A Feature Selection Framework for Software Defect Prediction Using ISFLA

Wenbin Bi¹, a, Fang Yu², b
¹College of Computer and Software, Dalian Neusoft University of Information, Dalian, China
²College of Information Engineering, Qingdao Bin Hai University, Qingdao, China

¹biwenbin@neusoft.edu.cn, b13614283271@139.com

Abstract. Aiming at the problem of feature space dimension reduction and large search space in feature selection of software defect, a defect prediction feature selection framework based on meta-heuristic search algorithm (ISFLA) is proposed. The framework improves generalization of predictions of unknown data samples, enhances the ability to search for features related to learning tasks, and completes further reductions in feature space dimensions. Using some NASA data sets, several common software defect prediction methods and ISFLA simulation experiments were carried out. The experimental results show that the software feature selection framework based on the improved shuffled frog leaping algorithm effectively improves the performance of software defect prediction.

1. Introduction
Software defect is the flaw that leads to software failure in software products. In order to reduce the number of software defects, it is necessary to complete software testing for software products. However, the quality assurance activities such as testing usually take place after the completion of software development. And because of the increasing complexity of current software, it is difficult to increase the quality of delivered software products only by relying on software testing. According to Pareto-Zipf-type law [1], the 80:20 empirical rule is operating here, i.e., a small amount of code (often quantified as 20% of the code) is responsible for the majority of software faults (often quantified as 80% of the faults). Software defect prediction, in turn, may be the best way to improve the quality of software systems without allocating too many resources, and, at the same time, to reduce the cost of developing software products.

Software defect prediction plays an important role in software system. The current cloud concept has spread all over the corner. In 2016, Huang et al. established a trusted cloud infrastructure to improve the reliability of cloud service [2]. Subsequently, Wang et al. used proxy re-signature technology to propose a shared data integrity audit scheme that supports user revocation [3]. In the above cloud storage, defect prediction also plays an important role. The purpose of software defect prediction is to find potential defects before delivery, and to improve the quality of software systems. Software defect prediction first appeared in the 1970s, and now there are hundreds of related software defect prediction models. Typical software defect prediction methods corresponding to software defect prediction model are linear discriminant analysis (LDA) [4], Boolean discriminant function (BDF) [5],
classification and regression tree (CART) [6], optimized set reduce (OSR) [7], clustering analysis (CA) [8], support vector machine (SVM) [9], artificial neural network (ANN) [10], etc.

Ant colony optimization (ACO) algorithms take inspiration from the foraging behavior of some ant species. These ants deposit pheromone on the ground in order to mark some favorable path that should be followed by other members of the colony [11]. ACO is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. This algorithm is a member of ant colony algorithms family, in swarm intelligence methods, and it constitutes some meta-heuristic optimization. To implement the algorithm it is necessary to get basic ideas of swarm intelligence. ACO is also applied to software defect prediction. However, due to the lack of direct communication between groups in ant colony optimization algorithm, it is easy to obtain local optimization or reduce the convergence speed of the parameter optimization process.

Shuffled Frog-Leaping Algorithm (SFLA) [12] aims to model and mimic the behavior of frogs searching for food laid on stones randomly located in a pond. It combines the advantages of the genetic-based meme-tic algorithm (MA) and the social behavior-based Particle Swarm Optimization (PSO) algorithm. The shuffled frog-leaping algorithm is a heuristic search algorithms [13]. It pays attention to the communication between frogs in each group and the information exchange between groups and groups, both local search and global collaborative search. SFLA have demonstrated effectiveness in a number of global optimization problems. SFLA has the advantages of strong search ability, simple algorithm, great flexibility and strong robustness.

This paper presents a feature selection framework based on Improved Shuffled Frog Leaping Algorithm (ISFLA) for software defect prediction. In our framework, we design a two-level structure to select the feature. The feature selection framework first carries out primary optimization in local scope, and then filters out the final feature subset in global scope according to the threshold. Finally, we conducted experiments on 11 National Aeronautics and Space Administration (NASA) data sets and compared them with other feature selection methods to demonstrate the effectiveness of our framework.

The main contributions of this paper are as follows:
1. The original meta heuristic algorithm is improved and used for feature selection. The improved algorithm has strong search ability and greater flexibility.
2. A feature selection framework based on heuristic search algorithms is proposed for software defect prediction.

2. Related Work
From a biological perspective, how can a group of frogs find food quickly and effectively? The methods of local optimization and information exchange among different sub-groups are used for reference. Firstly, a group of frogs was divided into several sub-groups. Secondly, the distance between each frog and food is used as a message to communicate with other frogs. Compare the farthest distance with the nearest distance, and then move the corresponding position to adjust the distance. When the location of each sub-group is optimized to a certain extent, then the sub-groups in the total group are mixed to meet the final set conditions.

In 2003, American scholars Eusuff and Lansey proposed a new swarm algorithm, SFLA (Shuffled Frog Leaping Algorithm). SFLA attempts to balance between a wide scan of a large solution space and also a deep search of promising location for a global optimum which can not be solved by traditional optimization techniques. It combines the benefits of a gene-based meme-tic algorithm (MA) and social behavior-based particle swarm optimization (PSO). MA is a gene-based optimization algorithm similar to a GA. In a GA, chromosomes are represented as a string consisting of a set of elements called "genes". Chromosomes in MA are represented by elements, called "memes". MA and GA differ in one aspect, i.e. MA implements a local search before cross-over or mutations to determine offspring. After the local search, new offspring that obtains better results than original offspring, replaces original offspring and thus the evolutionary process continue. PSO is an evolutionary algorithm in which individual solutions are called "particle" (analogous to the GA chromosome), but PSO does not
apply crossover and mutation to construct a new particle. Each particle changes its position and velocity based on the individual particle’s optimal solution and the corporate optimal solution until a global optimal solution is found.

The algorithm process of SFLA is as follows [14]: assuming the frog population size is Q, the initial population \( X = (X_1, X_2, \ldots, X_q) \). For the frog population generated by random method, the fitness function \( f(x_i) \) of each frog individual is calculated, and the fitness value is ranked in descending order according to the calculated fitness value. Then the whole frog population is divided into all frog populations. The whole frog population was divided into \( C \) subgroups, each of which contained \( L \) frogs, namely \( T = C \times L \).

In the optimization process, \( X_1 \) is put into the first subgroup, \( X_2 \) into the second subgroup, \( X_c \) into the \( C \) subgroup, \( X_{c+1} \) into the first subgroup, \( X_{c+2} \) into the second subgroup,... Until the last individual into \( X_c \). The frogs with the best fitness in each subgroup were recorded as \( G_b \) and the frogs with the worst fitness were recorded as \( G_w \). The frogs with the best fitness in the whole group were recorded as \( X_b \).

In each local search update strategy, only \( G_w \) is operated to improve its fitness. The internal search strategy of frog subgroup is as follows:

\[
sd_{(s)} = R \times (G_b - G_w) \tag{1}
\]

\[
G_w = G_w + sd_{(s)} - sd_{(max)} \leq sd_{(s)} \leq sd_{(max)} \tag{2}
\]

\( R \) denotes the random number in \([0, 1]\), and \( sd_{(max)} \) denotes the maximum step of frog movement.

The optimization rule of SFLA is that if the fitness of new frog individual \( G_n \) is greater than that of original frog individual \( G_w \), then \( G_n \) will replace \( G_w \) to become a new individual in frog subgroup. Otherwise, replace \( G_h \) with \( x_h \) and re-operate according to Formula (1) and Formula (2). If there is improvement, replace \( G_w \); if there is no improvement, randomly generate an individual to replace \( G_w \). Repeat the above until the frog population reaches the set number of iterations. When all frog sub-populations are searched locally, all individuals in the population are reordered, subgroups are divided, and then subgroups are searched locally, and so on, until the set population evolution algebra is reached.

3. Our Approach

3.1. Improved Shuffled Frog Leaping Algorithm

The new position of the worst frog may always be within the linear range between the current position and the optimal frog position. Even if the position of the worst frog is updated according to the random solution, it can not ensure that the random solution will be better. At the same time, the worst frog position is updated only according to the optimal frog position, and the communication with other frog individuals is lacking, which reduces the adaptability of the frog population. In the hybrid leaping frog algorithm, the moving step of the worst frog is uncertain, and the learning step may be too large to omit the global optimum. In the later stage of optimization using SFLA algorithm, the difference between Frog individuals in the population decreases and the moving step decreases, which reduces the convergence accuracy of the whole algorithm.

In order to make the worst frog individuals fully communicate with other frogs, other frog individuals in the subgroup are introduced into the moving step of the improved SFLA algorithm. At the same time, through the adjustment of trigonometric function, the improved SFLA algorithm performs better in both local search and global search. And, in order to improve the search accuracy,
the newly generated frog individuals search within the maximum radius between themselves and the optimal frog in their subgroup with their own origin.

\[
sd_{(k)} = 2 \times \sin\left(\frac{\pi}{2G}\right) \times R \times \left[ x(k) - x(k)_w \right] + R \times [x(k)_s - x(k)_n] \tag{3}
\]

\[
sd_{(g)} = 2 \times \sin\left(\frac{\pi}{2G}\right) \times R \times \left[ x(g) - x(k)_w \right] + R \times [x(k)_s - x(k)_n] \tag{4}
\]

\[
x_{new} = 2 \times \cos\left(\frac{\pi}{2G}\right) \times R \times r_{max} \times x(k)_s \tag{5}
\]

Where \( t = 1, 2, \ldots, G \), \( G \) are fixed evolutionary iterations, \( r_{max} \) is the maximum radius between the frog individual and the optimal frog in the subgroup, and \( R \) is the random number between \((0, 1)\). \( sd_{(k)} \) was the worst moving step of the \( K \)th frog population. \( x(k)_s \) is a randomly selected frog individual from the \( K \)th frog subgroup. \( x(k)_s, x(k)_w \) and \( x(g)_b \) represent the best, worst in the \( K \)th frog population and globally best frog individuals. \( x_{new} \) is a random new frog individual in frog subpopulation.

3.2. Feature selection framework based on ISFLA

There are two main difficulties in the current feature selection: A. Feature interaction problem. The problem is that there are two-two interactions, three-three interactions, and even higher-intensity interactions between features. On the one hand, a feature may have little relevance to the class label, but if the feature has a complementary relationship with other features, the performance of the classifier can be significantly improved. Therefore, removing such features will result in a selected subset of features. Not optimal. On the other hand, although a certain feature has a strong correlation with the class label, if it is put together with other features, it may have some redundancy, which may cause the performance of the classifier to decrease. B. The search space is large. The search space grows exponentially as the number of features increases (ie, the number of possible feature subsets is \( 2^n \) relative to \( n \) features). In most cases, searching for all possible subsets is not feasible. This thesis proposes an effective defect-based feature selection framework based on meta-heuristic search algorithm with the help of search-based software engineering. The combinatorial optimization problem in software engineering is solved by meta-heuristic search algorithm.

With the development of new technologies, more and more large-scale data has been produced. It is the basic requirement for feature selection to reduce the dimension of high-dimensional data to a suitable size by feature selection, while retaining as much information as possible in the original data set, and then sending the feature data subset to the data processing system. Therefore, the defect prediction feature selection framework based on the meta-heuristic search algorithm proposed in this paper will focus on the following four key issues:

1. Interpretability, that is, the feature selection framework has scientific significance.
2. Stability.
3. Try to avoid errors in the hypothesis test.
4. Try to control the complexity of the calculation.

This section will give the implementation details of the feature selection framework ISLLA proposed in the paper. This topic is divided into two parts: the primary and the preferred part of the software complexity measurement data set.

Primary section: Processing the test history data set. The main work is to describe the two types of erroneous data: "contradictory data" and "duplicate data". Combined with the idea of data mining, the
important influences of these two types of erroneous data on defect prediction are described and processed separately, and a more appropriate test historical data set is selected.

The preferred part: processing the complexity metric attribute set, including local optimization within the feature group and global information interaction. The local search evolution in the feature subgroup relies on the worst individuals in the subgroup to randomly move toward the optimal individuals in the subgroup, so that the differences of all individuals in the group are gradually reduced, resulting in a gathering effect. In the global information interaction part, the ranking is performed according to the fitness value, and the ranking of the fitness value is high (the feature correlation is high). At the same time, a threshold is needed to select the final subset of features.

Assuming that there are m×n feature subsets. The first stage: local optimization within the feature group.

The local search evolution in the feature subgroup relies on the worst individuals in the subgroup to randomly move toward the optimal individuals in the subgroup, so that the differences of all individuals in the group are gradually reduced, resulting in a gathering effect. Dynamically adjust the search step size and individual frog position using ISFLA algorithm.

Figure 1. Feature selection framework based on ISFLA
Sorting according to the fitness value, the ranking of the high fitness value (high correlation of features) is first. At the same time, a threshold is needed to select the final subset of features. The threshold setting formula calculates the mean of all frog individual positions.

\[ t = \frac{1}{n} \sum_{i=1}^{n} x_{(i)} \]  

(6)

4. **Empirical Study**
To investigate the validity of our approach, we design experiments on data sets in the Tera-PROMISE repository (http://openscience.us/repo/). All experiments are conducted on Open JDK 1.7 and Weka 3.7 (http://www.cs.waikato.ac.nz/ml/weka/).

4.1. **Experimental Subjects**
We select 11 NASA datasets as the subjects. These datasets are commonly used in software defect prediction. It indicated that the quality of datasets is important for software defect prediction, and there are constant, repeated, and inconsistent features in the original NASA datasets [15]. Moreover, it indicated that there are many repeated and inconsistent data in the original NASA datasets, and they provided the cleaned datasets [16]. Therefore, we use these cleaned NASA datasets, which can be obtained from the Tera-PROMISE repository.
The details of NASA databases are described in Table 1. It shows the names of these datasets (column 1) and the number of features (except for the class feature) in each dataset (column 2). Next, it shows the number of all samples, defective samples, and non-defective samples, respectively (columns 3-5). Finally, it shows the defect rate and imbalance ratio of each dataset (columns 6).

| Data set | Number of features | Number of all samples | Number of defective samples | Number of non-defective samples | Defect rate |
|----------|--------------------|-----------------------|-----------------------------|---------------------------------|-------------|
| CM1      | 37                 | 327                   | 42                          | 285                             | 12.8%       |
| JM1      | 21                 | 7782                  | 1672                        | 6110                            | 21.5%       |
| KC3      | 39                 | 194                   | 36                          | 158                             | 18.6%       |
| MC1      | 38                 | 1988                  | 46                          | 1942                            | 2.3%        |
| MC2      | 39                 | 125                   | 44                          | 81                              | 35.2%       |
| MW1      | 37                 | 253                   | 27                          | 226                             | 10.7%       |
| PC1      | 37                 | 705                   | 61                          | 644                             | 8.7%        |
| PC2      | 36                 | 745                   | 16                          | 729                             | 2.1%        |
| PC3      | 37                 | 1077                  | 134                         | 943                             | 12.4%       |
| PC4      | 37                 | 1287                  | 177                         | 1110                            | 13.8%       |
| PC5      | 38                 | 1711                  | 471                         | 1240                            | 27.5%       |

All features in the above data sets are numeric, and they are designed at the function level, including McCabe metrics [17] and Halstead metrics [18]. McCabe metrics use a flow graph to measure the complexity of software modules, and Halstead metrics use the number of operators and operands to measure the complexity of software modules. They consider that the more complicated modules tend to be defective. Based on these features, our approach aims to fully measure the correlation between different features and the class. What is more, we expect to select those features that are more relevant to the class.

4.2. Experimental Design

We design experiments to make comparisons with several feature selection approaches. These approaches are also the classification methods often used in current software defect prediction research. They are all feature ranking approaches that can be implemented in Weka. We take the default parameters of these approaches from Weka. We use the KNN model as the classifier, which can also be implemented in Weka. In our experiments, the parameter $K$(number of nearest neighbors) of KNN is set to 10, and the parameter 'distance Weighting' is set to 'Weight by 1/distance'. All experiments are conducted over 10 times 10-fold cross-validation.

Next, we should select a reasonable metric to measure the classification performance. Generally, precision, recall, accuracy, and F-measure metrics are commonly used in software defect prediction [19].

4.3. Experimental Results and Analysis

To fully compare the performance of different feature selection approaches, we evaluate all feature subsets obtained from the feature ranking list in sequence. Considering that SM and RF are feature weighted approaches based on distance measures, we first compare them as follows.

In the JM1 data set, the software modules in each data set are classified according to the number of errors ERROR_COUNT attribute. The software module that defines ERROR_COUNT ≥1 is an error-prone module, and vice versa. The traditional feature selection methods used here are Stepwise Regression (SR) [20], Ridge Regression (RR) [21] and Principal Component Analysis (PCA) [22]. The comparison of defect prediction results is shown in Table 2.
As can be seen from Table 2, SR has a poor prediction effect. Because the core of SR is to statistically test the original variables while introducing new variables, and to delete the variables with insignificant influences to achieve the purpose of dimensionality reduction. This can easily lead to an infinite loop, which can not be found the optimal subset of attributes. The prediction effect of RR is better than SR. However, RR can only compress the regression coefficient of a variable infinitely, but it cannot go to zero, which means that it cannot reduce the dimension of the attribute set. PCA has a better prediction effect, but this method cannot transform or combine the complexity measure attributes to generate a new attribute set, which changes the original physical meaning of the selected attribute. This is very disadvantageous for understanding the data generation process and the structure of the software defect prediction model. The ISFLA proposed in this paper reduces the complexity metric attribute set used for experiments. It can be seen from the simulation results that the four indicators of ISFLA are better than the other methods, and good defect prediction results are obtained.

**Table 2. Comparison of Prediction Results under JM1 Data Set**

| Feature selection method | Accuracy(%) | Precision(%) | Recall(%) | F-measure(%) |
|--------------------------|-------------|--------------|-----------|--------------|
| SR                       | 48.3        | 73.2         | 51.3      | 60.3         |
| RR                       | 55.3        | 78.9         | 56.9      | 66.1         |
| PCA                      | 68.3        | 83.9         | 72.6      | 77.8         |
| ISFLA                    | 91.1        | 92.1         | 80.2      | 85.7         |

**Table 3. Comparison of Prediction Results under CM1 Data Set**

| Feature selection method | Accuracy(%) | Precision(%) | Recall(%) | F-measure(%) |
|--------------------------|-------------|--------------|-----------|--------------|
| BP                       | 90.5        | 94.2         | 88.2      | 91.1         |
| CA                       | 88.2        | 89.2         | 82.4      | 85.7         |
| ISFLA                    | 92.3        | 95.1         | 89.7      | 92.3         |

**Table 4. Comparison of Prediction Results under KC1 Data Set**

| Feature selection method | Accuracy(%) | Precision(%) | Recall(%) | F-measure(%) |
|--------------------------|-------------|--------------|-----------|--------------|
| BP                       | 90.1        | 92.1         | 88.5      | 90.3         |
| CA                       | 86.8        | 85.3         | 76.4      | 80.6         |
| ISFLA                    | 91.5        | 93.9         | 88.7      | 91.2         |

**Table 5. Comparison of Prediction Results under PC1 Data Set**

| Feature selection method | Accuracy(%) | Precision(%) | Recall(%) | F-measure(%) |
|--------------------------|-------------|--------------|-----------|--------------|
| BP                       | 91.2        | 94.9         | 89.4      | 92.1         |
| CA                       | 87.4        | 89.5         | 82.4      | 85.8         |
| ISFLA                    | 92.6        | 95.2         | 90.1      | 92.6         |
Figure 3. Comparison of effects under Accuracy

Figure 4. Comparison of effects under Precision

Figure 5. Comparison of effects under Recall
Cluster Analysis (CA) [23] and BP neural network are the current common methods for predicting software defects. It can be seen from Table 3, Table 4 and Table 5 that for some NASA datasets, ISFLA has different degrees of predictive performance improvement. It shows that ISFLA not only can select a smaller subset of features, but also has an advantage in further removing irrelevant features. Further comparisons in Figures 3, 4, 5 and Figure 6 show that, in most cases, ISFLA is superior to BP and CA in the four indicators of Accuracy, Precision, Recall and F-measure.

We also find that there are indeed variations of the performance with different numbers of features. To be specific, the performance can be significantly improved at the beginning because the top features are more relevant to the class. As the number of features increases, the performance would be stable or may drop slightly due to some redundant or irrelevant features. We select ten data sets CM1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4 and PC5, and select the three classifiers NB, J48 and LR. Two benchmark methods are considered, one is the wrap feature selection method based on the forward selection strategy (FW), and the other is the wrap feature selection method based on the backward selection strategy (BW). With AUC as the performance evaluation standard, the larger the AUC, the better the classification performance.

The ROC (receiver operating characteristic curve) curve is also a commonly used measure to evaluate classification performance. The ROC curve is different from the traditional evaluation method. The ROC curve can be allowed to have an intermediate class according to the actual situation. When measuring the classification effect of the classifier, the classification performance is generally evaluated by the area AUC of the area surrounded by the ROC curve and the abscissa. The closer the ROC curve is to the upper left corner when evaluating the classification results, the better the classification effect representing the classification algorithm. That is, when AUC>0.5, the closer the AUC is to 1, the better the classification effect. When AUC is between 0.5 and 0.7, the classification performance is low. When the AUC is 0.7~0.9, the classification result is more accurate.
Table 6. AUC Value Based on J48 Classifier

| Data set | FW   | BW   | ISFLA |
|----------|------|------|-------|
| CM1      | 0.564| 0.577| 0.671 |
| KC3      | 0.629| 0.6595| 0.6724|
| MC1      | 0.704| 0.68 | 0.713 |
| MC2      | 0.616| 0.565| 0.684 |
| MW1      | 0.63 | 0.538| 0.6331|
| PC1      | 0.7355| 0.7065| 0.7462|
| PC2      | 0.606| 0.611| 0.726 |
| PC3      | 0.7095| 0.6545| 0.718 |
| PC4      | 0.851| 0.817| 0.8672|
| PC5      | 0.705| 0.666| 0.713 |

From the above experimental data, it is found that if the J48 classifier is used as the modeling method, the ISFLA proposed in this paper is better than FW and BW. If NB is used as the classifier, the proposed ISFLA is superior to FW and BW in most cases. If the LR classifier is used as the modeling method, the ISFLA proposed in this paper is superior to FW in most cases, and is better than BW. The ISFLA method proposed in this paper improves the classification performance of data sets and has a good classification effect.

Table 7. AUC Value Based on NB Classifier

| Data set | FW   | BW   | ISFLA |
|----------|------|------|-------|
| CM1      | 0.6235| 0.633| 0.661 |
| KC3      | 0.697| 0.707| 0.763 |
| MC1      | 0.777| 0.7365| 0.763 |
| MC2      | 0.686| 0.729| 0.758 |
| MW1      | 0.709| 0.701| 0.737 |
| PC1      | 0.812| 0.8065| 0.819 |
| PC2      | 0.7215| 0.716| 0.7393|
| PC3      | 0.7865| 0.8005| 0.8014|
| PC4      | 0.858| 0.866| 0.868 |
| PC5      | 0.72 | 0.715| 0.728 |

Table 8. AUC Value Based on LR Classifier

| Data set | FW   | BW   | ISFLA |
|----------|------|------|-------|
| CM1      | 0.6815| 0.655| 0.7016|
| KC3      | 0.677| 0.5965| 0.678 |
| MC1      | 0.754| 0.6945| 0.771 |
| MC2      | 0.701| 0.703| 0.7355|
| MW1      | 0.6475| 0.552| 0.651 |
| PC1      | 0.8545| 0.8095| 0.8514|
| PC2      | 0.772| 0.686| 0.7936|
| PC3      | 0.806| 0.7955| 0.8193|
| PC4      | 0.877| 0.895| 0.902 |
| PC5      | 0.733| 0.7415| 0.7436|

The defect prediction feature selection framework proposed in this paper deletes the error data and irrelevant metric attributes in the defect prediction in the early stage of the framework, and reduces the
dimension of the input data. In the later stage of the framework, the corresponding solutions are proposed for the two difficulties of feature selection, which improves the ability of defect prediction. Aiming at the existence of dimensionality disasters in software defect prediction research and software complexity metric attributes which are not related to defect prediction and existing feature selection algorithms can not satisfy the global optimal problem, a defect prediction based on meta heuristic search algorithm is proposed. Feature selection framework. Starting from the combinatorial optimization problem, the framework reduces the size of the data set and improves the accuracy of the classification.

Software defect prediction technology can obtain relevant experience from similar software development projects or projects under development to judge the reliability of newly developed products. This technology identifies software modules with errors in the early stages of software development, allowing software improvements to focus on modules that are more prone to defects, resulting in better cost-benefit ratios. Through the study of software defect prediction technology, it is possible to objectively evaluate software reliability, understand the quality status of the software, and determine whether the software meets the standards for delivery.

5. Conclusions and Future Work
This paper proposes and implements a software defect prediction feature selection framework ISFLA based on meta heuristic algorithm. Based on the empirical research on NASA dataset, the validity of ISFLA is verified and provides guidance for the effective use of ISFLA. The experimental results show that our approach performs better or more comparable than the compared approaches. Moreover, our approach performs better than the compared approaches on large-scale datasets and imbalanced datasets.

There are still many research points worthy of further attention in this research work, including: a) this paper mainly considers the NASA data set. In the next step, more data sets need to be considered to verify whether the conclusions obtained in this paper are applicable. b) In the implementation of ISFLA, other meta-heuristic algorithms can further try to use and verify their effectiveness and performance. c) We have evaluated our approach only on the KNN model. More feature prediction methods should be evaluated in the next work.

Acknowledgments
Liaoning Natural Fund Guidance Plan Project (No.20180550021), Dalian Science and Technology Star Project (No.2017RQ021) and 2019 Qingdao Binhai University-level Science and Technology Plan Research Project (No.2019KY09).

References
[1] Endres, A., & Rombach, D.(2003). A handbook of software and systems engineering. Reading: Addison Wesley.
[2] HUANG C, HE L, LIAO X, et al. Developing a trustworthy computing framework for clouds [J]. International Journal of Embedded Systems, 2016, 8(1): 59-68.
[3] TIAN H, MA Y, WEI B. Proxy re-designated versifier signature and its applications [J]. International Journal of Embedded Systems, 2017, 9(6): 535-547.
[4] Munson JC, Khoshgoftaar TM. The detection of fault-prone programs. IEEE Trans. on Software Engineering, 1992, 18(5): 423-433.
[5] Khoshgoftaar TM, Seliya N. Improving usefulness of software quality classification models based on boolean discriminant functions. In: Proc. of the 13th Int’l Symp. on Software Reliability Engineering. IEEE Computer Society Press, 2002. 221-230. http://portal.acm.org/citation.cfm?id=851033.856311
[6] Khoshgoftaar TM, Yuan X, Allen EB. Balancing misclassification rates in classification-tree models of software quality. Empirical Software Engineering, 2000, 5(4): 313-330.
[7] Briand LC, Basili V, Hetmanski CJ. Developing interpretable models with optimized set reduction for identifying high-risk software components. IEEE Trans. on Software Engineering, 1993, 19(11): 1028-1044.
[8] Zhong S, Khoshgoftaar TM, Seliya N. Analyzing software measurement data with clustering techniques. IEEE Intelligent Systems, 2004, 19(2): 20-27.

[9] Xing F, Guo P, Lyu MR. A novel method for early software quality prediction based on support vector machine. In: Proc. of the 16th IEEE Int’l Symp. on Software Reliability Engineering (ISSRE 2005). Chicago, 2005. 213-222. http://portal.acm.org/citation.cfm?id=1104997.1105250

[10] Khoshgoftaar TM, Lanning DL. A neural network approach for early detection of program modules having high risk in the maintenance phase. Journal of Systems and Software, 1995, 29(1): 85-91.

[11] Dorigo, M., Birattari and T. Stutzle, 2006. Ant colony optimization-artificial ants as a computational intelligence technique. IEEE Comput. Intell. Mag., 1: 28-39.

[12] Eusuff M M, Lansey K E. Optimization of water distribution network design using the shuffled frog leaping algorithm [J]. Journal of Water Sources Planning and Management. 2003. 129(3): 210-225.

[13] Eusuff M M, Lansey K E. Pasha F. Shuffled frog leaping algorithm: A memetic meta-heuristic for discrete optimization [J]. Engineering Optimization. 2006. 38(2): 129-151.

[14] Liu LQ, Wang LG, Huo JY. Shuffled Frog Leaping Algorithm Based on Fuzzy Threshold Compensation [J]. Computer Engineering. 2014. 40(5): 168-172.

[15] Gray, D., Bowes, D., Davey, N., et al., 2011. The misuse of the NASA metrics data program data sets for automated software defect prediction. Int. Conf. on Evaluation and Assessment in Software Engineering, p. 96-103. https://doi.org/10.1049/ic.2011.0012

[16] Shepperd, M., Song, Q., Sun, Z., et al., 2013. Data quality: some comments on the NASA software defect datasets. IEEE Trans. Softw. Eng., 39(9): 1208-1215. https://doi.org/10.1109/TSE.2013.11

[17] McCabe, T. J., 1976. A complexity measure. IEEE Trans. Softw. Eng., SE-2(4):308-320. https://doi.org/10.1109/TSE.1976.233837

[18] Halstead, M.H., 1977. Elements of Software Science. Elsevier, New York, USA.

[19] Jing, X., Ying, S., Zhang, Z., et al., 2014. Dictionary learning based software defect prediction. Int. Conf. https://doi.org/10.1145/2568225.2568320

[20] Fredman J H. Multivariate adaptive regression splines [J]. Annals of Statistics. 1991. 19(1): 1-67.

[21] Hastie T, Taylor J, Tibshirani R, et al. Forward stagewise regression and the monotone Lasso [J]. Electronic Journal of Statistic. 2007. 5(1): 1-29.

[22] Zhang J, Ge W, Hao KG. Research on Principal Component Analysis Method in Software Metrics [J]. Computer Technology and Development. 2006. 16(12): 144-148.

[23] Khoshgoftaar TM, Liu Y, Seliya N. A multi-objective module-order model for software quality enhancement [J]. IEEE Transactions on Evolutionary Computation, 2004, 8(6): 593-608.