IET Software

Research Article

Low-rank representation for semi-supervised software defect prediction

ISSN 1751-8806
Received on 24th August 2017
Revised 29th March 2018
Accepted on 7th May 2018
E-First on 11th June 2018
doi: 10.1049/iet-sen.2017.0198
www.ietdl.org

Zhi-Wu Zhang[1], *Xiao-Yuan Jing*[2,3] ✉, *Fei Wu*[3]
[1]*School of Computer, Nanjing University of Posts and Telecommunications, Nanjing, People's Republic of China*
[2]*State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan, People's Republic of China*
[3]*School of Automation, Nanjing University of Posts and Telecommunications, Nanjing, People's Republic of China*
✉ *E-mail: jingxy_2000@126.com*

Abstract: Software defect prediction based on machine learning is an active research topic in the field of software engineering. The historical defect data in software repositories may contain noises because automatic defect collection is based on modified logs and defect reports. When the previous defect labels of modules are limited, predicting the defect-prone modules becomes a challenging problem. In this study, the authors propose a graph-based semi-supervised defect prediction approach to solve the problems of insufficient labelled data and noisy data. Graph-based semi-supervised learning methods used the labelled and unlabelled data simultaneously and consider them as the nodes of the graph at the training phase. Therefore, they solve the problem of insufficient labelled samples. To improve the stability of noisy defect data, a powerful clustering method, low-rank representation (LRR), and neighbourhood distance are used to construct the relationship graph of samples. Therefore, they propose a new semi-supervised defect prediction approach, named low-rank representation-based semi-supervised software defect prediction (LRRSSDP). The widely used datasets from NASA projects and noisy datasets are employed as test data to evaluate the performance. Experimental results show that (i) LRRSSDP outperforms several representative state-of-the-art semi-supervised defect prediction methods; and (ii) LRRSSDP can maintain robustness in noisy environments.

1 Introduction

Software defect prediction is one of the most important research topics in software engineering, which can relieve the burden on software code inspection or software testing. Software defects arise from the development process of developers [1]. However, due to the effect of the project schedule and the complexity of the software systems, it is almost impossible to test every software part extensively under all possible conditions. Thus, accurately predicting whether a software module contains defects enables the organisation's limited resource to be reasonably allocated, and hence, it can help to improve the quality of the software systems [1, 2].

With the wide application of machine learning theories and methods to the field of software defect prediction, software defect prediction has evolved from early empirical formula estimation to prediction of defect proneness and distribution [3]. Machine learning-based static software defect prediction methods utilise software history repository data, analysis software code or development process by the metrics connected with software defect, and predict the defect-proneness of the test modules using machine learning method. Most of them assume that there are enough software defect data to build software defect prediction models, in which supervised classification techniques utilise software metric information and software defect label information when constructing software defect prediction models. However, in some cases, it is difficult to obtain enough labelled software defect data to build accurate prediction models since manually labelling a large number of software defect examples is both time consuming and labour intensive, while many unlabelled data can be easily collected [4]. In addition, in the process of mining the software history repository, type labelling and metrics of program modules and development process may cause noise when establishing a link between the modified log in the version control system and the defect report in the defect tracking system. Noises also may be caused in misclassifying the problem reports in bug tracking systems [5]. These noises seriously affect the performance of defect prediction algorithm.

These practical applications pose two challenges to the effectiveness of software defect prediction algorithms: (i) How to construct an effective defect prediction model under the limited labelled defect data scene? (ii) How to build an effective defect prediction model in noisy environments?

1.1 Motivation

For solving the problem of software defect prediction with limited labelled defect data, both labelled and unlabelled data are considered for model training in semi-supervised learning so that the abundant information becomes available. As shown in Fig. 1, the initial labelled data are rather small, and they cannot accurately describe the overall distribution of the data. Therefore, the supervised learning boundary, which is classified by using labelled data only, cannot distinguish different samples effectively. However, the unlabelled data are obviously abundant and their natural clusters can depict data distribution more appropriately. Therefore, the semi-supervised learning boundary, which is classified by using labelled and unlabelled data, can better distinguish different samples. Incorporating the information of unlabelled data into the learning process may obtain a better classification prediction.

In graph-based semi-supervised learning (GSSL), graph construction that represents the pairwise relationship among data plays an important role [6–8]. GSSL models the whole dataset as a graph and considers labelled data and unlabelled data in the dataset as the nodes of the graph, and then the edges between any two nodes are instantiated according to the similarity between the corresponding samples. For a given dataset, the core problem of graph construction is to determine the non-negative weight matrix that reflects the similar relation among nodes. A perfect similarity graph built by GSSL has the same number of independent connected components corresponding to the subspaces or classes, so by applying spectral clustering method, the labels can be propagated from the labelled samples to unlabelled samples over the graph [8]. Low-rank representation (LRR) is a promising method for constructing a weighted graph. The target of the LRR is

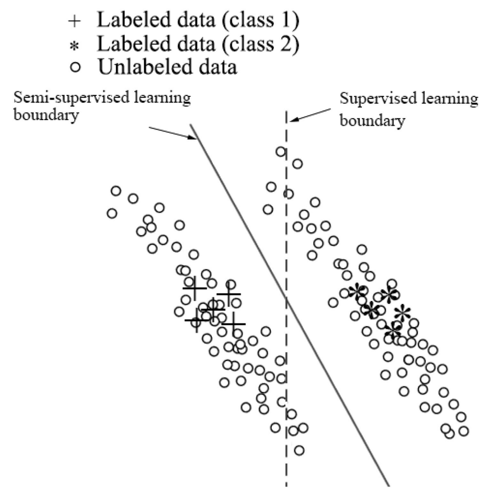

Fig. 1 *Example for semi-supervised learning*

to find the lowest-rankness representation among all candidates that can express the data vectors as linear combinations of the basis atoms in a proper dictionary [9]. The LRR guarantees that the sample coefficients from the same class are highly correlated, and it can suppress the noise in the data.

For the problem of software defect prediction, when the previous defect labels of modules in software history repository are limited, we can use few labelled defect data together with abundant unlabelled defect data to construct graph-based semi-supervised learning defect prediction model. For the problem of noise caused by type labelling and metrics of program modules and development process, the LRR, as an efficient weight graph construction method, can eliminate the influence of the noise in the data when constructing the data graph.

1.2 Contribution

In this paper, we propose a novel approach that employs LRR for semi-supervised software defect prediction. To summarise, the contributions of our study are as following three points:

(i) Aiming at the problem that the previous defect labels of modules are limited, a graph-based semi-supervised learning defect prediction approach is proposed.
(ii) During the construction of data relationship graph for semi-supervised learning, we use LRR method to construct the clustering relation of the data graph, which can effectively solve the noise problem in defect data. The nearest-neighbour distance is introduced in the construction of data relationship graph, which makes the data graph more accurate, as the LRR reflects the global structure of the data while the local neighbourhood structure can be retained. We, therefore, propose an LRR-based semi-supervised software defect prediction (LRRSSDP) approach.
(iii) Experimental results on the widely used software defect prediction datasets and noisy datasets show that the proposed LRRSSDP approach outperforms several representative state-of-the-art semi-supervised defect prediction methods and it can maintain robustness in noisy environments.

The remaining of this paper is organised as follows. Section 2 introduces the related work. Section 3 describes the methodology. Section 4 introduces the experimental setup. Section 5 shows the experimental results and analysis. Section 6 discusses the threats to validity. The conclusions and future works are drawn in Section 7.

2 Related work

2.1 Semi-supervised software defect prediction

In recent years, different types of semi-supervised learning techniques have been applied into the field of software defect prediction. Seliya and Khoshgoftaar investigated semi-supervised learning with the expectation maximisation (EM) algorithm for software defect prediction with limited defect data [10] and proposed a constraint-based semi-supervised clustering scheme [11]. Catal and Diri [12] used naïve Bayes algorithm to build a semi-supervised defect prediction model. Jiang *et al.* [13] proposed a semi-supervised software defect prediction approach named ROCUS. This method exploits the abundant unlabelled examples and tackles the class-imbalance problem. Li *et al.* [14] proposed two sample-based machine learning approaches for software defect prediction (CoForest and ACoForest). In their methods, random sampling with a semi-supervised learner is based on a sample and the remaining un-sampled modules, and active sampling with active semi-supervised learner can actively select several informative modules that are most helpful for learning a good prediction model. Thung *et al.* [15] proposed an active semi-supervised defect prediction approach. The approach actively selects a small subset of diverse and informative defect examples to label, and it is able to learn a good model while minimising the manual labelling effort. Catal [16] compared four semi-supervised classification methods for software defect prediction including low-density separation, EM, support vector machine, and class mass normalisation methods. He showed that low-density separation algorithm is better than support vector machine when the dataset is large and low-density separation-based prediction method is suggested for software defect prediction when the defect data are limited. Ma *et al.* [17] proposed an improved semi-supervised learning approach for software defect prediction involving limited defect labelled data and class imbalanced problem, which employs random undersampling technique to resample the original training set and updates training set in each round for co-train style algorithm. Abaei *et al.* [18] proposed an automated software fault prediction model using semi-supervised hybrid self-organising map (HySOM). HySOM is able to predict the label of the software modules in a semi-supervised manner in the absence of quality data. Wang *et al.* [19] proposed a non-negative sparse-based SemiBoost semi-supervised learning software defect prediction approach. This approach is capable of exploiting both labelled and unlabelled data and is formulated in a boosting framework. Zhang *et al.* [20] proposed a non-negative sparse graph-based label propagation approach for software defect classification and prediction, which uses not only a few labelled data but also abundant unlabelled ones to improve the generalisation capability.

2.2 Noise in software defect data

Many of the current defect prediction models are based on data from software repositories, which rarely consider the noise in these data. At present, most researches of software defect data noises are mainly to analyse the impact of noisy datasets on defect prediction methods. After studying the NASA MDP datasets, Khoshgoftaar and Seliya [21] suggested researchers not only to find a classification technique to improve the prediction accuracy, but also to solve the quality problem of software measurement data. At the same time, they proposed a technique to eliminate noise using the k-mean algorithm [22]. By analysing some open source projects, Bird and Bachmann [23, 24] found that many of the repaired defects did not link up with the corresponding modification logs and the dataset contained a lot of false negative (FN) noises. In addition, based on the BugCache method proposed by Kim *et al.* [25], they came to the conclusion that dataset with FN noise significantly reduces the performance of defect prediction methods. Nguyen *et al.* [26] also found the same problem in a business project. Research by Rahman *et al.* [27] showed that the FN noise and the size of the defect prediction datasets have important influence for the performance of defect prediction models. They found that even if there is FN noise in the dataset, it is still an effective method to improve the performance of the defect prediction model by trying to increase the size of the dataset. Herzig *et al.* [28] manually checked >7000 reports from five open source projects and found that most of the problem reports were classified as defect reports, but 33.8% of them were misclassified. Kim *et al.* [5] manually injected false positive (FP) and FN noises into the dataset by random means, and analysed the noise immunity

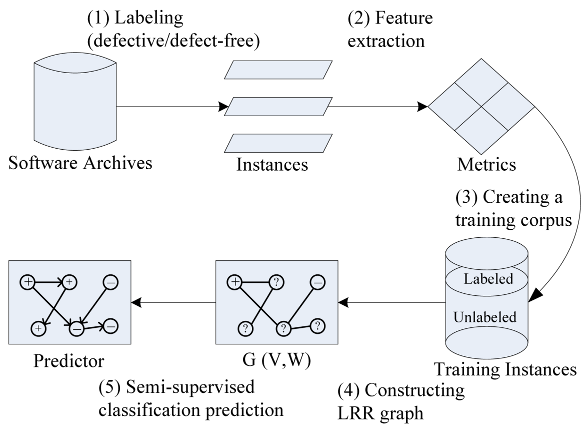

Fig. 2 *Defect prediction process of LRRSSDP*

of existing defect prediction methods. In addition, they proposed a cluster analysis-based method to detect and remove noise effectively. Tantithamthavorn *et al.* [29], based on the work of Herzig *et al.* [28], found that the noise generated by misclassification of the problem report was not random and different noise injection patterns had a significant impact on the final empirical results.

2.3 Low-rank representation

To capture the global structure of data, LRR is to construct the affinities of an undirected graph. LRR can correctly preserve the membership of samples that belong to the same class under suitable conditions. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be a set of n samples in the d-dimensional space. LRR $\boldsymbol{X} = \boldsymbol{AZ}$ aims at finding the lowest-rankness representation among all candidates that can express the data vectors as linear combinations of the basis in a proper dictionary $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m] \in \mathbb{R}^{d \times m}$, where $\boldsymbol{Z} = [z_1, z_2, \ldots, z_n]$ is the coefficient matrix with each z_i characterising how other samples contribute to the representation of $\boldsymbol{x}_i$. LRR seeks the lowest-rank solution by solving the following optimisation problem:

$$\min_{\boldsymbol{Z}} \operatorname{rank}(\boldsymbol{Z}) \quad \text{s.t.} \ \boldsymbol{X} = \boldsymbol{AZ} \tag{1}$$

Since the rank function is a NP-hard problem, the above optimisation problem can be relaxed to the following convex optimisation problem:

$$\min_{\boldsymbol{Z}} \| \boldsymbol{Z} \|_* \quad \text{s.t.} \ \boldsymbol{X} = \boldsymbol{AZ} \tag{2}$$

where $\| \bullet \|_*$ denotes the nuclear norm of a matrix which equals to the sum of its singular values. In real-world applications, the observation data are usually noisy. Based on the basic LRR, many efforts have been devoted to improve the LRR by imposing the penalty on low rank coefficients and noises. A more general rank minimisation problem is given as follows:

$$\min_{\boldsymbol{Z}} \| \boldsymbol{Z} \|_* + Q(\boldsymbol{Z}, \boldsymbol{E}) \quad \text{s.t.} \ \boldsymbol{X} = \boldsymbol{XZ} + \boldsymbol{E} \tag{3}$$

where Q is a penalty function on $\boldsymbol{Z}$ and $\boldsymbol{E}$. Zhuang *et al.* [30] defined $Q(\boldsymbol{Z}, \boldsymbol{E}) = \lambda_1 \| \boldsymbol{Z} \|_1 + \lambda_2 \| \boldsymbol{E} \|_{2,1}$ to construct a low-rank and sparse graph. Feng *et al.* [31] defined $Q(\boldsymbol{Z}, \boldsymbol{E}) = (\lambda/2) \| \boldsymbol{E} \|_F^2$ and produced an exactly block-diagonal similarity matrix by restricting the rank of Laplacian matrix. Fei *et al.* [9] proposed an LRR with adaptive distance penalty (LRRADP) to construct an affinity graph, which can not only capture the global structure of the whole data but also effectively preserve the neighbour relationship among samples.

In recent years, lots of efforts based on LRR have been made to exploit ways of constructing a discriminative graph for semi-supervised learning. Liu *et al.* [32] proposed a latent LRR method for subspace segmentation and feature extraction by approximating and using the unobserved data hidden in the observed data to construct the dictionary and resolve the issue of insufficient sampling. Zhang *et al.* [33] extended the latent LRR for subspace clustering by choosing the sparest solution in the solution set and pre-processing the noise data with robust PCA [34] to increase the robustness of the method. Siming and Zhouchen [35] showed that LRR can be approximated as a factorisation method and proposed an improved version of LRR which uses the corrected data as the dictionary instead of the noisy data. Fang *et al.* [36, 37] proposed a method combined the semi-supervised clustering learning with the non-negative LRR framework, which achieved acceptable clustering performance and was robust to different types of noises. Jing *et al.* [38] used the low-rank recovery and the semi-supervised regression technique to perform the missing data imputation for software effort estimation.

In this paper, LRRADP is improved and introduced into software defect prediction to solve the problem of constructing semi-supervised learning graphs and noise processing.

3 LRR-based semi-supervised software defect prediction

3.1 Problem formulation

In static software defect prediction, let $\boldsymbol{x}_i \in \mathbb{R}^d$ denote a program module sample, which is a column vector consisting of the defect metric values, and $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$ denote the software defect prediction dataset. $\boldsymbol{X}$ is a $d \times n$ matrix, where d denotes the number of software defect metric elements and n denotes the number of program modules in the software defect prediction dataset. A certain number of program modules in $\boldsymbol{X}$ have associated defect labels (defective or defect-free), and the remainders are unlabelled. l is the number of samples in the labelled dataset $\boldsymbol{X}_l$, and u is the number of samples in the unlabelled dataset $\boldsymbol{X}_u$. $\boldsymbol{X} = \{\boldsymbol{X}_l, \boldsymbol{X}_u\}$ $(l + u = n)$, where $\boldsymbol{X}_l = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_l\}$ and $\boldsymbol{X}_u = \{\boldsymbol{x}_{l+1}, \boldsymbol{x}_{l+2}, \ldots, \boldsymbol{x}_{l+u}\}$. Let $\boldsymbol{Y} = \{\boldsymbol{Y}_l, \boldsymbol{Y}_u\}$ be response labels where $\boldsymbol{Y}_l = \{y_1, y_2, \ldots, y_l\}$ are known and $\boldsymbol{Y}_u = \{y_{l+1}, y_{l+2}, \ldots, y_{l+u}\}$ are unknown. The observed labels in $\boldsymbol{Y}$ are binary variables, $y_i \in \{0, 1\}$, where 0 denotes the defect-free class and 1 denotes the defective class in software defect prediction. Our goal is to infer the missing labels $\boldsymbol{Y}_u$ of $\boldsymbol{X}_u$ using abundant unlabelled samples to assist limited labelled samples in semi-supervised learning to construct accurate defect classification prediction model. When there are noises in $\boldsymbol{X}$, the constructed semi-supervised defect prediction model can eliminate the influence of noises.

3.2 Framework of LRRSSDP

Fig. 2 shows the typical defect prediction process of LRRSSDP. The first step is to extract and label modules from software archives. A software module can be labelled as defective or defect-free according to whether it contains defects or not. The label data collected by mining software archives often contain noise. Then, defect prediction metrics, such as complexity metrics, are used as module features. The module features and labels are combined to create a training corpus. Then, the neighbourhood distance and LRR are used to construct the relationship graph of training instances. Finally, the semi-supervised classification predictions are used to predict the label of the unlabelled modules.

3.3 LRR graph construction

Graph-based semi-supervised learning relies on the construction of the graph that represents the relationship of the dataset. We create a weighted graph $G = (V, W)$ using the defect dataset, where V is the vertex set which denotes nodes of the graph corresponding to the defect data points and W is a symmetric non-negative weight matrix of the edge set which represents the connecting relationship among the nodes. A non-zero weight reflects the similar relations between corresponding nodes, while a zero weight indicates that there is no edge jointing them. An ideal weighted graph G is one in which nodes corresponding to points from the same class are connected to each other and there is no edge between any two nodes corresponding to points belonging to different classes. Most of the graph-based machine learning methods adopt a Gaussian function to calculate the edge weights of the graph, so the weight w_{ij} of the edge linking data $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are computed as

$$w_{ij} = \exp\left(\frac{-\| \boldsymbol{x}_i - \boldsymbol{x}_j \|^2}{(2\sigma^2)}\right) \tag{4}$$

where the variance σ is a free parameter determined by experience. According to Zhou *et al.* [39], σ will significantly affect the learning results, and it is difficult to determine an optimal σ. What is worse is that there is no reliable way for model selection if only a few labelled data are available.

To avoid the problem caused by the pairwise relationships as (4) in traditional graph-based methods, we propose to use the LRR to construct G. The graph identified by LRR obtains the representation of all the data under a global low-rank constraint and thus is better at capturing the global structures of data, such as multiple clusters. In k-nearest-neighbour graph (K-NNG), two vertices are connected by an edge, if the distance between the two vertices is among the kth smallest distances from one of the two vertices to the other vertex. According to the manifold hypothesis, the points in the local neighbourhood have similar properties, and thus these nearby points in the graph should have similar labels. Considering the local invariance property of the data, the nearest-neighbour distance of samples can be expressed as

$$\boldsymbol{D}_{ij} = \begin{cases} \| \boldsymbol{x}_i - \boldsymbol{x}_j \|_2^2, & \boldsymbol{x}_i \in N_k(\boldsymbol{x}_j) \ or \ \boldsymbol{x}_j \in N_k(\boldsymbol{x}_i) \\ \infty, & \text{other} \end{cases} \tag{5}$$

where $N_k(\boldsymbol{x}_j)$ represents the k-nearest neighbours of $\boldsymbol{x}_j$ and $\boldsymbol{x}_i \in N_k(\boldsymbol{x}_j)$ denotes $\boldsymbol{x}_i$ is one of the k-nearest neighbour of $\boldsymbol{x}_j$ (k is a positive integer, typically small). In the LRR, considering the nearest-neighbour relation, the weighted nearest-neighbour distance quadratic energy function can be expressed as

$$\sum_{i,j=1}^{n} \boldsymbol{D}_{ij}\boldsymbol{Z}_{ij} \tag{6}$$

where $\boldsymbol{Z}_{ij}$ is the non-negative weight between the ith and jth samples. Minimising formula (6) can assign small weight to the edge between samples with far distance. By combining formula (6) with the LRR, we obtain the following LRR:

$$\min_{\boldsymbol{Z}} \| \boldsymbol{Z} \|_* + \lambda \sum_{i,j=1}^{n} \boldsymbol{D}_{ij}\boldsymbol{Z}_{ij} \quad \text{s.t.} \ \boldsymbol{X} = \boldsymbol{X}\boldsymbol{Z}, \quad \boldsymbol{Z} \geq 0 \tag{7}$$

where $\boldsymbol{Z}$ is the LRR matrix of the dataset. In many real applications, the negative values lack physical interpretation, so they are restricted to non-negative values here. λ is a balance parameter to balance the ability to capture the global subspace structure and preserve the local the neighbourhood relativity between nearby points.

In real-world applications, software defect data are often noisy due to type labelling and metrics of program modules and development process issue. In such cases, the defect data do not entirely lie in a union of subspaces corresponding to their classes. So, we need to consider the case where data matrix $\boldsymbol{X}$ is a noisy matrix. We introduce an error matrix $\boldsymbol{E}$ to model the noise data, and get the following optimisation problem:

$$\begin{aligned} &\min_{\boldsymbol{Z},\boldsymbol{E}} \| \boldsymbol{Z} \|_* + \lambda_1 \| \boldsymbol{E} \|_1 + \lambda_2 \sum_{i,j=1}^{n} \boldsymbol{D}_{ij}\boldsymbol{Z}_{ij} \quad \text{s.t.} \ \boldsymbol{X} = \boldsymbol{X}\boldsymbol{Z} + \boldsymbol{E}, \\ &\boldsymbol{Z} \geq 0 \end{aligned} \tag{8}$$

The objective function of formula (8) has three terms. The first term seeks the lowest-rank representation of the whole data space. The second term indicates the sparse noises of the data and third term ensures that neighbour points in Euclidean space can be allocated relatively large edge weights. $\lambda_1 > 0$ and $\lambda_2 > 0$ are balance parameters to trade off among the LRR, noises and neighbour distance.

The solution of formulae (7) and (8) can be solved according to the methods provided by the LRRADP algorithm [9]. The minimiser $\boldsymbol{Z}^*$ of formula (8) can be considered as an improved LRR by embedding the nearest-neighbour distance, a column of which naturally represents the relationship of a sample with other samples. Since the nearest-neighbour distance generally guarantees that the representation coefficients of a sample are formed by the nearest neighbouring samples, the weighted graph identified by the optimal solution of the LRRSSDP can better capture both the global clustering structure of the whole data and the local nearest-neighbour relationships among the samples.

After obtaining optimal representation matrix $\boldsymbol{Z}^*$, we can build the graph adjacency structure and weight matrix from it. In practice, due to the data noise, the coefficient vector z_i^* of the point $\boldsymbol{x}_i$ is often dense with small values. With normalised coefficients vector of each sample (i.e. $z_i^* = z_i^*/\| z_i^* \|$), we make small coefficients zeros under a given threshold. After that, we can obtain a sparse $\hat{\boldsymbol{Z}}^*$, and define the graph weight matrix $\boldsymbol{W}$ as

$$\boldsymbol{W} = \frac{(\hat{\boldsymbol{Z}}^* + (\hat{\boldsymbol{Z}}^*)^{\mathrm{T}})}{2} \tag{9}$$

```
Algorithm 1: LRR graph construction
Input: Data matrix $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d \times n}$, regularised parameters $\lambda_1$ and $\lambda_2$, the number of nearest neighbours $n_{knn}(n_{knn} \leq \min(n, d))$, the threshold $\theta$
Output: The weight matrix $W$ of LRR graph

1. Normalise all the columns of $X$ to have unit $l^2$-norm.
2. According to the nearest-neighbour criterion and formula (5), compute the distance matrix $D$ of samples.
3. Solve the optimisation problem of formula (8) using LRRADP algorithm and obtain the optimal solution $Z^*$.
4. Normalise all column vectors of $Z^*$ by $z_i^* = z_i^*/\| z_i^* \|$, make small values under given threshold $\theta$ zeros, and obtain a sparse $\hat{Z}^*$.
5. Construct the LRR graph weight matrix $W$ by formula (9).
```

The method for constructing an LRR graph is summarised in Algorithm 1. The input data include data matrix, regularised parameters, the number of nearest neighbours and the threshold parameter. First, we normalise all the columns of $\boldsymbol{X}$ and let them have unit l^2-norm. This step is benefited for machine learning process. Second, according to the nearest-neighbour distance defined by formula (5), the algorithm computes the distance matrix $\boldsymbol{D}$ of all samples. Third, we solve the optimisation problem of formula (8) and obtain the optimal solution $\boldsymbol{Z}^*$. Then, we normalise all column vectors of $\boldsymbol{Z}^*$ and obtain a sparse $\hat{\boldsymbol{Z}}^*$ to deal with the data noise. Finally, we build the LRR graph weight matrix $\boldsymbol{W}$ and the graph adjacency structure is obtained at the same time. Therefore, the LRR graph is constructed.

3.4 Semi-supervised classification prediction

After obtaining the weight matrix $\boldsymbol{W}$ of LRR graph, the conventional semi-supervised classification method, such as Gaussian fields and harmonic functions (GFHFs), can be employed on the weight matrix to predict labels for unlabelled program modules. To address this issue, we define a label prediction matrix $\boldsymbol{F} = [\boldsymbol{F}_l\boldsymbol{F}_u]^{\mathrm{T}} \in \mathbb{R}^{n \times c}$, c is the number of defect classes, and the label of a sample $\boldsymbol{x}_i$ can be predicted by $y_i = \arg\max_j \boldsymbol{F}_{i,j}$. Let $\boldsymbol{Y} = [\boldsymbol{Y}_l \boldsymbol{Y}_u]^{\mathrm{T}} \in \mathbb{R}^{n \times c}$ be the initial label matrix, where $\boldsymbol{Y}_l$ and $\boldsymbol{Y}_u$ correspond to the labelled and unlabelled program modules, respectively. $\boldsymbol{Y}_{ij} = 1$ if the sample $\boldsymbol{x}_i$ is labelled and associated with the label $j\,(j = 1, 2, \ldots, c)$ and $\boldsymbol{Y}_{ij} = 0$ otherwise. The label prediction matrix $\boldsymbol{F}$ should satisfy the given labels $\boldsymbol{Y}_l$ and meanwhile smooth on the whole graph constructed by both labelled and unlabelled samples. That is, GFHF aims at minimising the following optimisation cost on a weight graph to recover the classification prediction matrix $\boldsymbol{F}$

$$g(\boldsymbol{F}) = (1/2)\sum_{i,j=1}^{n} \| \boldsymbol{F}_{i*} - \boldsymbol{F}_{j*} \|^2 \boldsymbol{W}_{i,j} + \lambda_\infty \sum_{i=1}^{l} \| \boldsymbol{F}_{i*} - \boldsymbol{Y}_{i*} \|^2 \\ = \mathrm{tr}(\boldsymbol{F}^{\mathrm{T}} \boldsymbol{L}_{\boldsymbol{W}} \boldsymbol{F}) + \mathrm{tr}(\boldsymbol{F} - \boldsymbol{F})^{\mathrm{T}} \boldsymbol{U} (\boldsymbol{F} - \boldsymbol{Y}) \tag{10}$$

where $\boldsymbol{F} \in \mathbb{R}^{n \times c}$ and $\boldsymbol{Y} \in \mathbb{R}^{n \times c}$ are the label prediction matrix and labelled matrix, respectively. $\boldsymbol{L}_{\boldsymbol{W}} = \boldsymbol{D} - \boldsymbol{W}$ is the Laplacian matrix of $\boldsymbol{W}$, in which $\boldsymbol{D}$ is a diagonal matrix with $\boldsymbol{D}_{ii} = \sum_{j=1}^{n} \boldsymbol{W}_{i,j}$. λ_∞ is a large enough parameter. $\boldsymbol{U}$ is a diagonal matrix with diagonal elements are λ_∞ and 0 corresponding to the labelled and unlabelled samples, respectively. By setting the derivative of g with respect to $\boldsymbol{F}$ to zero, the label prediction matrix can be obtained directly as follows:

$$\boldsymbol{F} = (\boldsymbol{L}_{\boldsymbol{W}} + \boldsymbol{U})^{-1} \boldsymbol{U} \boldsymbol{Y} \tag{11}$$

Finally, the label of each unlabelled sample can be identified as

$$\mathrm{Label}(k) = \arg\max_{j} \boldsymbol{F}_{k,j} \tag{12}$$

Table 1 21 metrics selected from NASA datasets

Metric	Description
loc	McCabe's line count of code
v(g)	McCabe 'cyclomatic complexity'
ev(g)	McCabe 'essential complexity'
iv(g)	McCabe 'design complexity'
n	halstead total operators and operands
v	halstead 'volume'
l	halstead 'program length'
d	halstead 'difficulty'
i	halstead 'intelligence'
e	halstead 'effort'
b	halstead effort estimate
t	halstead's time estimator
lOCode	halstead's line count
lOComment	halstead's count of lines of comments
lOBlank	halstead's count of blank lines
lOCodeAndComment	lines of comment and code
uniq_Op	unique operators
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
branchCount	branch count of the flow graph

Table 2 NASA MDP datasets used in the experiment

Project	Number of metrics	Number of defective modules	Total number of modules	Percentage of defective modules, %
JM1	22	1672	7755	21.56
KC1	22	314	1192	26.34
PC3	38	134	1073	12.49
PC4	38	177	1287	13.75
PC5	39	471	1691	27.85

4 Experimental setup

In this section, we describe the experimental setup in details, including benchmark datasets, evaluation measures, and experiment design.

4.1 Benchmark datasets

In this experiment, we applied widely used open datasets from software defect prediction studies, including NASA Metrics Data Program (MDP) and Eclipse 3.4 Buggy Files [5].

4.1.1 NASA MDP: The NASA MDP repository database is the open software defect database provided by NASA. We selected the five project datasets from it as the test data. Each project represents a NASA software system or subsystem that is programmed in C or Java language, which contains the corresponding defect-marking data and various static code metrics. The repository uses a bug tracking system to record the number of defects in every software module. The static code metrics of NASA MDP repository are generated according to the software scale and complexity measurement methods, such as Halsted, LOC, and McCabe, which are closely related to the quality of software. Table 1 shows the descriptions of 21 metrics information selected from NASA datasets. Table 2 gives the brief properties of five NASA datasets, including the number of metrics, the total number of modules, the number of defective modules and the percentage of them. As suggested in [40, 41], we have cleaned these datasets and removed the identical cases and the inconsistent cases.

4.1.2 Eclipse 3.4 buggy files: According to Kim's findings [5], there are 92.27 and 95.92% bugs reports linked to changes logs, respectively, in the SWT and Debug projects in Eclipse 3.4. The SWT dataset contains 1485 Java source files, among which 43.9% files are defective and 56.1% files are defect-free. The Debug dataset contains 1065 files, of which 24.7% are defective and 75.3% files are defect-free. Table 3 shows the category and name of the main metrics in the SWT and Debug projects.

4.2 Evaluation measures

In the experiment of software defect prediction, the widely used measurements are the probability of detection (PD), the probability of false alarm (PF), F-measure and the AUC. To evaluate the performance of the prediction model, we used the confusion matrix of the predictor as shown in Table 4. Here, TP, TN, FP, and FN are the number of defective instances to be correctly classified, the number of defect-free instances to be correctly classified, the number of defect-free instances that are incorrectly classified as defective instances and the number of defective instances that are incorrectly classified as defect-free instances, respectively.

Probability of detection (pd): The ratio is the number of the correctly classified defective modules to the total number of defective modules, which is defined as $\mathrm{pd} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$.

Probability of false alarm (pf): The ratio is the number of the wrongly classified defect-free modules to the total number of defect-free modules, which is defined as $\mathrm{pf} = \mathrm{FP}/(\mathrm{FP} + \mathrm{TN})$.

Precision: The ratio is the number of the correctly classified defective modules to the number of modules that are classified as defective, which is defined as $\mathrm{precision} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$.

A good prediction model expects to achieve a high value of pd and precision. However, there exists a trade-off between them. F-measure is the harmonic mean of pd and precision. High precision and pd result in high F-measure, and it can be defined as $\mathrm{F} - \mathrm{measure} = 2 * \mathrm{pd} * \mathrm{precision}/(\mathrm{pd} + \mathrm{precision})$.

AUC (the area under receiver operating characteristic (ROC) curve) measures the area under the ROC curve. ROC curve and AUC are widely used in machine learning for performance evaluation. The ROC curve plots pf on the x-axis and pd on the y-axis. ROC curve passes through points (0,0) and (1,1) by definition.

The range of the above measurements is [0,1], and an ideal defect prediction model should have high pd, F-measure, AUC,

Table 3 Main metrics of Eclipse 3.4 datasets used in the experiment

Category	Metric name
complexity metrics	LOC
	AverageComplexity
	MaximumComplexity
object-oriented metrics	LCOM
	DIT
	WMC
	CBO
	NOC
	RFC
	NIM
	NIV
	NCM
	NCV
	IFANIN
developer metrics	change_times

Table 4 Defect prediction confusion matrix

	Predict as defective	Predict as defect-free
defective modules	TP	FN
defect-free modules	FP	TN

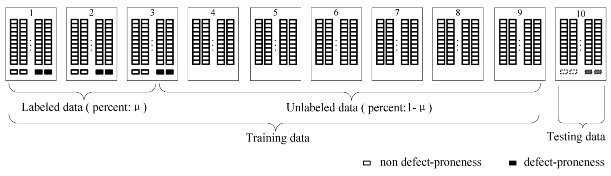

Fig. 3 *Experimental design of the semi-supervised scene*

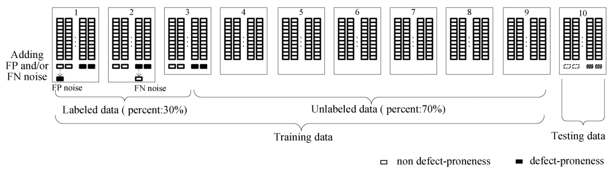

Fig. 4 *Experimental design of the noisy scene*

and low pf. In the experiment, we evaluated the performance of the semi-supervised defect prediction methods using the comprehensive evaluation measures F-measure and AUC.

4.3 Experimental design

In order to evaluate the performance of the proposed LRRSSDP approach in semi-supervised scene, we compared six representative state-of-the-art semi-supervised defect prediction methods, including iterative semi-supervised approach-fitting the fits (FTF) [41], random committee with under-sampling (ROCUS) [13], low-density separation (LDS) [16], active semi-supervised defect categorisation (ASDC) [15], non-negative sparse-based SemiBoost (NSSB) [19] and non-negative sparse graph-based label propagation (NSGLP) [20].

As shown in Fig. 3, in the experiment, we used five projects datasets in the widely used NASA MDP database as experimental data and used ten-fold cross-validation to evaluate the prediction results. We first randomly and equally partitioned the whole dataset into ten folds. We used nine folds as the training set and then used the remaining one fold as the testing set. For the training set, we randomly partitioned it into labelled and unlabelled sets according to a given labelled rate μ. In the experiment, we used three different labelled rates: 10, 20, and 30%. The random division treatment may affect the prediction performance. Therefore, we used the random division and performed prediction ten times, and reported the average F-measure and AUC in Section 5.

In order to evaluate the performance of the proposed LRRSSDP approach in a noisy scene, we also adopted the six semi-supervised methods above as compared methods. Besides, we used the SWT and Debug projects in Eclipse 3.4 as experimental data. As shown in Fig. 4, in the experiment, we used ten-fold cross-validation ten times to evaluate the prediction results. We first randomly and equally partitioned the whole dataset into ten folds. We used nine folds as the training set and then used the remaining one fold as the testing set. For the training set, we randomly partitioned it into labelled and unlabelled sets according to the labelled rate $\mu = 30\%$. In the training set, by changing the labels of the original labelled data from defective to defect-free or from defect-free to defective, we artificially added FN and FP noises to the original training dataset of SWT and Debug. For the testing set, we did not inject noise data. In the experiment, we injected noise into the training set in three cases: injecting FN noise, injecting FP noise, and injecting both FN and FP noises. The proportion of injected noise ranges from 10 to 60%. We used F-measure to evaluate the performance of semi-supervised defect prediction methods in noisy environments.

5 Experimental results and analysis

In this section, we first give the experimental results of semi-supervised learning effectiveness of the proposed approach and comparison methods on the widely used NASA MDP datasets, and then we give the experimental results of noise robustness in the two projects datasets of Eclipse 3.4.

5.1 Semi-supervised learning effectiveness experiment

Tables 5–7 show the prediction results of our approach and other compared methods on the five defect prediction projects of NASA MDP database, which are F-measure and AUC values at the labelled rates of 10, 20, and 30%, respectively. Table 8 gives the average values of the F-measure and AUC values at three labelled rates. From Tables 5–7, we can observe that LRRSSDP almost always outperforms the other compared methods for semi-supervised software defect prediction tasks. The performance improvement of LRRSSDP on the comprehensive measures of F-measure and AUC verifies that the method based on LRR can make full use of unlabelled samples to assist semi-supervised learning with labelled samples. From Table 8, we can observe that as compared with other methods, the LRRSSDP approach can, respectively, increase at least 0.02 (=0.52–0.50) and 0.02 (=0.78–0.76) on the average of F-measure and AUC.

In LRRSSDP, NSGLP, and NSSB, the construction of the data relationship graph is based on the similarity of the examples and the sparsity of their representation. Comparing LRRSSDP, NSGLP, and NSSB with other four semi-supervised defect prediction methods, FTF, ROCUS, LDS, and ASDC, we can find that, after carefully exploiting the sparsity and similarity of the examples, the LRRSSDP, NSGLP, and NSSB are able to dramatically improve the experimental performance on F-measure. Especially for LRRSSDP, even if the labelled rate is only 10%, the average performance improvement of F-measure still reaches 15.91% (=(0.51–0.44)/0.44). Their experimental performance on AUC also has the same trend. This fact suggests that constructing a low rank and sparse graph is beneficial for semi-supervised learning in software defect prediction.

Comparing LRRSSDP with NSGLP, which also uses label propagation algorithm but represents the data relationship using a non-negative sparse graph, we can find that although NSGLP may achieve the best performance on some datasets under certain label rate, the average F-measures and AUC are always much less than that of LRRSSDP. This fact verifies that the graph construction plays an important role in the graph based semi-supervised learning, and LRR is a promising method for constructing a weighted graph.

5.2 Noise robustness experiment

Fig. 5*a* shows how FN noise training set affects the prediction performance of different semi-supervised learning methods on

SWT dataset. As can be seen from the figure, the LRRSSDP approach has strong stability against FN noise. The F-measure on the original noiseless dataset is 0.75. With the increase of noise, the prediction results of LRRSSDP are still stable (F-measure value is about 0.74), even when the FN noise rate reaches 60%, the F-measure value also reaches 0.73. Considering the local neighbourhood relationship, NSGLP and NSSB also have a certain ability to resist FN noise. However, the F-measure values of the other compared methods drop significantly when the noise ratio increases gradually. When the noise ratio is over 40%, the performance of these methods is even lower than that of the random predictor.

Fig. 5*b* shows how FP noise training set affects the prediction performance of different semi-supervised learning methods on SWT dataset. As can be seen from the figure, the LRRSSDP approach has strong stability against FP noise (with F-measures around 0.74), even when the FP noise rate reaches 60%, the F-measure value also reaches 0.73. The compared methods NSGLP and NSSB also have a certain ability to resist FP noise, but other methods do not have such appearance.

Fig. 5*c* shows how FN and FP noise training sets affect the prediction performance of different semi-supervised learning methods on SWT dataset. As can be seen from the figure, the LRRSSDP approach has strong stability against FN and FP noises

Table 5 Average F-measure and AUC of all compared methods on NASA MDP datasets when $\mu = 0.1$

Dataset	Measure	FTF	ROCUS	LDS	ASDC	NSSB	NSGLP	LRRSSDP
JM1	F-measure	0.38	0.42	0.39	0.40	0.44	0.46	0.47
	AUC	0.65	0.74	0.71	0.71	0.71	0.72	0.75
KC1	F-measure	0.47	0.49	0.51	0.51	0.54	0.56	0.57
	AUC	0.62	0.66	0.68	0.68	0.71	0.72	0.73
PC3	F-measure	0.27	0.28	0.29	0.30	0.38	0.38	0.37
	AUC	0.62	0.63	0.64	0.62	0.65	0.66	0.66
PC4	F-measure	0.40	0.40	0.41	0.42	0.42	0.43	0.47
	AUC	0.70	0.71	0.73	0.72	0.74	0.76	0.78
PC5	F-measure	0.52	0.53	0.56	0.57	0.62	0.64	0.65
	AUC	0.76	0.80	0.79	0.80	0.80	0.81	0.82
Avg.	F-measure	0.41	0.42	0.43	0.44	0.48	0.49	0.51
	AUC	0.67	0.71	0.71	0.71	0.72	0.73	0.75

Table 6 Average F-measure and AUC of all compared methods on NASA MDP datasets when $\mu = 0.2$

Dataset	Measure	FTF	ROCUS	LDS	ASDC	NSSB	NSGLP	LRRSSDP
JM1	F-measure	0.39	0.42	0.45	0.41	0.46	0.47	0.53
	AUC	0.67	0.75	0.76	0.73	0.73	0.74	0.77
KC1	F-measure	0.49	0.50	0.52	0.53	0.55	0.56	0.57
	AUC	0.62	0.69	0.69	0.69	0.73	0.75	0.75
PC3	F-measure	0.28	0.29	0.31	0.32	0.39	0.39	0.39
	AUC	0.70	0.67	0.68	0.66	0.70	0.73	0.74
PC4	F-measure	0.41	0.41	0.42	0.44	0.43	0.44	0.46
	AUC	0.75	0.75	0.76	0.75	0.76	0.80	0.81
PC5	F-measure	0.52	0.53	0.57	0.59	0.63	0.64	0.66
	AUC	0.78	0.82	0.81	0.81	0.81	0.83	0.82
Avg.	F-measure	0.42	0.43	0.45	0.46	0.49	0.50	0.52
	AUC	0.70	0.74	0.74	0.73	0.75	0.77	0.78

Table 7 Average F-measure and AUC of all compared methods on NASA MDP datasets when $\mu = 0.3$

Dataset	Measure	FTF	ROCUS	LDS	ASDC	NSSB	NSGLP	LRRSSDP
JM1	F-measure	0.40	0.43	0.45	0.42	0.47	0.48	0.56
	AUC	0.68	0.76	0.78	0.74	0.78	0.80	0.83
KC1	F-measure	0.50	0.51	0.53	0.54	0.56	0.57	0.57
	AUC	0.65	0.70	0.72	0.71	0.76	0.77	0.77
PC3	F-measure	0.30	0.30	0.32	0.33	0.39	0.39	0.40
	AUC	0.71	0.69	0.72	0.72	0.74	0.75	0.78
PC4	F-measure	0.42	0.41	0.43	0.45	0.44	0.45	0.49
	AUC	0.78	0.77	0.77	0.79	0.79	0.81	0.82
PC5	F-measure	0.53	0.55	0.58	0.61	0.65	0.66	0.69
	AUC	0.79	0.83	0.82	0.82	0.82	0.84	0.85
Avg.	F-measure	0.43	0.44	0.46	0.47	0.50	0.51	0.54
	AUC	0.72	0.75	0.76	0.76	0.77	0.79	0.81

Table 8 Total average F-measure and AUC of all compared methods on NASA MDP datasets under three labelled rates

Measure	FTF	ROCUS	LDS	ASDC	NSSB	NSGLP	LRRSSDP
F-measure	0.42	0.43	0.45	0.46	0.49	0.50	0.52
AUC	0.70	0.73	0.74	0.73	0.75	0.76	0.78

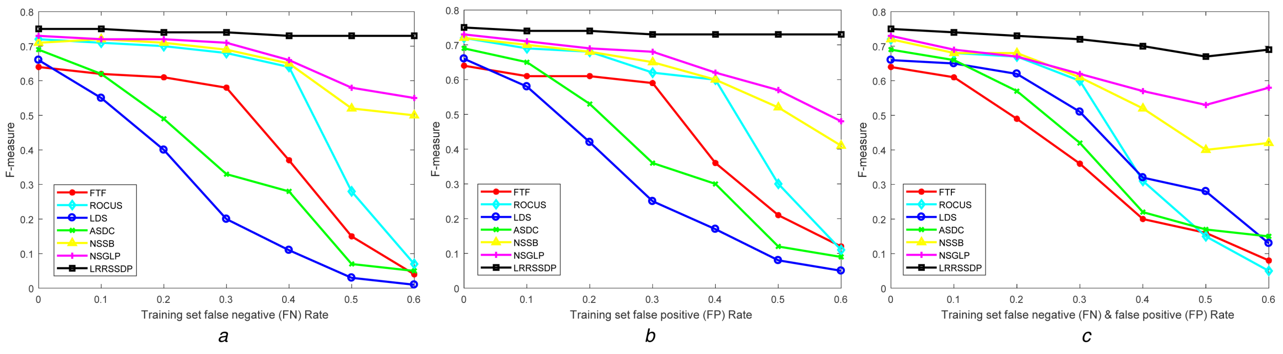

Fig. 5 *Impact of noises on defect prediction on SWT dataset*
(a) F-measure for FN training set, ***(b)*** F-measure for FP training set, ***(c)*** F-measure for FN and FP training set

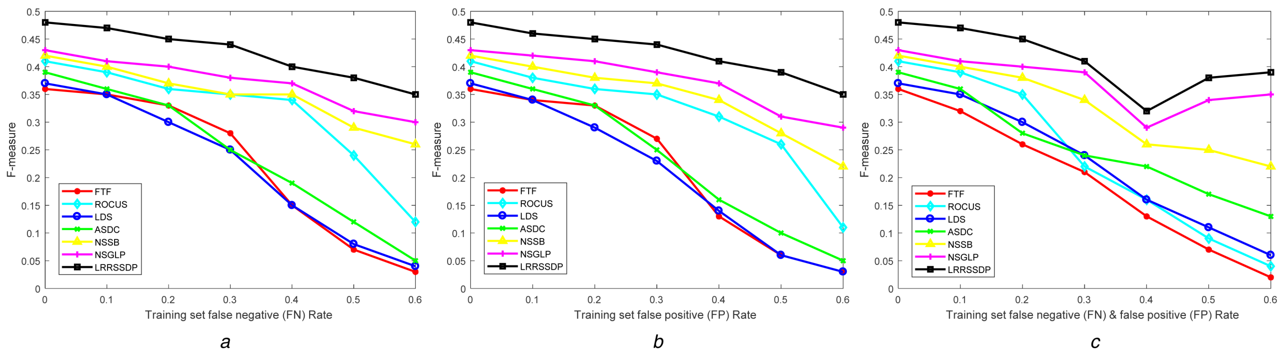

Fig. 6 *Impact of noises on defect prediction on Debug dataset*
(a) F-measure for FN training set, ***(b)*** F-measure for FP training set, ***(c)*** F-measure for FN and FP training set

(with F-measures around 0.71), even when the rate of FN and FP noise reaches 60%, the F-measure value also reaches 0.67. The compared methods NSGLP and NSSB also have a certain ability to resist FN and FP noises, but other methods do not have such appearance.

The three charts of Fig. 6 show how the three kinds of noise training sets affect the prediction performance of different semi-supervised learning methods on Debug dataset. Similar to the experimental results on the SWT dataset, the LRRSSDP approach has strong stability on FN noise training set and FP noise training set. However, the prediction results of the LRRSSDP approach on FN and FP mixed noise datasets fluctuate at the noise rate around 40%, but its performance has been better than other comparison methods. By analysing the reason, we can find that the number of defect instances in Debug dataset is small. When the FN noise rate or the FP noise rate is relatively large, the number of the noiseless data is small, and the LRRSSDP approach has no ability to resist the impact of noise data. Therefore, as the noise ratio increases, the prediction performance decreases gradually. On the training set with both FP and FN noise on Debug dataset, when the noise ratio reaches 40%, the F-measure appears minimum and the prediction performance is close to the random predictor.

According to Figs. 5 and 6, we can find that the LRRSSDP approach has strong stability against FN and FP noise. This is because that the LRR has the ability to resist noise, and our approach considers the local neighbourhood relationship, which is advantageous to eliminate abnormal noise points.

6 Threats to validity

We note some threats to the validity of our work in this section.

Construct validity: First of all, our prediction model covered only a small number of datasets from specific sources, namely, NASA MDP and Eclipse. The NASA MDP database used in our experiment uses static code metrics, although the validity of static code metrics has been widely debated, we adopt this metric because they are useful, easy to use, and widely used. Eclipse 3.4 is an open source project, and the types of noises introduced by open source software developers may be different from those introduced by employees in an effectively managed software organisation. We cannot generalise from current research to other datasets, since the attributes of these datasets may be different. Second, there are some potential problems, such as whether the defects are incompletely fixed, and whether the defects are recorded. Since we know that many researchers have used the open dataset studied in their studies, we consider that defects have been fully revealed and fixed in our study. Besides, we have cleaned the NASA MDP datasets and removed the identical cases and the inconsistent cases in them.

Internal validity: The orders of magnitude of the attributes measure values vary widely. In our method, we suppose the attributes have the same importance to building the defect model. We normalise the samples to have unit l^2-norm before constructing the LRR graph. Therefore, the internal validity threat caused by these attributes should be minor. A second issue affecting the internal validity is the value setting of the balance factor. We rely on the relationship among the LRR, noises and neighbour distance to empirically set such value. In other domain, the value setting of balance factor also depends on experience mostly. Finally, the simulated noise data used in our experiment may not reflect the realistic noise patterns. The noise injected in the training set is random, but in practice, the occurrences of some noises actually follow certain rules.

External validity: Our study to evaluate the proposed defect prediction model involves five systems from NASA MDP and two projects from Eclipse with different characteristics. Therefore, we claim that these samples extracted from real world and non-trivial applications include a credible number of defect categories. Moreover, the metrics we use cover major source code properties like size, readability, complexity. Despite these observations, we cannot guarantee that our findings apply to other metrics or systems, specifically to systems which are not measured by static code metric or to systems from different domains in industrial practice.

Since the domain of the applications and development teams of the projects in our study may be different from that of many other companies, it might be possible that our results do not generalise to the projects of other companies. Our method should encourage more researchers to conduct similar studies and develop more practical prediction models. Our study would be replicated with more projects and different metrics, and replaced by more sophisticated methods.

7 Conclusions and future works

Prediction of the defect-proneness of software modules is important for improving the quality of a software system since it is an efficient means to relieve the burden on software code inspection or testing. Traditional models of software defect prediction are built by supervised learning. The drawback of supervised learning method is that it cannot build effective prediction models when the historical defect labels of software modules are limited. When the training data are noisy, the performance of defect prediction model is often unsatisfactory. In this paper, we propose a novel LRRSSDP for semi-supervised learning in software defect prediction, which uses not only a few labelled data but also abundant unlabelled ones to improve the generalisation capability. LRRSSDP uses the nearest-neighbour distance and LRR to construct the relation graph of the defect data and uses the graph-based semi-supervised learning method to predict the defect proneness of unlabelled data.

As compared with several state-of-the-art representative semi-supervised software defect prediction methods, the experiments on the NASA MDP dataset show that the proposed LRRSSDP approach achieves the best performance with the highest average F-measure and AUC values. The experiments on the Eclipse noise dataset show that the LRRSSDP approach maintains the stability of the prediction performance in noisy environments. Experimental results demonstrate the effectiveness of our approach for semi-supervised defect prediction tasks, especially in noisy environments.

In the future work, how to construct an effective defect prediction model with very few historical training samples or without historical data will be a worthwhile research direction. Using a single sample to build a semi-supervised defect prediction model and using cross-company project data to improve the performance of defect prediction model are interesting research topics.

8 References

[1] Catal, C., Diri, B.: 'A systematic review of software fault prediction studies', *Expert Syst. Appl.*, 2009, **36**, (4), pp. 7346–7354

[2] Hall, T., Beecham, S., Bowes, D., *et al.*: 'A systematic literature review on fault prediction performance in software engineering', *IEEE Trans. Softw. Eng.*, 2012, **38**, (6), pp. 1276–1304

[3] Shi, Y., Li, M., Arndt, S., *et al.*: 'Metric-based software reliability prediction approach and its application', *Empir. Softw. Eng.*, 2017, **22**, (4), pp. 1579–1633

[4] Jing, X.Y., Wu, F., Dong, X., *et al.*: 'An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems', *IEEE Trans. Softw. Eng.*, 2017, **43**, (4), pp. 321–339

[5] Kim, S., Zhang, H., Wu, R., *et al.*: 'Dealing with noise in defect prediction'. Proc. Int. Conf. Software Engineering, Zurich, Switzerland, 2011, pp. 481–490

[6] Zhu, X.: 'Semi-supervised learning with graphs'. PhD thesis, Carnegie Mellon University, 2005

[7] Culp, M., Michailidis, G.: 'Graph-based semisupervised learning', *IEEE Trans. Pattern Anal. Mach. Intell.*, 2008, **30**, (1), pp. 174–179

[8] Li, S., Fu, Y.: 'Low-rank coding with b-matching constraint for semi-supervised classification'. Proc. Int. Joint Conf. Artificial Intelligence, Beijing, China, 2013, pp. 1472–1478

[9] Fei, L., Xu, Y., Fang, X., *et al.*: 'Low rank representation with adaptive distance penalty for semi-supervised subspace classification', *Pattern Recognit.*, 2017, **67**, pp. 252–262

[10] Seliya, N., Khoshgoftaar, T.M.: 'Software quality estimation with limited fault data: a semi-supervised learning perspective', *Softw. Qual. J.*, 2007, **15**, (3), pp. 327–344

[11] Seliya, N., Khoshgoftaar, T.M.: 'Software quality analysis of unlabeled program modules with semisupervised clustering', *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, 2007, **37**, (2), pp. 201–211

[12] Catal, C., Diri, B.: 'Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction', *Expert Syst.*, 2009, **26**, (5), pp. 458–471

[13] Jiang, Y., Li, M., Zhou, Z.-H.: 'Software defect detection with ROCUS', *J. Comput. Sci. Technol.*, 2011, **26**, (2), pp. 328–342

[14] Li, M., Zhang, H., Wu, R., *et al.*: 'Sample-based software defect prediction with active and semi-supervised learning', *Autom. Softw. Eng.*, 2012, **19**, (2), pp. 201–230

[15] Thung, F., Le, X.B.D., Lo, D.: 'Active semi-supervised defect categorization'. Proc. Int. Conf. Program Comprehension, Florence, Italy, 2015, pp. 60–70

[16] Catal, C.: 'A comparison of semi-supervised classification approaches for software defect prediction', *J. Intell. Syst.*, 2014, **23**, (1), pp. 75–82

[17] Ma, Y., Pan, W., Zhu, S., *et al.*: 'An improved semi-supervised learning method for software defect prediction', *J. Intell. Fuzzy Syst.*, 2014, **27**, (5), pp. 2473–2480

[18] Abaei, G., Selamat, A., Fujita, H.: 'An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction', *Knowl.-Based Syst.*, 2015, **74**, pp. 28–39

[19] Wang, T.J., Zhang, Z.W., Jing, X.Y., *et al.*: 'Non-negative sparse-based SemiBoost for software defect prediction', *Softw. Test. Verif. Reliab.*, 2016, **26**, (7), pp. 498–515

[20] Zhang, Z.W., Jing, X.Y., Wang, T.J.: 'Label propagation based semi-supervised learning for software defect prediction', *Autom. Softw. Eng.*, 2017, **24**, (1), pp. 47–69

[21] Khoshgoftaar, T.M., Seliya, N.: 'The necessity of assuring quality in software measurement data'. Proc. Int. Conf. Software Metrics, Chicago, IL, USA, 2004, pp. 119–130

[22] Tang, W., Khoshgoftaar, T.M.: 'Noise identification with the k-means algorithm'. Proc. Int. Conf. Tools with Artificial Intelligence, FL, USA, 2004, pp. 373–378

[23] Bird, C., Bachmann, A., Aune, E., *et al.*: 'Fair and balanced?: bias in bug-fix datasets'. Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering, Amsterdam, Netherlands, 2009, pp. 121–130

[24] Bachmann, A., Bird, C., Rahman, F., *et al.*: 'The missing links: bugs and bug-fix commits'. Proc. of the Eighteenth ACM SIGSOFT Int. Symp. on Foundations of Software Engineering, Santa Fe, NM, USA, 2010, pp. 97–106

[25] Kim, S., Zimmermann, T., Whitehead, E.J.Jr, *et al.*: 'Predicting faults from cached history'. Proc. Int. Conf. Software Engineering, Minneapolis, MI, USA, 2007, pp. 489–498

[26] Nguyen, T.H., Adams, B., Hassan, A.E.: 'A case study of bias in bug-fix datasets'. 17th Working Conf. on Reverse Engineering, Beverly, CA, USA, 2010, pp. 259–268

[27] Rahman, F., Posnett, D., Herraiz, I., *et al.*: 'Sample size vs. Bias in defect prediction'. Proc. of the 9th Joint Meeting on Foundations of Software Engineering, Saint Petersburg, Russia, 2013, pp. 147–157

[28] Herzig, K., Just, S., Zeller, A.: 'It's not a bug, it's a feature: how misclassification impacts bug prediction'. Proc. Int. Conf. Software Engineering, San Francisco, CA, USA, 2013, pp. 392–401

[29] Tantithamthavorn, C., McIntosh, S., Hassan, A.E., *et al.*: 'The impact of mislabeling on the performance and interpretation of defect prediction models'. Proc. Int. Conf. Software Engineering, San Francisco, CA, USA, 2013, pp. 812–823

[30] Zhuang, L., Gao, S., Tang, J., *et al.*: 'Constructing a nonnegative low-rank and sparse graph with data-adaptive features', *IEEE Trans. Image Process.*, 2015, **24**, (11), pp. 3717–3728

[31] Feng, J., Lin, Z., Xu, H., *et al.*: 'Robust subspace segmentation with block-diagonal prior'. Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 3818–3825

[32] Liu, G., Yan, S.: 'Latent low-rank representation for subspace segmentation and feature extraction'. Proc. Int. Conf. Computer Vision, Barcelona, Spain, 2011, pp. 1615–1622

[33] Zhang, H., Lin, Z., Zhang, C., *et al.*: 'Robust latent low rank representation for subspace clustering', *Neurocomputing*, 2014, **145**, pp. 369–373

[34] Candes, E.J., Li, X., Ma, Y., *et al.*: 'Robust principal component analysis', *J. ACM (JACM)*, 2011, **58**, (3), pp. 11–47

[35] Siming, W., Zhouchen, L.: 'Analysis and improvement of low rank representation for subspace segmentation', arXiv preprint arXiv:1107.1561, 2011

[36] Fang, X., Xu, Y., Li, X., *et al.*: 'Learning a nonnegative sparse graph for linear regression', *IEEE Trans. Image Process.*, 2015, **24**, (9), pp. 2760–2771

[37] Fang, X., Xu, Y., Li, X., *et al.*: 'Robust semi-supervised subspace clustering via non-negative low-rank representation', *IEEE Trans. Cybernetics*, 2016, **46**, (8), pp. 1828–1838

[38] Jing, X.Y., Qi, F., Wu, F., *et al.*: 'Missing data imputation based on low-rank recovery and semi-supervised regression for software effort estimation'. Proc. Int. Conf. Software Engineering, Austin, TX, USA, 2016, pp. 607–618

[39] Zhou, D., Bousquet, O., Lal, T.N., *et al.*: 'Learning with local and global consistency', Adv. Neural Inf. Process. Syst., Vancouver, Canada, 2004, pp. 321–328

[40] Shepperd, M., Song, Q., Sun, Z., *et al.*: 'Data quality: some comments on the NASA software defect datasets', *IEEE Trans. Softw. Eng.*, 2013, **39**, (9), pp. 1208–1215

[41] Lu, H., Cukic, B., Culp, M.: 'An iterative semi-supervised approach to software fault prediction'. Proc. Int. Conf. Predictive Models in Software Engineering, Banff, Canada, 2011, Article No. 15