evaluation function with a non-linear factor shown as follows

$$J(j) = I(Y; j) - a \times |S|^b \times \sum\nolimits_{s \in S} I(s; j), \quad a, b \in R \quad (8)$$

where, R is a real number set. In (6), the decreasing size of the metric subset will affect the effect of candidate metrics. However, it is too simple to eliminate the effect by dividing the size. No evidence shows there is a linear relation between the effect and the size. Hence, we introduce the non-linear factor to solve this problem and obtain (8). Contribution of metrics computes according to (8), which computes the importance of each metric independently.

Step 5: To determine whether to remove one metric is necessary. On removing one metric with the smallest contribution from the current metrics space, if the training error is less than before, it is assumed that removing this metric is necessary. Otherwise continue.
Step 6: Add a hidden neuron and go to *Step* 2.

We note that, in the above process, the number of hidden neurons of the ANN is determined automatically during reduction dimensionality, that is, the proposed ANN uses one simple criterion to decide when to add hidden neurons.

4.2 SFPM Model

In the software fault-prone prediction model, the SVM is regarded as a mapping from the metrics space to '1' or '−1'. Up to now, we can acquire software fault-prone predictor as follows:

Step 1. The input metric $\boldsymbol{x}$ was normalised, and obtained the normalised metric $\boldsymbol{X}$. They always fall within the interval [0, 1].
Step 2: We divided the preprocessed metrics set into two subsets randomly, that is, training subset and testing subset according to 2:1 ratio.
Step 3: Train the SVM model via an $\boldsymbol{x}$-fold cross-validation on the training subsets with selected metrics, and obtain SFPM.
Step 4: Predict the modules (i.e. fault-prone or non-fault-prone) using SFPM in the testing subset with selected metrics.
Step 5: For analysing the performance of the prediction model, compute the prediction performance measures values, obtain the performance evaluation and get prediction results.

5 Experiments

In this section, we discuss the conducted empirical study that evaluates the performance of SVM in prediction of fault-prone software modules. We used the open source WEKA [(Waikato environment for knowledge analysis) http://www.cs.waikato.ac.nz/ml/weka/] machine learning toolkit to conduct this paper. In this empirical study, we will evaluate SVM for the purpose of prediction fault-prone software modules with respect to its prediction performance against the four compared models [decision tree (DT), K-nearest neighbour (KNN), logistic regression (LR) and naïve bayes (NB)] and practitioners in the context of four NASA datasets.

5.1 Datasets

The datasets used in this paper study are four mission critical NASA software projects, which are publicly accessible from the repository of the NASA IV&V Facility Metrics Data Program [http://mdp.ivv.nasa.gov/index.html]. Two datasets (PC1 and CM1) are from software projects written in a procedural language (C) where a module in this case is a function. The other two datasets (KC1 and KC3) are from projects written in object-oriented languages (C++ and Java) where a module in this case is a method. Each dataset contains 21 software metrics at the module-level and the associated dependent Boolean variable: fault-prone (whether or not the module has any fault-proneness). Table 1 shows some main characteristics of these datasets.

The metrics are 21 static metrics including McCabe [32] , Halstead [33], line count and branch count. Table 2 lists these metrics.

The expression levels for each metric of all datasets were normalised to [0, 1]. The normalisation procedure is given in (9). $\text{Value}_{\max}$ is maximum original value. $\text{Value}_{\min}$ is

doi: 10.1049/iet-sen.2011.0138

Table 1 Characteristics of datasets

	Project	Language	No. of modules	% of fault-prone modules	Description
procedural	PC1	C	1107	6.9	flight software for an earth orbiting satellite
	CM1	C	496	9.7	NASA spacecraft instrument
object-oriented	KC1	C++	2107	15.4	storage management for receiving and processing ground data
	KC3	Java	458	6.3	collection, processing and delivery of satellite metadata

minimum original value.

$$X = \text{lower} + \left[(\text{upper} - \text{lower})\left(\frac{\text{Value} - \text{Value}_{\min}}{\text{Value}_{\max} - \text{Value}_{\min}}\right)\right] \quad (9)$$

5.2 Parameters initialisation

There are some parameters in this study, which need to be specified by the user. These are described as follows:

• ANN: The initial connection weights for an ANN were randomly chosen in the range between −1.0 and 1.0. The learning rate and momentum term for training of ANN were chosen as 0.1–0.15 and 0.8–0.9, respectively. For ANN, the number of training epochs υ was between 30. The initial weight values, momentum term and learning rate are the parameters of the well-known BP algorithm [27]. According to some preliminary runs and the suggestion of the previous study [34] these values were set. They were not meant to be optimal. Through a large number of experiments, the training error threshold values δ for PC1, CM1, KC1 and KC3 dataset were set to 0.002, 0.01, 0.001 and 0.003, respectively. The outputs were encoded by the 1-of-2 representation of 2 classes. The most commonly used winner-takes-all method was used for selecting the ANN output. The hidden and output neuron functions were defined by the logistic sigmoid function $f(x) = 1/(1 + \exp(-x))$.

• SVM: The regularisation parameter (C) was set to 1 and the bandwidth (γ) of the kernel function was set to 0.5.
• LR: The method of optimisation was the maximisation of log-likelihood.
• KNN: The number of observations (k) in the set of closest neighbour was set to 5.
• NB: It does not require any parameters to pass.
• DT: It uses the well-known C4.5 algorithm to generate a decision tree. The confidence factor used for pruning was set to 25% and the minimum number of instances per leaf was set to 2.
• For evaluation function, after a large number of experiments, we let $a = 1.0$, $b = -3$.

In addition, a software fault-prone threshold of 0.5 was used for all the models to predict a module as fault-prone if its predicted probability is higher than the threshold.

5.3 Cross-validation

An x-fold cross-validation [35] was used to evaluate the performance of the prediction approaches. In x-fold cross-validation, the original training data is randomly partitioned into x subsets. Of the x subsets, a single subset is retained as the test data for testing the SFPM, and the remaining $x - 1$ subsets are used as training sets. The cross-validation process is then repeated x times (the folds) with each of the x subsets used exactly once as a testing set. The x results from the folds then can be averaged to produce a single

Table 2 Method-level metrics

No.	Metrics	Type	Description
1	LOCode	line count	number of lines of statement
2	LOComment	line count	number of lines of comment
3	LOBlank	line count	number of lines of blank
4	LOCec	line count	number of lines code and comment
5	LOC	McCabe	total number of lines
6	BR	branch	total number of branch count
7	N_1	basic halstead	number of unique operators
8	N_1	basic halstead	total number of operators
9	N_2	basic halstead	number of unique operands
10	N_2	basic halstead	total number of operands
11	$V(g)$	McCabe	cyclomatic complexity
12	$IV(g)$	McCabe	design complexity
13	$EV(g)$	McCabe	essential complexity
14	N	derived halstead	total number of operands and operators
15	V	derived halstead	volume on minimal implementation
16	L	derived halstead	program length = V/N
17	D	derived halstead	difficulty = $1/L$
18	E	derived halstead	effort to write program = V/L
19	T	derived halstead	time to write program = E/18 s
20	B	derived halstead	effort estimate
21	I	derived halstead	intelligent content

Table 3 A confusion atrix

		Predicted class	
		Class = 0	Class = 1
actual class	class = 0	f_{00}	f_{01}
	class = 1	f_{10}	f_{11}

estimation. The advantage of this method over repeated random sub-patterning is that all observations are used for both training and testing, and each observation is used for testing exactly once [36]. In order to reduce sample bias, ten-fold cross-validation was used in the experiment.

5.4 Prediction performance measures

The performance of prediction approaches for two-class (i.e. fault-prone and nonfault-prone) problem is commonly evaluated using the pattern set in the confusion matrix [37], which is shown in Table 3.

In Table 3, f_{ij} expresses that the number of actual class = i is classified into class = j, where $i, j \in \{0,1\}$. We will use the commonly used prediction performance measures: accuracy, recall and precision to evaluate the prediction approaches quantitatively. These measures are derived from the confusion matrix.

5.4.1 Prediction accuracy: PA is also known as correct prediction rate. It is defined as the ratio of the number of classes correctly classified to the total number of classes. It is calculated as follows

$$\text{Prediction accuracy} = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}} \tag{10}$$

5.4.2 Recall: Recall is known as fault detection rate. It is defined as the ratio of the number of classes correctly predicted as fault-prone to the total number of classes that are actually fault-prone. It is calculated as follows

$$\text{Recall} = \frac{f_{11}}{f_{11} + f_{10}} \tag{11}$$

5.4.3 Precision: Precision is also known as correctness. It is defined as the ratio of the number of classes correctly predicted as non-fault-prone to the total number of classes predicted as non-fault-prone. It is calculated as follows

$$\text{Precision} = \frac{f_{00}}{f_{00} + f_{01}} \tag{12}$$

Both recall and precision are important performance measures. Higher the precision, less the effort wasted in testing and inspection; and higher the recall, fewer fault-prone modules go undetected. Generally, we need a trade-off between recall and precision.

5.4.4 F-measure: F-measure considers both recall and precision equally important by taking their harmonic mean [37]. It is calculated as follows

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

5.5 Results and discussions

5.5.1 Performance of ANN: Table 4 shows the results of ANN over 30 independent runs on four datasets, respectively. Here, SD and % refer to standard deviation and percentage of selected metrics, respectively.

From Table 4, ANN was able to select a smaller number of metrics for different datasets. After reducing dimensionality, current dimensionality of metrics space does not exceed 30% of the dimensionality of the original metrics space. Therefore it can be seen that the proposed reduction dimensionality approach can simplify ANN architectures. A very difficult work to decide the number of hidden neurons in ANN is achieved automatically.

After contribution of metrics is computed according to (8), the importance of each metrics is computed independently. In these experiments, the result of the ANN selected metrics set with the most number is listed in Table 5.

For each dataset, in addition to the selected metrics, the remaining metrics have been removed. This shows that the selected metrics are relevant metrics for predicting software fault-proneness, and the remaining metrics are irrelevant.

Table 4 Performance of ANN for different datasets

Dataset	Parameters			No. of metrics			No. of hidden neurons	
	A	b	δ	Mean	SD	%	Mean	SD
PC1	1.0	− 3	0.002	6.13	0.48	29.19	3.79	0.85
CM1	1.0	− 3	0.01	5.16	0.37	24.57	2.52	0.41
KC1	1.0	− 3	0.001	5.71	1.06	27.19	3.20	0.36
KC3	1.0	− 3	0.003	2.69	0.54	12.81	4.03	0.69

Table 5 Most number metrics selected by ANN in each dataset

Dataset	The most number metrics selected by ANN	No. of metrics
PC1	n_1, n_2, IOBlank, IOCec, IOComment, I, $V(g)$	7
CM1	n_1, n_2, IOBlank, IOComment, I, LOC, $IV(g)$	7
KC1	BR, n_2, IOBlank, IOComment, I, D, V	7
KC3	$EV(g)$, N, IOCec, IOC	4

doi: 10.1049/iet-sen.2011.0138

Obviously, it does not mean that such selection results are optimal.

5.5.2 Performance of SFPM: Table 6 presents the best prediction result of SFPM over four datasets on the testing subset with metrics set of Table 5, respectively. Here, G_1 and G_2 refer to fault-prone and non-fault-prone, respectively. A_1 and A_2 refer to no. of metrics and no. of hidden neurons, respectively.

From the results in Table 6, it is found that software fault-prone modules can be predicted by SFPM accurately, which shows SFPM has the effectiveness to indicate fault-prone software.

5.5.3 Comparison with other models: For comparison of the prediction performance of SFPM and the other four compared models with selected metrics on the testing subset, we calculated PA and *F*-measure of each dataset. We also compared SFPM to SVM with the original metrics. The experimental mean results are reported in Table 7 over 30 independent runs.

Results from PC1: It can be observed that SFPM exceeds four models in *F*-measure. The DT do not exceed (0.963). However, the difference was not significant to compare with SFPMs (0.961).

Results from CM1, KC1 and KC3: It can be observed that NNFS exceeds all the compared models in *F*-measure.

In addition, it can be observed that the performance SFPM is always higher than SVMs, which shows that to remove redundant and misleading metrics can indeed improve the prediction performance of SFPM.

We will also compare our results with the literature [24]'s. We note that [24] use the CMS technique for selecting metrics. However, the selected metric subsets by CMS and ANN are not exactly the same. The experimental mean results are reported in Table 8 over 30 independent runs.

Results from PC1: It can be observed that DT exceeds all other models in PA. The SFPM is not the highest (94.03).

Table 6 Predict results of NNFS

Pattern set	PC1			CM1			KC1			KC3		
	$A_1 = 5, A_2 = 4$			$A_1 = 5, A_2 = 3$			$A_1 = 6, A_2 = 3$			$A_1 = 2, A_2 = 4$		
	G_1	G_2	Total	G_1	G_2	Total	G_1	G_2	Total	G_1	G_2	Total
actual												
G_1	23	9	32	16	9	25	106	29	135	10	8	18
G_2	15	322	337	5	135	140	46	521	567	4	131	135
total	38	331	369	21	144	165	152	550	702	14	139	153
percent	PER = 6.50%			PER = 8.48%			PER = 10.68%			PER = 7.84%		

Table 7 Mean prediction results

Model	PC1		CM1		KC1		KC3	
	PA, %	*F*-measure	PA, %	*F*-measure	PA, %	*F*-measure	PA, %	*F*-measure
SFPM	94.03	0.961	91.17	0.951	87.74	0.909	93.36	0.968
SVM	62.72	0.653	60.25	0.638	64.51	0.674	58.43	0.626
LR	93.86	0.957	90.91	0.926	86.53	0.902	93.32	0.966
KNN	90.31	0.936	81.20	0.899	81.42	0.852	93.21	0.961
NB	88.67	0.910	76.74	0.853	86.36	0.895	92.96	0.962
DT	94.14	0.963	83.29	0.908	83.29	0.850	92.64	0.961

Table 8 Mean prediction results

Model	PC1		CM1		KC1		KC3	
	PA, %	*F*-measure	PA, %	*F*-measure	PA, %	*F*-measure	PA, %	*F*-measure
SFPM	94.03	0.961	91.17	0.951	87.74	0.909	93.36	0.968
SVM	62.72	0.653	60.25	0.638	64.51	0.674	58.43	0.626
SVM[a]	93.10	0.964	90.69	0.951	84.59	0.916	93.28	0.965
LR	93.86	0.957	90.91	0.926	86.53	0.902	93.32	0.966
LR[a]	93.19	0.964	90.17	0.948	85.55	0.916	93.42	0.966
KNN	90.31	0.936	81.20	0.899	81.42	0.852	93.21	0.961
KNN[a]	91.82	0.956	83.27	0.906	85.55	0.919	92.50	0.960
NB	88.67	0.910	76.74	0.853	86.36	0.895	92.96	0.962
NB[a]	89.21	0.941	86.74	0.926	82.86	0.900	92.81	0.962
DT	94.14	0.963	83.29	0.908	83.29	0.850	92.64	0.961
DT[a]	93.58	0.966	89.82	0.946	84.56	0.914	93.11	0.964

[a]Proposed in [24].

However, the difference was not significant to compare with DTs (94.14). Similarly, the DT^a (0.966) exceeds all other models in *F*-measure. However, the difference was not significant to compare with SFPMs (0.961).

Results from CM1: It can be observed that SFPM exceeds all other models in PA and *F*-measure, respectively. SFPM has the best performance.

Results from KC1: It can be observed that SFPM exceeds all other models in PA. SFPM has the best performance. Similarly, KNN^a (0.919) exceeds all other models in *F*-measure. The NNFS is not as good as the SVM^a (0.916), LR^a (0.916) and DT^a (0.914), which shows that [24]'s performance is better than SFPMs.

Results from KC3: It can be observed that KNN^a exceeds all other models in PA. The SFPM is not the highest (93.36). However, the difference was not significant to compare with KNN^a's (93.42). SFPM (0.968) exceeds all other models in *F*-measure. SFPM has the best performance.

We also note that the performance of SVM and SVM^a is very different. The SVM^a is much better than SVM, which shows that metric selection is absolutely necessary.

5.5.4 Threats to validity: In the proposed approach, there is the issue that may affect the training/test results. Namely, the choice of parameters a and b has an impact on the estimated function. By the large number of experimental analysis and research, we found that, for a large dataset, the range of the parameter b is $[-2, 0]$; for a small dataset, the parameter b does not exceed -2. Parameter a is the adjustment coefficient; it is recommended that the range is [0.5, 1]. If not in this range, the evaluation function has a bad performance.

6 Conclusions

This paper investigated the application of ANN and SVM to develop the software fault-prone prediction approach. The proposed prediction approach identifies fault-prone software modules using only software metrics. The main advantages of the proposed approach are as follows:

1. In this paper, ANN is used for reducing dimensionality of the metrics space, and SVM is used for prediction software modules that are fault-prone. SFPM has been proven to be very effective for establishing a relationship between software metrics and fault-proneness.
2. The proposed prediction approach is very simple; the experiment results confirm that performance of the proposed approach is generally satisfactory.
3. Utilisation of the proposed approach requires neither extra cost nor expert's knowledge for the development team since the proposed approach is based solely on software metrics. Deciding the number of hidden neurons in ANN is achieved automatically.
4. The proposed reduction dimensionality approach is designed according to the SBS stratagem, which makes the proposed reduction dimensionality approach not ignore the completeness of the metrics.

7 Acknowledgments

This work was supported by the social science foundation from Chinese Ministry of Education (Grant no. 11YJAZH040), and the Science and Technology Research Programme of Wuhan of China (Grant no. 201210121023).

8 References

1 Boehm, B.W.: 'Papaccio P.N., Understanding and controlling software costs', *IEEE Trans. Softw. Eng.*, 1988, **14**, (10), pp. 1462–1477
2 Porter, A., Selby, R.: 'Empirically guided software development using metric-based classification trees', *IEEE Softw.*, 1990, **7**, (2), pp. 46–54
3 Cartwright, M., Shepperd, M.: 'An empirical investigation of an object-oriented software system', *IEEE Trans. Softw. Eng.*, 2000, **26**, (8), pp. 786–796
4 Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: 'Benchmarking classification models for software defect prediction: a proposed framework and novel findings', *IEEE Trans. Softw. Eng.*, 2008, **34**, (4), pp. 485–496
5 Pendharkar, P.C.: 'Exhaustive and heuristic search approaches for learning a software defect prediction model', *Eng. Appl. Artif. Intell.*, 2010, **23**, (1), pp. 34–40
6 Guyon, I., Elisseeff, A.: 'An introduction to variable and feature selection', *J. Mach. Learn. Res.*, 2003, **3**, (3), pp. 1157–1182
7 Maldonado, S., Weber, R.: 'A wrapper method for feature selection using support vector machines', *Inf. Sci.*, 2009, **179**, (13), pp. 2208–2217
8 Oh, I.S., Lee, J.S., Moon, B.R.: 'Hybrid genetic algorithms for feature selection', *IEEE Trans. Pattern Anal. Mach. Intell.*, 2004, **26**, (11), pp. 1424–1437
9 Sivagaminathan, R.K., Ramakrishnan, S.: 'A hybrid approach for feature subset selection using neural networks and ant colony optimization', *Expert Syst. Appl.*, 2007, **33**, (1), pp. 49–60
10 Liu, H., Yu, L.: 'Toward integrating feature selection algorithms for classification and clustering', *IEEE Trans. Knowl. Data Eng.*, 2005, **17**, (4), pp. 491–502
11 Zhang, H., Sun, G.: 'Feature selection using Tabu search method', *Pattern Recognit.*, 2002, **35**, (3), pp. 701–711
12 Stearns, S.D.: 'On selecting feature for pattern classifiers'. Third Int. Conf. on Pattern Recognition, Coronado, CA, 1976, pp. 71–75
13 Sindhwani, V., Rakshit, S., Deodhare, D., Erdogmus, D., Principe, J.C., Niyogi, P.: 'Feature selection in MLPs and SVMs based on maximum output information', *IEEE Trans. Neural Netw.*, 2004, **15**, (4), pp. 937–948
14 Verikas, A., Bacauskiene, M.: 'Feature selection with neural networks', *Pattern Recognit. Lett.*, 2002, **23**, (11), pp. 1323–1335
15 Setiono, R., Liu, H.: 'Neural network feature selector', *IEEE Trans. Neural Netw.*, 1997, **8**, (3), pp. 654–662
16 Jolliffe, I.T.: 'Principal component analysis' (Springer, 1986)
17 Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., Thambidurai, P.: 'Object-oriented software fault prediction using neural networks', *Inf. Softw. Technol.*, 2007, **49**, (5), pp. 483–492
18 Hochman, R., Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J.P.: 'Using the genetic algorithm to build optimal neural networks for fault-prone module detection'. Seventh Int. Symp. on Software Reliability Engineering, 30 October–2 November 1996, pp. 152–162
19 Sachindra, J., Jayadeva, Ganesh, R., Suresh, C.: 'Using sequential unconstrained misnimization techniques to simplify SVM solvers', *Neurocomputing*, 2012, **77**, (1), pp. 253–260
20 Lai, C., Reinders, M.J.T., Wessels, L.: 'Random subspace method for multivariate feature selection', *Pattern Recognit. Lett.*, 2006, **27**, (10), pp. 1067–1076
21 Hsu, C., Huang, H., Schuschel, D.: 'The ANNIGMA-wrapper approach to fast feature selection for neural nets', *IEEE Trans. Syst. Man Cybern. B, Cybern.*, 2002, **32**, (2), pp. 207–212
22 Khoshgoftaar, T.M., Szabo, R.: 'Using neural networks to predict software faults during testing', *IEEE Trans. Reliab.*, 1996, **45**, (3), pp. 456–462
23 Adriano, L.I.O., Petronio, L.B., Ricardo, M.F.L., Márcio, L.C.: 'GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation', *Inf. Softw. Technol.*, 2010, **52**, (11), pp. 1155–1166
24 Karim, O.E., Mahmoud, O.E.: 'Predicting defect-prone software modules using support vector machines', *J. Syst. Softw.*, 2008, **81**, (5), pp. 649–660
25 Ohlsson, N., Helander, M., Wohlin, C.: 'Quality improvement by identification of fault-prone modules using software design metrics'. Int. Conf. on Software Quality, 28–30 October 1996, pp. 1–13
26 Bishop, C.M.: 'Neural networks for pattern recognition' (Oxford University Press, New York, 1995)
27 Haykin, S.: 'Neural networks' (Prentice-Hall, 1999)
28 Su, Y.S., Huang, C.Y.: 'Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models', *J. Syst. Softw.*, 2007, **80**, (4), pp. 606–615

doi: 10.1049/iet-sen.2011.0138

29 Battiti, R.: 'Using mutual information for selecting features in supervised neural net learning', *IEEE Trans. Knowl. Data Eng.*, 2005, **17**, (9), pp. 1199–1207
30 Amiri, F., Yousefi, M.M.R., Lucas, C., Shakery, A., Yazdani, N.: 'Mutual information-based feature selection for intrusion detection systems', *J. Netw. Comput. Appl.*, 2011, **34**, (4), pp. 1184–1199
31 Liu, H.W., Sun, J.G., Liu, L., Zhang, H.J.: 'Feature selection with dynamic mutual Information', *Pattern Recognit.*, 2009, **42**, (7), pp. 1330–1339
32 McCabe, T.J., Butler, C.W.: 'Design complexity measurement and testing', *Commun. ACM*, 1989, **32**, (12), pp. 1415–1423
33 Halstead, M.H.: 'Elements of software science' (Elsevier, New York, 1977)
34 Gasca, E., Sanchez, J.S., Alonso, R.: 'Eliminating redundancy and irrelevance using a new MLP-based feature selection method', *Pattern Recognit.*, 2006, **39**, (2), pp. 313–315
35 Kohavi, R.: 'A study of cross-validation and bootstrap for accuracy estimation and model selection'. Fourteenth Int. Joint Conf. on Artificial Intelligence, 1995, pp. 1137–1143
36 Duan, K., Keerthi, S., Poo, A.: 'Evaluation of simple performance measures for tuning SVM hyperparameters', Technical Report, Department of Mechanical Engineering, National University of Singapore, Singapore, 2001
37 Richard, P., Dennis, C.: 'Cross-validation of regression models', *J. Am. Stat. Assoc.*, 1984, **79**, (387), pp. 575–583