The Journal of Systems & Software 180 (2021) 111026

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network☆

Kun Zhu [a], Shi Ying [a,*], Nana Zhang [a], Dandan Zhu [b]

[a] School of Computer Science, Wuhan University, China
[b] Artificial Intelligence Institute, Shanghai Jiaotong university, China

ARTICLE INFO

Article history:
Received 15 May 2020
Received in revised form 27 April 2021
Accepted 31 May 2021
Available online 16 June 2021

Keywords:
Software defect prediction
Metaheuristic feature selection
Whale optimization algorithm
Convolutional neural network
Kernel extreme learning machine

ABSTRACT

Software defect prediction aims to identify the potential defects of new software modules in advance by constructing an effective prediction model. However, the model performance is susceptible to irrelevant and redundant features. In addition, previous studies mainly use traditional data mining or machine learning techniques for defect prediction, the prediction performance is not superior enough. For the first issue, motivated by the idea of search based software engineering, we leverage the recently proposed whale optimization algorithm (WOA) and another complementary simulated annealing (SA) to construct an enhanced metaheuristic search based feature selection algorithm named EMWS, which can effectively select fewer but closely related representative features. For the second issue, we employ a hybrid deep neural network — convolutional neural network (CNN) and kernel extreme learning machine (KELM) to construct a unified defect prediction predictor called WSHCKE, which can further integrate the selected features into the abstract deep semantic features by CNN and boost the prediction performance by taking full advantage of the strong classification capacity of KELM. We conduct extensive experiments for feature selection or extraction and defect prediction across 20 widely-studied software projects on four evaluation indicators. Experimental results demonstrate the superiority of EMWS and WSHCKE.

1. Introduction

Defective software can lead to unexpected results after deployment, and even in the worst case, can cause great economic loss to the company or organization (Hosseini et al., 2019). Currently, software defect prediction is a very important software quality assurance technique, which can extract historical defect data from software repositories and leverage data mining or machine learning algorithm to construct the effective prediction model, thus detecting the defect proneness of new software modules (Ma et al., 2012). An effective defect prediction model can help developers or maintainers efficiently detect the potentially defective modules by reasonably allocating limited software development and maintenance resources (Ma 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 (Xu et al., 2019). But not all the features are helpful to the prediction performance of the model, which may contain some irrelevant and redundant features. Jiarpakdee et al. (2016) demonstrated 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. Prior study (Kondo et al., 2019) has proven that feature selection or extraction can avoid the problem of multicollinearity (FARRAR and Glauber, 1967) and the curse of dimensionality (Bellman, 1957). Therefore, it is very necessary to conduct feature selection or extraction for defect datasets in software defect prediction. Recently, some feature selection or extraction methods (Ni et al., 2017; Kondo et al., 2019) have been proposed to remove or combine irrelevant and redundant features for software defect datasets.

The purpose of feature selection is to select an optimal representative feature subset instead of the original feature set by evaluating the contribution of module features, while feature extraction techniques decrease the number of features by constructing new, combined features from the original features (Seiffert et al., 2014). Previous studies (Ghotra et al., 2017; Kondo et al., 2019) mainly utilized a single metaheuristic or machine learning algorithm to conduct feature selection or extraction in software defect prediction, which showed that correlation-based feature selection methods are one of the best performing methods, such as BestFirst search (a heuristic search algorithm using

☆ Editor: Alexander Serebrenik.
* Corresponding author.
E-mail address: yingshi@whu.edu.cn (S. Ying).

https://doi.org/10.1016/j.jss.2021.111026
0164-1212/

greedy hillclimbing with backtracking) (Ghotra et al., 2017), genetic search (Goldberg, 1989; Ghotra et al., 2017), and they are just metaheuristic intelligent algorithms. In recent years, metaheuristic algorithms have been widely used in many optimization problems, and feature selection is also one of the key research domains where metaheuristic algorithms have been investigated. However, due to the stochastic nature of metaheuristic algorithms, there is no guarantee to search the best combination of features in feature selection problem. Also, the No-Free-Lunch (NFL) theorem (Turabieh et al., 2019) for optimization problems declares that there is no optimizer that is effective enough to solve all optimization problems, e.g., the feature selection problem that needs to deal with numerous software projects in software defect prediction. Moreover, the exploitation of the optimal solution (intensification) and the exploration of the search space (diversification) are two contradictory goals that must be taken into account simultaneously when employing a metaheuristic algorithm, especially for the feature selection problem that is relatively random in finding the optimal solution (Talbi, 2009; Jensen and Shen, 2004). A good balance between these two contradictory goals can boost the searching performance of the metaheuristic algorithm (Jensen and Shen, 2004), especially for the feature selection problem which requires high search performance. One of the best performing methods, BestFirst search, has strong exploitation capability (local search) but poor exploration capability, so the capability to search for optimal solution can be further enhanced. These reasons motivate our attempts to investigate more superior feature selection methods.

According to the two contradictory goals mentioned above, metaheuristic algorithms can be divided into two categories: exploitation oriented algorithms based on single-solution (e.g., BestFirst search (Ghotra et al., 2017), simulated annealing (SA) (Jensen and Shen, 2004) and exploration oriented algorithms based on population (e.g.,whale optimization algorithm (WOA) Mirjalili and Lewis, 2016, ant colony optimization Kashef and Nezamabadi-pour, 2015), in which BestFirst search and SA are the hill climbing mechanism based algorithms, and SA also adopts an annealing mechanism, so SA can be regarded as an upgraded version of the BestFirst search. One method to achieve the balance of two contradictory goals is to use a hybrid algorithm in which at least two metaheuristic algorithms are combined to enhance the performance of each algorithm, nevertheless, this hybrid metaheuristic feature selection method has not been investigated thoroughly in software defect prediction. In this paper, we leverage a recently proposed exploration (global search) oriented algorithm – **W**hale optimization algorithm (WOA) (Mirjalili and Lewis, 2016) and another complementary exploitation (local search) oriented algorithm – **S**imulated annealing (SA) (Jensen and Shen, 2004) to construct an **E**nhanced **M**etaheuristic search based feature selection algorithm named EMWS in the context of software defect prediction, which adopts roulette wheel selection as the selection mechanism in the exploration stage and Logistic Regression algorithm as the classifier of feature selection. (Mafarja and Mirjalili, 2017) utilized this combination mechanism of WOA and SA to build two hybrid models (Low-Level Teamwork Hybrid (LTH) and High-Level Relay Hybrid (HRH)) in intelligent algorithm optimization. Different from their studies, we first apply this combination mechanism to feature selection in software defect prediction, and adopt different selection mechanism in the exploration stage and different classifier of feature selection. The enhanced EMWS algorithm can take full advantage of the strong local search capability of SA to enhance the weak exploitation performance of WOA, and leverage strong global search capability of WOA to boost the weak exploration of SA a large extent simultaneously. Where the WOA algorithm is a recently proposed exploration oriented optimization algorithm that shows many advantages in some optimization problems, such as strong global search capability, simple structure, few adjustment parameters and high flexibility (Mirjalili and Lewis, 2016), which can complement SA perfectly. We first utilize the WOA algorithm to find the global optimal solution, and then employ the SA algorithm to further search the optimal solution on the basis of the existing optimal solution, and iterative loop in turn. This is because the solution obtained by WOA may not be good enough under limited iterations or the search may trap in the local optimum, but SA allows accepting a worse solution with a certain probability, and can escape from local optimum by allowing hill-climbing moves towards worse objective function values, so SA can further enhance the solution or overcome the problem of stagnating for local optima of WOA and help this solution jump out of local optimum based on the optimal solution obtained by WOA. The unique properties of WOA and SA can be integrated to a hybrid metaheuristic algorithm to achieve better search performance than using each one separately in the feature selection problem.

Previous studies (Razavian et al., 2014; van de Wolfshaar et al., 2015) showed that the performance of a prediction model largely depends on two key factors: feature representation and classifier. Motivated by these studies, we expect to find a strong distinguishing feature representation for each software project and a powerful classifier with superior performance. For an excellent defect feature representation, it not only requires the features having the strong discriminant inter-class separability, but also reserves unaltered intra-class compactness (Guo et al., 2017b). In recent years, due to the capability to extract semantic features with such strong discriminative capability compared to ordinary machine learning methods, convolutional neural network (CNN) have been highlighted in many fields, such as semantic search (Shen et al., 2014), speech recognition (Anvarjon et al., 2020). Prior researches (Guo et al., 2017b; Duan et al., 2018; Ren et al., 2017) have demonstrated that the deep semantic features have stronger discriminating capacity for different classes (defective or non-defective). In a traditional CNN (N network layers) framework, the previous (N-1) layers are used to extract the high-level deep semantic features, and the final softmax layer is used as a classifier to discriminate these features. However, the softmax does not require intra-class compactness and inter-class separation, and cannot take full advantage of these discriminative deep semantic features extracted by previous convolutional layers (Duan et al., 2018). Also, the softmax is more suitable for multi-classification tasks (Ren et al., 2017), while the software defect prediction in this paper is only for binary classification tasks. In addition, (Duan et al., 2018) verified that the classification capability of the softmax in CNN is inferior to other classifiers, such as ELM (extreme learning machine). The ELM proposed (Huang et al., 2006) has been verified to be an efficient and fast classification algorithm due to its good generalization performance, fast training speed without iteration operation, ease of implementation, fewer super-parameters, and little human intervention (Huang et al., 2005). Unlike the traditional gradient-based learning algorithms, the ELM was initially proposed for generalized single-hidden-layer feed-forward neural (SLFN) networks, and overcome the learning rate, local minima, learning epochs and stopping criteria (Huang et al., 2005, 2006). What is more, ELM can be widely adopted to deal with binary classification tasks (Yang et al., 2015b), and take full advantage of the extracted discriminative deep semantic features (Kim et al., 2017). Compared to ELM, KELM (kernel extreme learning machine) introduces a nonlinear Gaussian kernel, and it is an upgraded version of ELM because this Gaussian kernel can better classify features with nonlinear relationships (Huang et al., 2015). From these investigation, we can easily judge that the KELM

classifier should be effective when applied to CNN structure. Motivated by above facts, to integrate the advantages of CNN and KELM, we replace the softmax classifier by the KELM classifier in the last layer of CNN. As a consequence, we build a unified defect predictor called WSHCKE (**H**ybrid **C**onvolutional Neural Network and **K**ernel **E**xtreme Learning Machine according to the features selected by EM**WS**) by constructing the CNN as a data-driven deep semantic feature extractor and the cascaded KELM as a powerful classifier instead of the conventional used softmax classifier in the CNN framework.

The main contributions of this paper are as follows:

(1) We leverage the recently proposed whale optimization algorithm (WOA) and another complementary simulated annealing (SA) to construct an enhanced metaheuristic feature selection algorithm named EMWS that selects an optimal representative feature subset, which can take full advantage of the strong local search capability of SA to enhance the exploitation performance of WOA, and leverage strong global search capability of WOA to boost the exploration of SA simultaneously. To the best of our knowledge, this is the first attempt to apply the hybrid exploration oriented algorithm based on population (WOA) and exploitation oriented algorithm based on single-solution (SA) to feature selection in software defect prediction.

(2) We construct a unified defect predictor called WSHCKE that adopts a hybrid deep neural network — CNN and KELM, which can further integrate the defect features into the data-driven abstract deep semantic features by CNN and boost the prediction performance by utilizing the strong discriminative capacity of KELM.

(3) We conduct extensive experiments for feature selection and extraction and defect prediction across 20 software projects from three large open source datasets. We compare the EMWS algorithm with eleven state-of-the-art feature selection or extraction approaches, and compare the WSHCKE predictor with nine classic defect predictors. The experimental results demonstrate that the EMWS algorithm and the WSHCKE predictor can achieve superior prediction performance on four evaluation indicators.

The reminder of this paper is organized as follows. We describe background on whale optimization algorithm and simulated annealing in Section 2. Section 3 introduces data preprocessing, including class imbalance processing and data standardization. Section 4 details feature selection based on the enhanced EMWS algorithm. Section 5 details the proposed WSHCKE model. Section 6 shows the experimental setup, including benchmark datasets, evaluation indicators, experimental design, parameter settings, Scott–Knott Effect Size Difference (ESD) test and Wilcoxon signed-rank test and Cliff's Delta effect size analysis. Section 7 evaluates and discusses the performance of the EMWS algorithm and the WSHCKE predictor. Section 8 introduces the threats to validity. Section 9 describes the related work. We conclude the paper and describe the future work in Section 10.

2. Background

We leverage an enhanced metaheuristic algorithm named EMWS that contains whale optimization algorithm (WOA) and simulated annealing (SA) to select an optimal representative defect feature subset. In this section, we introduce WOA and SA algorithms respectively.

2.1. Whale optimization algorithm

The whale optimization algorithm is a recently proposed metaheuristic algorithm based on stochastic population, which mimics the intelligent foraging behavior — the bubble-net feeding for the humpback whales, Mirjalili and Lewis (2016). The WOA has good properties, such as strong global search capability, simple structure, few adjustment parameters and high flexibility.

The WOA consists of two stages: exploitation stage (shrinking encircle prey, spiral bubble-net feeding mechanism), and exploration stage (search for prey).

In the exploitation stage, the humpback whales constantly adjust their position according to the position of the prey when hunting. The WOA assumes that the current optimal candidate solution is the position of target prey or the position closest to the target prey (An optimal candidate solution refers to a candidate feature subset in the context of software defect prediction, and the number of selected features is not necessarily the minimum.). After defining the optimal position, other whales will move towards the optimal position. In the context of software defect prediction, the movement of whales towards the prey is analogous to the process of searching for the optimal feature subset by calculating fitness value (calculate the fitness value of the search agent — whale by the fitness function). The mathematical expressions for this process are as follows:

$$\overrightarrow{D_1} = |\overrightarrow{C}.\overrightarrow{X^*}(t) - \overrightarrow{X}(t)|, \tag{1}$$

$$\overrightarrow{X}(t+1) = \overrightarrow{X^*}(t) - \overrightarrow{A}.\overrightarrow{D_1}, \tag{2}$$

where t represents the current iteration number, $\overrightarrow{X}(t)$ denotes the current solution, $\overrightarrow{X}(t+1)$ represents the target coordinate vector after the next iteration, $\overrightarrow{X^*}(t)$ denotes the optimal solution obtained so far. $\overrightarrow{A}$ and $\overrightarrow{C}$ are coefficient vectors, which are calculated as in Eqs. (3) and (4), respectively:

$$\overrightarrow{A} = 2\vec{a}.\vec{r} - \vec{a}, \tag{3}$$

$$\overrightarrow{C} = 2\vec{r}, \tag{4}$$

where $\overrightarrow{a}$ decreases linearly from 2 to 0, which is updated with $\overrightarrow{a} = \frac{2t}{M}$, M is the maximum number of iterations, $\overrightarrow{r}$ is a random vector in [0, 1].

The humpback whales swim with a shrinking encircling mechanism and move towards the prey with a spiral-shaped path. The shrinking encircling mechanism is simulated by decreasing the value of $\overrightarrow{a}$ in Eq. (3). The adjustment of $\overrightarrow{A}$ and $\overrightarrow{C}$ controls the area where a solution can be located in the neighborhood of the optimal solution. Then the spiral-shaped path is calculated according to the distance between the current solution and the optimal solution in the spiral bubble-net feeding mechanism, as shown in Eqs. (5) and (6):

$$\overrightarrow{X}(t+1) = \overrightarrow{D_2}.e^{bl}.cos(2\pi l) + \overrightarrow{X^*}(t), \tag{5}$$

$$\overrightarrow{D_2} = |\overrightarrow{X^*}(t) - \overrightarrow{X}(t)|, \tag{6}$$

where $\overrightarrow{D_2}$ represents the distance between the current position of the humpback whale and the current optimal individual (the optimal solution obtained so far), b denotes a constant which defines the logarithmic spiral shape, l represents a random number in $[-1, 1]$.

To model the shrinking encircling mechanism and the upward spiral-shaped path simultaneously, we assume that there is a probability of 50% to choose between them to update the position of whales during the optimization, as shown in Eq. (7):

$$\begin{cases} \overrightarrow{X}(t+1) = \overrightarrow{X^*}(t) - \overrightarrow{A}.\overrightarrow{D_1} & \text{when p<0.5} \\ \overrightarrow{X}(t+1) = \overrightarrow{D_2}.e^{bl}.cos(2\pi l) + \overrightarrow{X^*}(t) & \text{when } p \geq 0.5, \end{cases} \tag{7}$$

where p is a random number in [0, 1]. The p ($p \in [0, 1]$) value is randomly chosen, that is, there is a probability of 50% to choose the shrinking encircling mechanism ($p < 0.5$) (i.e., Eq. (2)) or the upward spiral-shaped path ($p \geq 0.5$) (i.e., Eq. (5)).

Fig. 1 shows two feeding search mechanisms for WOA in the exploitation stage. X^* denotes the best solution achieved so far. Fig. 1(a) visualizes the possible positions (circle) from the position (X, Y) of whale towards the position (X^*, Y^*) of prey by setting $A \in [0, 1)$ in the shrinking encircling mechanism. Fig. 1(b) visualizes the spiral updating position mechanism between the position of whale (X, Y) and the position of prey (X^*, Y^*) by setting $l \in [-1, 1]$.

In the exploration stage, the humpback whales also randomly search for prey to update the position based on the change of the coefficient for $\overrightarrow{A}$. If $\overrightarrow{A}$ exceeds the range of $[-1, 1]$, the distance $\overrightarrow{D}$ will be updated according to roulette wheel selection, rather than randomly selection, and the whales will deviate from the original target prey in order to find new prey, which gives the WOA a certain global search capability. This mechanism can be mathematically modeled as follows:

$$\overrightarrow{D} = |\overrightarrow{C} . \overrightarrow{X}_{rw} - \overrightarrow{X}|, \tag{8}$$

$$\overrightarrow{X}(t+1) = \overrightarrow{X}_{rw} - \overrightarrow{A} . \overrightarrow{D}, \tag{9}$$

where $\overrightarrow{X}_{rw}$ denotes the location of a whale individual in the current population according to roulette wheel selection.

2.2. Simulated annealing

Simulated annealing (Jensen and Shen, 2004) is a stochastic computational algorithm that seeks global extrema (i.e., global optimal solution) for various intelligent optimization problems, and this algorithm mimics the process of annealing in metallurgy which involves initial heating and controls cooling of the material. Simulated annealing can obtain a global optimal solution (In the context of software defect prediction, a global optimal solution refers to a feature subset with the minimum number of selected features corresponds to a software project), this is because it allows accepting a worse solution with a certain probability, and can escape from local optimum by allowing hill-climbing moves towards worse objective function values in hope that a global optimum solution is found. An key characteristic of simulated annealing is that it has strong local search capability, so we can combine SA and WOA to overcome the problem of stagnating for local optima of WOA.

Simulated annealing exists the current optimal solution and a newly generated solution at each iteration. By comparing these two solutions, the improved solutions are always accepted if $\delta \leq 0$, $\delta = L(X'') - L(X')$, while the worse solutions are accepted with a certain probability determined by the Boltzmann probability P in order to escape local optima and achieve the global optima if $P = e^{-\delta/T}$, where δ is the difference between the fitness of the optimal solution and the generated neighbor, T is an important parameter called the "temperature" for the convergence of the solution, which typically decreases as the number of iterations increases according to a certain cooling schedule. For the cooling schedule, a typical option is to use the proportional method, i.e., $T_k = \alpha . T_{k-1}$. The α denotes a cooling proportional coefficient, $0 < \alpha < 1$, which shows that the temperature typically decreases as the number of iterations increases.

3. Data preprocessing

In this paper, data preprocessing includes two aspects: class imbalance processing and data standardization.

3.1. Class imbalance processing

Class imbalance is a common problem in software defect datasets. The distribution of software defect in a project is roughly in line with the Pareto principle, namely 20% of the program modules contain about 80% of the defects, and the number of defective modules (a few class) is far less than that of the non-defective modules (majority class). If there is the serious class imbalance problem in a software project, the prediction performance of the model is relatively poor.

In this paper, we adopt the SMOTUNED (Synthetic Minority Oversampling TUNED) algorithm (Agrawal and Menzies, 2018; Tantithamthavorn et al., 2020) for class imbalance processing, which is an upgraded auto-tuning version of simple SMOTE (Chawla et al., 2002). The SMOTUNED employs DE (differential evolution (Storn and Price, 1997)) to explore the parameter search space for three important parameters, including number of neighbors $k \in [1, 20]$, number of synthetic instances $M \in \{50, 100, 200, 400\}$ and power parameter $r \in [0.1, 5]$ for the Minkowski distance metric, thereby generating the optimal control parameter combination for different software projects. (Agrawal and Menzies, 2018; Tantithamthavorn et al., 2020) have proved that the SMOTUNED can out-perform the previous state-of-the-art class imbalance techniques.

This step is critical for software defect prediction because it can help the prediction model not bias towards non-defective modules (majority class), thus improving the performance of defect prediction.

3.2. Data standardization

The distribution of the defect feature values is of large difference, even not in the same order of magnitude. If the original feature values are used for defect prediction, the function of the higher values in the comprehensive analysis will be highlighted, and the function of the lower values will be relatively weakened. To ensure the reliability of the prediction results, we need to standardize the original feature values, and the processed feature values should be consistent with the original distribution.

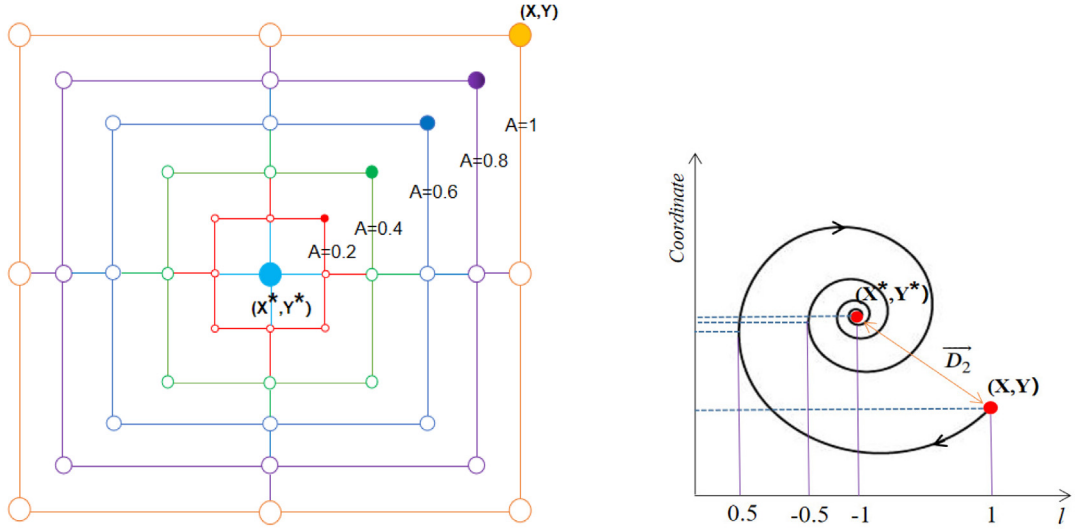
In this paper, we use the z-score standardization method (Yang et al., 2017), for each value f_i of the feature f, the normalized value z_i is calculated as follows:

$$z_i = \frac{f_i - mean(f)}{std(f)}, \tag{10}$$

where $mean(f)$ and $std(f)$ are the mean and variance of the initial values in feature f, respectively.

4. Feature selection based on the enhanced EMWS algorithm

In this paper, we combine the powerful unique properties of WOA and SA to integrate an enhanced metaheuristic feature selection optimization algorithm, which can take full advantage of the strong local search capability of SA to enhance the weak exploitation performance of WOA, and leverage strong global search capability of WOA to boost the weak exploration of SA a large extent simultaneously, thereby achieving better search performance than using each one separately in the feature selection problem of software defect prediction. We first utilize the WOA algorithm to find the global optimal solution, and then employ the SA algorithm to further search the optimal solution on the basis of the existing optimal solution, and iterative loop in turn. This is because the solution obtained by WOA may not be good enough under limited iterations or the search may trap in the local optimum, but SA allows accepting a worse solution with a certain probability, and can escape from local optimum by



(a) Shrinking encircle prey (b) Spiral bubble-net feeding mechanism

Fig. 1. Two feeding search mechanisms for WOA in the exploitation stage.

allowing hill-climbing moves towards worse objective function values, so SA can further enhance the solution or overcome the problem of stagnating for local optima of WOA and help this solution jump out of local optimum based on the optimal solution obtained by WOA.

In the context of software defect prediction, an optimal candidate solution refers to a candidate feature subset (The number of selected features is not necessarily the minimum.), and a optimal solution refers to a feature subset with the minimum number of selected features that corresponds to a software project. The solution refers to a feature subset corresponds to a software project, which is a one-dimensional binary vector that contains N elements, where N is the number of features in a software project. Each element in the vector is represented by "1" or "0". The "1" denotes that this corresponding feature can be selected, otherwise not.

The EMWS algorithm considers two optimization objectives into the feature selection problem in software defect prediction. One optimization objective is to minimize the number of selected features and another objective is to minimize classification error. The smaller the number of selected features and the smaller the classification error, the better the solution. Previous studies (Mafarja and Mirjalili, 2017; Hamdani et al., 2007; Xue et al., 2013) also adopted the same two optimization goals for feature selection. In order to balance the number of selected features (minimum) and the classification error (minimum) in each solution, we empirically set two priori coefficients for these two objectives by the decision maker, and integrate the two objectives into an objective function to obtain the optimal solution by weight sum. Since the EMWS is wrapper-based feature selection algorithm, we adopt Logistic Regression (LR) as the classifier of the EMWS algorithm. In this paper, we choose the following fitness function to evaluate the solution in the EMWS algorithm, which can be defined as follows:

$$fitness = \lambda \mu_{|C|}(\mathrm{LR}) + \omega \frac{|C|}{|N|}, \tag{11}$$

where $\mu_{|C|}(\mathrm{LR})$ denotes the classification error of the Logistic Regression classifier, $|N|$ denotes the total number of features in each software project, $|C|$ presents the cardinality of the selected feature subset, λ and ω are two parameters corresponding to the importance of classification quality (i.e., minimize classification error) and feature subset length, respectively, $\lambda \in [0, 1]$ and $\omega = 1 - \lambda$ (Here we use the values of λ and ω derived from Mafarja and Mirjalili, 2017; Emary et al., 2016.).

Note that we only use the EMWS algorithm for feature selection on each software project, not the final prediction process (the latter WSHCKE predictor is specifically designed for classification). Since the exploration in the WOA algorithm (as shown in Eqs. (8) and (9)) depends on the position change of each search individual according to a randomly selected solution, meanwhile, in order to preserve the diversity capability of the EMWS algorithm, we apply the roulette wheel selection mechanism to select search individuals (solutions) from the population in the exploration stage, and this mechanism can give more chance to the weak solutions.

The pseudo-code of the enhanced metaheuristic EMWS algorithm is as shown in Algorithm 1.

The fitness function of the enhanced EMWS algorithm can be calculated by Eq. (11). Both WOA and SA algorithms search and evaluate solutions based on this fitness function respectively. In steps 1–2, we first randomly initialize the population $X \leftarrow X_i$, $i \in [1, N]$ and an initial temperature T_0, and calculate the fitness value of each solution. In step 3, we can select the best solution X^* based on the calculated fitness value. The entire EMWS algorithm begins to loop until reaching the maximum number of iterations M in step 4. For each search agent $i \in [1, N_a]$, we first update a series of important parameters, including the variable vector $\overrightarrow{a}$, two coefficient vectors $\overrightarrow{A}$ and $\overrightarrow{C}$, and two random numbers l and p according to Eqs. (3), (4), (5), (7) in steps 5–6. In steps 7–16, to model the shrinking encircling mechanism ($p < 0.5$) and the upward spiral-shaped path ($p \geq 0.5$) simultaneously, we assume that there is a probability of 50% to choose between them to update the position of whales during the optimization, where the p ($p \in [0, 1]$) value is randomly chosen. For the shrinking encircling mechanism ($p < 0.5$), we update the position of the current solution in the exploitation phase according to the best solution found so far when $|\overrightarrow{A}| < 1$ (Eq. (2)), and update the position of the current solution and search for prey in the exploration phase according to the roulette wheel selection mechanism when $|\overrightarrow{A}| \geq 1$ (Eq. (9)). The upward spiral-shaped path is calculated according to the distance between the current solution and the optimal solution in the spiral bubble-net feeding mechanism when $p \geq 0.5$ (Eq. (5)). In steps 18–20, we check if any solution goes beyond the search space and amend it, and

Algorithm 1 Pseudo-code of the enhanced metaheuristic EMWS algorithm

Input:
number of search agents: N_a, maximum number of iterations: M, initial temperature T_0, the temperature cooling schedule: α, maximum number of iterations for SA: M_{sa}, maximum number of iterations at each temperature T: $F(T)$.

Output:
feature subsets X_f and their training and testing classification error rates.

1: Initialize the whales population: $X \leftarrow X_i$, $i \in [1, N]$, the temperature: $T \leftarrow T_0$;
2: Calculate the fitness function value of each solution;
3: X^* = the best solution;
4: **while** $t \leq M$ **do**
5: **for** i←1 to N_a **do**
6: Update $\overrightarrow{a}$, $\overrightarrow{A}$, $\overrightarrow{C}$, l and p;
7: **if** $p < 0.5$ **then**
8: **if** $|\overrightarrow{A}| < 1$ **then**
9: Use Eq. (2) to update the position of the current solution with a shrinking encircling mechanism;
10: **else if** $|\overrightarrow{A}| \geq 1$ **then**
11: Select a solution X_{rw} according to roulette wheel selection;
12: Use Eq. (9) to update the position of the current solution and search for prey;
13: **end if**
14: **else if** $p \geq 0.5$ **then**
15: Use Eq. (5) to update the position of the current solution with the upward spiral-shaped path;
16: **end if**
17: **end for**
18: Check if any solution goes beyond the search space and amend it;
19: Calculate the fitness value of each solution;
20: Update X^*, if there is a better solution;
21: $X' \leftarrow X^*$;
22: **while** $t_{sa} \leq M_{sa}$ **do**
23: **for** j←1 to F(T) **do**
24: $X'' \leftarrow Next(X')$;
25: $\delta = L(X'') - L(X')$;
26: **if** $\delta < 0$ **then**
27: $X' \leftarrow X''$;
28: **else if** $Random(0, 1) < e^{-\delta/T}$ **then**
29: $X' \leftarrow X''$;
30: **end if**
31: **end for**
32: $T_k = \alpha . T_{k-1}$;
33: **end while**
34: **end while**
35: Calculate the classification error rate of the solutions (feature subsets) X' on the test set;
36: $X_f \leftarrow X'$;
37: **return** feature subsets X_f and their training and testing classification error rates

calculate the fitness value of each solution. If there is a better solution, we will update this solution as the current best solution X^*.

After each iteration of WOA, the output X^* of WOA is assigned to the input (the initial solution) X' of SA in step 21, and SA also follows up with an iteration. In step 22, when $t_{sa} \leq M_{sa}$, the SA goes to work. The repetition schedule function $F(T)$ denotes

Algorithm 2 Simple pseudo-code for feature selection using EMWS in software defect prediction

Input:
Software projects after z-score standardization: D; maximum number of iterations: M.

Output:
Feature subset X_f for each software project.

1: Initialize the population constructed by a one-dimensional binary vector for each software project D;
2: Calculate the fitness function value of EMWS according to Eq. (11);
3: Obtain the current best feature subset X^*;
4: **while** $t \leq M$ **do**
5: Conduct feature selection by WOA according to Eqs. (2), (5), (9);
6: Calculate the fitness value of each feature subset;
7: Update X^*, if there is a better feature subset;
8: The output X^* of WOA is assigned to the input X' of SA;
9: Conduct feature selection by SA;
10: **end while**
11: Calculate the classification error rate of feature subsets X' on the test set for each software project;
12: $X_f \leftarrow X'$;
13: **return** Feature subset X_f for each software project

maximum number of iterations conducted at each temperature T in step 23. In steps 24–25, the neighbor function $Next(X')$ can generate new solutions X'' at the neighborhood of X', and the difference δ between the fitness value of the generated neighbor $L(X'')$ and the optimal solution $L(X')$ can produce a move gain. In steps 26–30, through comparing these two solutions, the improved solutions are always accepted if $\delta < 0$, while the worse solutions are accepted with a certain probability determined by the Boltzmann probability P in order to escape local optima and achieve the global optima if $Random(0, 1) < P = e^{-\delta/T}$, where $Random(0, 1)$ denotes a random number in (0,1), T is an important parameter called the "temperature" for the convergence of the solution, which typically decreases as the number of iterations increases according to a certain cooling schedule. In step 32, a typical option for the cooling schedule is to use the proportional method, i.e., $T_k = \alpha . T_{k-1}$; $(0 < \alpha < 1)$, and the α denotes a cooling proportional coefficient. After M iterations, we calculate the classification error rate of the solutions (feature subsets) X' on the test set, and each solution X' is the final selected feature subset X_f for each software project.

Fig. 2 shows an example of the EMWS algorithm used for feature selection. Assuming there are 10 features in the original feature set, we use a one-dimensional binary vector that contains 10 elements to represent a feature subset (i.e., a solution). The "1" denotes that this corresponding feature can be selected, and "0" is the opposite. We first employ the WOA to select the feature subset by a series of search processes such as shrinking encircle prey, spiral updating position corresponding to Fig. 1(a), (b) and search for prey according to the roulette wheel selection mechanism in step 1, and utilize SA to further select the corresponding optimal feature subset by two cooling mechanisms (i.e., the improved solutions are always accepted or accepting a worse solution with a certain probability) based on the existing feature subset obtained by WOA in step 2. Through a series of iterations for steps 1,2, we can achieve an optimal feature subset (an optimal solution, i.e., two elements in the purple cell) for each software project.

In Algorithm 2, we give a simple pseudo code for feature selection using EMWS in defect prediction corresponds to Algorithm

Fig. 2. An example of the EMWS algorithm used for feature selection.

1. We first utilize the WOA algorithm to find the global optimal solution, and then employ the SA algorithm to further search the optimal solution on the basis of the existing optimal solution, and iterative loop in turn. This is because the solution obtained by WOA may not be good enough under limited iterations or the search may trap in a local optimum, and SA can further enhance the solution or help this solution jump out of local optimum based on the optimal solution. In addition, we take full advantage of the strong local search capability of SA to enhance the weak exploitation performance of WOA, and leverage strong global search of WOA to boost the weak exploration of SA a large extent simultaneously. The unique properties of WOA and SA can be integrated to a hybrid metaheuristic algorithm to achieve better performance than using each one separately.

5. The proposed WSHCKE model

In this section, we construct a unified defect predictor called WSHCKE that adopts a hybrid deep neural network — CNN and KELM, which can further integrate the defect features into the data-driven abstract deep semantic features by CNN and boost the prediction performance by utilizing the strong discriminative capacity of KELM. Therefore, this predictor contains two functional modules: (1) Deep semantic feature representation based on CNN. The module includes the following network structure: input layer, convolutional layer, Relu nonlinear activation function, flatten layer, fully connected (FC) layer and softmax layer (The softmax layer only participates in the end-to-end training.). (2) Defect prediction based on KELM. Compared to the softmax, the KELM has been verified to be an efficient and fast classification algorithm due to its good generalization performance, fast training speed without iteration operation, ease of implementation, fewer super-parameters, and little human intervention. What is more, KELM can be widely adopted to deal with binary classification tasks, and take full advantage of the extracted discriminative deep semantic features. Motivated by above facts, to integrate the advantages of CNN and KELM, we replace the softmax classifier by the KELM classifier in the last layer of CNN.

Our WSHCKE predictor consists of two training stages and a testing stage. In training stage 1, we train an end to end CNN network, which includes two components: a CNN feature extractor represented by the network weights λ_{CNN} and a softmax classifier. The CNN with the network weights λ_{CNN} can be utilized to extract deep semantic feature representation in the training set. In training stage 2, the CNN feature extractor trained in training stage 1 is cascaded with the KELM classifier represented by the network weights ϑ_{KELM}, the KELM classifier behind FC is used for the classification task based on the deep semantic features extracted by CNN (i.e., the output of FC is adopted as the input of KELM). In testing stage, the WSHCKE predictor (CNN with KELM) with the trained weights λ_{CNN} and ϑ_{KELM} is used to predict whether the defect instances in the test set are defective.

The network architecture of the WSHCKE model is shown in Fig. 3.

5.1. Deep semantic feature representation based on CNN

In training stage 1, the CNN has the following network structure: an input layer, three convolutional layers, three Relu nonlinear activation functions, a flatten layer, a fully connected layer and a softmax layer. Since the feature dimension of each project is different after feature selection, the network parameters of the WSHCKE predictor for each project are also different. We take the 10-dimensional features selected by the EMWS algorithm as an example, as shown in Fig. 3. The 10-dimensional features refer to the number of feature dimensions for each project after feature selection, and dimensions and the number of features are same. We first process the 10-dimensional features into 16-dimensional features by adding six columns of zero at the end, and then reshape the 16-dimensional features into a 4×4 matrix, thereby facilitating the subsequent convolution operations. The original 10-dimensional features are not convenient for subsequent convolution processing since the width and height of the reshaped matrix are inconsistent. Next, we perform three convolution operations. The number of neurons in each convolutional layer is 32, 64, and 16, respectively, and convolution kernel sizes are 3×3, 3×3, and 4×4, respectively. We add a Relu nonlinear activation function in each convolutional layer. After completing these operations, we convert the 2-dimensional feature maps into 1-dimensional vectors ($X_i^{ln} \in Z^{w' \times 1}$, $i = 1, \ldots, N$) by the flatten and FC layers. The softmax layer only participates in the end-to-end training in training stage 1. Prior researches (Hoang et al., 2019; Wang et al., 2016a; Guo et al., 2017a) have demonstrated that the deep semantic features have stronger discriminating capacity for different classes (defective or non-defective). The CNN can extract such deep semantic features hidden behind defect data through a series of convolution operations. We employ three convolution operations to continuously enhance the nonlinear fitting capability of the network and extract stronger discriminative features through a series of parameters adjustment. We add a Relu nonlinear activation function in each convolutional layer, which can boost the nonlinear relationship between layers.

The training data on each software project is defined as $X_f = \{(X_i, Y_i) | i = 1, \ldots, N\}$, where $X_i \in Z^{w \times h}$ (w and h are the width and height of the matrix reshaped for each defect instance) is the ith training instance, $Y_i \in Z^m$ is the class label (defective and non-defective) corresponding to X_i, and N is the number of the training data on each project.

Convolutional layer: In the convolutional layer, we use $\varphi_{i,j}^{c,d}$ to represent the value of an unit at the position (c, d) in the jth feature map of the ith layer , as shown in Eq. (12):

$$\varphi_{i,j}^{c,d} = \psi\left(b_{i,j} + \sum_{\tau} \sum_{e=0}^{E_i-1} \sum_{f=0}^{F_i-1} w_{i,j,\tau}^{e,f} \varphi_{\tau(i-1)}^{(e+c)(f+d)}\right), \tag{12}$$

where $w_{i,j,\tau}^{e,f}$ represents the value at the position (e, f) of the kernel connected to the feature map, E_i and F_i denote the height and

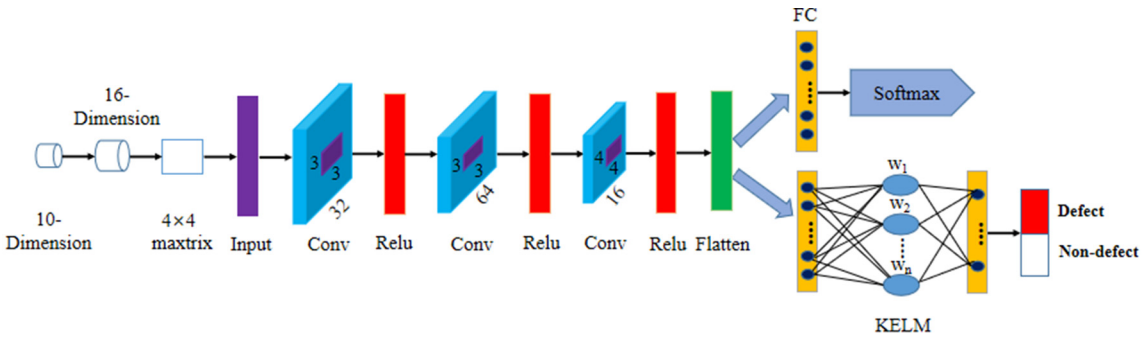


Fig. 3. Network architecture of the proposed WSHCKE model.

width of the filter kernel, respectively, $b_{i,j}$ denotes the bias of the feature map when τ indexes over the feature map set in the (i-1)th layer connected to the convolutional layer.

The convolutional layer can transform the low-level feature representation of the defect instances to the deep semantic feature representation through a nonlinear mapping. The convolution operation can be rewritten as follows:

$$\varphi_j = \psi(\sum w_{i,j} \otimes \varphi_{i-1}), \tag{13}$$

where $\otimes$ denotes the convolution operation, $w_{i,j}$ represents the value of the ith layer in the jth feature map, φ_{i-1} denotes the output of the (i -1)layer, and φ_j represents the output of the jth feature map in the convolutional layer.

After completing all training, we extract the deep semantic features of the last training behind the fully connected layer and feed them to the next KELM classifier. The deep semantic features integrated by CNN are defined as the feature vector V, and the parameters (i.e., the network weights) involved in this process are defined as λ_{CNN}, so the feature vector V_i is also expressed as follows:

$$V_i = g_{\lambda_{\text{CNN}}}(X_i), \tag{14}$$

where $g_{\lambda_{\text{CNN}}}(.)$ represents the mapping function for the feature vector V_i integrated from X_i.

5.2. Defect prediction based on KELM

As mentioned before, KELM has many advantages in classification compared to softmax. Therefore, we replace the softmax layer with the KELM classifier at the end of CNN in training stage 2. The KELM can take full advantage of the extracted strong discriminative deep semantic features with intra-class compactness and inter-class separation to predict whether the instance modules are defective. The network structure of KELM is shown in Fig. 4.

We regard these deep semantic features of the last training behind the fully connected (FC) layer as the input of the KELM. The KELM classifier is defined as follows: The input of the KELM is deep semantic feature vector $V \in Z^{n_i}$ integrated by CNN, n_i is the number of the input neurons, n_h is the number of the hidden neurons, n_o is the number of the output neurons (Since the defect data is divided to two classes: defective and non-defective, n_o is equal to 2), $w \in Z^{n_i \times n_h}$ denotes the input weight matrix from the input layer to the hidden layer, $b \in Z^{n_h}$ denotes the bias vector of the hidden layer, $\beta \in Z^{n_h \times n_o}$ denotes the output weight matrix from the hidden layer to the output layer and $O \in Z^{n_o}$ denotes the output vector of the KELM.

Therefore, new training data can be formed to train the KELM classifier, which is defined as D_V:

$$D_V = \{(V_i, Y_i) | V_i \in Z^{n_i}, i = 1, \ldots, N\}, \tag{15}$$

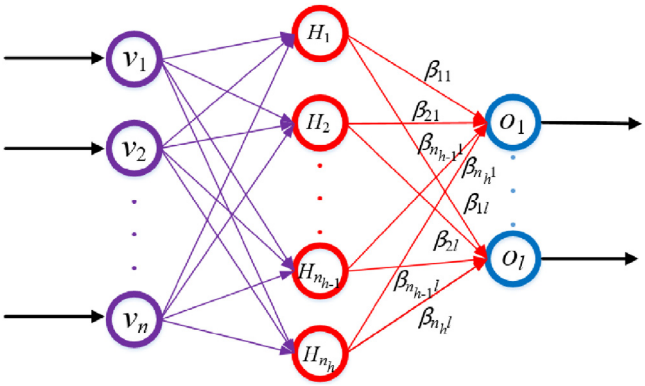


Fig. 4. The network structure of KELM.

where V_i represents the feature vector of each defect instance, n represents the number of the defect instances for training on each project .

Next the output of the hidden layer $h(V_i) \in Z^{n_h}$ and the output of the KELM (i.e., the output of the output layer) $O_i \in Z^{n_o}$ can be represented respectively as follows:

$$h(V_i) = f(wV_i + b). \tag{16}$$

$$O_i = h(V_i)\beta. \tag{17}$$

For the training set of each software project, the output of the hidden layer $H \in Z^{n \times n_h}$ and the corresponding output label matrix $U \in Z^{n \times n_o}$ can be represented respectively as follows:

$$H = [h(V_1), h(V_2), \ldots, h(V_n)]^T. \tag{18}$$

$$U = [\mu_{Y_1}, \mu_{Y_2}, \ldots, \mu_{Y_n}]^T. \tag{19}$$

The standard ELM classifier can approximate any instance with zero error, and the rule also applies to our KELM. In other words, for a specific defect training set D_V, there exist w, b and β that make the following Eq. (30) true:

$$O_i = \mu_{Y_i}. \tag{20}$$

So far we can obtain the output weight matrix β by minimizing the mean square error, the final β can be calculated according to the above Eqs. (18)–(20):

$$\beta = H^{\dagger}U, \tag{21}$$

where $H^{\dagger}$ denotes the Moore–Penrose generalized inverse of the H.

The output of the KELM classifier can be rewritten as Eq. (22):

$$g = H\beta = h(V_i)H^{\dagger}U. \tag{22}$$

From the above derivation process of KELM, we can notice that n_h is the only one hyperparameter in the KELM classifier, so the

calculation is relatively simple, which is also an advantage of the KELM classifier.

In standard ELM, the hidden layer of ELM maps data from the original feature space to the higher dimensional space, where each dimension corresponds to a hidden unit. However, the new high-dimensional space can be regarded as a feature space where the ELM just solves a linear transformation. To solve this problem, we adopt the KELM based on Gaussian kernel, which can achieve the nonlinear transformation from the lower dimensional feature space to the higher dimensional space, and the equation of the Gaussian kernel is as follows:

$$G(x, x', \sigma) = e^{-\frac{\|x-x'\|^2}{2\sigma^2}}, \tag{23}$$

where σ is the bandwidth, which controls the range of radial action.

Therefore, we can define a kernel matrix for KELM as follows:

$$\Phi_{\text{KELM}} = HH^T : \Phi_{\text{KELM}_{i,j}} = h(V_i).h(V_j) = K(V_i, V_j). \tag{24}$$

The output of KELM can be represented as follows:

$$O(V) = h(V)H^T\left(\frac{I}{C} + HH^T\right)^{-1}T = \begin{bmatrix} K(V, V_1) \\ \cdots \\ \cdots \\ K(V, V_N) \end{bmatrix}^{\mathrm{T}} \left(\frac{I}{C} + \Phi_{\text{KELM}}\right)^{-1}T, \tag{25}$$

where I is an identity matrix and its dimension is the same as that of HH^T. A positive value 1/C is added to the diagonal of HH^T, which can achieve better generalization performance.

The network weights λ_{CNN} trained by the CNN and the network weights ϑ_{KELM} of the KELM classifier can be determined in the training stage. Next, in the test stage, the output of CNN and KELM and the final prediction label are denoted as follows:

The deep semantic feature vector V_{test} extracted by CNN with the trained network weights λ_{CNN} can be represented as follows according to the above Eq. (14):

$$V_{test} = g_{\lambda_{\text{CNN}}}(X_{test}), \tag{26}$$

where X_{test} denotes the test set in the prediction stage.

The output of the KELM with the trained network weights ϑ_{KELM} can be calculated as follows according to the Eqs. (16) and (17):

$$h_{\vartheta_{\text{KELM}}}(V_{test}) = f(wV_{test} + b). \tag{27}$$

$$O_{test} = h_{\vartheta_{\text{KELM}}}(V_{test})\beta. \tag{28}$$

The final prediction label O_l is denoted as follows:

$$O_l = arg\ \max_m O^m_{test}, \tag{29}$$

where O^m_{test} is the mth component of the O_{test}, and O_l can determine whether the test instances are defective.

The pseudo-code of the proposed WSHCKE predictor is as shown in Algorithm 3.

In training stage 1 (steps 1–24), for steps 1–2, we first process each defect instance for each project after feature selection into the dimension which can reshape into matrix by adding columns of zero at the end, and then reshape the dimensional features to a matrix (where $X_i \in Z^{w\times h}$, w=h, $Y_i \in Z^m$), thereby facilitating the subsequent convolution operations. The original dimensional features are not convenient for subsequent convolution processing. In steps 3–24, the CNN begins to conduct iterative training, and the maximum number of iterations is set to I_{max} and the number of network layers is set to L_{CNN}. When the network layer is the *INPUT* layer, we feed this defect instance matrix into the CNN structure; when the network layer is the convolutional (*CONV*) layer, the convolution operation integrates the defect features according to Eq. (13) and employs *Relu* nonlinear activation function to map nonlinear relationships, thereby generating the network weights $w^{e,f}_{i,j,\tau}$ and biases $b_{i,j}$; when the network layer is the *Flatten* layer, the layer flattens the matrix back to corresponding features; when the network layer is the Fully connected (*FC*) layer, the layer converts the 2-dimensional feature maps into 1-dimensional vectors ($X^{l_n}_i \in Z^{w'\times 1}$, $i = 1, \ldots, N$); when the network layer is the *Softmax* layer, the network adopts sparse cross entropy loss function and participates in the end-to-end training of CNN. When the number of iterations $I_t > I_{max}$ or the error $E > \xi$, we extract the deep semantic features with the trained weights λ_{CNN} behind the *FC* layer after the last training, otherwise, the CNN continues to carry out back propagation.

In training stage 2 (steps 25–28), we first feed the extracted deep semantic features with the trained weights λ_{CNN} to the KELM classifier, and then calculate the output matrix H of hidden layer according to Eqs. (16) and (18) and the output weight matrix β of the KELM according to Eqs. (19) and (21), thus obtaining the output of the KELM with the trained weights ϑ_{KELM} : $O(V) = h(V)H^T\left(\frac{I}{C} + HH^T\right)^{-1}T$.

In testing stage (step 29), we employ the trained WSHCKE (CNN with KELM) with the trained weights λ_{CNN} and ϑ_{KELM} to predict whether the defect instances in the test set are defective.

6. Experimental setup

In this section, we show the experimental setup, including benchmark datasets, evaluation indicators, experimental design, parameter settings, Scott–Knott Effect Size Difference (ESD) test and Wilcoxon signed-rank test and Cliff's Delta effect size analysis.

6.1. Benchmark datasets

In this paper, we use a total of three datasets, including PROMISE,[1] (Jureczko and Madeyski, 2010) ReLink,[2] (Wu et al., 2011) and AEEEM,[3] (D'Ambros et al., 2012) which are publicly available and commonly used benchmark datasets in software defect prediction studies (Kondo et al., 2019; Lu et al., 2014; Li et al., 2019a; Yan et al., 2017; Chen and Ma, 2015; Nam et al., 2018). We conduct extensive experiments on 20 open source software projects from these three datasets, including 12 projects from PROMISE, 3 projects from ReLink, and 5 projects from AEEEM.

The PROMISE dataset is collected by Jureczko and Madeyski (2010). Each project in the PROMISE dataset has 20 metrics (features), which contain Object-Oriented (OO) metrics, McCabe's cyclomatic metrics, CK metrics. The ReLink dataset is collected by Wu et al. (2011), which contains three projects: Apache, Safe, ZXing, and these projects all have 26 metrics. The AEEEM dataset is collected by D'Ambros et al. (2012), and each project in the AEEEM dataset contains 61 metrics: 5 previous-defect metrics, 5 entropy-of-change metrics, 17 source code metrics, 17 entropy-of-source-code metrics, and 17 churn-of-source-code metrics.

Table 1 summarizes the basic information of 20 projects in three datasets, which shows the project names, the number of features, the number of instances, the number of defective instances, the number of non-defective instances, defective ratio (%) and imbalance ratio. From Table 1, we observe that each project in these three datasets is class imbalanced, among which the

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

[2] http://www.cse.ust.hk/~scc/ReLink.htm

[3] http://bug.inf.usi.ch/

Table 1
Statistics for 20 software projects in three datasets.

Datasets	Projects	# of features	# of instances	# of defective instances	# of non-defective instances	Defective ratio (%)	Imbalance ratio
PROMISE	ant-1.6	20	351	92	259	26.21	2.82
	ant-1.7	20	745	166	579	22.28	3.49
	camel-1.4	20	872	145	727	16.63	5.01
	camel-1.6	20	965	188	777	19.48	4.13
	ivy-2.0	20	352	40	312	11.36	7.80
	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
	synapse-1.2	20	256	86	170	33.59	1.98
	xalan-2.5	20	803	387	416	48.19	1.07
	xerces-1.2	20	440	71	369	16.14	5.20
ReLink	Apache	26	194	98	96	50.52	0.98
	Safe	26	56	22	34	39.29	1.55
	ZXing	26	399	118	281	29.57	2.38
AEEEM	JDT	61	997	206	791	20.66	3.84
	LC	61	691	64	627	9.26	9.80
	EQ	61	324	129	195	39.81	1.51
	PDE	61	1492	209	1283	14.01	6.14
	ML	61	1862	245	1617	13.16	6.60

imbalance ratio of jedit-4.3, ZXing and LC in these three datasets reach the highest 43.73, 2.38 and 9.80, respectively. Therefore, it is very necessary to conduct class imbalance processing on these software projects. In addition, each instance in any project contains a certain number of features and a dependent variable that denotes the number of defects in an instance module. We label the instance module as 1 if it contains one or more defects; otherwise, we label it as 0.

6.2. Evaluation indicators

We use *F1*, *MatthewsCorrelationCoefficient*(*MCC*), $G-measure$ and *AUC* to evaluate the performance of the models, which are widely used in defect prediction studies (Ma et al., 2012; Wang et al., 2016b; Li et al., 2012; Zhu et al., 2020; Zhou et al., 2019; Xu et al., 2019). The performance evaluation is based on the classification results in a confusion matrix, which describes how a prediction model classifies different defect categories compared to their actual classification, as shown in Table 2.

F1: The comprehensive evaluation for harmonic mean of precision and recall, as shown in Eq. (30):

$$F1 = \frac{2 * recall * precision}{recall + precision}. \tag{30}$$

MCC (Matthews Correlation Coefficient): The correlation between the observed and predicted binary classification with values in [−1,1], which is a comprehensive evaluation by considering *TP*, *TN*, *FP*, and *FN*, as shown in Eq. (31):

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}. \tag{31}$$

G − measure: The comprehensive evaluation for harmonic mean of pd and $1 - pf$, as shown in Eq. (32).

$$G - measure = \frac{2 * pd * (1 - pf)}{pd + (1 - pf)}. \tag{32}$$

AUC : AUC denotes the **A**rea **U**nder the *R*eceiver *O*perating *C*haracteristic (*ROC*) **C**urve (**AUC**), and the curve can be drawn in a two-dimensional space with pf (probability of false alarm) as *x*-axis and pd (probability of detection) as *y*-axis. AUC is independent from the cutoff threshold used to decide whether an instance is classified as positive or negative.

The larger these four indicator values, the better the prediction performance.

6.3. Experimental design

In this paper, we conduct extensive experiments for feature selection or extraction and defect prediction, and set up the following six sets of state-of-the-art baseline methods corresponding to six research questions (RQs) in Section 7:

(1) Eleven state-of-the-art feature selection or extraction approaches (for RQ1), including Principal Component Analysis (PCA) (Kondo et al., 2019), Factor Analysis (FA) (Ali et al., 2017), Diffusion Maps (DM) (Pascual et al., 2015), Stochastic Neighbor Embedding (SNE) (Bunte et al., 2012), Locality Preserving Projection (LPP) (Chen et al., 2017), Neighborhood Preserving Embedding (NPE) (Zhao et al., 2013), Locally Linear Coordination (LLC) (Huang et al., 2009), Manifold Charting (MC) (Saini et al., 2013) and Maximally Collapsing Metric Learning (MCML) (Globerson and Roweis, 2005), which are all feature extraction approaches. In addition, two correlation-based feature selection approaches — Genetic Search (GS) (Goldberg, 1989; Ghotra et al., 2017) and BestFirst (BF) search (Ghotra et al., 2017) are also included. We compare the EMWS algorithm with eleven state-of-the-art feature selection or extraction approaches using the same defect predictor WSHCKE. For our EMWS algorithm, we also report that what is the minimum number of selected features while achieving the minimum error on each project in RQ1a.

(2) Nine classic defect predictors (for RQ2), including Naive Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT), K-Nearest Neighbor (KNN) and Random Forest (RF). Moreover, three deep learning-based models — Deep Belief Network (DBN) (Yang et al., 2015a), Convolutional Neural Network (CNN) (Hoang et al., 2019), Defect Prediction based on Deep Forest (DPDF) (Zhou et al., 2019) are also included. Since these nine defect predictors all use features selected by the EMWS algorithm, they are renamed as WSNB, WSSVM, WSLR, WSDT, WSKNN, WSRF, WSDBN, WSCNN and WSDPDF respectively in this paper, where WS represents **W**OA-**S**A.

(3) Same baseline approaches as baseline approach (2) in terms of efficiency (for RQ3). We compare WSHCKE with nine classic defect predictors in terms of time cost, so as to verify whether the time taken by our WSHCKE predictor is within an acceptable range.

(4) Generalization performance of eleven state-of-the-art feature selection or extraction approaches in baseline approach (1) and nine classic defect predictors in baseline approach (2) (for

Table 2
Confusion matrix.

	Positive (Predicted)	Negative (Predicted)
True (Actual)	TP	FN
Flase (Actual)	FP	TN
Precision	$\frac{TP}{TP+FP}$	
Recall or pd (probability of detection)	$\frac{TP}{TP+FN}$	
pf (probability of false alarm)	$\frac{FP}{FP+TN}$	

Algorithm 3 Pseudo-code of the proposed WSHCKE predictor

Input:
$X_f = \{(X_i, Y_i)|i = 1, ..., N\}$, maximum iteration number: I_{max}, minimum error: ξ, the network layers of the CNN : L_{CNN}.
Output:
The final prediction result: R.

Process each defect instance to the dimension which can reshape into matrix by adding columns of zero; /* Training stage 1 (steps 1-24) */
Reshape each defect instance into a matrix (where $X_i \in Z^{w\times h}$, $Y_i \in Z^m$);
for $I_t \leftarrow 1$ to I_{max} **do**
  for $l_n \leftarrow 1$ to L_{CNN} **do**
    if *network.layer*[l_n] = *INPUT* **then**
      Feed the defect instance matrix into the CNN structure;
    else if *network.layer*[l_n] = *CONV* **then**
      Integrate the defect features according to Eq. (13);
      Use *Relu* nonlinear activation function to map nonlinear relationships;
      Generate the network weights $w_{i,j,\tau}^{e,f}$ and biases $b_{i,j}$;
    else if *network.layer*[l_n] = *Flatten* **then**
      Flatten the matrix back to corresponding features;
    else if *network.layer*[l_n] = *FC* **then**
      Convert the 2-dimensional feature maps into 1-dimensional vectors ($X_i^{l_n} \in Z^{w'\times 1}$, i = 1,...,N);
    else if *network.layer*[l_n] = *Softmax* **then**
      Adopt sparse cross entropy loss function and participate in the end-to-end training of CNN;
    end if
  end for
  if $I_t > I_{max}$ || $E > \xi$ **then**
    Extract the deep semantic features with the trained weights λ_{cnn} behind the *FC* layer after the last training;
  else
    Conduct back-propagation;
  end if
end for
Feed the extracted deep semantic features with the trained weights λ_{CNN} to the KELM classifier; /* Training stage 2 (steps 25-28) */
Calculate the output matrix H of hidden layer according to Eq. (16) and Eq. (18);
Calculate the output weight matrix β of the KELM according to Eq. (19) and Eq. (21);
Obtain the output of the KELM with the network weights ϑ_{KELM}: $O(V) = h(V)H^T(\frac{I}{C} + HH^T)^{-1}T$;
Use the trained WSHCKE with the trained weights λ_{CNN}, ϑ_{KELM} to predict whether the defect instances in the test set are defective.; /* Testing stage (step 29) */
Obtain the final prediction result R;
return R

RQ4). To investigate the generalization capacity of our EMWS feature selection algorithm and WSHCKE predictor, we also compare EMWS with eleven state-of-the-art feature selection or extraction approaches in baseline method (1) using the same defect predictor respectively (i.e., each of the nine classic defect predictors), and compare WSHCKE with nine classic defect predictors in baseline method (2) using the same feature selection approach respectively (i.e., each of eleven feature selection or extraction approaches).

(5) Original defect features without feature selection. To verify whether the features selected by the EMWS algorithm have advantages in prediction performance compared with the original defect features without feature selection, we also compare WSHCKE using EMWS with not using EMWS.

(6) Adopt Naive Bayes (NB) and K-Nearest Neighbor (KNN) as classifiers instead of Logistic Regression (LR) classifier in the EMWS feature selection algorithm. To investigate the feature selection capability of our EMWS algorithm more comprehensively and the impact of using different feature selection classifiers within EMWS on prediction performance, we replace LR classifier with NB and KNN classifiers in the EMWS feature selection algorithm, and conduct defect prediction using the same defect predictor WSHCKE after feature selection.

6.4. Parameter settings

In the paper, feature selection is only performed on the training set. We first perform feature selection on the training set, thereby selecting a feature subset from this training set. Then the corresponding test set also retains the same features as the training set, thus ensuring that the training set and the test set have the same feature dimensions. Only when the feature dimensions for the training set and the test set are consistent, we can employ deep learning or machine learning models to predict whether the defect instances are defective.

For our EMWS feature selection algorithm, some important parameters need to be set or adjusted. To determine the number of search agents, we first set up six groups of search agents, the numbers of which are 5,10,15,20,25,30, and conduct feature selection using the EMWS algorithm respectively. Then we feed the features after feature selection to the WSHCKE predictor, and finally the number of search agents that can achieve the highest AUC value on most software projects can be determined. The number of search agents cannot be too few or too many, otherwise the prediction performance is not ideal. The maximum number of iterations for the entire EMWS is set to 200. The maximum number of iterations and the number of iterations at each temperature T for SA are set to 60 and 20, respectively.

The parameter settings of eleven baseline feature selection or extraction algorithms are detailed as follows. For PCA, the PCA eigenvalue is adopted as intrinsic dimensionality estimator. For FA, the dimensionality reduction is conducted by the EM (Expectation Maximization) algorithm, and the regularization parameter is set to 1E-9 and the maximum number of iterations is set to 200. For DM, the variable *sigma* for the variance of the Gaussian used in the affinity computation is set to 1, and the

variable *alpha* that determines the operator applied on the graph is set to 1. For SNE, we conduct the SNE algorithm through a Gaussian kernel with fixed perplexity (default = 30), and set the following parameters: learning rate: 0.05, the maximum number of iterations: 2000, initial momentum: 0.5, final momentum: 0.8. For LPP, the number of neighbors is specified by 12. The variable *sigma* that determines the bandwidth of the Gaussian kernel is defaulted to 1. For NPE, the number of neighbors is specified by 12. For LLC, we set the parameters as follows: tolerant error: 1E-5, the number of neighbors: 12, the maximum number of iterations: 200, the number of mixture: 20. For MC, we set the parameters as follows: the maximum number of iterations: 200, the number of mixture: 20. For MCML, the maximum number of iterations is set to 200. For BF, it adopts forward search as search strategy. For GS, the population size is set to 100, the maximum number of iterations is set to 200. The crossover probability and mutation probability are set as 0.5 and 0.01 respectively.

For our WSHCKE predictor, some network parameters need to be set or adjusted. As mentioned in 5.1, since the feature dimension of each project is different after feature selection, the network parameters of the WSHCKE predictor for each project are also different. We take the 10-dimensional features selected by the EMWS algorithm as an example. In the CNN structure, the CNN has the following network structure: an input layer, three convolutional layers, three Relu nonlinear activation functions, a flatten layer, a fully connected layer and a softmax layer. We first process 10-dimensional features selected by the EMWS algorithm to 16-dimensional features by adding six columns of zero at the end, and then reshape into a 4×4 matrix, so that these features can be better for convolution operation. Next, we perform three convolution operations. The number of convolution kernels is in each convolutional layer is 32, 64, and 16 respectively, and convolution kernel sizes are 3×3, 3×3, and 4×4 respectively. We add a Relu nonlinear activation function in each convolutional layer. Unlike images, our software project data is not complex, so we determine the number of convolutional layers through ablation analysis, i.e., gradually increase the number of convolutional layers and select the number of convolutional layers with the highest AUC value, which is also a commonly used method for adjusting simple neural network parameters. Meanwhile, we manually set the convolution kernel size (generally the height is equal to the width, and the value is relatively small) and the number of convolution kernels (generally 2^n, n = 3, 4.....) according to our experience, thereby determining the convolution kernel size and the number of convolution kernels with the highest AUC value respectively. We also manually set the batch size (generally 2^n, n = 3, 4.....) and the learning rate (generally 10^{-n}, n = 1, 2, 3, 4, 5) according to our experience, thus determining the batch size and the learning rate with the highest AUC value respectively. Moreover, we adopt some optimization strategies, such as the early stop training iteration strategy before overfitting in a small number of software projects, to boost prediction performance. We adopt sparse softmax cross entropy as loss function in this paper.

In the KELM structure, a salient characteristic of KELM is that the hidden layer need not be tuned (Huang et al., 2012). The ELM is originally proposed to employ random computational neurons in the hidden layer, which are independent of the training data (Huang et al., 2012). However, since the KELM is empty when it is created, we need to add a certain number of neurons to an empty KELM to make it work. When the number of neurons exceeds a certain number, the ELM may slow down and the prediction performance may decrease. This is because the computational complexity is proportional to a qube of number of neurons. We take the 10-dimensional features selected by the EMWS algorithm as an example. We first set up eight groups of neurons, the numbers of which are 50, 100, 150, 200, 250, 300, 350, 400, and then conduct experiment respectively, and finally determine the number of neurons with the highest AUC.

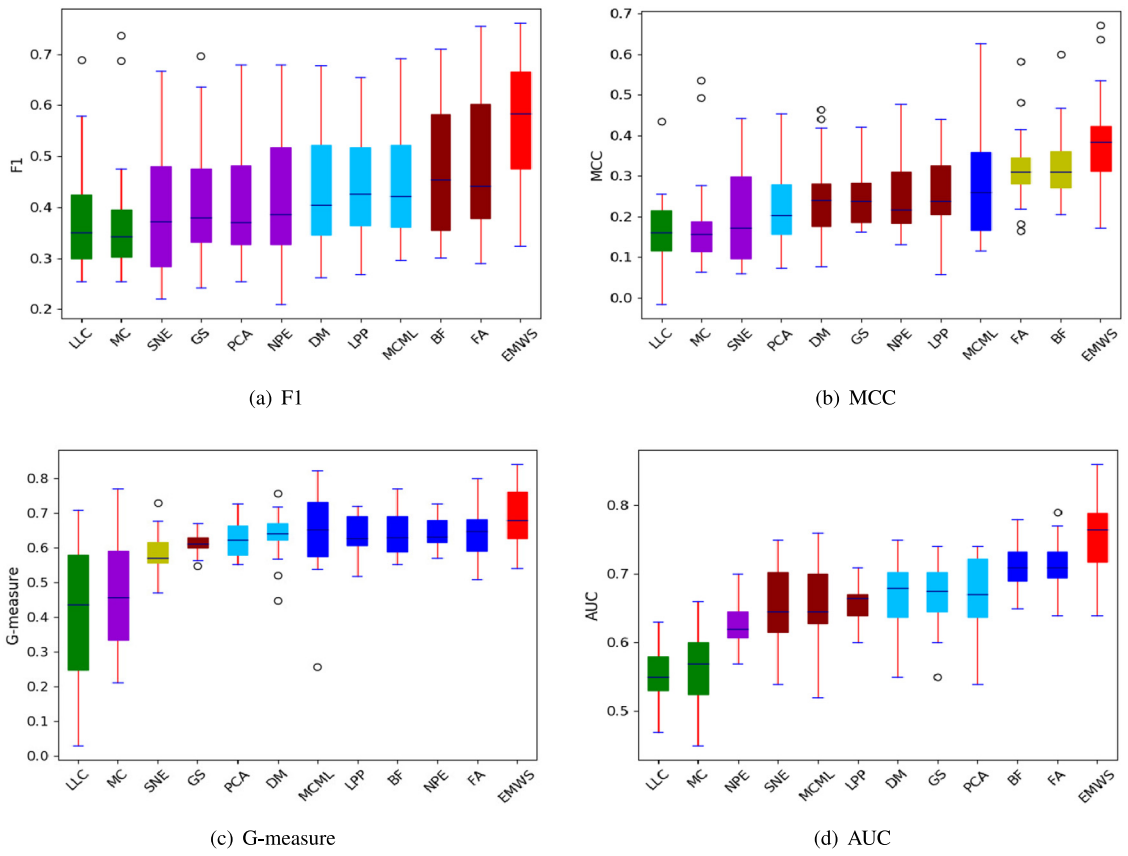
The parameter settings of nine baseline predictors are detailed as follows. For WSNB, we adopt the kernel estimator that can achieve the best AUC values on most software projects. For WSSVM, we adopt the Gaussian kernel as the kernel function. As mentioned by Hsu and Lin (2002), the cost parameter $C = 2^{-2}$, 2^{-1}, …, 2^{12} while the kernel parameter $gamma = 2^{-10}$, 2^{-9}, …, 2^{4}. The optimal parameter combination for the cost parameter C and the kernel parameter *gamma* can be achieved according to the highest AUC value by the grid search. For WSLR, the random state adopts *None* and the distribution adopts *normal*, and the tolerant error is set to 1E-4. For WSDT, the criterion adopts *gini*, and the minimum sample leaf is set to 1, and the minimum sample split is set to 2. For WSKNN, the number of neighbors with the highest AUC value is set to 1 by setting different numbers of neighbors. For WSRF, the number of generated trees is set to 10 and the number of variables for random feature selection is set to 2, and the criterion adopts *gini*. In addition, we do not limit the maximum depth of the trees as suggested by Elish and Elish (2008). For WSCNN, the parameter settings and the parameter adjustment methods are the same as the WSHCKE predictor. Ror WSDBN, we adopt the same parameter settings and network structure as in Yang et al. (2015a). For WSDPDF, we adopt the same parameter settings method as in Zhou et al. (2019) to select the number of decision trees in each forest.

6.5. Scott–Knott effect size difference (ESD) test

In this paper, we first employ the Scott–Knott Effect Size Difference (ESD) test (Tantithamthavorn et al., 2017, 2019) to rank multiple feature selection or extraction approaches and multiple defect predictors in terms of four indicators. Then we exhibit the boxplots with the Scott–Knott ESD test to visual the performance differences for these feature selection or extraction approaches and defect predictors. Each color denotes a rank, and there is a statistically significant performance difference between the two approaches in different ranks. The Scott–Knott Effect Size Difference (ESD) test is an alternative approach of the Scott–Knott test, which is a mean comparison approach that utilizes the hierarchical clustering to partition multiple approaches into statistically distinct groups with non-negligible difference, and considers the magnitude of the difference (i.e., effect size) for multiple approaches within a group and between groups.

6.6. Wilcoxon signed-rank test and Cliff's Delta effect size analysis

We also employ Wilcoxon signed-rank test (Fan et al., 2019) and Cliff's Delta effect size analysis (Li et al., 2019b) to check whether our models are statistically significant or not. The Wilcoxon signed-rank test (Fan et al., 2019) can be used for data pairs, and does not demand the underlying experimental data to follow any data distribution. Since multiple comparisons may induce the false discovery rate, we further employ the Benjamini–Hochberg correction method (Benjamini and Hochberg, 1995) to adjust the p-values in the following Tables 5, 8, 10, 11 and Fig. 9. At the 95% confidence level, p-values that are less than 0.05 represent that the differences between approaches are statistically significant, while p-values that are 0.05 or larger represent that the differences are not statistically significant. To measure the effectiveness between our methods and the baselines, we also leverage Cliff's delta (δ) (Li et al., 2019b) to measure the effect size between our models and the baselines. The Cliff's delta indicates a non-parametric effect size evaluation that measures the difference degree between two approaches. The δ is a evaluation



(a) F1 (b) MCC

(c) G-measure (d) AUC

Fig. 5. The Scott–Knott ESD ranking for EMWS compared with eleven baseline feature selection or extraction approaches in terms of four evaluation indicators. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Cliff's Delta and the effectiveness level.

Cliff's Delta ($\|\delta\|$)	Effectiveness Level
$\|\delta\| < 0.147$	Negligible (N)
$0.147 \leq \|\delta\| < 0.33$	Small (S)
$0.33 \leq \|\delta\| < 0.474$	Medium (M)
$0.474 \leq \|\delta\|$	Large (L)

of how often the values in one method are larger than the values in the other method. The δ values are in the interval $[-1,1]$, where $\delta = 1$ or -1 indicates that all values in one method are larger or smaller than those of the other method, and 0 indicates that all values in the two methods are completely overlapping. Table 3 presents the different δ values and their corresponding effectiveness levels.

7. Experimental results and analysis

We focus on the performance of the EMWS algorithm and the WSHCKE predictor, and answer and discuss the following six research questions (RQs):

RQ1: Does the enhanced metaheuristic EMWS algorithm outperform eleven state-of-the-art feature selection or extraction approaches in software defect prediction?

To validate the effect for the features selected by the enhanced metaheuristic EMWS algorithm on the prediction performance of the subsequent WSHCKE model, we compare EMWS with eleven state-of-the-art feature selection or extraction approaches using the same defect predictor WSHCKE across total 20 open source software projects from three datasets in terms of F1, MCC, G-measure and AUC, including PCA, FA, DM, SNE, LPP, NPE, LLC, MC, MCML, GS and BF. GS and BF are two correlation-based feature selection approaches, while the rest are feature extraction approaches.

Table 4 depicts the F1, MCC, G-measure and AUC of all twelve feature selection or extraction approaches across total 20 software projects from three datasets, including 12 projects from PROMISE, 3 projects from ReLink, 5 projects from AEEEM. In Table 4, we record the average performance indicators of all projects on each dataset (PROMISE, ReLink, AEEEM), and the average performance indicators of 20 projects in all three datasets (ALL). Note that the highest value of each row is marked in bold. From Table 4, we can observe that the EMWS algorithm can achieve the best average performance from the point of four indicators except for the G-measure indicator on the ReLink dataset. More specifically, for all three datasets (ALL), the average F1 (0.5607) by EMWS gains improvements between 16.93% (for FA) and 48.29% (for LCC) with an average improvement of 31.05%, the average MCC (0.3976) by EMWS achieves improvements between 20.89% (for BF) and 134.02% (for LLC) with an average improvement of 65.41%, the average G-measure (0.6950) by EMWS yields improvements between 7.87% (for FA) and 73.10% (for LCC) with an average improvement of 19.64%, and the average AUC (0.7616) by EMWS obtains improvements between 6.56% (for FA) and 37.57% (for LCC) with an average improvement of 18.14% compared with eleven baseline feature selection or extraction approaches.

To conduct a visual comparison of the prediction performance differences between EMWS and eleven baseline feature selection or extraction approaches, we exhibit the boxplots with the Scott–Knott ESD test in terms of four evaluation indicators. Fig. 5(a)(b)(c)(d) manifest the results of the Scott–Knott ESD test for all twelve feature selection or extraction approaches

Table 4
Four average indicator values for EMWS compared with eleven baseline feature selection or extraction approaches.

Datasets	Indicators	PCA	FA	DM	SNE	LPP	NPE	LLC	MC	MCML	GS	BF	EMWS
PROMISE	F1	0.4267	0.4408	0.4217	0.4259	0.4411	0.4307	0.362	0.3773	0.4553	0.4220	0.4347	**0.4851**
	MCC	0.2321	0.2977	0.2172	0.225	0.2546	0.2435	0.1432	0.1838	0.3038	0.2386	0.3060	**0.3525**
	G-measure	0.6333	0.6198	0.6119	0.5973	0.6385	0.6385	0.3095	0.4586	0.6351	0.6175	0.6131	**0.6455**
	AUC	0.6752	0.7134	0.6688	0.6655	0.6645	0.6372	0.5624	0.5715	0.6669	0.6779	0.7090	**0.7445**
ReLink	F1	0.5217	0.5905	0.5684	0.5217	0.5684	0.5660	0.4397	0.4229	0.5400	0.5278	0.5789	**0.6632**
	MCC	0.2032	0.3437	0.3246	0.2032	0.3246	0.2995	0.2255	0.1754	0.2664	0.2233	0.3545	**0.4162**
	G-measure	0.5882	**0.6786**	0.6691	0.5882	0.6691	0.6549	0.5518	0.2948	0.6412	0.6099	0.6648	0.6706
	AUC	0.7334	0.7656	0.7427	0.6200	0.6707	0.5745	0.5378	0.5543	0.7367	0.7008	0.7534	**0.7856**
AEEEM	F1	0.3944	0.5345	0.4574	0.3570	0.4300	0.3842	0.3980	0.3999	0.4174	0.3995	0.5166	**0.6909**
	MCC	0.2153	0.3651	0.3008	0.1631	0.2532	0.2785	0.2120	0.2202	0.2492	0.2800	0.3696	**0.4840**
	G-measure	0.6164	0.6865	0.6718	0.5702	0.6290	0.6486	0.5553	0.5226	0.6388	0.6054	0.6902	**0.7989**
	AUC	0.6467	0.7089	0.6486	0.6223	0.6427	0.6146	0.5490	0.5578	0.6148	0.6443	0.7129	**0.7920**
ALL	F1	0.4225	0.4795	0.4420	0.4103	0.4456	0.4246	0.3781	0.3872	0.4487	0.4216	0.4693	**0.5607**
	MCC	0.2251	0.3216	0.2500	0.2043	0.2586	0.2579	0.1699	0.1946	0.2844	0.2506	0.3289	**0.3976**
	G-measure	0.6276	0.6443	0.6342	0.5883	0.6374	0.6427	0.4015	0.4684	0.6366	0.6133	0.6404	**0.6950**
	AUC	0.6696	0.7147	0.6664	0.6488	0.6575	0.6259	0.5536	0.5634	0.6542	0.6682	0.7121	**0.7616**

Table 5
P-value and Cliff's deltas (δ) for EMWS compared with eleven baseline feature selection or extraction approaches in terms of four evaluation indicators.

Against	F1		MCC		G-measure		AUC	
	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)
PCA	0.0028	0.52(L)	0.0015	0.73(L)	0.0607	0.41(M)	0.0039	0.72(L)
FA	0.0092	0.31(S)	0.0252	0.41(M)	0.1471	0.31(S)	0.0076	0.50(L)
DM	0.0028	0.41(M)	0.0015	0.60(L)	0.0607	0.36(M)	0.0045	0.80(L)
SNE	0.0028	0.54(L)	0.0015	0.70(L)	0.0224	0.63(L)	0.0026	0.82(L)
LPP	0.0033	0.46(M)	0.0015	0.62(L)	0.1074	0.32(S)	0.0014	0.89(L)
NPE	0.0033	0.50(L)	0.0015	0.61(L)	0.0679	0.33(S)	0.0014	0.96(L)
LLC	0.0028	0.67(L)	0.0015	0.87(L)	0.0061	0.81(L)	0.0014	1(L)
MC	0.0028	0.63(L)	0.0015	0.77(L)	0.0061	0.69(L)	0.0014	0.99(L)
MCML	0.0151	0.43(M)	0.0160	0.48(L)	0.1961	0.27(S)	0.0026	0.85(L)
GS	0.0028	0.52(L)	0.0015	0.70(L)	0.0311	0.55(L)	0.0033	0.80(L)
BF	0.0030	0.38(M)	0.0437	0.38(M)	0.0332	0.33(S)	0.0076	0.56(L)

across total 20 software projects in terms of F1, MCC, G-measure and AUC, respectively. Each color denotes a rank: approaches in different ranks have a statistically significant difference in the prediction performance of the model. The blue line in each box denotes the median indicator value for each feature selection or extraction approach. The *x*-axis represents twelve feature selection or extraction approaches; the *y*-axis represents different evaluation indicators. From Fig. 5(a)(b)(c)(d), we can find that the enhanced metaheuristic EMWS algorithm is in the highest rank (the red box), which indicates that the EMWS can achieve the optimal prediction performance in terms of four evaluation indicators compared with eleven baseline feature selection or extraction approaches. We also find that the median value gained by EMWS is higher than those gained by eleven baseline feature selection or extraction approaches in terms of four evaluation indicators respectively, which fully validates the superiority of EMWS.

To comprehensively reflect the superiority of EMWS, we employ the Wilcoxon signed-rank test to validate whether the performance differences between EMWS and eleven baseline feature selection or extraction approaches are statistically significant, and further employ the Benjamini–Hochberg correction method to adjust the p-values. At the 95% confidence level, p-values that are less than 0.05 represent that the performance differences between approaches are statistically significant. In addition, we also leverage Cliff's delta (δ) to measure the effect size between EMWS and eleven baseline feature selection or extraction approaches. The Cliff's delta values (δ) and the corresponding effectiveness (E) level is shown in Table 3. A positive δ value represents that the EMWS approach can enhance the prediction performance in terms of the effect size. Table 5 depicts p-values and Cliff's deltas (δ) of EMWS compared with eleven baseline feature selection or extraction approaches in terms of four evaluation indicators. In terms of F1, MCC and AUC, the corrected p-values are less than 0.05 (significant) and the δ is larger than 0.147 (not negligible), and even the δ is greater than 0.474 (L) in terms of AUC, which are consistent with the observations in Table 4 and Fig. 5(a)(b)(d). In terms of G-measure, the corrected p-values are less than 0.05 (significant) in five feature extraction approaches and the δ is larger than 0.147 (not negligible).

To sum up, the enhanced metaheuristic EMWS algorithm can achieve the best prediction performance using the same defect predictor WSHCKE in terms of four evaluation indicators compared with eleven state-of-the-art feature selection or extraction approaches. This may be because the enhanced EMWS algorithm can take full advantage of the strong local search capability of SA to enhance the weak exploitation performance of WOA, and leverage strong global search capability of WOA to boost the weak exploration of SA a large extent simultaneously. The unique properties of WOA and SA can be integrated to achieve better search performance than using each one separately, so as to search for the optimal feature subset for each software project.

> **Conclusion 1**: The enhanced metaheuristic EMWS algorithm performs better than eleven state-of-the-art feature selection or extraction approaches. The EMWS achieves the average 31.05%, 65.41%, 19.64% and 18.14% performance improvements compared with eleven feature selection or extraction approaches using the same defect predictor WSHCKE across total 20 projects in terms of F1, MCC, G-measure and AUC, respectively.

RQ1a: For the enhanced metaheuristic EMWS algorithm, what is the minimum number of selected features while achieving the minimum error on each project?

For our EMWS algorithm, we also report that what is the minimum number of selected features while achieving the minimum error on each project. We perform five experiments on each

Table 6
Average selected features and classification error achieved by the enhanced EMWS algorithm.

Projects	# of total features	# of selected features	Selected proportion	Error	Fitness
ant-1.6	20	6.4	0.32	0.0811	0.1662
ant-1.7	20	9.2	0.46	0.1296	0.1497
camel-1.4	20	10.8	0.54	0.1170	0.1346
camel-1.6	20	10.6	0.53	0.1365	0.1560
ivy-2.0	20	8.2	0.41	0.0577	0.1001
jedit-4.2	20	9	0.45	0.0753	0.0985
jedit-4.3	20	7.6	0.38	0.0845	0.1109
poi-2.0	20	8.2	0.41	0.0795	0.1132
prop-6	20	7	0.35	0.0842	0.0911
synapse-1.2	20	7.8	0.39	0.1059	0.1695
xalan-2.5	20	8	0.40	0.2068	0.2726
xerces-1.2	20	7.8	0.39	0.0786	0.1240
Apache	26	12.8	0.49	0.1753	0.2212
Safe	26	9.6	0.37	0.0589	0.1214
ZXing	26	10.8	0.42	0.1815	0.2018
JDT	61	21.4	0.35	0.0961	0.1227
LC	61	22.6	0.37	0.0383	0.0649
EQ	61	20	0.33	0.1283	0.1819
PDE	61	17.6	0.29	0.0823	0.1005
ML	61	14	0.23	0.0934	0.1046

Table 7
Four average indicator values for WSHCKE compared with nine classic defect predictors.

Datasets	Indicators	WSNB	WSSVM	WSLR	WSDT	WSKNN	WSRF	WSDBN	WSCNN	WSDPDF	WSHCKE
PROMISE	F1	0.3008	0.3419	0.3764	0.3338	0.3690	0.4099	0.3903	0.4166	0.4322	**0.4851**
	MCC	0.2126	0.2003	0.3001	0.2448	0.2448	0.2927	0.2986	0.3120	0.3319	**0.3525**
	G-measure	0.5122	0.5061	0.5709	0.5398	0.5895	0.5536	0.5649	0.5465	0.6117	**0.6455**
	AUC	0.5523	0.5732	0.6939	0.5613	0.5957	0.6853	0.6313	0.6568	0.729	**0.7442**
ReLink	F1	0.4818	0.5298	0.5545	0.5492	0.5892	0.6038	0.5655	0.5895	0.6433	**0.6632**
	MCC	0.2848	0.2942	0.3575	0.3840	0.3240	0.3188	0.3467	0.3766	0.3897	**0.4162**
	G-measure	0.5262	0.5751	0.5947	0.5925	0.6547	0.6067	0.6333	0.6207	0.6422	**0.6706**
	AUC	0.5923	0.6346	0.6875	0.6567	0.6576	0.7154	0.6756	0.6909	0.7423	**0.7856**
AEEEM	F1	0.4744	0.5110	0.6053	0.5072	0.5681	0.6588	0.5864	0.5997	0.6825	**0.6913**
	MCC	0.2908	0.3182	0.3746	0.3449	0.3268	0.3785	0.3449	0.3522	0.4060	**0.4840**
	G-measure	0.5390	0.6166	0.7094	0.6758	0.6882	0.6221	0.6586	0.6873	0.7281	**0.7989**
	AUC	0.6142	0.6689	0.7123	0.6754	0.6987	0.7342	0.6865	0.7061	0.7222	**0.7923**
ALL	F1	0.3788	0.4057	0.4591	0.4015	0.4450	0.4998	0.4625	0.4846	0.5236	**0.5607**
	MCC	0.2415	0.243	0.3270	0.2848	0.2754	0.3212	0.3161	0.3286	0.3587	**0.3976**
	G-measure	0.5214	0.5449	0.6157	0.5856	0.6244	0.5783	0.5984	0.5911	0.6500	**0.6950**
	AUC	0.5738	0.6063	0.6981	0.6019	0.6306	0.7018	0.6506	0.6738	0.7276	**0.7616**

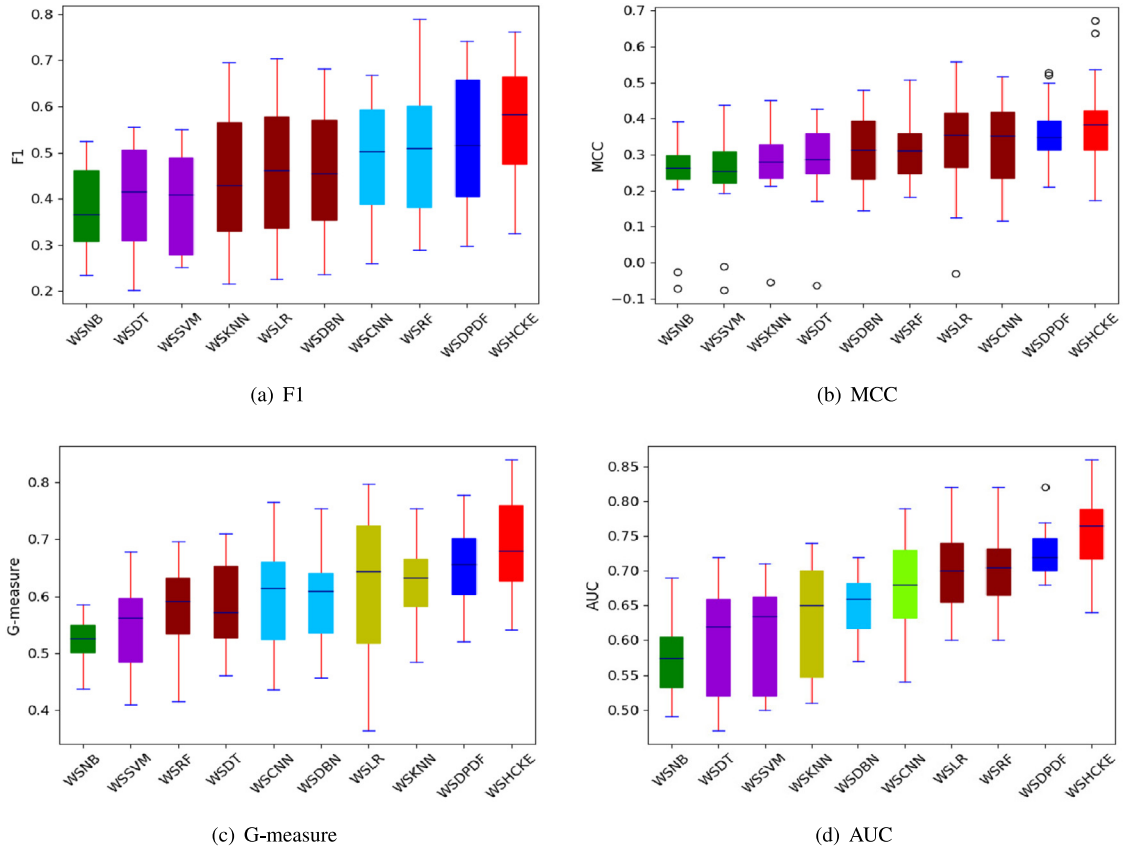
project with the EMWS algorithm, and record the average results of five experiments. Table 6 depicts the average classification accuracy, average number of selected features, average selected proportion and average fitness value achieved by the EMWS algorithm on each project. In the subsequent WSHCKE model, we adopt the features of the round in which the number of selected features ranks in the middle of five experiments on each project.

RQ2: Is our WSHCKE predictor superior to nine classic defect predictors?

Since we adopt the WSHCKE as the defect predictor in this paper, this question is designed to investigate the effectiveness of the new defect predictor WSHCKE. We compare the WSHCKE predictor with nine classic predictors using the same feature selection approach EMWS, including WWSNB, WSSVM, WSLR, WSDT, WSKNN, WSRF, WSDBN, WSCNN and WSDPDF, where the last three approaches are deep learning-based models and WS represents WOA-SA.

Table 7 shows the F1, MCC, G-measure and AUC of all ten defect predictors across total 20 software projects from three datasets, including 12 projects from PROMISE, 3 projects from ReLink, 5 projects from AEEEM. In Table 7, we record the average performance indicators of all projects on each dataset (PROMISE, ReLink, AEEEM) and the average performance indicators of 20 projects in all three datasets (ALL). Note that the highest value of each row is marked in bold. From Table 7, we can find that our WSHCKE predictor can achieve the optimal average performance in terms of four indicators. More specifically, for all three datasets (ALL), the average F1 (0.5607) by WSHCKE yields improvements between 7.09% (for WSDPDF) and 48.02% (for WSNB) with an average improvement of 25.58%, the average MCC (0.3976) by WSHCKE obtains improvements between 10.84% (for WSDPDF) and 64.64% (for WSNB) with an average improvement of 35.03%, the average G-measure (0.6950) by WSHCKE gains improvements between 6.92% (for WSDPDF) and 33.29% (for WSNB) with an average improvement of 18.28%, and the average AUC (0.7616) by WSHCKE achieves improvements between 4.67% (for WSDPDF) and 32.73% (for WSNB) with an average improvement of 17.56% compared with nine classic defect predictors.

To conduct a visual comparison of the prediction performance differences between WSHCKE and nine baseline defect predictors, we exhibit the boxplots with the Scott–Knott ESD test in terms of four evaluation indicators. Fig. 6(a)(b)(c)(d) visualize the results of the Scott–Knott ESD test for all ten defect predictors across total 20 software projects in terms of F1, MCC, G-measure and AUC, respectively. The blue line in each box represents the median indicator value for each defect predictor. The *x*-axis denotes ten defect predictors; the *y*-axis denotes different evaluation indicators. From Fig. 6(a)(b)(c)(d), we can observe that the WSHCKE predictor is in the top rank (the red box), which represents that WSHCKE predictor can achieve the best prediction performance in terms of four evaluation indicators compared with nine classic defect predictors. We also observe that the median value gained by WSHCKE is higher than those gained by nine classic defect



(a) F1 (b) MCC

(c) G-measure (d) AUC

Fig. 6. The Scott–Knott ESD ranking for WSHCKE compared with nine classic defect predictors in terms of four evaluation indicators. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 8
P-value and Cliff's deltas (δ) for WSHCKE compared with nine classic defect predictors in terms of four evaluation indicators.

Against	F1		MCC		G-measure		AUC	
	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)
WSNB	0.0009	0.70(L)	0.0029	0.71(L)	0.0011	0.93(L)	0.0014	0.98(L)
WSSVM	0.0009	0.61(L)	0.0023	0.66(L)	0.0011	0.73(L)	0.0014	0.93(L)
WSLR	0.0067	0.38(M)	0.1089	0.24(S)	0.0299	0.33(S)	0.0247	0.54(L)
WSDT	0.0009	0.61(L)	0.0023	0.49(L)	0.0011	0.59(L)	0.0014	0.92(L)
WSKNN	0.0009	0.45(M)	0.0058	0.57(L)	0.0059	0.41(M)	0.0014	0.85(L)
WSRF	0.0147	0.26(S)	0.0029	0.40(M)	0.0017	0.64(L)	0.0125	0.53(L)
WSDBN	0.0041	0.40(M)	0.0127	0.34(M)	0.0059	0.52(L)	0.0014	0.89(L)
WSCNN	0.0067	0.34(M)	0.0127	0.27(S)	0.0017	0.53(L)	0.0072	0.66(L)
WSDPDF	0.1788	0.16(S)	0.0068	0.20(S)	0.0011	0.24(S)	0.0221	0.41(M)

predictors in terms of four evaluation indicators respectively, which fully validates the superiority of our WSHCKE predictor.

To further validate the performance of the WSHCKE predictor, we utilize the Wilcoxon signed-rank test to verify whether the performance differences between WSHCKE and nine baseline defect predictors are statistically significant. Moreover, we also employ Cliff's delta (δ) to measure the effect size between WSHCKE and nine baseline defect predictors. Table 8 shows p-values with the Benjamini–Hochberg correction and Cliff's deltas (δ) of WSHCKE compared with nine baseline defect predictors in terms of four evaluation indicators. In terms of G-measure and AUC, the corrected p-values are less than 0.05 (significant) and the δ is larger than 0.147 (not negligible). In terms of F1 and MCC, the corrected p-values are less than 0.05 (significant) except for WSDPDF (0.1788) in F1 and WSLR (0.1089) in MCC and the δ is larger than 0.147 (not negligible).

In conclusion, the WSHCKE predictor can achieve the optimal prediction performance using the same feature selection approach EMWS in terms of four evaluation indicators compared with nine classic defect predictors. This may be because: (1) The CNN can extract the data-driven abstract deep semantic features hidden behind defect data, which not only have the strong inter-class separability, but also reserve unaltered intra-class compactness. Prior researches (Guo et al., 2017b; Duan et al., 2018; Ren et al., 2017) have demonstrated that the deep semantic features have stronger discriminating capacity for different classes (defective or non-defective). (2) The KELM used has good generalization performance, ease of implementation, fewer super-parameters, and little human intervention (Huang et al., 2005). As a consequence, we integrate the advantages of CNN and KELM to construct the unified defect predictor WSHCKE, which can boost the prediction performance.

Table 9
Running time for WSHCKE compared with nine baseline defect predictors (in seconds).

	WSNB	WSSVM	WSLR	WSDT	WSKNN	WSRF	WSDBN	WSCNN	WSDPDF	WSHCKE
PCA	0.26	0.37	0.25	0.26	0.25	0.28	20.62	26.29	1.72	31.72
FA	0.31	0.51	0.31	0.43	0.32	0.35	26.67	36.68	2.39	38.41
DM	0.35	0.45	0.33	0.49	0.36	0.38	28.98	32.11	2.42	40.97
SNE	0.29	0.45	0.26	0.31	0.25	0.28	18.83	25.35	1.65	32.52
LPP	0.32	0.44	0.30	0.33	0.33	0.36	23.53	33.42	1.92	36.21
NPE	0.31	0.43	0.32	0.34	0.32	0.34	17.98	25.15	1.62	30.95
LLC	0.25	0.36	0.24	0.26	0.26	0.28	16.09	26.07	1.52	33.54
MC	0.25	0.32	0.24	0.40	0.24	0.29	15.89	25.46	1.46	29.44
MCML	0.22	0.42	0.22	0.25	0.23	0.23	17.83	26.92	1.57	29.53
GS	0.29	0.38	0.29	0.36	0.28	0.33	24.43	33.96	1.95	36.91
BF	0.28	0.37	0.30	0.31	0.26	0.34	22.78	32.44	1.89	35.76
EMWS	0.32	0.47	0.33	0.39	0.30	0.36	32.98	41.39	2.53	45.45
Avg	0.29	0.42	0.28	0.35	0.29	0.32	22.22	29.88	1.89	34.87

Conclusion 2: The WSHCKE predictor outperforms nine classic baseline predictors. The WSHCKE achieves the average 25.58%, 35.03%, 18.28% and 17.56% performance improvements compared with nine baseline predictors with the same feature selection method EMWS across total 20 projects in terms of F1, MCC, G-measure and AUC, respectively.

RQ3: How efficient is our WSHCKE predictor compared to nine classic defect predictors?

Since the time cost is a key indicator for a defect predictor, this question is designed to explore the efficient of WSHCKE compared with nine baseline predictors, including WWSNB, WSSVM, WSLR, WSDT, WSKNN, WSRF, WSDBN, WSCNN and WSDPDF, so as to verify whether the time taken by our WSHCKE predictor is within an acceptable range.

Table 9 shows the running time for WSHCKE compared with nine baseline defect predictors based on each of the twelve feature selection or extraction approaches. From Table 9, we can observe that the WSHCKE predictor costs the most running time (34.87s) across total 20 projects. Compared to CNN, the KELM in WSHCKE also requires a little training time, so the running time of WSHCKE is a little longer than that of CNN. Compared with the other eight predictors, not only the KELM classifier needs a little training time, but also the determination of the network structure and parameters of CNN needs iterative training and a series of convolution operations take more time, so our WSHCKE predictor takes the most training time.

Although our WSHCKE predictor costs the most running time, we believe that this running time is still within an acceptable range and is applicable in practice, which is equivalent to sacrificing efficiency for effectiveness.

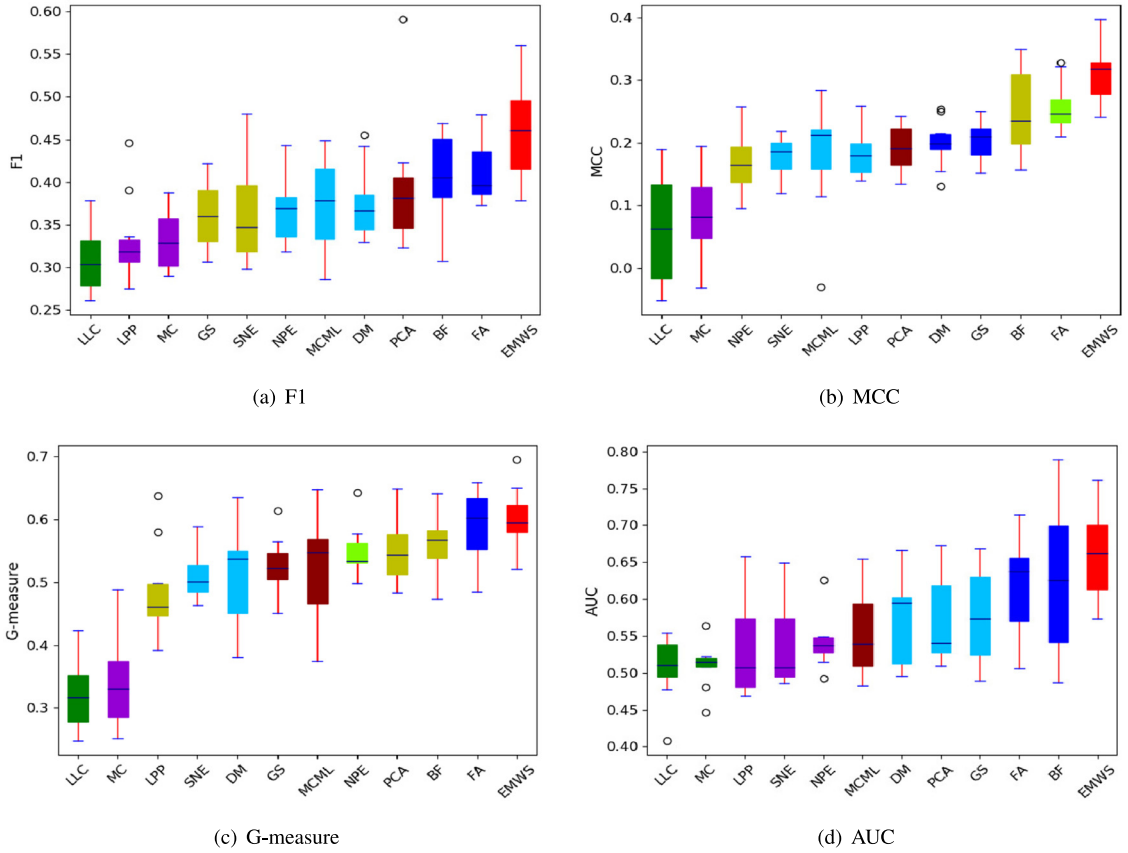
Conclusion 3: Compared with nine baseline defect predictors, the WSHCKE predictor costs the most running time across total 20 projects, but this running time is still within an acceptable range.

RQ4: What is the generalization capacity of our EMWS feature selection algorithm and WSHCKE predictor in software defect prediction?

In general, the stronger the generalization capacity, the better the prediction performance of the model. In this question, we investigate the generalization capacity of the EMWS feature selection algorithm and the WSHCKE predictor. In RQ1, we compare the EMWS algorithm with eleven state-of-the-art feature selection or extraction approaches using the same defect predictor WSHCKE. In RQ2, we compare the WSHCKE predictor with nine classic defect predictors using the same feature selection approach WSHCKE. Different from RQ1 and RQ2, we compare EMWS with eleven state-of-the-art feature selection or extraction approaches in baseline method (1) using the same defect predictor respectively (i.e., each of the nine classic defect predictors), and compare WSHCKE with nine classic defect predictors in baseline method (2) using the same feature selection or extraction approach respectively (i.e., each of eleven feature selection or extraction approaches). More specifically, we pair eleven baseline feature selection or extraction approaches with nine classic baseline predictors in this question, and then perform experiments across all 20 software projects from three datasets in terms of F1, MCC, G-measure and AUC.

Figs. 7, 8 visualize the boxplots with the Scott–Knott ESD ranking used to verify the generalization capacity of all twelve feature selection or extraction approaches and all ten defect predictors across total 20 projects in terms of four evaluation indicators, respectively. In Fig. 7, the *x*-axis represents twelve feature selection or extraction approaches; the *y*-axis represents different evaluation indicators. The blue line in each box denotes the median indicator value for each feature selection or extraction approach. From Fig. 7, we can find that the EMWS algorithm is in the highest rank (the red box), and the highest rank contains only EMWS, which demonstrates that the EMWS algorithm can achieve the optimal prediction performance in terms of four evaluation indicators compared with eleven state-of-the-art feature selection or extraction approaches. We also find that the median value gained by EMWS is higher than those gained by eleven baseline feature selection or extraction approaches in terms of F1, MCC and AUC respectively. However, this EMWS algorithm is not the best performance performer according to the median value on G-measure, and it is second only to FA. In Fig. 8, the *x*-axis denotes ten defect predictors; the *y*-axis denotes different evaluation indicators. The blue line in each box represents the median indicator value for each defect predictor. From Fig. 8, we can observe that the WSHCK predictor is in the top rank (the red box), and the top rank contains only WSHCK, which verifies that the WSHCK predictor can achieve the best prediction performance in terms of four evaluation indicators compared with nine baseline defect predictors. We also observe that the median value obtained by WSHCK is higher than those obtained by nine baseline defect predictors in terms of four evaluation indicators respectively.

Tables 10, 11 exhibit the p-values with the Benjamini–Hochberg correction and Cliff's deltas (δ) used to verify the generalization capacity of the EMWS algorithm and the WSHCKE predictor in terms of four evaluation indicators. In Table 10, the corrected p-values are less than 0.05 except for PCA (0.0593) on F1, FA (0.3863) on G-measure and BF (0.1141) on AUC, which indicates that the differences between EMWS and eleven baseline feature selection or extraction approaches are statistically significant. In addition, the δ is larger than 0.147 in terms of four evaluation indicators except for FA (0.04) on G-measure, and even all greater than 0.474 in terms of four evaluation indicators except for FA (0.34) and BF (0.24) on AUC, which indicates that the performance differences are not negligible and even large.



(a) F1 (b) MCC

(c) G-measure (d) AUC

Fig. 7. The Scott–Knott ESD ranking used to verify the generalization capacity of EMWS in terms of four evaluation indicators. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

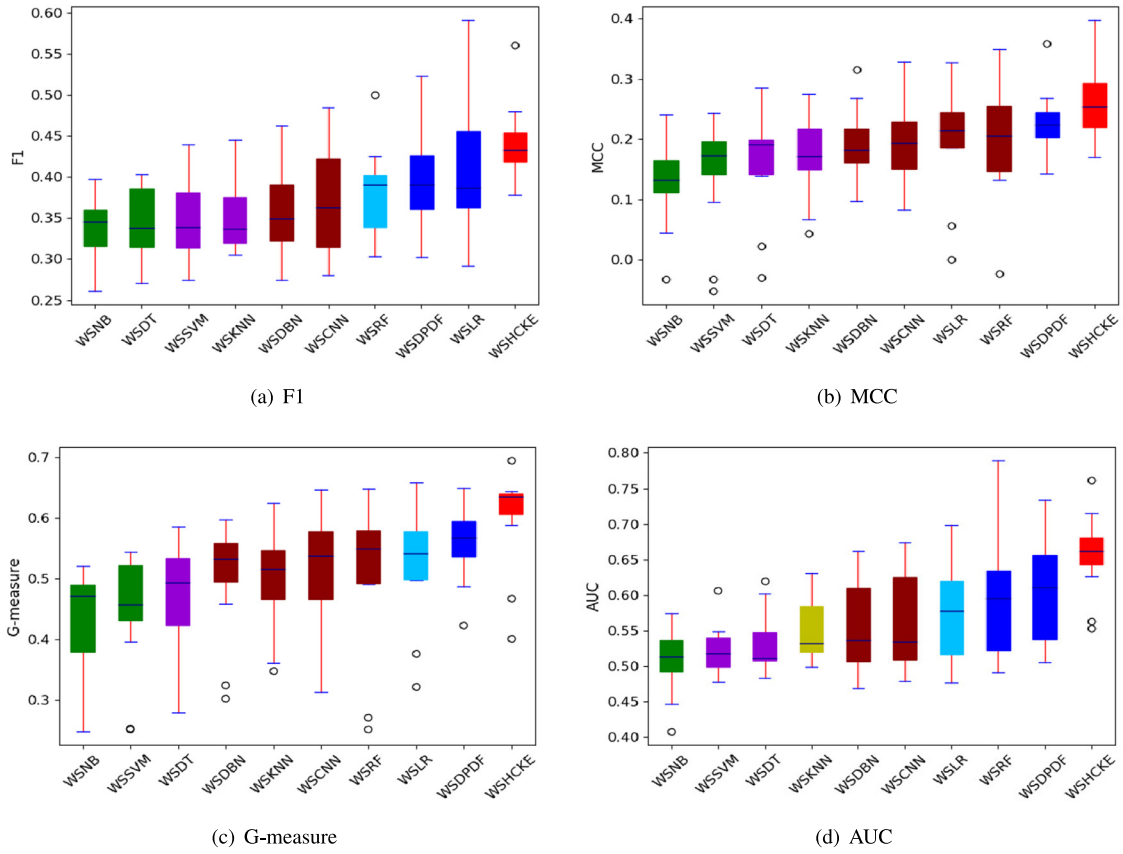
Table 10
P-value and Cliff's deltas (δ) used to verify the generalization capacity of EMWS in terms of four evaluation indicators.

Against	F1		MCC		G-measure		AUC	
	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)
PCA	0.0593	0.64(L)	0.0062	0.97(L)	0.0404	0.56(L)	0.0062	0.68(L)
FA	0.0312	0.59(L)	0.0088	0.56(L)	0.3863	0.04(N)	0.0062	0.34(M)
DM	0.0080	0.84(L)	0.0062	0.92(L)	0.0080	0.58(L)	0.0062	0.74(L)
SNE	0.0084	0.76(L)	0.0062	1(L)	0.0080	0.86(L)	0.0062	0.88(L)
LPP	0.0080	0.90(L)	0.0062	0.96(L)	0.0080	0.78(L)	0.0076	0.82(L)
NPE	0.0080	0.84(L)	0.0062	0.96(L)	0.0080	0.66(L)	0.0062	0.94(L)
LLC	0.0080	1(L)	0.0062	1(L)	0.0080	1(L)	0.0062	1(L)
MC	0.0080	0.98(L)	0.0062	1(L)	0.0080	1(L)	0.0062	1(L)
MCML	0.0080	0.72(L)	0.0062	0.94(L)	0.0153	0.52(L)	0.0062	0.80(L)
GS	0.0080	0.90(L)	0.0062	0.94(L)	0.0080	0.76(L)	0.0062	0.66(L)
BF	0.0084	0.52(L)	0.0093	0.48(L)	0.0095	0.48(L)	0.1141	0.24(S)

Table 11
P-value and Cliff's deltas (δ) used to verify the generalization capacity of WSHCKE in terms of four evaluation indicators.

Against	F1		MCC		G-measure		AUC	
	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)	p-value	δ(E)
WSNB	0.0028	0.96(L)	0.0040	0.88(L)	0.0037	0.82(L)	0.0028	0.92(L)
WSSVM	0.0028	0.85(L)	0.0040	0.79(L)	0.0037	0.81(L)	0.0028	0.97(L)
WSLR	0.1167	0.35(M)	0.0048	0.44(M)	0.0037	0.57(L)	0.0028	0.63(L)
WSDT	0.0028	0.90(L)	0.0048	0.71(L)	0.0037	0.76(L)	0.0028	0.93(L)
WSKNN	0.0028	0.86(L)	0.0040	0.61(L)	0.0037	0.69(L)	0.0028	0.88(L)
WSRF	0.0028	0.63(L)	0.0186	0.40(M)	0.0042	0.54(L)	0.0229	0.51(L)
WSDBN	0.0028	0.65(L)	0.0040	0.61(L)	0.0037	0.72(L)	0.0028	0.76(L)
WSCNN	0.0028	0.60(L)	0.0040	0.63(L)	0.0037	0.60(L)	0.0028	0.71(L)
WSDPDF	0.0068	0.50(L)	0.0068	0.33(M)	0.0047	0.50(L)	0.0053	0.42(M)

In Table 11, the corrected p-values are less than 0.05 except for WSLR (0.1167) on F1, which indicates that the differences between WSHCKE and nine baseline defect predictors are statistically significant. Moreover, the δ is larger than 0.147 in terms



(a) F1 (b) MCC

(c) G-measure (d) AUC

Fig. 8. The Scott–Knott ESD ranking used to verify the generalization capacity of WSHCKE in terms of four evaluation indicators. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of four evaluation indicators, and even all greater than 0.474 in terms of four evaluation indicators except for WSLR (0.35) on F1, WSLR (0.44), WSRF (0.40), WSDPDF (0.33) on MCC and WSDPDF (0.42) on AUC, which indicates that the performance differences are not negligible and even large. These observations are consistent with the results in Figs. 7, 8, which fully validates the superiority of the EMWS algorithm and the WSHCKE predictor.

Based on the above experimental results and analysis, we can conclude that our EMWS algorithm and WSHCKE predictor can achieve the optimal prediction performance compared with eleven state-of-the-art feature selection or extraction approaches and nine classic baseline defect predictors, which can indicate that the EMWS algorithm and the WSHCKE predictor have strong generalization capacity. The reasons for the superior performance of EMWS and WSHCKE have been explained in RQ1 and RQ2. In addition, our WSHCKE predictor employs the features after feature selection or extraction to predict whether the instance modules are defective, and these features may have strong robustness, which also enhances the generalization capacity of the WSHCKE predictor to a certain extent. Also, for our WSHCKE predictor, we adopt an early stop training iteration strategy before overfitting in a small number of software projects, which also boost the prediction performance of the model.

Conclusion 4: Our EMWS algorithm and WSHCKE predictor have strong generalization performance in terms of F1, MCC, G-measure and AUC compared with eleven state-of-the-art feature selection or extraction approaches and nine classic baseline defect predictors.

RQ5: Do the features selected by the EMWS algorithm have advantage in prediction performance compared with the original defect features without feature selection?

To answer this question, we also compare the prediction performance using EMWS with not using EMWS on ten defect predictors. Table 12 shows the performance comparison for different defect predictors with EMWS and without EMWS in terms of four evaluation indicators. Compared with ten defect predictors not using EMWS, these predictors using EMWS can achieve average improvements of 12.21%, 14.85%, 12.22% and 9.94% in terms of F1, MCC, G-measure and AUC, respectively. We utilize Wilcoxon signed-rank test and Cliff's Delta effect size to check whether these approaches are statistically significant. We can observe that the p-values are less than 0.05 in terms of four evaluation indicators, which indicates that the differences between ten defect predictors with EMWS and without EMWS are statistically significant in terms of four evaluation indicators. Moreover, the δ is larger than 0.147 in terms of four evaluation indicators, and even greater than 0.474 in terms of G-measure and AUC, which shows that the performance differences are not negligible and even large.

Therefore, the features selected by the EMWS algorithm can significantly boost the prediction performance of the models compared with the original defect features without feature selection, this is because the EMWS feature selection approach can search for the optimal feature subset combination by utilizing

Table 12
The performance comparison for different predictors with EMWS and without EMWS in terms of four evaluation indicators.

Predictors	F1 p = 0.0069, δ(E) = 0.44(M)		MCC p = 0.0125, δ(E) = 0.42(M)		G-measure p = 0.0051, δ(E) = 0.64(L)		AUC p = 0.0.0051, δ(E) = 0.56(L)	
	Without	With	Without	With	Without	With	Without	With
WSNB	0.3209	0.3788	0.1823	0.2415	0.4689	0.5214	0.5223	0.5738
WSSVM	0.3327	0.4057	0.2478	0.2430	0.4578	0.5449	0.5489	0.6063
WSLR	0.4678	0.4591	0.2834	0.3270	0.5635	0.6157	0.6289	0.6981
WSDT	0.3829	0.4015	0.2530	0.2848	0.5034	0.5856	0.5623	0.6019
WSKNN	0.3608	0.4450	0.2789	0.2754	0.5189	0.6244	0.5670	0.6306
WSRF	0.4532	0.4998	0.2790	0.3212	0.5523	0.5783	0.6440	0.7018
WSDBN	0.3899	0.4625	0.2673	0.3161	0.5301	0.5984	0.5998	0.6506
WSCNN	0.4389	0.4846	0.2609	0.3286	0.5470	0.5911	0.6156	0.6738
WSDPDF	0.4623	0.5236	0.3012	0.3587	0.5812	0.6500	0.6520	0.7276
WSHCKE	0.5090	0.5607	0.3406	0.3976	0.6278	0.6950	0.6864	0.7616
Avg	0.4118	0.4621	0.2694	0.3094	0.5351	0.6005	0.6027	0.6626

strong global search capability of WOA and strong local search capability of SA, which eliminates irrelevant and redundant features that seriously degrade the prediction performance.

> **Conclusion 5**: The features selected by the EMWS algorithm have advantage in prediction performance compared with the original defect features without feature selection.

RQ6: What is the impact of using different classifiers in the EMWS feature selection algorithm on prediction performance?

To investigate the feature selection capability of our EMWS algorithm more comprehensively and the impact of using different feature selection classifiers within EMWS on prediction performance, we replace LR classifier with NB and KNN classifiers in the EMWS feature selection algorithm, and conduct defect prediction using the same defect predictor WSHCKE after feature selection. Fig. 9 visualizes the impact comparison for three classifiers in the EMWS algorithm on prediction performance in terms four evaluation indicators. We can observe that the LR classifier can achieve the best prediction results in most cases, followed by KNN, and NB is the worst in most cases. We employ Wilcoxon signed-rank test with the Benjamini–Hochberg correction and Cliff's Delta effect size to measure whether these classifiers are statistically significant. We can observe that the corrected p-values are less than 0.05 compared with NB in terms of four evaluation indicators, which indicates that the differences between LR and NB are statistically significant. Compared with KNN, the corrected p-values are less than 0.05 in terms of F1 and MCC, which indicates that the differences between LR and KNN are statistically significant in terms of these two indicators. In terms of G-measure and AUC, the corrected p-values are more than 0.05 compared with KNN. In addition, compared with NB, the δ is larger than 0.147 in terms of four evaluation indicators, which shows that the performance differences are not negligible. Compared with KNN, the δ is larger than 0.147 in terms of F1, MCC and AUC, which shows that the performance differences are not negligible. But the δ is less than 0.147 in terms of G-measure.

> **Conclusion 6**: Compared with NB and KNN, the LR classifier can achieve the best prediction performance in most cases.

8. Threats to validity

In this section, we discuss three kinds of validity threats that may affect our experimental results.

8.1. Internal validity

Internal validity is mainly concerned with uncontrolled aspects that may threaten our experimental results, such as errors in the experiment. We examine our experiment process carefully, but there may still be errors that we have not noticed.

8.2. External validity

External validity involves the quality and universality of the datasets. In this paper, we conduct extensive experiments on 20 open source software projects from three datasets, including 12 projects from PROMISE, 3 projects from ReLink and 5 projects from AEEEM, which are publicly available and commonly used benchmark datasets in software defect prediction studies (Kondo et al., 2019; Lu et al., 2014; Li et al., 2019a; Yan et al., 2017; Chen and Ma, 2015; Nam et al., 2018). In addition, these software projects belong to different application fields, cover a long time and are written with different programming languages. Therefore, we believe that these datasets used in the paper are large enough and have a certain universality.

8.3. Construct validity

Construct validity is related to the applicability of our evaluation indicators. In this paper, we use four evaluation indicators, namely F1, MCC, G-measure and AUC, which have been widely used in previous software defect prediction studies (Ma et al., 2012; Wang et al., 2016b; Li et al., 2012; Zhu et al., 2020; Zhou et al., 2019; Xu et al., 2019), so we believe that the construct validity should be acceptable. In addition, the parameter settings may also affect our experimental results. Recent study (Tantithamthavorn et al., 2019) has pointed out that defect prediction models with different parameter settings may produce different results. In order to reduce the threat of parameter settings, we plan to use the most advanced automated parameter optimization techniques in more experiments.

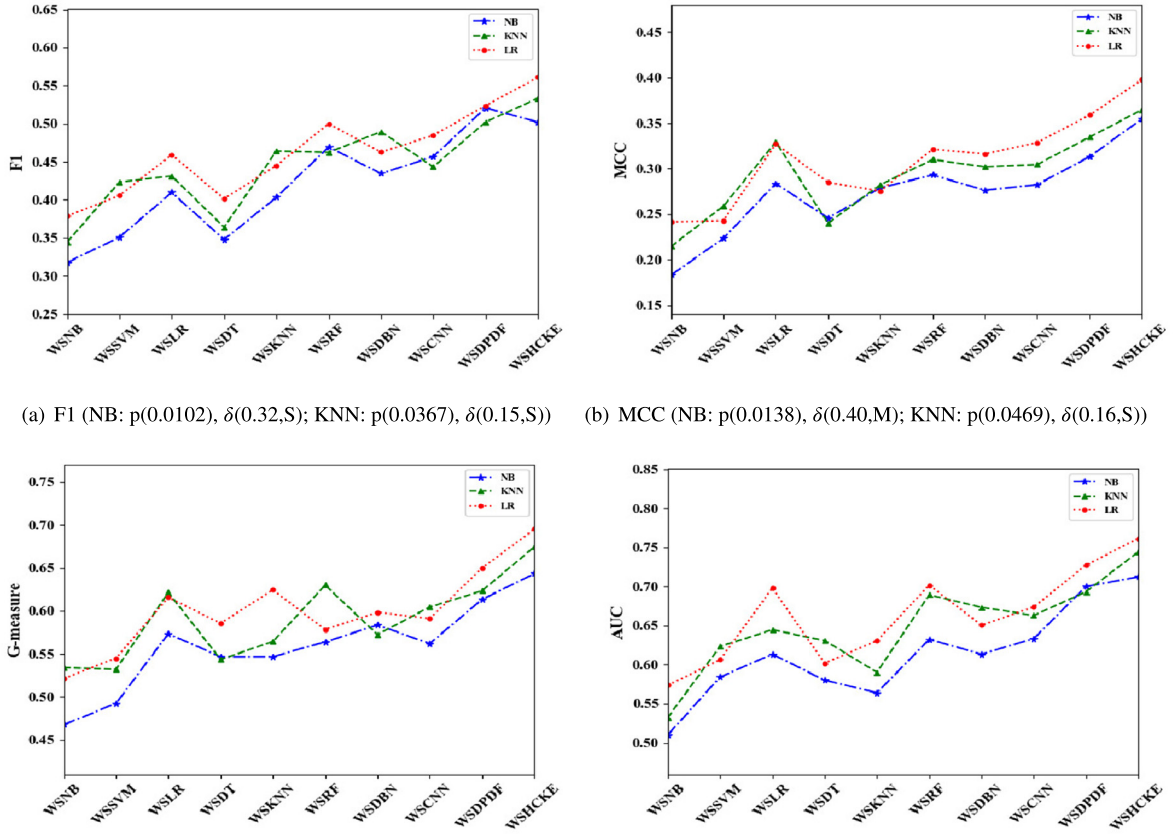
9. Related work

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

9.1. Software defect prediction

Existing software defect prediction methods can be roughly divided into the following two aspects.

Some researchers focused on how to leverage machine learning algorithms to construct effective defect prediction models (Mori and Uchihira, 2019; Li et al., 2012). Chen et al. (2018) proposed a succinct but effective defect prediction learner called FFT (Fast-and-Frugal Trees). The experimental results showed



(a) F1 (NB: p(0.0102), δ(0.32,S); KNN: p(0.0367), δ(0.15,S))

(b) MCC (NB: p(0.0138), δ(0.40,M); KNN: p(0.0469), δ(0.16,S))

(c) G-measure (NB: p(0.0102), δ(0.46,M); KNN: p(0.2411), δ(0.12,N))

(d) AUC (NB: p(0.0102), δ(0.38,M); KNN: p(0.1849), δ(0.16,S))

Fig. 9. The impact comparison for different classifiers in the EMWS algorithm on prediction performance in terms of four evaluation indicators.

that FFT can outperform other software analytics tools (e.g., Expectation Maximization (EM), Simple Logistic (SL)). Elish and Elish (2008) used support vector machine (SVM) to construct the defect prediction model and compared it with eight statistical and machine learning models on four NASA projects. Mori and Uchihira (2019) proposed a new classification model named superposed Naive Bayes (SNB), which transforms a Naive Bayes ensemble into a simple Naive Bayes model through linear approximation. Lu et al. (2014) leveraged active learning to conduct defect prediction model, and it can significantly improve the prediction performance. Li et al. (2012) proposed a novel semi-supervised learning method called ACoForest, which can sample the prediction modules that are most helpful for the model training. Li et al. (2019a) proposed a novel two-stage ensemble learning (TSEL) approach for defect prediction, which contains two stages: ensemble multi-kernel domain adaptation stage and ensemble data sampling stage. Herbold et al. (2018) replicated 24 cross-project defect prediction methods proposed by researchers between 2008 and 2015 and evaluated their performance on software projects from five different data groups. Abaei et al. (2013) proposed the self-organizing mapping (SOM) prediction model with the threshold, which can help testers to mark modules without experts. Wang et al. (2016b) proposed a SemiBoost defect prediction method named NSSB based on non-negative sparse graphs, and used the adaboost algorithm to improve the performance of the model. The experimental results showed that the NSSB method can effectively solve the issues of class imbalance and label instances insufficiency. There are few researches on software defect prediction using deep learning techniques (e.g., DBN, CNN, deep forest). Yang et al. (2015a) leveraged deep belief network (DBN) to construct a group of expressive features from a group of initial change features. The experimental results showed that the DBN can significantly improve the prediction performance of Just-In-Time defect prediction compared to two baseline models in six open source projects. Hoang et al. (2019) proposed an end-to-end Just-In-Time defect prediction framework called DeepJIT, which can automatically extract feature representation from code changes and commit messages by convolutional neural network (CNN). Experimental results showed that the variant of DeepJIT — DeepJIT-Combined can achieve the best prediction performance. Zhou et al. (2019) constructed a novel defect prediction model called DPDF by deep forest, which can identify more important features by utilizing a new cascade strategy and convert random forest classifiers into a layer-by-layer structure. Experimental results proved the effectiveness of their model. Note that in this paper, we adopt DBN, CNN and DPDF as baseline defect predictors and compare them with our WSHCKE predictor.

Different from the above methods, some researchers paid attention to data preprocessing (Seiffert et al., 2014; Li et al., 2012). Rodríguez et al. (2014) compared different methods for different data preprocessing problems, such as sampling method, cost sensitive method and integration method. The final experimental results showed that the above methods can significantly improve the prediction performance. Li et al. (2012) proposed a two-stage semi-supervised integration learning method. The results showed that the method has better prediction capability for unbalanced dataset compare with the classical machine learning methods. Seiffert et al. (2014) analyzed eleven different algorithms and seven different data sampling techniques, and found that class imbalance and data noise have the negative impact on the prediction performance.

For the two aspects mentioned above, especially the first aspect, the prediction performance is still not superior enough. Different from the previous studies that mainly used machine learning methods, we try to employ more advanced the deep learning techniques to further boost the prediction performance in this paper. Moreover, Yu et al. (2015) conducted image classification for the digestive organs by combining CNN with extreme learning machine (ELM). Unlike their research, we combine CNN with kernel extreme learning machine (KELM) to conduct our software defect prediction task, and adjust the network structure and parameters to make them suitable for our software defect datasets.

9.2. Feature selection and extraction methods in software defect prediction

Recently, feature selection or extraction techniques have been introduced into the field of software defect prediction, which can be used for eliminating irrelevant and redundant features Kondo et al. (2019), Xu et al. (2016a). Feature selection methods decrease the number of features by selecting an optimal representative feature subset, while feature extraction methods reduce the number of features by building new, combined features from the original features.

Feature selection methods are mainly divided into two types: filter and wrapper. The wrapper-based methods must contain a learning algorithm (e.g., classification algorithm) in the search process. For the filter-based methods, the search of feature space depends on the feature itself rather than a specific learning algorithm. The computation speed of the filter-based method is faster, but the performance cannot be guaranteed, and the case of the filter-based method is the opposite (Liu and Motoda, 1998). At present, most prior studies mainly utilized feature selection methods to conduct defect prediction, feature extraction methods have not been thoroughly investigated in software defect prediction.

Most previous studies used filter-based methods for feature selection in software defect prediction (Wang et al., 2011, 2010; Khoshgoftaar et al., 2012). Wang et al. (2011) combined multiple feature selection techniques to select the representative feature subset using ensemble learning. Liu et al. (2014) proposed three new cost-sensitive based feature selection methods — Cost-Sensitive Variance Score (CSVS), Cost-Sensitive Laplacian Score (CSLS), and Cost-Sensitive Constraint Score (CSCS), which can incorporate cost information into traditional feature selection methods. The experimental results showed that these methods can outperform existing single-stage cost-sensitive classifiers and traditional cost-blind feature selection methods. Wang et al. (2010) conducted a comprehensive empirical study that surveys seventeen different feature ranking methods which contain commonly-used feature ranking methods, the signal-to-noise filter method and threshold-based feature ranking methods. The experimental results showed that ensembles of very few rankers are very effective and even outperform ensembles of multiple rankers. Khoshgoftaar et al. (2012) compared seven filter-based feature ranking techniques (e.g., information gain (IG), gain ratio (GR)) on sixteen defect projects. Yu et al. (2017) proposed a feature selection method based on feature spectral clustering and feature ranking (FSCR) to predict the number of software defects. Ghotra et al. (2017) investigated 30 feature selection methods, including 6 filter-based subset methods, 11 filter-based ranking methods, 12 wrapper-based subset methods, and a no feature selection method and 21 classification methods on PROMISE and NASA datasets. The experimental results verified that a correlation-based feature selection method — BestFirst search performs better than other feature selection methods.

There are few studies on applying wrapper-based feature selection techniques to software defect prediction. Xu et al. (2016a) researched the impact of 32 feature selection techniques (including four wrapper-based techniques) on software defect prediction, and the experimental results showed that these feature selection techniques have significant performance differences on each software project. Kasinathan et al. (2015) surveyed two wrapper-based feature selection methods, seven feature ranking methods and an embedded selection method on multiple software projects.

For feature extraction methods, most prior studies adopted principal component analysis (PCA) D'Ambros et al. (2010), Rathore and Gupta (2014) and kernel principal component analysis (KPCA) Xu et al. (2016a) for feature extraction in software defect prediction. D'Ambros et al. (2010) utilized PCA to conduct class-level defect prediction, which can void the issue of multicollinearity among the independent variables. Rathore and Gupta (2014) compared PCA with some feature selection methods. The experimental results demonstrated that the PCA is one of the best-performing methods. Xu et al. (2016a) leveraged KPCA to project the original data into a latent feature space by nonlinear mapping, and the experimental results verified that the effectiveness of KPCA.

Different from previous feature selection studies in software defect prediction, we leverage the recently proposed whale optimization algorithm (WOA) and another complementary simulated annealing (SA) to construct an enhanced metaheuristic feature selection algorithm instead of a single algorithm. As far as we know, this is the first attempt to apply a hybrid algorithm that contains exploration oriented algorithm based on population (WOA) and exploitation oriented algorithm based on single-solution (SA) to feature selection in software defect prediction. In addition, unlike two hybrid models (Low-Level Teamwork Hybrid (LTH) and High-Level Relay Hybrid (HRH)) using WOA and SA proposed by Mafarja and Mirjalili (2017), we adopt roulette wheel selection as the selection mechanism in the exploration stage and Logistic Regression algorithm as the classifier of feature selection in this paper, and Mafarja and Mirjalili (2017) do not apply their models to software defect prediction (just used for intelligent algorithm optimization).

9.3. The application of deep learning techniques in software engineering

In recent years, some researchers have already applied deep learning techniques (e.g., CNN and DBN) to improve tasks in software engineering. Yang et al. (2015a) combined fourteen change-level features to generate new features through deep belief network (DBN) for software defect prediction. Zhou et al. (2019) used the deep forest classifier to construct the defect prediction model. This model can detect more important defect features through a new cascade method, which transforms random forest classifiers into a layer-by-layer structure. Ferrari et al. (2018) verified that which extent NLP can be practically applied to detect defects in the requirement documents of the railway signaling manufacturer. Lam et al. (2015) combined deep learning with information retrieval, thereby enhancing the performance of defect location. Huo et al. (2016) used CNN to learn the features from the source code and the text in the defect report, and then combined two kinds of features as a kind of unified feature for defect location. Hoang et al. (2019) proposed an end-to-end Just-In-Time defect prediction framework called DeepJIT, which can automatically extract feature representation from code changes and commit messages by convolutional neural network (CNN). Experimental results showed that the variant of DeepJIT — DeepJIT-Combined can achieve the best prediction performance. Ha and

Zhang (2019) utilized deep feedforward neural network (FNN) and the L1 regularization technique to predict the performance for highly configurable software systems. Experimental results verified the superiority of their approach on eleven public available datasets. LeClair et al. (2019) proposed a neural model to generate coherent summaries in many cases by combining code structure from the abstract syntax tree (AST) with words from code, and the experimental results showed that this model can outperform three state-of-the-art baseline models from software engineering (SE) literature and natural language processing (NLP) literature.

The deep learning techniques have also been used for test report classification (Wang et al., 2017), link prediction in developer online forums (Xu et al., 2016b), software traceability (Guo et al., 2017a) and so on.

10. Conclusion and future work

In this work, we construct an enhanced metaheuristic feature selection algorithm named EMWS that effectively selects fewer but closely related representative features for each software project, which can take full advantage of the strong local search capability of SA to enhance the weak exploitation performance of WOA, and leverage strong global search capability of WOA to boost the weak exploration of SA simultaneously. We also leverage a hybrid deep neural network — CNN and KELM to construct a unified defect predictor called WSHCKE according to the defect features selected by the EMWS algorithm, which can further integrate the defect features into more robust deep semantic features by the CNN and boost the prediction performance by utilizing the strong classification capacity of the KELM classifier. We conduct extensive experiments for feature selection and extraction and defect prediction on 20 widely-studied software projects with four evaluation indicators. By comparing the EMWS algorithm with eleven state-of-the-art feature selection or extraction methods and comparing the WSHCKE model with nine classic defect predictors, the experimental results verify that the EMWS and the WSHCKE can outperform these baseline methods basically.

In future work, we plan to evaluate our models in more open source and commercial projects, and use automated parameter optimization techniques to adjust the parameter settings. In addition, we plan to extend our models to multi-source cross-project or cross-version defect prediction and effort-aware defect prediction.

CRediT authorship contribution statement

Kun Zhu: Conceptualization, Software, Methodology, Experiment, Writing. **Shi Ying:** Supervision, Methodology. **Nana Zhang:** Data curation, Writing - original draft, Methodology, Experiment. **Dandan Zhu:** Experiment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by the National Science Foundation of China (Grant Nos. 62072342 and 61672392), and in part by the National Key Research and Development Program of China (Grant No. 2016YFC1202204).

References

Abaei, G., Rezaei, Z., Selamat, A., 2013. Fault prediction by utilizing self-organizing Map and Threshold. In: 2013 IEEE International Conference on Control System, Computing and Engineering. ICCSCE 2013, Penang, Malaysia, November 29–Dec. 1, 2013. pp. 465–470.

Agrawal, A., Menzies, T., 2018. Is "better data" better than "better data miners"?: on the benefits of tuning SMOTE for defect prediction. In: Proceedings of the 40th International Conference on Software Engineering. ICSE 2018, Gothenburg, Sweden, May 27–June 03, 2018. pp. 1050–1061.

Ali, M.U., Ahmed, S., Ferzund, J., Mehmood, A., Rehman, A., 2017. Using PCA and factor analysis for dimensionality reduction of bio-informatics data. CoRR abs/1707.07189.

Anvarjon, T., Mustaqeem, Kwon, S., 2020. Deep-net: A lightweight CNN-based speech emotion recognition system using deep frequency features. Sensors 20 (18), 5212.

Bellman, R., 1957. Dynamic Programming. Princeton University Press, Princeton.

Benjamini, Y., Hochberg, Y., 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. R. Stat. Soc. 57 (1), 289–300.

Bunte, K., Haase, S., Biehl, M., Villmann, T., 2012. Stochastic neighbor embedding (SNE) for dimension reduction and visualization using arbitrary divergences. Neurocomputing 90, 23–45.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. J. Artificial Intelligence Res. 16, 321–357.

Chen, D., Fu, W., Krishna, R., Menzies, T., 2018. Applications of psychological science for actionable analytics. In: Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018. pp. 456–467.

Chen, M., Ma, Y., 2015. An empirical study on predicting defect numbers. In: The 27th International Conference on Software Engineering and Knowledge Engineering. SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, pp. 397–402.

Chen, Y., Zhao, X., de Rijke, M., 2017. Top-N recommendation with high-dimensional side information via locality preserving projection. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. Shinjuku, Tokyo, Japan, August 7–11, 2017. pp. 985–988.

D'Ambros, M., Lanza, M., Robbes, R., 2010. An extensive comparison of bug prediction approaches. In: Proceedings of the 7th International Working Conference on Mining Software Repositories. MSR 2010 (Co-Located with ICSE), Cape Town, South Africa, May 2–3, 2010, Proceedings. pp. 31–41.

D'Ambros, M., Lanza, M., Robbes, R., 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empir. Softw. Eng. 17 (4–5), 531–577.

Duan, M., Li, K., Yang, C., Li, K., 2018. A hybrid deep learning CNN-ELM for age and gender classification. Neurocomputing 275, 448–461.

Elish, K.O., Elish, M.O., 2008. Predicting defect-prone software modules using support vector machines. J. Syst. Softw. 81 (5), 649–660.

Emary, E., Zawbaa, H.M., Hassanien, A.E., 2016. Binary ant lion approaches for feature selection. Neurocomputing 213, 54–65.

Fan, G., Diao, X., Yu, H., Yang, K., Chen, L., 2019. Deep semantic feature learning with embedded static metrics for software defect prediction. In: 26th Asia-Pacific Software Engineering Conference. APSEC 2019, Putrajaya, Malaysia, December 2–5, 2019. pp. 244–251.

FARRAR, D.A., Glauber, R.R., 1967. Multicollinearity in regression analysis:The problem revisited. Rev. Econ. Stat. 49 (1), 92–107.

Ferrari, A., Gori, G., Rosadini, B., Trotta, I., Bacherini, S., Fantechi, A., Gnesi, S., 2018. Detecting requirements defects with NLP patterns: an industrial experience in the railway domain. Empir. Softw. Eng. 23 (6), 3684–3733.

Ghotra, B., McIntosh, S., Hassan, A.E., 2017. A large-scale study of the impact of feature selection techniques on defect classification models. In: Proceedings of the 14th International Conference on Mining Software Repositories. MSR 2017, Buenos Aires, Argentina, May 20–28, 2017. pp. 146–157.

Globerson, A., Roweis, S.T., 2005. Metric learning by collapsing classes. In: Advances in Neural Information Processing Systems 18. Neural Information Processing Systems, NIPS 2005, December 5-8 2005, Vancouver, British Columbia, Canada, pp. 451–458.

Goldberg, D.E., 1989. Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley.

Guo, J., Cheng, J., Cleland-Huang, J., 2017. Semantically enhanced software traceability using deep learning techniques. In: Proceedings of the 39th International Conference on Software Engineering. ICSE 2017, Buenos Aires, Argentina, May 20–28, 2017. pp. 3–14.

Guo, J., Yuan, Y., Zhang, C., 2017. Joint supervision for discriminative feature learning in convolutional neural networks. In: Computer Vision–Second CCF Chinese Conference, CCCV 2017, Tianjin, China, October 11–14, 2017, Proceedings, Part II. pp. 509–520.

Ha, H., Zhang, H., 2019. DeepPerf: performance prediction for configurable software with deep sparse neural network. In: Proceedings of the 41st International Conference on Software Engineering. ICSE 2019, Montreal, QC, Canada, May 25–31, 2019. pp. 1095–1106.
Hamdani, T.M., Won, J., Alimi, A.M., Karray, F., 2007. Multi-objective Feature Selection with NSGA II. In: Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 11–14, 2007, Proceedings, Part I. pp. 240–247.
Herbold, S., Trautsch, A., Grabowski, J., 2018. A comparative study to benchmark cross-project defect prediction approaches. IEEE Trans. Softw. Eng. 44 (9), 811–833.
Hoang, T., Dam, H.K., Kamei, Y., Lo, D., Ubayashi, N., 2019. DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction. In: Proceedings of the 16th International Conference on Mining Software Repositories. MSR 2019, 26–27 May 2019, Montreal, Canada. pp. 34–45.
Hosseini, S., Turhan, B., Gunarathna, D., 2019. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. Softw. Eng. 45 (2), 111–147.
Hsu, C., Lin, C., 2002. A comparison of methods for multiclass support vector machines. IEEE Trans. Neural Netw. 13 (2), 415–425.
Huang, G., Huang, G., Song, S., You, K., 2015. Trends in extreme learning machines: A review. Neural Netw. 61, 32–48.
Huang, Q., Wang, H., Xu, Q., Bi, W., 2009. Semi-supervised Learning with Locally Linear Coordination for Face Recognition. In: Fifth International Conference on Natural Computation. ICNC 2009, Tianjian, China, 14–16 August 2009, 6 Volumes. pp. 255–259.
Huang, G., Zhou, H., Ding, X., Zhang, R., 2012. Extreme learning machine for regression and multiclass classification. IEEE Trans. Syst. Man Cybern. B 42 (2), 513–529.
Huang, G., Zhu, Q., Siew, C.K., 2005. Extreme learning machine: a new learning scheme of feedforward neural networks. In: IEEE International Joint Conference on Neural Networks.
Huang, G., Zhu, Q., Siew, C.K., 2006. Extreme learning machine: Theory and applications. Neurocomputing 70 (1–3), 489–501.
Huo, X., Li, M., Zhou, Z., 2016. Learning unified features from natural and programming languages for locating buggy source code. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. IJCAI 2016, New York, NY, USA, 9–15 July 2016. pp. 1606–1612.
Jensen, R., Shen, Q., 2004. Semantics-preserving dimensionality reduction: Rough and fuzzy-rough-based approaches. IEEE Trans. Knowl. Data Eng. 16 (12), 1457–1471.
Jiarpakdee, J., Tantithamthavorn, C., Ihara, A., Matsumoto, K., 2016. A study of redundant metrics in defect prediction datasets. In: 2016 IEEE International Symposium on Software Reliability Engineering Workshops. ISSRE Workshops 2016, Ottawa, on, Canada, October 23–27, 2016. pp. 51–52.
Jureczko, M., Madeyski, L., 2010. Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering. PROMISE 2010, Timisoara, Romania, September 12–13, 2010. p. 9.
Kashef, S., Nezamabadi-pour, H., 2015. An advanced ACO algorithm for feature subset selection. Neurocomputing 147, 271–279.
Kasinathan, M., Rallapalli, A., Neti, L.B.M., 2015. Impact of feature selection techniques on bug prediction models. In: Proceedings of the 8th India Software Engineering Conference. ISEC 2015, Bangalore, India, February 18–20, 2015. pp. 120–129.
Khoshgoftaar, T.M., Gao, K., Napolitano, A., 2012. An empirical study of feature ranking techniques for software quality prediction. Int. J. Softw. Eng. Knowl. Eng. 22 (2), 161–183.
Kim, J., Kim, J., Jang, G., Lee, M., 2017. Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection. Neural Netw. 87, 109–121.
Kondo, M., Bezemer, C., Kamei, Y., Hassan, A.E., Mizuno, O., 2019. The impact of feature reduction techniques on defect prediction models. Empir. Softw. Eng. 24 (4), 1925–1963.
Lam, A.N., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N., 2015. Combining deep learning with information retrieval to localize buggy files for bug reports (N). In: 30th IEEE/ACM International Conference on Automated Software Engineering. ASE 2015, Lincoln, NE, USA, November 9–13, 2015. pp. 476–481.
LeClair, A., Jiang, S., McMillan, C., 2019. A neural model for generating natural language summaries of program subroutines. In: Proceedings of the 41st International Conference on Software Engineering. ICSE 2019, Montreal, QC, Canada, May 25–31, 2019. pp. 795–806.
Li, Z., Jing, X., Zhu, X., Zhang, H., Xu, B., Ying, S., 2019a. Heterogeneous defect prediction with two-stage ensemble learning. Autom. Softw. Eng. 26 (3), 599–651.
Li, Z., Jing, X., Zhu, X., Zhang, H., Xu, B., Ying, S., 2019b. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. IEEE Trans. Softw. Eng. 45 (4), 391–411.
Li, M., Zhang, H., Wu, R., Zhou, Z., 2012. Sample-based software defect prediction with active and semi-supervised learning. Autom. Softw. Eng. 19 (2), 201–230.
Liu, M., Miao, L., Zhang, D., 2014. Two-stage cost-sensitive learning for software defect prediction. IEEE Trans. Reliab. 63 (2), 676–686.
Liu, H., Motoda, H., 1998. Feature Selection for Knowledge Discovery and Data Mining. In: The Springer International Series in Engineering and Computer Science, vol. 454, Kluwer.
Lu, H., Kocaguneli, E., Cukic, B., 2014. Defect prediction between software versions with active learning and dimensionality reduction. In: 25th IEEE International Symposium on Software Reliability Engineering. ISSRE 2014, Naples, Italy, November 3–6, 2014. pp. 312–322.
Ma, W., Chen, L., Yang, Y., Zhou, Y., Xu, B., 2016. Empirical analysis of network measures for effort-aware fault-proneness prediction. Inf. Softw. Technol. 69, 50–70.
Ma, Y., Luo, G., Zeng, X., Chen, A., 2012. Transfer learning for cross-company software defect prediction. Inf. Softw. Technol. 54 (3), 248–256.
Mafarja, M.M., Mirjalili, S., 2017. Hybrid whale optimization algorithm with simulated annealing for feature selection. Neurocomputing 260, 302–312.
Mirjalili, S., Lewis, A., 2016. The whale optimization algorithm. Adv. Eng. Softw. 95, 51–67.
Mori, T., Uchihira, N., 2019. Balancing the trade-off between accuracy and interpretability in software defect prediction. Empir. Softw. Eng. 24 (2), 779–825.
Nam, J., Fu, W., Kim, S., Menzies, T., Tan, L., 2018. Heterogeneous defect prediction. IEEE Trans. Softw. Eng. 44 (9), 874–896.
Ni, C., Liu, W., Gu, Q., Chen, X., Chen, D., 2017. FeSCH: A feature selection method using clusters of hybrid-data for cross-project defect prediction. In: 41st IEEE Annual Computer Software and Applications Conference, Vol. 1. COMPSAC 2017, Turin, Italy, July 4–8, 2017. pp. 51–56.
Pascual, Ángela.F., González, A.M., García, J.D., Dorronsoro, J.R., 2015. Diffusion maps for dimensionality reduction and visualization of meteorological data. Neurocomputing 163, 25–37.
Rathore, S.S., Gupta, A., 2014. A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction. In: 7th India Software Engineering Conference, Chennai. ISEC '14, Chennai, India–February 19–21, 2014. pp. 7:1–7:10.
Razavian, A.S., Azizpour, H., Sullivan, J., Carlsson, S., 2014. CNN features off-the-shelf: An astounding baseline for recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. CVPR Workshops 2014, Columbus, OH, USA, June 23–28, 2014. pp. 512–519.
Ren, Y., Zhao, P., Sheng, Y., Yao, D., Xu, Z., 2017. Robust softmax regression for multi-class classification with self-paced learning. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. IJCAI 2017, Melbourne, Australia, August 19–25, 2017. pp. 2641–2647.
Rodríguez, D., Herraiz, I., Harrison, R., Dolado, J.J., Riquelme, J.C., 2014. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In: 18th International Conference on Evaluation and Assessment in Software Engineering. EASE '14, London, England, United Kingdom, May 13–14, 2014. pp. 43:1–43:10.
Saini, S., Rambli, D.R.A., Sulaiman, S.B., Zakaria, M.N.B., 2013. Human pose tracking in low-dimensional subspace using manifold learning by charting. In: 2013 IEEE International Conference on Signal and Image Processing Applications. Melaka, Malaysia, October 8–10, 2013. pp. 258–263.
Seiffert, C., Khoshgoftaar, T.M., Hulse, J.V., Folleco, A., 2014. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. Inform. Sci. 259, 571–595.
Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G., 2014. Learning semantic representations using convolutional neural networks for web search. In: 23rd International World Wide Web Conference. WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume. pp. 373–374.
Storn, R., Price, K.V., 1997. Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. 11 (4), 341–359.
Talbi, E., 2009. Metaheuristics–from Design To Implementation. Wiley.
Tantithamthavorn, C., Hassan, A.E., Matsumoto, K., 2020. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Trans. Softw. Eng. 46 (11), 1200–1219.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. 43 (1), 1–18.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2019. The impact of automated parameter optimization on defect prediction models. IEEE Trans. Softw. Eng. 45 (7), 683–711.
Turabieh, H., Mafarja, M.M., Li, X., 2019. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. Expert Syst. Appl. 122, 27–42.
Wang, J., Cui, Q., Wang, S., Wang, Q., 2017. Domain adaptation for test report classification in crowdsourced testing. In: 39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track. ICSE-SEIP 2017, Buenos Aires, Argentina, May 20–28, 2017. pp. 83–92.
Wang, H., Khoshgoftaar, T.M., Hulse, J.V., Gao, K., 2011. Metric selection for software defect prediction. Int. J. Softw. Eng. Knowl. Eng. 21 (2), 237–257.

Wang, H., Khoshgoftaar, T.M., Napolitano, A., 2010. A comparative study of ensemble feature selection techniques for software defect prediction. In: The Ninth International Conference on Machine Learning and Applications. ICMLA 2010, Washington, DC, USA, 12–14 December 2010. pp. 135–140.

Wang, S., Liu, T., Tan, L., 2016a. Automatically learning semantic features for defect prediction. In: Proceedings of the 38th International Conference on Software Engineering. ICSE 2016, Austin, TX, USA, May 14-22, 2016. pp. 297–308.

Wang, T., Zhang, Z., Jing, X., Liu, Y., 2016b. Non-negative sparse-based semiboost for software defect prediction. Softw. Test. Verif. Reliab. 26 (7), 498–515.

van de Wolfshaar, J., Karaaba, M.F., Wiering, M.A., 2015. Deep convolutional neural networks and support vector machines for gender recognition. In: IEEE Symposium Series on Computational Intelligence. SSCI 2015, Cape Town, South Africa, December 7–10, 2015. pp. 188–195.

Wu, R., Zhang, H., Kim, S., Cheung, S., 2011. ReLink: recovering links between bugs and changes. In: SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5–9, 2011. pp. 15–25.

Xu, Z., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., Tang, Y., Zhang, T., 2019. Software defect prediction based on kernel PCA and weighted extreme learning machine. Inf. Softw. Technol. 106, 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. In: 27th IEEE International Symposium on Software Reliability Engineering. ISSRE 2016, Ottawa, on, Canada, October 23–27, 2016. pp. 309–320.

Xu, B., Ye, D., Xing, Z., Xia, X., Chen, G., Li, S., 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ASE 2016, Singapore, September 3–7, 2016. pp. 51–62.

Xue, B., Zhang, M., Browne, W.N., 2013. Particle swarm optimization for feature selection in classification: A multi-objective approach. IEEE Trans. Cybern. 43 (6), 1656–1671.

Yan, M., Fang, Y., Lo, D., Xia, X., Zhang, X., 2017. File-level defect prediction: Unsupervised vs. Supervised models. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM 2017, Toronto, on, Canada, November 9–10, 2017. pp. 344–353.

Yang, X., Lo, D., Xia, X., Sun, J., 2017. TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. Inf. Softw. Technol. 87, 206–220.

Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J., 2015a. Deep learning for just-in-time defect prediction. In: 2015 IEEE International Conference on Software Quality, Reliability and Security. QRS 2015, Vancouver, BC, Canada, August 3–5, 2015. pp. 17–26.

Yang, Y., Wu, Q.M.J., Wang, Y., Zeeshan, K.M., Lin, X., Yuan, X., 2015b. Data partition learning with multiple extreme learning machines. IEEE Trans. Cybern. 45 (8), 1463–1475.

Yu, J., Chen, J., Xiang, Z.Q., Zou, Y., 2015. A hybrid convolutional neural networks with extreme learning machine for WCE image classification. In: 2015 IEEE International Conference on Robotics and Biomimetics. ROBIO 2015, Zhuhai, China, December 6–9, 2015. pp. 1822–1827.

Yu, X., Ma, Z., Ma, C., Gu, Y., Liu, R., Zhang, Y., 2017. FSCR: a feature selection method for software defect prediction. In: The 29th International Conference on Software Engineering and Knowledge Engineering. Pittsburgh, PA, USA, July 5-7, 2017, Wyndham Pittsburgh University Center, pp. 351–356.

Zhao, L., Zou, D., Gao, G., 2013. Subsampling based neighborhood preserving embedding for image classification. In: Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing. IIH-MSP 2013, Beijing, China, October 16-18, 2013. pp. 358–360.

Zhou, T., Sun, X., Xia, X., Li, B., Chen, X., 2019. Improving defect prediction with deep forest. Inf. Softw. Technol. 114, 204–216.

Zhu, K., Zhang, N., Ying, S., Wang, X., 2020. Within-project and cross-project software defect prediction based on improved transfer naive bayes algorithm. CMC-Comput. Mater. Contin. 63 (2), 891–910.

Kun Zhu is currently pursuing the Ph.D. degree in the School of Computer Science, Wuhan University. His research interests include software engineering and deep learning.

Shi Ying is now a professor in the School of Computer Science, Wuhan University. His main research interests include software engineering, and artificial intelligence.

Nana Zhang is currently pursuing the Ph.D. degree in the School of Computer Science, Wuhan University. His research interests include software engineering and deep learning.

Dandan Zhu is currently engaged in postdoctoral work at the Artificial Intelligence Institute, Shanghai Jiaotong university. His research interests include software project management, deep learning and artificial intelligence.