Computers, Materials & Continua CMC, vol.65, no.1, pp.279-308, 2020

Software Defect Prediction Based on Stacked Contractive Autoencoder and Multi-Objective Optimization

Nana Zhang[1], Kun Zhu[1], Shi Ying[1,*] and Xu Wang[2]

Abstract: Software defect prediction plays an important role in software quality assurance. However, the performance of the prediction model is susceptible to the irrelevant and redundant features. In addition, previous studies mostly regard software defect prediction as a single objective optimization problem, and multi-objective software defect prediction has not been thoroughly investigated. For the above two reasons, we propose the following solutions in this paper: (1) we leverage an advanced deep neural network—Stacked Contractive AutoEncoder (SCAE) to extract the robust deep semantic features from the original defect features, which has stronger discrimination capacity for different classes (defective or non-defective). (2) we propose a novel multi-objective defect prediction model named SMONGE that utilizes the Multi-Objective NSGAII algorithm to optimize the advanced neural network—Extreme learning machine (ELM) based on state-of-the-art Pareto optimal solutions according to the features extracted by SCAE. We mainly consider two objectives. One objective is to maximize the performance of ELM, which refers to the benefit of the SMONGE model. Another objective is to minimize the output weight norm of ELM, which is related to the cost of the SMONGE model. We compare the SCAE with six state-of-the-art feature extraction methods and compare the SMONGE model with multiple baseline models that contain four classic defect predictors and the MONGE model without SCAE across 20 open source software projects. The experimental results verify that the superiority of SCAE and SMONGE on seven evaluation metrics.

Keywords: Software defect prediction, deep neural network, stacked contractive autoencoder, multi-objective optimization, extreme learning machine.

1 Introduction

Software defect prediction is a very important software quality assurance technology, which can predict the defect proneness of new software modules in advance [Zhang, Zheng, Zou et al. (2016)]. An effective defect prediction model can help developers or

[1] School of Computer Science, Wuhan University, Wuhan, 430072, China.

[2] Department of Computer Science, Vrije University Amsterdam, Amsterdam, 1081HV, The Netherlands.

[*] Corresponding Author: Shi Ying. Email: yingshl@whu.edu.cn.

Received: 13 April 2020; Accepted: 28 April 2020.

testers to detect the potentially defective modules by reasonably allocating limited resources [Tantithamthavorn, McIntosh, Hassan et al. (2016)].

When building the software defect datasets, researchers can inspect software defect modules by designing many software features based on the software development process or code complexity, so these defect datasets may be high dimensional [Xu, Liu, Luo et al. (2018)]. But not all the features are helpful to the performance of the defect prediction model since the datasets may contain some irrelevant and redundant features. Jiarpakdee et al. [Jiarpakdee, Tantithamthavorn, Ihara et al. (2016)] demonstrate that 10%-67% of features in the 101 open source defect datasets are irrelevant or redundant, and these features seriously degrade the prediction performance and increase the training time of the model. Therefore, it is very necessary to conduct feature selection or extraction for defect datasets in software defect prediction. For the above reason, some feature selection or extraction methods are proposed to solve the high-dimensional problem of software defect datasets by removing irrelevant and redundant features [Kondo, Bezemer, Kamei et al. (2019); Xu, Liu, Yang et al. (2016)]. Feature selection techniques reduce the number of features by selecting an optimal representative and important feature subset, while feature extraction techniques decrease the number of features by constructing new, combined features from the original features [Kondo, Bezemer, Kamei et al. (2019)]. At present, most previous studies mainly leverage feature selection techniques for defect prediction, feature extraction techniques have not been thoroughly investigated in software defect prediction. Because feature selection techniques directly remove some features, which will lead to the loss of some feature information, we adopt feature extraction technique in this paper. For feature extraction techniques, most researchers use Principal Component Analysis (PCA) [Kondo, Bezemer, Kamei et al. (2019)] to conduct software defect prediction. Traditional features extracted by PCA focus on the statistical features of software modules and these features are easily affected by the unbalanced data, so the inherent structure information hidden behind the original defect features may not be represented fully. Currently, deep learning techniques have been successfully applied in many fields by constructing a deep network architecture to automatically learn deep semantic feature representation, such as speech recognition [Mohamed, Dahl and Hinton (2012)], image classification [Krizhevsky, Sutskever and Hinton (2012)], traffic sign classification [Zhang, Wang, Lu et al. (2019)], concentration prediction of PM10 [Oh, Song, Kim et al. (2019)] etc. Previous studies [Guo, Cheng and Cleland-Huang (2017); Wang, Liu and Tan (2016)] have verified that the deep semantic features have stronger discrimination capacity for different classes (defective or non-defective). For these reasons above, we leverage an advanced deep neural network-Stacked Contractive AutoEncoder (SCAE) [Rifai, Vincent, Muller et al. (2011); Ning, Chen, Tie et al. (2018)] to extract the robust deep semantic features from the original defect features. On the one hand, SCAE adopts the Frobenius norm of Jacobian matrix as the regularization penalty term, which can enhance the locally invariant and robust encoding representation. On the other hand, the unsupervised deep network SCAE is stacked by some unsupervised contractive autoencoders (CAE), and the hidden layer of each subnetwork serves as the input layer for the next subnetwork, thereby further improving the robustness and discrimination capacity of deep feature representation. The SCAE can not only prevent the deep network from overfitting but also effectively provide a deep combination of basic features with its excellent nonlinear mapping capability.

Currently, most previous studies utilize classical machine learning methods to build defect prediction models, but these traditional methods have some inevitable flaws. For instance, traditional machine learning methods usually require complex feature engineering, and the prediction performance is not good enough, and their adaptability and migration capacity are not strong enough [Wang, Liu and Tan (2016)]. Based on the above analysis, we adopt an advanced neural network—Extreme Learning Machine (ELM) to construct defect prediction model according to the robust deep semantic features extracted by the SCAE in this paper. The ELM has obvious advantages in classification, including strong discrimination capacity, good generalization performance and fast training speed [Huang, Zhu and Siew (2006)].

At present, search-based software engineering has become a research hotspot in the field of software engineering because it can provide automated or semi-automated solutions for software engineering problems with large-scale complex problem space, which may have multiple competing or even conflicting objectives based on state-of-the-art Pareto optimal solutions [Ni, Chen, Wu et al. (2019)]. Prior studies mostly treat software defect prediction as a single objective optimization problem, and multi-objective software defect prediction has not been thoroughly investigated. In this paper, we propose a novel multi-objective defect prediction model named SMONGE, which leverages the Multi-Objective NSGAII algorithm to optimize the number of hidden neurons and output weight norm of ELM based on state-of-the-art Pareto optimal solutions according to the features extracted by SCAE. We mainly consider two objectives. One objective is to maximize the performance of the constructed defect prediction model, which refers to the benefit of the prediction model. Another objective is to minimize the output weight norm, which is related to the cost of the prediction model. Therefore, we need to make a compromise between these two contradictory objectives.

The main contributions of this paper are as follows:

(1) We leverage an advanced deep neural network-Stacked Contractive AutoEncoder (SCAE) to extract the robust deep semantic features from the original defect features, which has stronger discrimination capacity for different classes (defective or non-defective).

(2) Motivated by the idea of search based software engineering, we propose a novel multi-objective defect prediction model named SMONGE that utilizes the multi-objective NSGAII algorithm to optimize two objectives of the advanced ELM predictor based on state-of-the-art Pareto optimal solutions. One objective is to maximize the model performance, which refers to the benefit of the prediction model. Another objective is to minimize the output weight norm, which is related to the cost of the prediction model. To the best of our knowledge, it is the first time that the multi-objective NSGAII algorithm is used to optimize the advanced neural network-ELM.

(3) To verify the performance of SCAE and SMONGE, we conduct extensive experiments for feature extraction and defect prediction across 20 software defect projects from large open source datasets. We compare the SCAE with six state-of-the-art feature extraction methods, and compare the SMONGE model with multiple baseline models that contain four classic defect predictor and the MONGE model without SCAE. The experimental results demonstrate that the effectiveness of SCAE and SMONGE on seven evaluation metrics.

The reminder of this paper is organized as follows. Section 2 describes the background and related work. Section 3 details feature extraction based on SCAE. Section 4 details the proposed SMONGE model. Section 5 shows the experimental setup, including benchmark datasets, evaluation metrics and baseline models. Section 6 evaluates the performance of SCAE and SMONGE. Section 7 introduces the threats to validity. We conclude this paper and describe the future work in Section 8.

2 Background and related work

In this section, we introduce the typical software defect prediction models, feature selection and extraction methods for software defect prediction, and the application of deep learning techniques in software engineering.

2.1 Software defect prediction

Software defect prediction is a research hotspot in software engineering domain, which can be used to identify potential defective modules in advance by constructing the effective prediction model, and then allocate more testing resources on these defective modules [Tantithamthavorn, McIntosh, Hassan et al. (2017, 2016)]. The granularity of the modules can be classified as component, file, class or code change [Yasutaka, Takafumi, Shane et al. (2016)].

Existing software defect prediction methods focus on how to use machine learning methods to construct effective defect prediction models [Ren, Qin, Ma et al. (2014); Chen and Ma (2015); Lu, Kocaguneli and Cukic (2014)]. Chen et al. [Chen and Ma (2015)] conduct extensive empirical studies by using six regression algorithms and find that decision tree regression can achieve best performance. Lu et al. [Lu, Kocaguneli and Cukic (2014)] use active learning method to conduct defect prediction model, and the method can significantly improve the prediction effect. Nam et al. [Nam, Pan and Kim (2013)] successfully apply Transfer Component Analysis (TCA) technique to software defect prediction. Abaei et al. [Abaei, Rezaei and Selamat (2013)] propose the self-organizing mapping (SOM) prediction model with the threshold, and it can help testers to mark modules without experts.

Previous studies mostly regard software defect prediction as a single objective optimization problem, and multi-objective software defect prediction has not been thoroughly investigated. As far as we know, only the MOFES method proposed by Ni et al. [Ni, Chen, Wu et al. (2019)] considers the multi-objective optimization in software defect prediction, but their study only considers feature selection as a multi-objective optimization problem. Different from the study of Ni et al. [Ni, Chen, Wu et al. (2019)], we regard the optimization of the defect prediction model as a multi-objective optimization problem, and the model leverages the NSGAII algorithm to optimize two objectives of the advanced ELM predictor.

2.2 Feature selection and extraction for software defect prediction

Recently, feature selection and extraction techniques have been applied to software defect prediction, which can eliminate irrelevant and redundant features in the defect datasets [Kondo, Bezemer, Kamei et al. (2019); Xu, Liu, Yang et al. (2016)]. Feature selection methods reduce the number of features in a model by selecting an optimal representative

feature subset, while feature extraction methods decrease the number of features by constructing new, combined features from the original features [Kondo, Bezemer, Kamei et al. (2019)]. Feature selection methods are mainly divided into two types: filter-based feature ranking methods or wrapper-based feature subset selection methods [Majdi and Seyedali (2017)]. Most previous studies mainly use feature selection techniques for defect prediction, while feature extraction techniques have not been thoroughly investigated in software defect prediction.

Previous studies have applied many feature selection techniques to software defect prediction [Liu, Miao and Zhang (2014); Khoshgoftaar, Gao and Napolitano (2012); Gao, Khoshgoftaar and Wang (2011)]. Liu et al. [Liu, Miao and Zhang (2014)] propose three new cost-sensitive based feature selection methods, including Cost-Sensitive Variance Score (CSVS), Cost-Sensitive Laplacian Score (CSLS), and Cost-Sensitive Constraint Score (CSCS), which incorporate cost information into traditional feature selection methods. Khoshgoftaar et al. [Khoshgoftaar, Gao and Napolitano (2012)] compare seven filter-based feature ranking techniques (e.g., information gain (IG), gain ratio (GR)) on sixteen defect datasets. Gao et al. [Gao, Khoshgoftaar and Wang (2011)] verify the performance of hybrid feature selection framework based on seven filter-based methods and three feature subset search methods, and the experimental results show that the reduced features are unable to adversely affect the performance of the prediction model in most cases. Xu et al. [Xu, Liu, Yang et al. (2016)] investigate the impact of 32 feature selection techniques on the software defect prediction, and the experimental results verify that these feature selection techniques have significant performance differences on each dataset. Ni et al. [Ni, Chen, Wu et al. (2019)] use five different multi-objective optimization algorithms (i.e., MOCell, SPEA2, NSGA-II, PAES and SMSEMOA) to conduct feature selection respectively, and the experimental results verify that the effectiveness of the multi-objective optimization feature selection algorithms. For feature extraction techniques, most researchers use principal component analysis (PCA) [Kondo, Bezemer, Kamei et al. (2019)] to conduct feature extraction in software defect prediction. Marco et al. [Marco, Michele and Romain (2010)] adopt PCA to conduct class-level defect prediction, which can void the problem of multicollinearity among the independent variables. Rathore et al. [Rathore and Gupta (2014)] compare PCA with feature selection techniques. The experimental results prove that PCA is one of the best-performing techniques.

Different the previous studies, we leverage an advanced deep neural network—stacked contractive autoencoder (SCAE) to effectively learn the robust deep semantic feature representation from the original defect features, which has stronger discrimination capacity for different classes.

2.3 The application of deep learning techniques in software engineering

Recently, some researchers adopt deep learning techniques to improve various tasks in the field of software engineering [Yang, David and Zhang (2015); Wang, Liu and Tan (2016); Gu, Zhang, Zhang et al. (2016)]. Gu et al. [Gu, Zhang, Zhang et al. (2016)] utilize the RNN encoder-decoder to address the problem of retrieving API call sequences based on the user's natural language query. Wang et al. [Wang, Liu and Tan (2016)] leverage deep belief network (DBN) to learn deep semantic features automatically. The experimental results verify that the deep semantic features-based method outperforms traditional software

metrics. Yang et al. [Yang, David and Zhang (2015)] propose a novel just-in-time defect prediction model named Deeper, which can combine initial change features into high-level features by deep belief network (DBN), and then utilize the new high-level features to construct the defect prediction model.

Deep learning techniques are also used for software traceability [Guo, Cheng, Cleland-Huang et al. (2017)], test report classification [Wang, Cui, Wang et al. (2017)], link prediction in developer online forums [Xu, Ye, Xing et al. (2016)] and so on.

3 Feature extraction based on stacked contractive autoencoder

In this paper, after class imbalance processing (SMOTE) [Chawla, Bowyer, Hall et al. (2002)] and data normalization (min-max) [Witten, Frank and Hall (2011)] operations, we utilize an advanced unsupervised deep neural network-stacked contractive autoencoder (SCAE) to extract the robust deep semantic features from the original defect features with its nonlinear mapping capability, which can properly characterize the complex data structures and increase the probability of linear separability of the data. SCAE is a variant of regularized autoencoder, which adopts the Frobenius norm of Jacobian matrix of encoder activations as the regularization penalty term, so as to form a localized space contraction and yield robust features on the activation layer. In addition, SCAE regards the hidden layer of each subnetwork as the input layer of next subnetwork, which further enhances the robustness and the discrimination capacity of deep feature representation.

The training process for SCAE is as follows. A basic autoencoder subnetwork consists of two parts: encoder and decoder. The encoder *f(x)* is used to output the representation $h \in R^{d_h}$ after feature extraction, while the decoder *g(h)* reconstructs the original input $x \in R^{d_x}$ and output *r* from the output *h* of the encoder by minimizing the cost function.

The internal structures of the encoder *f(x)* and decoder *g(h)* are all mapping functions with nonlinear activation functions, as shown in Eqs. (1) and (2):

$$h = f(x) = s_f(Wx + b_h), \tag{1}$$

$$r = g(h) = s_g(W'h + b_r), \tag{2}$$

where s_f and s_g represent the nonlinear activation functions of encoder and decoder, respectively.

we adopt the *sigmoid()* as the nonlinear activation function in this paper, $sigmoid(z) = \frac{1}{1+e^{-z}}$. W represents the $(d_x \times d_h)$-dimension weight from the input layer to the hidden layer, and W' represents the $(d_h \times d_x)$-dimension weight from the hidden layer to the reconstruction layer. $b_h \in R^{d_h}$ and $b_r \in R^{d_x}$ denote the bias vectors of encoder and decoder, respectively. The parameters of the autoencoder are shown below: $\theta = \{W, W', b_h, b_r\}$.

In order to improve the robustness of small perturbations around the training points and learn a mapping with stronger contraction effect on the training instances, we introduce a penalty term that penalizes the highly sensitive inputs to increase the robustness of the network in the form of the mapping *f(x)* of the encoder with respect to the Frobenius norm of the Jacobian matrix of the input *x* , and the sensitivity penalty term is the sum of squares of all partial derivatives for the extracted features according to input dimensions, as shown

in Eq. (3):

$$||J_f(x)||_F^2 = \sum_{ij}\left(\frac{\partial h_j(x)}{\partial x_i}\right)^2 \tag{3}$$

Assume the training set is $D_{tr,}$ we learn the parameters of the SCAE by minimizing the reconstruction error and penalizing the gradient. The entire loss function of SCAE is as follows:

$$L_{SCAE}(\theta) = \sum_{x\in D_{tr}} \left(L\left(x, g\left(f(x)\right)\right) + \lambda||J_f(x)||_F^2\right)$$

$$= \sum_{x\in D_{tr}}\left(-\frac{1}{n}\sum_x [y\ln a + (1-y)\ln(1-a)] + \lambda\sum_{ij}\left(\frac{\partial h_j(x)}{\partial x_i}\right)^2\right), \tag{4}$$

where L is the reconstruction error in the form of cross entropy loss (nonlinear error), and λ is a superparameter that controls the intensity of regularization, n presents the number of classes, y denotes the true value classification and a denotes the prediction value.

Multiple contractive autoencoders can be stacked to construct an unsupervised deep neural network SCAE with more than one hidden layer. The schematic diagram of SCAE is shown in Fig. 1, in which the output of previous hidden layer is the input of next hidden layer. In this paper, we train a SCAE with four contractive autoencoders to extract and reconstruct the defect features, where the output of the hidden layer for first contractive autoencoder is extracted as first-order feature representation, and then the first-order feature representation is regarded as the input of the hidden layer for second contractive autoencoder, and the same strategy is also used for the subsequent contractive autoencoders. Based on the above strategy, the SCAE can learn the first-order feature, second-order feature, third-order feature and fourth-order representations from the original defect features.

Through the continuous stacking process, the SCAE can extract more robust and abstract deep semantic features from the original defect features than a single contractive autoencoder. In addition, since the SCAE is an unsupervised model, it not only can prevent the training network from overfitting when the number of labeled defect instances is relatively small, but also can effectively achieve a deep combination of defect features with its nonlinear mapping capacity.

4 The proposed multi-objective SMONGE model

In this section, we propose a novel multi-objective defect prediction model called SMONGE that leverages the multi-objective NSGAII algorithm to optimize the number of hidden neurons and output weight norm of extreme learning machine (ELM). We first derive the training process of ELM, and then present the multi-objective optimization problem and our multi-objective SMONGE model.

4.1 Extreme learning machine

Different from traditional single hidden-layer feedforward neural network (SLFN), for ELM, the connection weights of the input layer and the hidden layer and the biases of the hidden layer can be assigned randomly, and need not be adjusted after setting. The connection weights between the hidden layer and the output layer do not need to be tuned

iteratively through the back propagation process of network error, which can be determined once by solving a linear model [Huang, Zhou, Ding et al. (2012)]. In addition, ELM has obvious advantages in classification, including strong classification capacity, fast training speed and easily adjust parameters. The network structure of ELM is shown in Fig. 2.

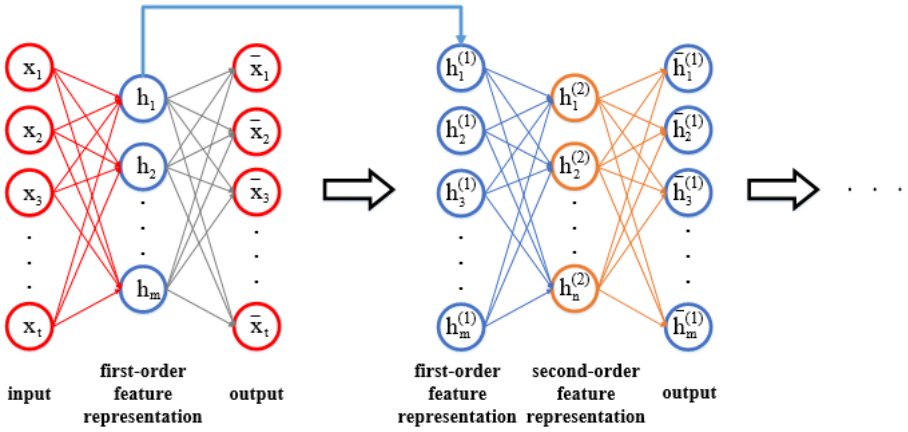


Figure 1: The schematic diagram of SCAE

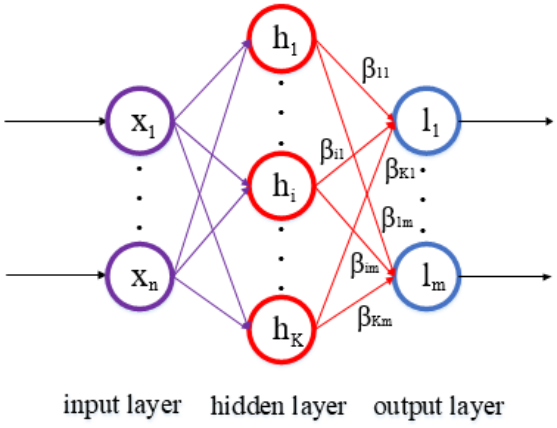


Figure 2: The network structure of ELM

Given a training dataset consisting of N arbitrary instances $\{(x_j, l_j)\}_{j=1}^{N}$, where $x_j = [x_{j1}, x_{j2}, \dots, x_{jn}]^T \in R^n$ contains n input features and $l_j = [l_{j1}, l_{j2}, \dots, l_{jm}]^T \in R^m$contains m output labels. For a standard SLFN with K hidden neurons, its output can be expressed as follows:

$$\sum_{i=1}^{K} \beta_i g(w_i \cdot x_j + b_i) = o_j, j = 1, 2, \dots, N, \quad (5)$$

where $g(.)$ denotes the activation function, $w_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T \in R^n$, $i=1,2,\dots,K$ represents the input weight vector between the input neurons and ith hidden neuron, b_i denotes the bias of the ith hidden neuron, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T \in R^m$ denotes the output

weight vector between the *i*th hidden neuron and output neurons, $o_j = [o_{j1}, o_{j2}, \dots, o_{jm}]^T \in R^m$ represents the network output value.

The learning goal of SLFN is to minimize the output error, which can be expressed as follows:

$$\sum_{j=1}^{N} ||o_j - l_j|| = 0. \tag{6}$$

The Eq. (6) can approximate zero error if there are suitable β_i, w_i, and b_i, which has been proved in Huang et al. [Huang, Chen and Siew (2006)]. Therefore, the Eq. (5) can be rewritten as Eq. (7):

$$\sum_{i=1}^{\mathrm{K}} \beta_i g(w_i \cdot x_j + b_i) = l_j, j = 1, 2, \dots, N. \tag{7}$$

The Eq. (7) can be transformed into a matrix form, as shown in Eq. (8):

$$H\beta = L, \tag{8}$$

where H is the output of the hidden neurons, β is the output weight, and L is the expected output.

H, β, L can be expressed respectively as follows:

$$\mathrm{H} = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_K \cdot x_1 + b_K) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_K \cdot x_N + b_{\mathrm{K}}) \end{bmatrix}_{N \times K}. \tag{9}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_K^T \end{bmatrix}_{K \times m}. \tag{10}$$

$$L = \begin{bmatrix} L_1^T \\ \vdots \\ L_N^T \end{bmatrix}_{N \times m}. \tag{11}$$

The output weight β can be computed by solving the linear least squares problem.

$$\beta = \mathrm{H}^{+}\mathrm{L}, \tag{12}$$

where H^{+} represents the Moore-Penrose generalized inverse of the matrix H.

4.2 Multi-objective NSGAII optimization based extreme learning machine

In this paper, we adopt extreme learning machine based on multi-objective NSGAII optimization to construct our defect prediction model, so as to transform software defect prediction into a multi-objective optimization problem based on state-of-the-art Pareto optimal solutions. We mainly consider two objectives. One objective is to maximize the performance (i.e., accuracy) of the constructed defect prediction model, which refers to the benefit of the prediction model. Another objective is to minimize the output weight norm as much as possible, which is related to the cost of the prediction model. There is a serious contradiction between these two objectives in most cases. The smaller the output weight norm, the smaller the influence of each feature component, which is equivalent to reducing the number of parameters, thus realizing the limitation of the model space. The simpler the model, the lower the cost, and the less likely it is to produce overfitting phenomenon. However, the performance of the prediction model may reduce to some extent as the weight

output norm decreases, and vice versa. Therefore, we should make a compromise between these two contradictory objectives.

In this section, we first give some definitions for multi-objective optimization. Then, we define the multi-objective optimization problem for software defect prediction. Finally, we introduce extreme learning machine based on multi-objective NSGAII optimization.

4.2.1 Definitions for multi-objective optimization

We give the following five definitions for multi-objective optimization based on Pareto optimal solutions. Since there are two optimization objectives in this paper, we take two optimization objectives as example.

Definition 1 (Multi-objective Optimization Problem)

$$F(x) = \big(f_1(x), f_2(x)\big), \tag{13}$$

$$s.t. x \in \Omega$$

where x is the decision vector and Ω is the decision space. $F(x)$: $\Omega \longrightarrow R^2$ contains two objective functions and R^2 represents the objective space.

Definition 2 (Pareto Dominance) Suppose x can dominate y, then $x \prec y$, if and only if

$\forall i \in \{1,2\}, f_1(x) \leq f_2(y), and\ \exists j \in \{1,2\}, s.t. f_j(x) < f_j(y)$.

Definition 3 (Pareto Optimal Solution and Pareto Optimal Vector) If and only if x^* is not dominated by other solutions, the solution x^* is called Pareto optimal solution. *F(x*)* is called a Pareto optimal vector.

Definition 4 (Pareto Optimal Set) The Pareto optimal set is composed by all the Pareto optimal solutions.

Definition 5 (Pareto Front) In the objective space, the surface composed by the target value vectors corresponding to all the Pareto optimal solutions is called Pareto front.

4.2.2 The multi-objective optimization problem

In this paper, we leverage the multi-objective NSGAII algorithm to optimize the number of hidden neurons and output weight norm of ELM. Therefore, the individual of the multi-objective optimization model includes the number of hidden neurons H and the control parameter λ of output weight norm. We define the initialized individual and population as follows:

$$I_{k,G} = [H_{k,G}, \lambda_{k,G}], \tag{14}$$

$$\phi_G = \{I_{1,G}, I_{2,G}, \dots, I_{NP,G}\}, \tag{15}$$

where $I_{k,G}$ is the kth individual of the Gth evolution generation and ϕ_G is the population with NP individuals. NP denotes the size of population. The output weight norm adopts the L2 norm, $\lambda \in (0,1]$.

In the multi-objective SMONGE model, the mean square error (MSE) and the control of output weight norm are regarded as two computable objectives. For each evolution generation of the SMONGE model, we utilize the parameter vector of each individual to

calculate the corresponding output weight according to the Eq. (12). The individual vector is generated by real encoding.

The objective function for the training MSE is defined as follows:

$$f_1(H) = \frac{1}{H}\sum_{h=1}^{H}(f(x) - y)^2. \tag{16}$$

Another objective is the control of output weight norm, as shown in Eq. (17):

$$f_2(\lambda) = \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(w_{i,j}^{l})^2, \tag{17}$$

where S_l denotes the number of neurons in *l*th layer, n_l denotes the number of network layers (ELM has three network layers), $w_{i,j}^{l}$ denotes the parameter between the *j*th neuron in *(l+1)*th layer and the *i*th neuron in *l*th layer, and λ denotes the weight decay parameter.

Considering the minimization of the above two objectives, we propose the multi-objective SMONGE model for software defect prediction, which is defined as follows:

$$\min_{H,\lambda} F(H,\lambda) = \{f_1(H), f_2(\lambda)\}^T. \tag{18}$$

$$s.t. \lambda \in (0,1]$$

The training MSE is used to enhance the classification accuracy while the control of output weight norm aims to make the cost of the ELM as low as possible and prevent overfitting. In the multi-objective SMONGE model, these two contradictory objectives are optimized simultaneously, thereby finding the Pareto optimal solutions.

4.2.3 The multi-objective SMONGE model

Since the ELM has strong classification capacity, we adopt ELM for software defect prediction. In order to further improve the prediction capacity of ELM, we utilize the multi-objective NSGAII algorithm to optimize number of hidden neurons and output weight norm of ELM, which is the above multi-objective optimization problem.

The learning process of the proposed SMONGE model is shown in Algorithm 1. In Algorithm 1, we first randomly initialize the population $\phi_0 = \{I_{k,0} | k = 1, 2, \ldots, NP\}$ and calculate the fitness values (multi-objective functions) of the initialized population by Eq. (18) in Steps 1 and 2. We can combine NSGAII and ELM closely to form a multi-objective optimization problem for software defect prediction by minimizing the multi-objective functions. Since ELM is used for software defect prediction in this paper, we need to calculate the output weight $\beta_{k,G}$ of ELM by Eq. (12) in Step 5. Next, we adopt the generateNewPop() function to produce new population Q_k by continuous selection, crossover and mutation in Step 6, and combine parent ϕ_k and offspring population Q_k to generate the p_k in Step 7. By the non-dominated sorting for F_i, we can obtain a set of classification subsets (all nondominated fronts of p_k) $F = (F_1, F_2, \ldots)$ in Step 8. We calculate crowding-distance for F_i (a measure of solutions density in the neighborhood), and select part of the individual F_i to merge into the new population ϕ_{k+1} until the population size reaches *NP* in Steps 11-15. Then, we establish a partial order relationship for F_i, and choose the first $(NP - |\phi_{k+1}|)$ elements of F_i until ϕ_{k+1} is filled in Steps 16-17.

After enough population evolution, SMONGE will meet the termination criteria and converge to a stable solutions. Finally, SMONGE can return all Pareto optimal solutions

in the current population, thereby obtaining software defect prediction results. In this process, in order to comprehensively reflect the performance of the defect prediction model, we also implement other prediction metrics, including accuracy, precision, recall, F1, pf, G-measure and MCC.

Algorithm 1 The multi-objective SMONGE model

Input:

Population size: *NP*, maximum evolution generation: *MAXGEN*, evolving population: ϕ_G

Output:

Pareto optimal solutions: R

Randomly initialize the population: $\phi_0 = \{I_{k,0} | k = 1, 2, \ldots, NP\}$;
Calculate the fitness values of the initialized population by Eq. (18);
while $G \leq MAXGEN$ **do**
  for *k=1, 2, …, NP* **do**
    Calculate the output weight $\beta_{k,G}$ by Eq. (12);
    $Q_k \longleftarrow generateNewPop(\phi_k)$;
    $p_k \longleftarrow \phi_k \cup Q_k$;
    $F \longleftarrow fast - don - dominated - sort(p_k)$;
    $\phi_{k+1} \longleftarrow null$;
    $i \longleftarrow 0$;
    while $|\phi_{k+1}| + |F_i| \leq NP$ **do**
      crowing-distance-assignment (F_i);
      $\phi_{k+1} \longleftarrow \phi_{k+1} \cup F_i$;
      $i \longleftarrow i + 1$;
    end while
    $sort(F_{i,} \prec n)$;
    $\phi_{k+1} \longleftarrow \phi_{k+1} \cup F_i[1: (NP - |\phi_{k+1}|)]$;
  end for
  $G \longleftarrow G + 1$;
end while
return Pareto optimal solutions: R

5 Experimental setup

In this section, we introduce the experimental setup, including benchmark datasets, evaluation metrics and baseline methods. We conduct the experiments on a 3.6 GHz i7-4790 CPU machine with 8 GB RAM.

5.1 Benchmark datasets

To verify the effectiveness of SCAE and SMONGE, we conduct extensive experiments on 20 real software projects (i.e., 5 projects from the NASA data repository and 15 projects from the PROMISE data repository), which are open source and commonly used benchmark datasets in software defect prediction studies [Tantithamthavorn, McIntosh, Hassan et al. (2016)]; Chen and Ma (2015); Hosseini, Turhan and Gunarathna (2019); Peters, Menzies and Layman (2015)]. The basic attributes of NASA (the first five rows) and PROMISE (the latter fifteen rows) are shown in Tab. 1, including project name, the number of features, the number of instances, the number of defective instances, the number

of non-defective instances, defective ratio, and imbalance ratio. For the NASA dataset, we can observe that the defect ratio of PC2 is the smallest with 2.15%, and the defect ratio of KC2 is the largest with 20.50%. The imbalance ratio varies from 3.88 to 45.56. Tab. 2 describes the features of 5 projects from the NASA data repository, which tabulates the 20 common features among the 5 projects and other 19 specific features for each project (The symbol ✓ represents that the project has a certain feature, while the symbol ✘ represents that the project does not have a certain feature). For the PROMISE dataset, we can observe that the defect ratio of jedit-4.3 is the smallest with 2.24%, and the defect ratio of xerces-init is the largest with 47.53%. The imbalance ratio varies from 1.10 to 43.73. Tab. 3 describes all features of 15 projects from the PROMISE data repository. Each instance in any project contains 20 object-oriented features and a dependent variable that presents the number of defects.

For all these software defect projects, we adopt the SMOTE (Synthetic Minority Oversampling Technique) algorithm [Chawla, Bowyer, Hall et al. (2002)] for class imbalance processing and the min-max method [Witten, Frank and Hall (2011)] for data normalization in this paper. In addition, we perform 10 times 10-fold cross-validation to evaluate the performance of these models in this paper.

Table 1: The statistics of 20 projects from the NASA and PROMISE data repository

Projects	#features	#instances	#defective instances	#non-defective instances	Defective ratio (%)	Imbalance ratio
KC2	21	522	107	415	20.50	3.88
MC1	38	1988	46	1942	2.31	42.22
MC2	39	125	16	109	12.80	6.81
PC1	37	705	61	644	8.65	10.56
PC2	36	745	16	729	2.15	45.56
ant-1.4	20	178	40	138	22.47	3.45
ant-1.5	20	293	32	261	10.92	8.16
ant-1.6	20	351	92	259	26.21	2.82
ant-1.7	20	745	166	579	22.28	3.49
ivy-1.4	20	241	16	225	6.64	14.06
ivy-2.0	20	352	40	312	11.36	7.80
jedit-4.0	20	306	75	231	24.51	3.08
jedit-4.1	20	312	79	233	25.32	2.95
jedit-4.2	20	367	48	319	13.08	6.65
jedit-4.3	20	492	11	481	2.24	43.73
poi-2.0	20	314	37	277	11.78	7.49
prop-6	20	660	66	594	10.00	9.00
xerces-1.2	20	440	71	369	16.14	5.20
xerces-1.3	20	453	69	384	15.23	5.57
xerces-init	20	162	77	85	47.53	1.10

Table 2: The feature description of 5 projects from the NASA data repository

Features	KC2	MC1	MC2	PC1	PC2
1. LOC_EXECUTABLE	✓	✓	✓	✓	✓
2. LOC_CODE_AND_COMMENT	✓	✓	✓	✓	✓
3. LOC_COMMENTS	✓	✓	✓	✓	✓
4. LOC_TOTAL	✓	✓	✓	✓	✓
5. DESIGN_COMPLEXITY	✓	✓	✓	✓	✓
6. ESSENTIAL_COMPLEXITY	✓	✓	✓	✓	✓
7. BRANCH_COUNT	✓	✓	✓	✓	✓
8. HALSTEAD_ERROR_EST	✓	✓	✓	✓	✓
9. HALSTEAD_DIFFICULTY	✓	✓	✓	✓	✓
10. HALSTEAD_EFFORT	✓	✓	✓	✓	✓
11. HALSTEAD_CONTENT	✓	✓	✓	✓	✓
12. HALSTEAD_LENGTH	✓	✓	✓	✓	✓
13. HALSTEAD_LEVEL	✓	✓	✓	✓	✓
14. HALSTEAD_PROG_TIME	✓	✓	✓	✓	✓
15. HALSTEAD_VOLUME	✓	✓	✓	✓	✓
16. NUM_OPERANDS	✓	✓	✓	✓	✓
17. NUM_OPERATORS	✓	✓	✓	✓	✓
18. NUM_UNIQUE_OPERANDS	✓	✓	✓	✓	✓
19. NUM_UNIQUE_OPERATORS	✓	✓	✓	✓	✓
20. CYCLOMATIC_COMPLEXITY	✓	✓	✓	✓	✓
21. GLOBAL_DATA_DENSITY	✘	✘	✓	✓	✓
22. LOC_BLANK	✓	✓	✓	✓	✘
23. CALL_PAIRS	✘	✓	✓	✓	✓
24. CYCLOMATIC_DENSITY	✘	✓	✓	✓	✓
25. DECISION_COUNT	✘	✓	✓	✓	✓
26. DESIGN_DENSITY	✘	✓	✓	✓	✓
27. EDGE_COUNT	✘	✓	✓	✓	✓
28. ESSENTIAL_DENSITY	✘	✓	✓	✓	✓
29. CONDITION_COUNT	✘	✓	✓	✓	✓
30. GLOBAL_DATA_COMPLEXITY	✘	✓	✓	✘	✘
31. DECISION_DENSITY	✘	✘	✓	✓	✓
32. MAINTENANCE_SEVERITY	✘	✓	✓	✓	✓
33. MODIFIED_CONDITION_COUNT	✘	✓	✓	✓	✓
34. NUMBER_OF_LINES	✘	✓	✓	✓	✓
35. MULTIPLE_CONDITION_COUNT	✘	✓	✓	✓	✓
36. NORMALIZED_CYLOMATIC_COMPLEXITY	✘	✓	✓	✓	✓
37. NODE_COUNT	✘	✓	✓	✓	✓
38. PERCENT_COMMENTS	✘	✓	✓	✓	✓
39. PARAMETER_COUNT	✘	✓	✓	✓	✓

Table 3: The feature description of 15 projects from the PROMISE data repository

1. Lines of Code (LOC)	11. Lack of Cohesion in Methods (LOCM)
2. Weighted Methods per Class (WMC)	12. Inheritance Coupling (IC)
3. Depth of Inheritance Tree (DIT)	13. Afferent Couplings (Ca)
4. Data Access Metric (DAM)	14. Coupling Between Methods (CBM)
5. Number of Children (NOC)	15. Efferent Couplings (Ce)
6. Measure of Aggregation (MOA)	16. Average Method Complexity (AMC)
7. Coupling between Object Classes (CBO)	17. Arithmetic mean value of CC (Avg_CC)
8. Measure of Functional Abstraction (MFA)	18. Greatest Value of CC (Max_CC)
9. Response for a Class (RFC)	19. Lack of Cohesion in Methods (LOCM3)
10. Cohesion Among Methods of Class (CAM)	20. Number of Public Methods (NPM)

5.2 Evaluation metrics

In this paper, we adopt seven widely used evaluation metrics–accuracy, precision, recall, F1, pf, G-measure and MCC [Kondo, Bezemer, Kamei et al. (2019); Nam, Pan and Kim (2013); He, Shu, Yang et al. (2012); Herbold, Trautsch and Grabowski (2018)] to evaluate the models. These evaluation metrics can be defined based on confusion matrix, which lists all four possible classification results, i.e., TP, FP, FN and TN, as shown in Tab. 4.

Table 4: Confusion matrix

	Positive (Predicted)	Negative (Predicted)
True (Actual)	TP	FN
Flase (Actual)	FP	TN

Accuracy: The ratio of correctly predicted defect files to all files.

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN}. \tag{19}$$

Precision: The ratio of correctly predicted defect files to all files predicted to be defective.

$$precision = \frac{TP}{TP+FP}. \tag{20}$$

Recall or pd (probability of detection): The ratio of correctly predicted defect files to all truly defective files.

$$recall(pd) = \frac{TP}{TP+FN}. \tag{21}$$

F1: The harmonic means between precision and recall.

$$F1 = \frac{2\times precision\times recall}{precision+recall}. \tag{22}$$

pf (probability of false alarm): The ratio of the number of non-defective instances that are wrongly classified as defective to the total number of non-defective instances.

$$pf = \frac{FP}{FP+TN}. \tag{23}$$

G-measure: The harmonic means of *pd* and 1-*pf*.

$$G-measure = \frac{2\times pd\times(1-pf)}{pd+(1-pf)}. \tag{24}$$

MCC (Matthews correlation coefficient): The correlation between the actual and predicted outputs which is a comprehensive evaluation by considering TP, TN, FP and FN.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}. \quad (25)$$

Except for pf, the larger the values of these metrics, the better the prediction performance.

5.3 Baseline methods

To verify the performance of SCAE and SMONGE, we conduct extensive experiments for feature extraction and software defect prediction.

For feature extraction, we compare the deep neural network SCAE with six state-of-the-art feature extraction methods, including Maximally Collapsing Metric Learning (MCML) [Globerson and Roweis (2005)], Stochastic Neighbor Embedding (SNE) [Parviainen (2016)], Manifold Charting (MC) [Saini, Rambli, Sulaiman et al. (2013)], Locality Preserving Projection (LPP) [Lu, Wang, Zou et al. (2018)], Locally Linear Embedding (LLE) [Ji, Liu, Cao et al. (2017)], Locally Linear Coordination (LLC) [Huang, Wang, Xu et al. (2009)].

For software defect prediction, we compare the SMONGE model with four classic defect predictor, include Decision Tree (DT), K-Nearest Neighbor (KNN), Naive Bayes (NB), Support Vector Machine (SVM). Since these four defect predictors all use features extracted by SCAE, they are named as SDT, SKNN, SNB and SSVM respectively in this paper, where S denotes SCAE.

In addition, we also compare SMONGE with the multi-objective NSGAII optimization based the ELM predictor that does not use features extracted by SCAE, and the method is named MONGE.

6 Experimental results

We introduce the experimental results by the following three research questions (RQ) in the section.

RQ1: How about the performance of the deep neural network SCAE compared to six state-of-the-art feature extraction methods in software defect prediction?

To validate the effectiveness of the deep semantic features extracted by SCAE, we compare the SCAE with six state-of-the-art feature extraction methods with the same defect predictor-the multi-objective NSGAII optimization based ELM, including MCML, SNE, MC, LPP, LLE and LLC. We conduct extensive experiments across 20 projects in terms of F1, pf, G-measure and MCC.

Tabs. 5-8 show the F1, pf, G-measure and MCC of SCAE and six state-of-the-art feature extraction methods across all 20 projects. Note that the best value of each project is in bold font. From Tabs. 5, 7 and 8, we can observe that our method SCAE achieves the best average performance in terms of F1, G-measure and MCC. More specifically, the average F1 (0.8088) by SCAE achieves improvements between 4.62% (for SNE) and 28.75% (for LLC) with an average improvement of 13.25%, the average G-measure (0.7675) by SCAE yields improvements between 2.51% (for SNE) and 42.95% (for MC) with an average improvement of 21.99% and the average MCC (0.5694) by SCAE gains improvements between 6.61% (for SNE) and 87.73% (for MC) with an average improvement of 43.39%.

In addition, from Tab. 6, we can find that our method SCAE is not ideal in terms of pf, but it is better than MCML, MC and LLE (the smaller the pf, the better the performance).

Fig. 3 depicts the box-plots of four metrics for our method SCAE and six feature extraction methods across all 20 projects. From Figs. 3(a), 3(c) and 3(d), we can find that the median values achieved by SCAE are higher than those achieved by six feature extraction methods from the point of F1, G-measure and MCC, respectively, which can fully demonstrate the superiority of our method SCAE, and the cases are consistent with the observations in Tabs. 5, 7 and 8. Moreover, the lowest F1, G-measure and MCC by SCAE are higher than the median values by MCML, MC, LLE and LLC, respectively.

Conclusion: Our method SCAE outperforms six state-of-the-art feature extraction methods in terms of F1, G-measure and MCC. The SCAE yields the average 13.25%, 21.99% and 43.39% performance improvements compared with six feature extraction methods across all 20 projects in terms of F1, G-measure and MCC.

Table 5: The F1 for our method SCAE compared with six feature extraction methods

Datasets	MCML	SNE	MC	LPP	LLE	LLC	SCAE
KC2	0.7368	0.7848	0.4727	**0.8293**	0.6667	0.5806	0.7857
MC1	0.7098	0.6000	0.6957	0.7939	**0.8868**	0.7219	0.7867
MC2	0.6957	0.7619	**0.7692**	0.6667	0.7000	0.5000	0.7273
PC1	0.8834	0.8662	0.7536	0.6667	**0.8929**	0.4554	0.8606
PC2	0.7727	**0.9221**	0.8917	0.6000	0.915	0.661	0.8659
ant-1.4	0.6857	**0.7879**	0.7500	0.7273	0.7273	0.7179	0.7647
ant-1.5	0.7385	**0.9180**	0.7273	0.7500	0.8000	0.6957	0.8197
ant-1.6	0.8060	0.8254	0.7595	0.6792	0.7606	0.7467	**0.8438**
ant-1.7	0.6719	0.7742	0.7114	0.7778	0.6957	0.7000	**0.8472**
ivy-1.4	0.6500	0.7317	0.5263	0.6471	0.7568	0.6486	**0.8000**
ivy-2.0	0.7385	**0.7931**	0.4545	0.7241	0.7826	0.7077	0.7797
jedit-4.0	0.7018	0.7500	0.6875	0.7241	0.7273	0.3636	**0.7719**
jedit-4.1	0.7660	0.6522	0.5833	0.7111	0.7742	0.5143	**0.8163**
jedit-4.2	0.7945	0.8205	0.5000	0.7353	**0.8605**	0.2917	0.8354
jedit-4.3	0.8807	0.9204	**0.9369**	0.9174	0.8548	0.8932	0.8952
poi-2.0	0.5965	0.7692	0.5217	**0.8525**	0.7500	0.7606	0.8308
prop-6	0.6569	0.7869	0.7344	**0.8305**	0.6667	0.8130	0.7759
xerces-1.2	0.6301	0.8276	0.7257	0.7765	0.7273	0.5867	**0.8471**
xerces-1.3	0.7619	**0.8378**	0.6383	0.8611	0.6667	0.6190	0.7857
xerces-init	0.6316	0.3333	0.5882	0.7000	0.6364	0.5882	**0.7368**
Avg	0.7254	0.7731	0.6713	0.7485	0.7624	0.6282	**0.8088**

Table 6: The pf for our method SCAE compared with six feature extraction methods

Datasets	MCML	SNE	MC	LPP	LLE	LLC	SCAE
KC2	0.4651	0.1860	**0.0465**	0.186	0.8837	0.0930	0.2558
MC1	0.4872	0.6250	0.2359	0.2205	**0.2154**	0.2308	0.3179
MC2	0.6667	0.3333	0.8333	**0.1667**	0.3333	**0.1667**	0.5000
PC1	0.2830	0.2453	0.1887	0.6250	0.3208	**0.0377**	0.3396
PC2	0.4865	0.1486	0.2027	0.6250	0.1486	**0.0946**	0.2838
ant-1.4	0.6154	**0.3846**	**0.3846**	0.4615	0.6250	0.7692	0.4615
ant-1.5	0.5000	0.1667	0.8333	0.2500	0.4167	**0.0417**	0.2917
ant-1.6	0.4000	0.2500	0.8500	**0.1500**	0.6000	0.7500	0.2500
ant-1.7	0.3673	**0.1837**	0.5918	0.2041	0.7500	0.2245	0.3265
ivy-1.4	0.3333	0.2963	0.3704	**0.1852**	**0.1852**	0.2593	0.2222
ivy-2.0	0.3889	0.2222	**0.1944**	0.2778	0.4167	0.4167	0.2500
jedit-4.0	0.5455	0.2273	0.7727	0.5455	0.7727	**0.0909**	0.4545
jedit-4.1	0.1818	0.2727	0.4091	0.1818	0.5909	**0.0455**	0.1818
jedit-4.2	0.1667	0.2500	0.2083	0.1250	0.3750	**0.0417**	0.2500
jedit-4.3	0.1818	0.1818	0.1364	0.1364	0.4091	**0.0909**	0.1136
poi-2.0	0.3846	0.3846	**0.1538**	0.1923	0.7692	0.5385	0.3077
prop-6	0.5574	0.2623	0.3770	**0.1803**	0.6250	0.2459	0.2131
xerces-1.2	0.2500	0.2813	0.9375	0.3125	0.6250	0.3438	**0.2188**
xerces-1.3	0.4048	0.1905	0.6905	**0.1429**	0.8750	0.5476	0.3810
xerces-init	0.5000	**0.1250**	0.3750	0.5000	0.7500	0.3750	0.3750
Avg	0.4083	**0.2608**	0.4395	0.2834	0.5343	0.2702	0.2997

RQ2: How about the prediction performance of the proposed multi-objective SMONGE model compared to four classic defect predictors with the same feature extraction method SCAE?

Our multi-objective SMONGE model combines the feature extraction method SCAE and the ELM optimized by the multi-objective NSGAII algorithm. Since we adopt the ELM optimized by the multi-objective NSGAII algorithm as the defect predictor in this paper, this question is designed to evaluate the effectiveness of the SMONGE model compared with four classic defect predictors with the same feature extraction method SCAE, including SDT, SKNN, SNB and SSVM.

Tabs. 9-12 show the F1, pf, G-measure and MCC of the SMONGE model compared with those of four classic predictors across all 20 projects, respectively. Note that the best value of each project is in bold font. From Tabs. 9, 11 and 12, we can find that the SMONGE model can achieve the best average performance in terms of F1, G-measure and MCC (expect for SSVM) across all 20 projects. More specifically, the average F1 (0.8088) by SMONGE achieves improvements between 3.75% (for SKNN) and 19.88% (for SNB) with an average improvement of 9.09% and the average G-measure (0.7675) by SMONGE yields improvements between 0.67% (for SDT) and 16.87% (for SNB) with an average improvement of 4.84% compared with four classic predictors with the same feature

extraction method SCAE. From the point of MCC, the SMONGE model can gain an average improvement of 14.79% compared with four classic predictors, and it is only 0.2% worse than SSVM. Moreover, from Tab. 10, the SMONGE is not the best predictor from the point of pf, but it is only worse than SDT and SSVM.

Table 7: The G-measure for our method SCAE compared with six feature extraction methods

Datasets	MCML	SNE	MC	LPP	LLE	LLC	SCAE
KC2	0.6639	0.7940	0.4848	**0.8316**	0.2078	0.6015	0.7825
MC1	0.6309	0.4800	0.7081	0.7916	**0.8671**	0.7307	0.7591
MC2	0.4571	**0.6957**	0.2817	0.6593	0.6512	0.5063	0.5926
PC1	0.8162	**0.8188**	0.7424	0.506	0.8046	0.4605	0.7738
PC2	0.6653	**0.9138**	0.8761	0.4800	0.9078	0.6778	0.8298
ant-1.4	0.5195	**0.7197**	0.6957	0.6437	0.5275	0.3700	0.6642
ant-1.5	0.6234	**0.8946**	0.2843	0.7368	0.7068	0.7003	0.7777
ant-1.6	0.7013	0.7800	0.2586	0.6770	0.5427	0.3889	**0.7941**
ant-1.7	0.6372	0.7631	0.5385	0.7623	0.3902	0.6933	**0.7742**
ivy-1.4	0.6933	0.7631	0.5903	0.6984	0.7959	0.7018	**0.8296**
ivy-2.0	0.7243	**0.8131**	0.5074	0.7490	0.7368	0.6925	0.7977
jedit-4.0	0.5797	**0.7454**	0.3612	0.5899	0.3675	0.3797	0.6735
jedit-4.1	0.7660	0.6575	0.5750	0.7182	0.5737	0.5228	**0.8090**
jedit-4.2	0.7754	0.7742	0.5089	0.7292	0.7460	0.2960	**0.7857**
jedit-4.3	0.8597	0.8923	**0.9186**	0.9018	0.7429	0.8880	0.8866
poi-2.0	0.5900	0.7080	0.5432	**0.8361**	0.3750	0.6102	0.7826
prop-6	0.5637	0.7801	0.7044	**0.8321**	0.5060	0.8045	0.7813
xerces-1.2	0.6330	0.7819	0.1175	0.7333	0.5275	0.5826	**0.8174**
xerces-1.3	0.7210	0.8459	0.4548	**0.8712**	0.2192	0.5623	0.7474
xerces-init	0.5714	0.3544	0.5882	0.6087	0.3784	0.5882	**0.6931**
Avg	0.6596	0.7487	0.5369	0.7178	0.5787	0.5878	**0.7675**

Fig. 4 shows the box-plots of four metrics for our SMONGE model and four classic predictors with the same feature extraction method SCAE across all 20 projects. In Figs. 4 (a), 4(c) and 4(d), the median value by SMONGE is higher than that of four predictors in terms of F1, G-measure and MCC (expect for SSVM in terms of G-measure and MCC), respectively. In particular, the median value by SMONGE is higher than the maximum value of SNB in terms of F1, G-measure and MCC. In addition, we can observe that the median pf by SMONGE is only higher than that of SKNN and SNB respectively from Fig. 4(b), which also shows that the SMONGE is not good enough in terms of pf.

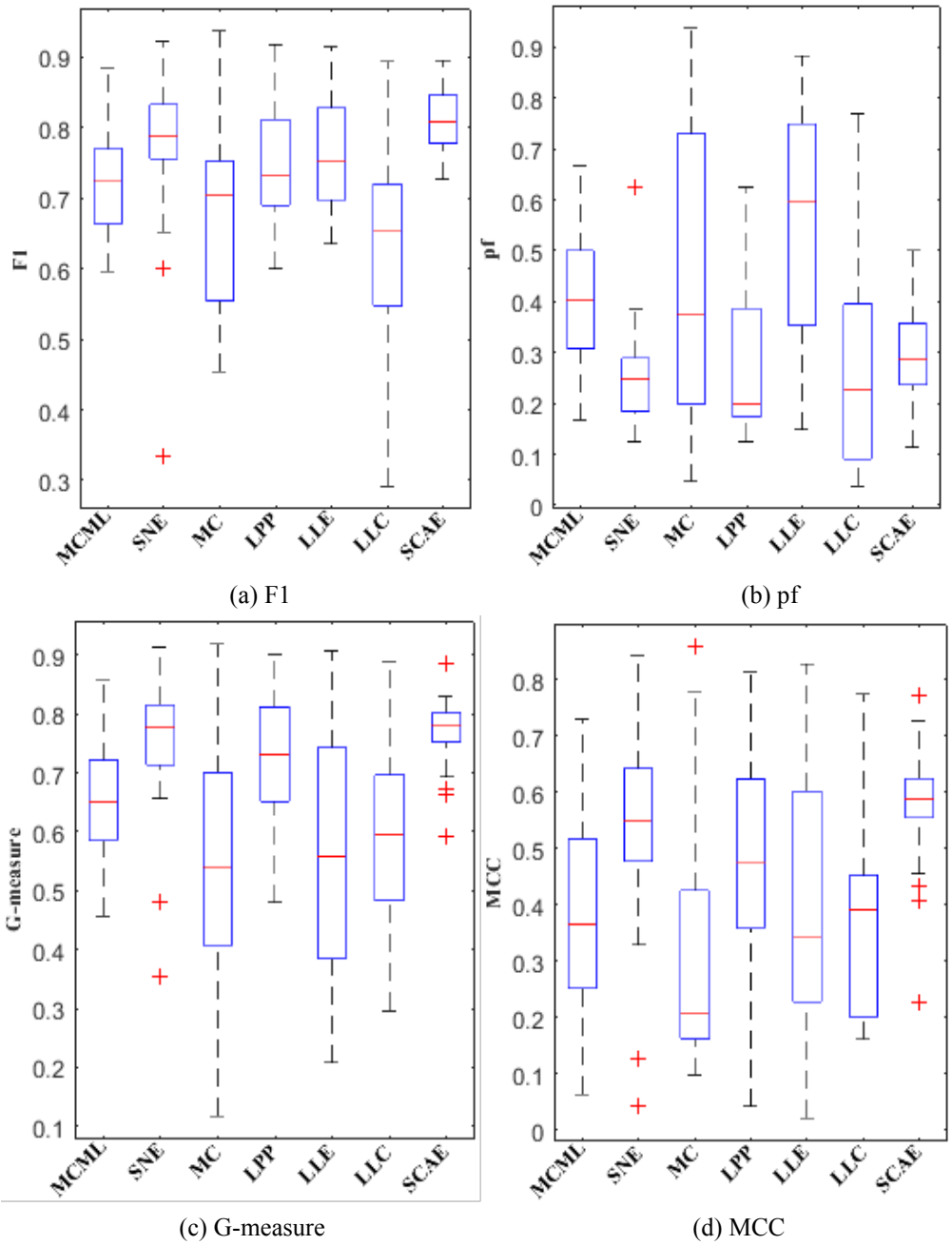
Conclusion: Our multi-objective SMONGE model performs better than four classic predictors in terms of F1, G-measure and MCC (expect for SSVM) on average. The SMONGE achieves the average 9.09%, 4.84% and 14.79% performance improvements compared with four classic defect predictors across all 20 projects in terms of F1, G-measure and MCC.

Table 8: The MCC for our method SCAE compared with six feature extraction methods

Datasets	MCML	SNE	MC	LPP	LLE	LLC	SCAE
KC2	0.4332	**0.5896**	0.3616	0.6636	0.1761	0.4041	0.5698
MC1	0.3491	0.0435	0.4263	0.5838	**0.7666**	0.4664	0.5459
MC2	0.0636	**0.3825**	0.1124	0.3678	0.2901	0.2066	0.2273
PC1	0.6975	0.6610	0.4880	0.1674	**0.7246**	0.3297	0.6325
PC2	0.5058	**0.8438**	0.7801	0.0435	0.8285	0.4812	0.7273
ant-1.4	0.2038	**0.5017**	0.4242	0.3523	0.3105	0.2339	0.4326
ant-1.5	0.3493	**0.8130**	0.2251	0.4721	0.5118	0.5439	0.5805
ant-1.6	0.4602	0.5577	0.1444	0.409	0.2739	0.1607	**0.5938**
ant-1.7	0.2715	0.5264	0.2162	0.521	0.1818	0.399	**0.6105**
ivy-1.4	0.3811	0.5263	0.1826	0.4359	0.5878	0.4041	**0.6534**
ivy-2.0	0.5058	**0.6233**	0.1961	0.4949	0.6124	0.4402	0.5958
jedit-4.0	0.2725	**0.4918**	0.1424	0.3214	0.2800	0.1979	0.4554
jedit-4.1	0.5382	0.3285	0.1506	0.4624	0.4500	0.3835	**0.6171**
jedit-4.2	0.5409	0.5422	0.1741	0.4880	**0.5922**	0.1952	0.5706
jedit-4.3	0.7295	0.8192	**0.8577**	0.8133	0.6642	0.7740	0.7718
poi-2.0	0.1817	0.4623	0.2717	**0.6765**	0.3721	0.4072	0.6102
prop-6	0.2312	0.5667	0.4401	**0.6643**	0.1674	0.6186	0.5627
xerces-1.2	0.2988	0.5844	0.0972	0.4752	0.3105	0.1795	**0.6410**
xerces-1.3	0.5274	0.6924	0.1961	**0.7403**	0.0215	0.2021	0.5816
xerces-init	0.1690	0.1273	0.1806	0.2901	0.0327	0.1806	**0.4085**
Avg	0.3855	0.5341	0.3033	0.4721	0.4077	0.3604	**0.5694**

RQ2a: For the novel multi-objective SMONGE model, what is the minimum output weight norm of ELM while achieving the best accuracy on each project?

For the proposed multi-objective SMONGE model, the model utilizes the multi-objective NSGAII algorithm to optimize two objectives of the advanced ELM predictor based on state-of-the-art Pareto optimal solutions. One objective is to maximize the model performance, which refers to the benefit of the prediction model. Another objective is to minimize the output weight norm, which is related to the cost of the prediction model. Therefore, we show the best classification accuracy and the minimum output weight norm of ELM gained by the multi-objective SMONGE model on each project in Tab. 13.



(a) F1 (b) pf

(c) G-measure (d) MCC

Figure 3: The box-plots (the traditional mode) for our method SCAE compared with six feature extraction methods in terms of four metrics

Table 9: The F1 for SMONGE compared with four classic defect predictors

Datasets	SDT	SKNN	SNB	SSVM	SMONGE
KC2	0.7752	0.7757	0.7708	**0.8177**	0.7857
MC1	0.7510	**0.7928**	0.6405	0.7000	0.7867
MC2	**0.7663**	0.7023	0.5000	0.6666	0.7273
PC1	0.7766	0.7325	0.6521	0.8167	**0.8606**
PC2	0.7735	0.7567	0.6041	0.7999	**0.8659**
ant-1.4	0.7647	0.7500	0.6666	**0.8162**	0.7647
ant-1.5	0.7927	**0.8372**	0.7428	0.8165	0.8197
ant-1.6	0.7183	0.8157	0.7341	0.8275	**0.8438**
ant-1.7	0.7531	0.8134	0.7613	0.8279	**0.8472**
ivy-1.4	0.7959	0.7914	0.7422	**0.8045**	0.8000
ivy-2.0	**0.8135**	0.8015	0.7333	0.6700	0.7797
jedit-4.0	0.7353	0.7520	0.7594	0.6419	**0.7719**
jedit-4.1	0.7472	0.7173	0.7123	0.7916	**0.8163**
jedit-4.2	0.8000	0.8259	0.6666	0.8223	**0.8354**
jedit-4.3	0.7288	0.8915	0.6046	0.8162	**0.8952**
poi-2.0	0.8108	0.8108	0.6041	0.5688	**0.8308**
prop-6	0.7637	0.7141	0.5322	0.4904	**0.7759**
xerces-1.2	0.8375	0.8345	0.6212	0.7967	**0.8471**
xerces-1.3	0.7750	0.7734	0.7654	0.7732	**0.7857**
xerces-init	**0.7766**	0.7023	0.6800	0.7000	0.7368
Avg	0.7728	0.7796	0.6747	0.7482	**0.8088**

RQ3: Do the deep semantic features extracted by the deep neural network SCAE (for SMONGE) have advantage in prediction performance compared with the original defect features without SCAE (for MONGE)?

Prior researches demonstrate that the performance of a prediction model is usually determined by only a few features [Menzies, Greenwald and Frank (2007)]. Feature extraction technique is an effective means to alleviate this problem by constructing new, combined features from the original features. We adopt the deep neural network SCAE to extract deep semantic features of software defects in this paper. In this experiment, we compare the deep semantic features extracted by SCAE (for SMONGE) with the original defect features without SCAE (for MONGE), so as to validate the effect of the SCAE on ELM based on the multi-objective NSGAII optimization.

Fig. 5 presents the average precision, recall, F1, pf, G-measure and MCC of the features extracted by SCAE (for SMONGE) compared with the original features without feature extraction method SCAE (for MONGE) with the same defect predictor-the multi-objective NSGAII optimization based ELM on the NASA and PROMISE, respectively. From Fig. 5, we can observe that the features extracted by SCAE perform better than the original features without SCAE on all evaluation indicators. More specifically, the average precision (0.7550), recall (0.8657), F1 (0.8052), pf (0.3394), G-measure (0.7476) and MCC (0.5406) by SCAE yield improvement 3.20%, 0.16%, 1.86%, 10.57%, 3.85% and 7.03% compared

with the original features without feature extraction method SCAE on NASA respectively, and the average precision (0.7718), recall (0.8580), F1 (0.8100), pf (0.2865), G-measure (0.7743) and MCC (0.5790) by SCAE achieve improvement 4.23%, 7.67%, 6.03%, 8.60%, 5.94% and 19.06% compared with the original features without feature extraction method SCAE on PROMISE respectively.

Conclusion: Deep semantic features extracted by deep neural network SCAE (for SMONGE) can boost the prediction performance of ELM based on the multi-objective NSGAII optimization compared with the original defect features without SCAE (for MONGE).

Table 10: The pf for SMONGE compared with four classic defect predictors

Datasets	SDT	SKNN	SNB	SSVM	SMONGE
KC2	**0.1895**	0.1975	0.5000	0.2526	0.2558
MC1	0.3259	0.3109	0.3393	**0.1339**	0.3179
MC2	0.2340	0.3875	**0.1250**	0.2500	0.5000
PC1	0.1702	0.3875	0.1489	**0.1422**	0.3396
PC2	0.2127	0.3888	0.1333	**0.1087**	0.2838
ant-1.4	0.1702	0.3809	0.1772	**0.1450**	0.4615
ant-1.5	0.3151	0.3090	0.3389	**0.0852**	0.2917
ant-1.6	0.3238	0.2772	0.2500	**0.0535**	0.2500
ant-1.7	0.2966	0.2520	0.2222	**0.1016**	0.3265
ivy-1.4	0.3260	0.3025	0.6562	**0.1739**	0.2222
ivy-2.0	0.2737	0.3218	0.5102	**0.1399**	0.2500
jedit-4.0	0.3285	0.3582	0.3421	**0.1363**	0.4545
jedit-4.1	0.2745	0.3269	0.3000	**0.1290**	0.1818
jedit-4.2	0.3050	0.3121	0.2181	**0.1082**	0.2500
jedit-4.3	0.3089	0.1908	0.1898	0.1450	**0.1136**
poi-2.0	0.2241	0.2372	0.5000	**0.0815**	0.3077
prop-6	0.2139	0.3715	0.3658	**0.0706**	0.2131
xerces-1.2	0.2361	**0.1518**	0.4615	0.2146	0.2188
xerces-1.3	0.2577	0.2676	0.3584	**0.1433**	0.3810
xerces-init	0.1702	0.3875	**0.1612**	0.3846	0.3750
Avg	0.2578	0.3060	0.3149	**0.1500**	0.2997

Table 11: The G-measure for SMONGE compared with four classic defect predictors

Datasets	SDT	SKNN	SNB	SSVM	SMONGE
KC2	0.7801	0.7769	0.6527	**0.8017**	0.7825
MC1	0.7300	**0.7650**	0.6457	0.7155	0.7591
MC2	**0.7555**	0.6768	0.5308	0.75	0.5926
PC1	0.7751	0.6966	0.6648	**0.8191**	0.7738
PC2	0.7657	0.7196	0.6272	0.8060	**0.8298**
ant-1.4	0.7647	0.7188	0.6903	**0.8197**	0.6642
ant-1.5	0.7725	0.8057	0.7428	**0.8217**	0.7777

ant-1.6	0.7125	0.7892	0.7467	**0.8368**	0.7941
ant-1.7	0.7439	0.8060	0.7739	**0.8353**	0.7742
ivy-1.4	0.7656	0.7771	0.4974	0.8104	**0.8296**
ivy-2.0	0.7862	0.7644	0.6248	0.6878	**0.7977**
jedit-4.0	0.7196	**0.7388**	0.7352	0.6573	0.6735
jedit-4.1	0.7566	0.7250	0.7207	**0.8117**	0.8090
jedit-4.2	0.7608	0.7862	0.6946	**0.8273**	0.7857
jedit-4.3	0.7233	0.8789	0.6334	0.8197	**0.8866**
poi-2.0	**0.8108**	**0.8108**	0.5631	0.5855	0.7826
prop-6	0.7676	0.6870	0.5838	0.5060	**0.7813**
xerces-1.2	0.8185	**0.8443**	0.5728	0.7997	0.8174
xerces-1.3	0.7644	0.7639	0.7292	**0.7804**	0.7474
xerces-init	**0.7751**	0.6768	0.7043	0.7225	0.6931
Avg	0.7624	0.7604	0.6567	0.7607	**0.7675**

7 Threats to validity

In this section, we introduce the potential threats to validity of our method, including internal validity, external validity and construct validity.

7.1 Internal validity

Internal validity is mainly concerned with uncontrolled internal factors that may affect our experimental results, such as errors in the experiment. We check all experiment process carefully, but there may still be errors in the experiment that we don't notice.

7.2 External validity

External validity involves that whether our experimental results can be generalized to other software subjects. To guarantee the representative of software subjects used in this paper, we use 15 projects from the PROMISE data repository and 5 projects from the NASA data repository, which are commonly used projects in previous software defect prediction studies [Tantithamthavorn, McIntosh, Hassan et al. (2016)]; Chen and Ma (2015); Hosseini, Turhan and Gunarathna (2019); Peters, Menzies and Layman (2015)]. Moreover, these software projects belong to different application fields and cover a long time.

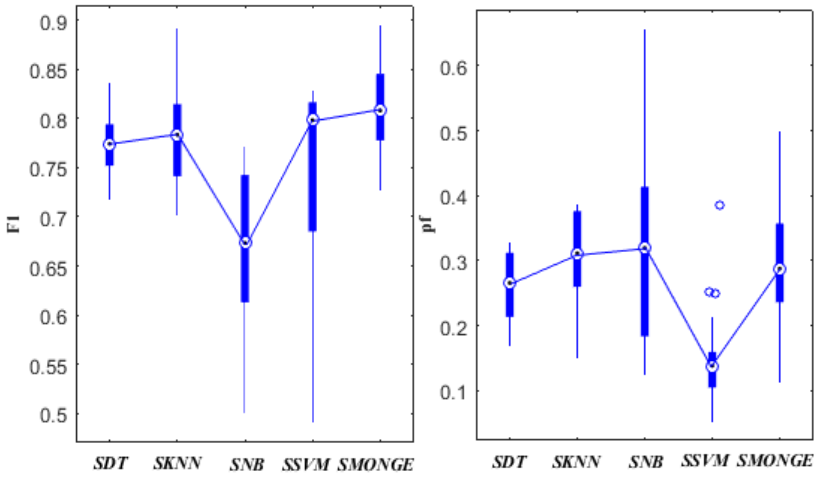
Table 12: The MCC for SMONGE compared with four classic defect predictors

Datasets	SDT	SKNN	SNB	SSVM	SMONGE
KC2	0.5633	0.5556	0.4899	**0.6164**	0.5698
MC1	0.4739	**0.5573**	0.2922	0.4916	0.5459
MC2	**0.5099**	0.3725	0.2971	0.4780	0.2273
PC1	0.5563	0.4279	0.4110	**0.6425**	0.6325
PC2	0.5310	0.4992	0.3869	0.6331	**0.7273**
ant-1.4	0.5388	0.4862	0.4299	**0.6434**	0.4326
ant-1.5	0.5786	**0.6766**	0.5088	0.6683	0.5805

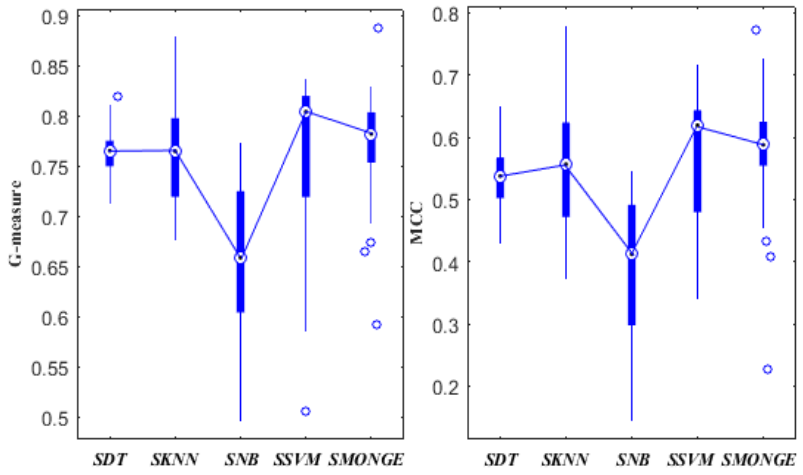
ant-1.6	0.4298	0.5997	0.4930	**0.7171**	0.5938
ant-1.7	0.4942	0.6243	0.5472	**0.6846**	0.6105
ivy-1.4	0.5716	0.5801	0.2982	0.6220	**0.6534**
ivy-2.0	0.5901	0.5664	0.3810	0.4516	**0.5958**
jedit-4.0	0.4490	**0.5170**	0.4976	0.4143	0.4554
jedit-4.1	0.5144	0.4571	0.4419	**0.6375**	0.6171
jedit-4.2	0.5434	0.6231	0.4129	**0.6672**	0.5706
jedit-4.3	0.4504	**0.7790**	0.3458	0.6434	0.7718
poi-2.0	0.6249	**0.6280**	0.1460	0.3988	0.6102
prop-6	0.5364	0.3896	0.1744	0.3402	**0.5627**
xerces-1.2	0.6512	**0.6881**	0.1498	0.5995	0.6410
xerces-1.3	0.5311	0.5319	0.4962	0.5777	**0.5816**
xerces-init	**0.5563**	0.3725	0.4604	0.4812	0.4085
Avg	0.5347	0.5466	0.3830	**0.5704**	0.5694

Table 13: The best classification accuracy and the minimum output weight norm of ELM achieved by the multi-objective SMONGE model

Datasets	**Accuracy**	Norm	**Datasets**	**Accuracy**	Norm
KC2	0.7831	3.9937	ivy-2.0	0.7937	14.0805
MC1	0.7686	34.4994	jedit-4.0	0.7234	23.3308
MC2	0.6471	145.5707	jedit-4.1	0.8085	60.2265
PC1	0.8217	1.7726e+05	jedit-4.2	0.7969	136.0188
PC2	0.8493	1.3278e+03	jedit-4.3	0.8866	93.3310
ant-1.4	0.7143	731.5775	poi-2.0	0.8036	299.3705
ant-1.5	0.7925	9.9551	prop-6	0.7815	1.2676e+03
ant-1.6	0.8077	3.9052	xerces-1.2	0.8243	61.6106
ant-1.7	0.8103	3.6909	xerces-1.3	0.7662	43.7408
ivy-1.4	0.8222	4.6603	xerces-init	0.7059	2.6412

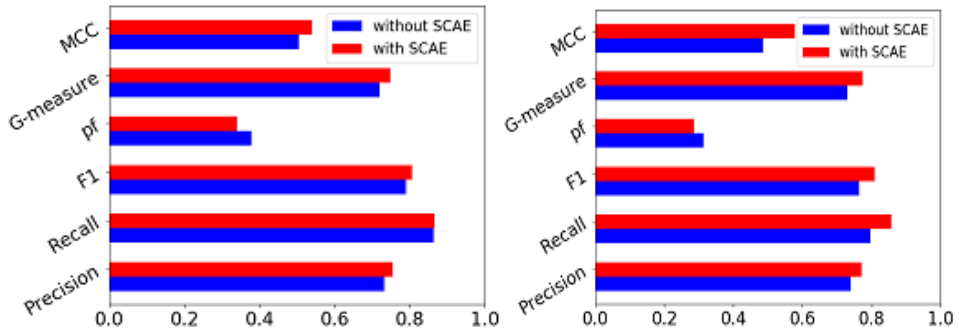


(a) F1 (b) pf



(c) G-measure (d) MCC

Figure 4: The box-plots (the compact mode) for our proposed SMONGE compared with four classic predictors in terms of four metrics



(a) NASA (b) PROMISE

Figure 5: The average performance comparison of SMONGE with SCAE and without SCAE on NASA and PROMISE datasets

7.3 Construct validity

Construct validity is related to whether the evaluation metrics used in our study reflect the real-world situation. To minimize the threat, we use seven evaluation metrics, including accuracy, precision, recall, F1, pf, G-measure and MCC which have been widely used in recent software defect prediction studies [Kondo, Bezemer, Kamei et al. (2019); Nam, Pan and Kim (2013); He, Shu, Yang et al. (2012); Herbold, Trautsch and Grabowski (2018); Zhu, Zhang, Ying et al. (2020)], so we believe that the construct validity should be acceptable.

8 Conclusion

In this work, we apply an advanced feature extraction method and a novel multi-objective optimization model to software defect prediction. First, we utilize an advanced deep neural

network SCAE to extract the robust deep semantic features, which has stronger discrimination capacity for different classes. Second, we propose a novel multi-objective defect prediction model called SMONGE, which leverages the multi-objective NSGAII algorithm to optimize two objectives of the advanced ELM predictor based on state-of-the-art Pareto optimal solutions. One objective is to maximize the model performance, which refers to the benefit of the prediction model. Another objective is to minimize the output weight norm, which is related to the cost of the prediction model. We conduct extensive experiments for feature extraction and defect prediction across 20 software defect projects from large open source datasets, and the experimental results verify that the effectiveness of SCAE and SMONGE.

In future work, to verify generalization capability and practicability of SCAE and SMONGE, we will evaluate SCAE and SMONGE in more open source and commercial projects. In addition, we plan to leverage the multi-objective NSGAII algorithm to optimize more classifiers in software defect prediction.

Funding Statement: This work is supported in part by the National Science Foundation of China (Grant Nos. 61672392, 61373038), and in part by the National Key Research and Development Program of China (Grant No. 2016YFC1202204).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Abaei, G.; Rezaei, Z.; Selamat, A. (2013): Fault prediction by utilizing self-organizing map and threshold. *IEEE International Conference on Control System, Computing and Engineering*, pp. 465-470.

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; Kegelmeyer, W. P. (2002): SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321-357.

Chen, M.; Ma, Y. (2015): An empirical study on predicting defect numbers. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, pp. 397-402.

Gao, K.; Khoshgoftaar, T. M.; Wang, H. J.; Seliya, N. (2011): Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software: Practice and Experience*, vol. 41, no. 5, pp. 579-606.

Globerson, A.; Roweis, S. T. (2005): Metric learning by collapsing classes. *Annual Conference on Neural Information Processing Systems*, pp. 451-458.

Gu, X. D; Zhang, H. Y.; Zhang, D. M.; Kim, S. H. (2016): Deep API learning. *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 631-642.

Guo, J.; Cheng, J.; Cleland-Huang, J. (2017): Semantically enhanced software traceability using deep learning techniques. *Proceedings of the International Conference on Software Engineering*, pp. 3-14.

He, Z.; Shu, F.; Yang, Y.; Li, M.; Wang, Q. (2012): An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, vol. 19, no. 2, pp. 167-199.

Herbold, S.; Trautsch, A.; Grabowski, J. (2018): A comparative study to benchmark cross-project defect prediction approaches. *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811-833.

Hosseini, S.; Turhan, B.; Gunarathna, D. (2019): A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111-147.

Huang, G. B.; Chen, L.; Siew, C. K. (2006): Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879-892.

Huang, G. B.; Zhou, H. M.; Ding, X. J.; Zhang, R. (2012) Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems Man & Cybernetics*, vol. 42, no. 2, pp. 513-529.

Huang, G.; Zhu, Q.; Siew, C. (2006): Extreme learning machine: theory and applications, *Neurocomputing*, vol. 70, no. 1, pp. 489-501.

Huang, Q. H.; Wang, H. J.; Xu, Q.; Bi, W. Z. (2009): Semi-supervised learning with locally linear coordination for face recognition. *Proceedings of the International Conference on Natural Computation*, pp. 255-259.

Ji, R. R.; Liu, H.; Cao, L. J.; Liu, D.; Wu, Y. J. et al. (2017): toward optimal manifold hashing via discrete locally linear embedding. *IEEE Transactions on Image Processing*, vol. 26, no. 11, pp. 5411-5420.

Jiarpakdee, J.; Tantithamthavorn, C.; Ihara, A.; Matsumoto, K. (2016): A study of redundant metrics in defect prediction datasets. *Proceedings of the International Symposium on Software Reliability Engineering Workshops*, pp. 51-52.

Khoshgoftaar, T. M.; Gao, K.; Napolitano, A. (2012): An empirical study of feature ranking techniques for software quality prediction. *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 2, pp. 161-183.

Kondo, M.; Bezemer, C. P.; Kamei, Y.; Ahmed, E. H.; Osamu, M. (2019): The impact of feature reduction techniques on defect prediction models. *Empirical Software Engineering*, vol. 24, no. 4, pp. 1925-1963.

Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012): Imagenet classification with deep convolutional neural networks. *Annual Conference on Neural Information Processing Systems*, pp. 1106-1114.

Liu, M.; Miao, L.; Zhang, D. (2014): Two-stage cost-sensitive learning for software defect prediction. *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 676-686.

Lu, G. F.; Wang, Y.; Zou, J.; Wang, Z. Q. (2018): Matrix exponential based discriminant locality preserving projections for feature extraction. *Neural Networks*, vol. 97, no. 1, pp. 127-136.

Lu, H.; Kocaguneli, E.; Cukic, B. (2014): Defect prediction between software versions with active learning and dimensionality reduction. *IEEE 25th International Symposium on Software Reliability Engineering*, pp. 312-322.

Majdi, M. M.; Seyedali, M. (2017): Hybrid whale optimization algorithm with simulated annealing for feature selection. *Neurocomputing*, vol. 260, no. 10, pp. 302-312.

Marco, D. A.; Michele, L.; Romain, R. (2010): An extensive comparison of bug prediction approaches. *Proceedings of the 7th International Conference on Mining Software Repositories*, pp. 31-41.

Menzies, T.; Greenwald, J.; Frank, A. (2007): Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13.

Mohamed, A.; Dahl, G. E.; Hinton, G. E. (2012): Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14-22.

Nam, J.; Pan, S. J.; Kim, S. (2013): Transfer defect learning. *Proceedings of the 2013 International Conference on Software Engineering*, pp. 382-391.

Ni, C.; Chen, X.; Wu, F. F.; Shen, Y. X.; Gu, Q. (2019): An empirical study on pareto based multi-objective feature selection for software defect prediction. *Journal of Systems and Software*, vol. 152, no. 6, pp. 215-238.

Ning, L. V.; Chen, C.; Tie, Q.; Arun, K. S. (2018): Deep learning and superpixel feature extraction based on contractive autoencoder for change detection in SAR images. *IEEE Transactions on Industrial Informatics*, pp. 5530-5538.

Oh, B. D.; Song, H. J.; Kim, J. D.; Park, C. Y.; Kim, Y. S. (2019): Predicting concentration of PM10 using optimal parameters of deep neural network. *Intelligent Automation and Soft Computing*, vol. 25, no. 2, pp. 343-350.

Parviainen, E. (2016): A graph-based N-body approximation with application to stochastic neighbor embedding. *Neural Networks*, vol. 75, no. 1, pp. 1-11.

Peters, F.; Menzies, T.; Layman, L. (2015): Lace2: better privacy-preserving data sharing for cross project defect prediction. *Proceedings of the International Conference on Software Engineering*, pp. 801-811.

Rathore, S. S.; Gupta, A. (2014): A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction. *Proceedings of the 7th India Software Engineering Conference*, pp. 1-10.

Ren, J.; Qin, K.; Ma, Y.; Luo, G. (2014): On software defect prediction using machine learning. *Journal of Applied Mathematics*, vol. 2014, no. 1, pp. 1-8.

Rifai, S.; Vincent, P.; Muller, X.; Xavier, G.; Yoshua, B. (2011): Contractive auto-encoders: explicit invariance during feature extraction. *ICML*, pp. 833-840.

Saini, S.; Rambli, D. R.; Sulaiman, S. B.; Zakaria, M. N. (2013): Human pose tracking in low-dimensional subspace using manifold learning by charting. *IEEE International Conference on Signal and Image Processing Applications*, pp. 258-263.

Tantithamthavorn, C.; McIntosh, S.; Hassan, A. E.; Matsumoto, K. (2017): An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1-18.

Tantithamthavorn, C.; McIntosh, S.; Hassan, A. E.; Matsumoto, K. (2016): Automated parameter optimization of classification techniques for defect prediction

models. *Proceedings of the 38th International Conference on Software Engineering*, pp. 321-332.

Wang, J. J.; Cui, Q.; Wang, S.; Wang, Q. (2017): Domain adaptation for test report classification in crowdsourced testing. *Proceedings of the International Conference on Software Engineering*, pp. 83-92.

Wang, S.; Liu, T. Y.; Tan, L. (2016): Automatically learning semantic features for defect prediction. *IEEE/ACM 38th IEEE International Conference on Software Engineering*, pp. 297-308.

Witten, I. H.; Frank, E.; Hall, M. A. (2011): Data mining: practical machine learning tools and techniques, third edition. *ACM Sigmod Record*, vol. 31, no. 1, pp. 76-77.

Xu, B. W.; Ye, D. H.; Xing, Z. C.; Xia. X.; Chen, G. B. et al. (2016): Predicting semantically linkable knowledge in developer online forums via convolutional neural network. *Proceedings of the International Conference on Automated Software Engineering*, pp. 51-62.

Xu, Z.; Liu, J.; Luo, X.; Yang, Z. J.; Zhang Y. F. et al. (2018): Software defect prediction based on kernel PCA and weighted extreme learning machine. *Information and Software Technology*, vol. 106, no. 2, pp. 182-200.

Xu, Z.; Liu, J.; Yang, Z.; An, G.; Jia, X. (2016): The impact of feature selection on defect prediction performance: an empirical comparison. *Proceedings of the 27th International Symposium on Software Reliability Engineering*, pp. 309-320.

Yang, X. L.; David, L.; Zhang, Y.; Sun, J. L. (2015): Deep learning for just-in-time defect prediction. *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security*, pp. 17-26.

Yasutaka, K.; Takafumi, F.; Shane, M.; Kazuhiro, Y.; Naoyasu, U. et al. (2016): Studying just-in-time defect prediction using cross-project models. *Empirical Software Engineering*, vol. 21, no. 5, pp. 2072-2106.

Zhang, F.; Zheng, Q.; Zou, Y.; Hassan, A. E. (2016): Cross-project defect prediction using a connectivity-based unsupervised classifier. *Proceedings of the 38th International Conference on Software Engineering*, pp. 309-320.

Zhang, J. M.; Wang, W.; Lu, C. Q.; Wang, J.; Sangaiah, A. K. (2019): Lightweight deep network for traffic sign classification. *Annals of Telecommunications* (to be published).

Zhu, K.; Zhang, N.; Ying, S.; Wang, X. (2020): Within-project and cross-project software defect prediction based on improved transfer naive Bayes algorithm. *Computers, Materials & Continua*, vol. 63, no. 2, pp. 891-910.