DOI: http://dx.doi.org/10.26483/ijarcs.v9i1.5476

ISSN No. 0976-5697

Volume 9, No. 1, January-February 2018

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

SOFTWARE DEFECT PREDICTION USING DEEP BELIEF NETWORK WITH L1-REGULARIZATION BASED OPTIMIZATION

Manjula C
Associate Professor, MCA Department,
PESIT- BSC, Karnataka,India

Lilly Florence
Professor, MCA Department,
Adiyamman College of Engineering,
Tamil Nadu,India

Abstract: Software defect prediction is considered as most interesting area for researchers in the field of software engineering. This process of defect prediction, identifies the bug during automated testing process which prevents the development of faulty software module. According to this process, previous archives of software modules are considered for analyzing the quality based on data mining and machine leaning concepts, which identifies the faults in software modules. Several techniques have been presented for software defect prediction using data mining techniques and machine learning techniques but achieving desired accuracy in performance is a challenging task for researchers. To address this issue, in this work we have presented a new approach deep learning for software defect prediction by combining genetic algorithm optimization process for feature subspace reduction and deep belief network for pattern learning. Deep belief networks are further improved by applying L1-regularization scheme resulting in better learning process by reducing the overfitting error. This combined model is implemented on SPIE lab software defect database. An extensive experimental study is carried out which shows that proposed approach achieves higher accuracy when compared with state-of-the-art software defect prediction techniques.

Keywords: Software Metrics, Software Quality, Software Defect Prediction, Deep Belief Network, Genetic Algorithm, L1-reguralization

1. INTRODUCTION

Modern software modules and development techniques are growing rapidly. Software based applications are widely adopted in real-time & daily life scenarios. Maintaining the quality and reliability is a critical issue in the software industries and the growth of industry also depends on the quality of software applications. Generally, huge software applications are complex and failure –prone in nature which is caused due to code defect/bugs during code development [1]. In order to deal with this issue, researchers have found that early prediction of bugs or code defects can improve the quality of the software application. Various researches have been carried out for early bug prediction which is known as software defect prediction [2]. According to this process of bug prediction, classifiers are built which predict the defective area in the code which can help to the code reviewers to allocate their effort for defected part instead of analyzing complete code. This process is considered as most cost-effective process of software quality management.

These techniques of software defect prediction are based on the software metrics which can be computed from the different levels of the software such as: file-level, component-level, process-level, class-level, quantitative level and method-level, which are used for software testing purpose. Further, software defect prediction can be performed by using statistical methods, machine learning techniques, artificial intelligence techniques etc. In general, software defect prediction technique contains two main phases: feature extraction and classifier model development using machine learning algorithm. Feature extraction techniques are based on the development of discriminative feature extraction technique for better defect prediction. Features can be McCabe features, Halstead features and CK features for object-oriented programs. Meanwhile, various machine learning algorithms such as SVM (Support Vector Machine) [3], ANN (Artificial Neural Network) [3], DT (Decision Tree) [4] and Naïve Bayes etc. have been developed for software defect prediction. However, existing techniques suffer from multiple issues such as classification performance and dataset training error. resulting in the degraded performance analysis for software defect prediction. Unfortunately, performance of these classification techniques still remain unaddressed issues for researchers. In order to overcome this, researchers have focused on the development of new sophisticated techniques based on the machine learning. Recently, Mussa et al. [5] presented software defect prediction technique for software bug prediction where particle swarm optimization and genetic algorithm are used for optimization process. Software defect prediction identifies defective or non-defective identities and attempts to reduce the misclassification rate. For better performance, sophisticated technique is used where in feature selection and two-stage cost sensitive learning are incorporated. Similarly, Ryu et al. [6] also introduced a hybrid approach where two classifiers Nearest –Neighbor and Naïve Bayes are used for achieving local and global information of the data. These techniques pose various challenges such as configuration and validation of sophisticated techniques, optimization etc. which may lead to improper identification of software bugs. Hence, a general and efficient framework for software defect prediction is required.

In order to develop the software defect prediction model, data mining techniques have been adopted widely and for analysis, open source software defect repositories such as NASA software defect datasets [8] are used. These datasets contains various metrics values such as LOC (Line of code), program length and number of blank lines etc. Based on these attributes, a machine learning approach is implemented which learns the complete pattern and returns a trained datasets which is further used for testing purpose. By considering data mining technique as base, Arar et al. [9] developed a new approach for software defect prediction using Naïve Bayes classification scheme. To improve the analysis, feature dependent naïve Bayes classifier model is developed where features are computed based on the dependency on the other features.

For software defect prediction artificial intelligence also plays important role. Quah et al. [10] discussed about the software quality assessment using neural network modeling by focusing on software maintenance and reliability. To achieve this objective, ward neural network and general regression neural network are presented. Similarly, Kanmani et al. [11] also presented neural network based approach for software defect prediction where two neural network models are used along with object-oriented metrics. This study shows that neural networks achieve better performance in terms of defect prediction accuracy. Further, performance of neural network can be enhanced by applying deep neural network techniques. However, deep neural network models are considered as prone to the overfitting error which may lead to the inaccurate performance of software defect prediction. These studies show that considerable amount of research work has been carried out in this field of software defect prediction but still prediction accuracy remains a challenging task along with the development of general framework for software defect prediction.

In order to overcome these issues, here we present an improved technique of software defect prediction. As discussed before neural network gives better performance which can be further improved by using Deep Neural Network (DNN) or Deep Belief Networks (DBN) due to its significant learning scheme. However, DBN suffers from the issue of overfitting. Hence, to deal with this issue, we present L1-reguralization scheme. First we present a classification study using DBN. Next, Genetic algorithm is applied for feature optimization. The optimized features are further processed through DBN using L1-regularization scheme.

Rest of the article is organized as follows: Section 2describes recent studies in the field of software defect prediction, proposed model is presented in section 3, experimental analysis is demonstrated in section 4and finally concluding remarks are given in section 5.

2. LITERATURE REVIEW

In this section we review the studies based on the software defect prediction techniques. The problem of software defect prediction can be considered as learning problem in software engineering. Several researches have been reported on this topic. He et al. [2] discussed about the software defect prediction techniques. Generally, software defect prediction techniques utilize the historical data for further software bug prediction but due to insufficient dataset the desired performance cannot be achieved. This issue can be addressed using metric set based evaluation for training and testing dataset. First of all, authors created predictor model based on the six classifiers which are later validated using Top-k metrics using statistical measurement. A general framework is needed for software defect prediction which can be used as benchmark for SDP. Lessmann et al. [12] discussed about this issue and presented an experimental study using 22 classifiers which are tested for 10 publicly available databases. This technique of software defect prediction depends on the feature extraction techniques. Along with feature extraction, feature selection and optimization plays important role. Several studies have been presented on feature selection and optimization techniques. In [13], Chen et al. presented a combined model for software defect prediction where feature selection and optimization are performed. Optimal feature selection is considered as most challenging task. Hence authors presented search based feature selection techniques. Further, optimization process is carried out by formulating a multi-objective optimization problem. Similarly, Hosseini et al. [14] also focused on the feature selection process for improving the Optimization. In this work, nearest neighbor filtering scheme is implemented along with genetic algorithm which helps to generate the accurate validation sets. These sets can be considered for training and testing dataset.

However, various techniques have been discussed for software defect prediction but time and complexity remains challenging and unaddressed issues. Sabharwal et al. [15] presented a low complex technique for feature selection for long search space. For this purpose, feature ranking based scheme is implemented which performs search space reduction and selection of feature subset. This process performs both the tasks simultaneously which reduces the computation time. Maua et al. [16] discussed about feature optimization scheme using multi-population genetic algorithm. In this process, multi-level selection algorithms are involved which require additional optimization process. To overcome this, authors presented multi-level genetic algorithm for SDP by using different stages of selection. In this process, colonization and migration operators are considered along with multi-objective evolutionary algorithm. Afzal et al. [17] focused on benchmarking the feature extraction technique and reviewed feature subset selection techniques. In this work, defect prediction experiments are conducted by using information gain attribute ranking (IG); Relief (RLF); principal component analysis (PCA); correlation-based feature selection (CFS); consistency-based subset evaluation (CNS); wrapper subset evaluation (WRP); and an evolutionary computation method, genetic programming (GP) for PROMISE dataset where C4.5 and naïve Bayes (NB) classifiers are used for measuring the classification performance.

Other than feature selection and feature optimization techniques, machine learning is also important stage for software defect detection and prediction. Researchers have presented various promising techniques for software defect prediction using machine learning techniques. Li et al. [18] presented convolutional neural network based scheme for using effective feature generation and learning scheme. According to this technique, initially tokens are generated and encoded into numerical vectors. Later, these encoded vectors are fed to the convolutional network for learning process. According to Wang et al. [19] software defect prediction also depends on the historical dataset and software structure pattern analysis. Sometimes, software datasets suffer from class imbalance problem where learning becomes a very complex task. This issue is addressed by using multiple kernel learning algorithm which helps to map the historical defect data into a higher dimensional feature space and express better and later ensemble learning can be implemented which uses series of weak classifiers to reduce the majority class and helps to achieve better prediction performance. In [20], authors presented metric learning based model for software defect prediction. Tomar et al. [21] studied about software defect prediction and found that efficient software defect predictor model is needed for improving the software defect prediction which can identify the bugs in the software modules without affecting the overall performance and working of the software module. As discussed before, learning becomes a crucial step where data distribution is random and imbalanced in nature.

Computational complexity and implementation cost is a challenging task for researchers. Conventional techniques require more time and human effort for software testing. Hryszkoet al. [22] discussed about complexity issues in software defect prediction in industries and presented a comprehensive study on cost effectiveness for software defect prediction. An extensive experimental study is carried out for defect prediction and cost effectiveness is analyzed. Study shows that early prediction can avoid the degraded software development and can reduce the implementation cost for software defect prediction model.

Above discussed studies are mainly focused on the feature extraction, feature selection, optimization and learning techniques. Feature extraction process includes the analysis of all metrics for pattern learning. Feature extraction and selection schemes are discussed which shows improvement in software defect prediction by selecting optimal features during computation. Finally, machine learning schemes are discussed which are used for pattern learning and classification. Various techniques have been presented in the literature to improve the software defect prediction performance, but, development of techniques with significant accuracy, performance, less-complexity still remain as challenging task. There is a need to develop an efficient model to deal with these issues. Next section discusses an approach proposed by the authors for addressing these issues.

3. PROPOSED MODEL

This section presents details about the proposed model for software defect prediction. From the previous section it is clear that artificial intelligence is a promising technique for software bug prediction. Hence, we focus on an improved artificial intelligence model known as Deep Belief Neural network (DBN).The process of DBN suffers from the overfitting issue which may lead to improper classification. To overcome this, we use L1-regularization scheme which helps to reduce the overfitting error. Moreover, proposed model also includes genetic algorithm optimization process. The complete proposed approach can be categorized as: (a) Genetic algorithm optimization (b) DBN implementation (c) L1- regularization modeling in DBN. .

3.1 Genetic Algorithm:

First of all, we study about genetic algorithm and its implementation for software defect dataset. Genetic algorithm is widely adopted for various optimization and feature selection problems.

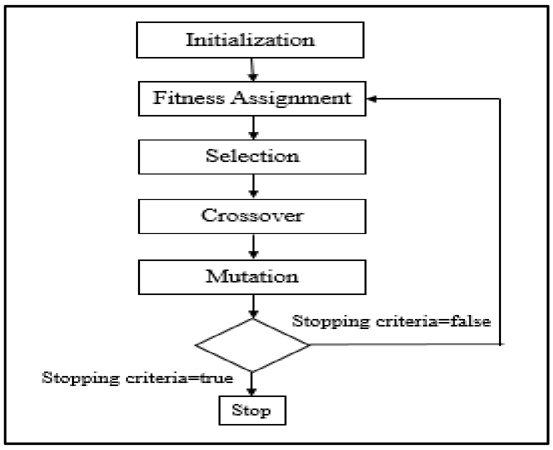

Fig..1 Genetic algorithm

Genetic algorithm is described as heuristic search and optimization approach which is inspired by nature. Development of genetic algorithm contains various stages. In this section we briefly describe various stages of genetic algorithm. This technique is evolutionary optimization technique which initiates with the initial population and tends to achieve global optimal solution for the given problem and stops when the desired optimal criteria parameters are achieved.

A basic architecture model of genetic algorithm is depicted in Fig 1. GA contains multiple stages to achieve the optimal solution which are as follows: (a) initialization (b) fitness assignment (c) selection (d) crossover (e) mutation.

(a) ***Initialization***:

This is the first step of genetic algorithm where individuals are initialized and an initial population is generated. At these stage, all initial individual and randomly generated and considered as initial population which is used for further analysis.

(b) Fitness assignment

After generating the initial population, next task is to assign fitness value to each individual. In order to perform this, we apply rank based method for fitness computation which helps to sort the selection error. This sorting process is useful for identifying the lowest selection error and its corresponding individual. Fitness assignment can be denoted as:

$$\varphi(i) = k.\mathcal{R}(i) \qquad i = 1,2,3, \ldots\ldots N \qquad (1)$$

Here k denotes selective pressure whose values are fixed between 1 to 2. Generally, greater value will result in fittest solution and $\mathcal{R}(i)$ denotes the rank of individuals.

(c) Selection

In next phase, selection operation is performed where selection operator selects the individuals which can be combined and used for next generation formulation. According to this process, individuals which are have good fitness values, are considered to survive as they are more fitted to the environment. Hence, this section is made based on the individual's fitness level. Here, half of the population is selected for further operation.

(d) Crossover

In previous stage, half of the population is selected which is used in this process where crossover is performed. Crossover operator recombines the best selected population and generates new population. In this process, randomly two individuals are considered and combinedto generate four off-springs for new population. This process continues until the size of new population is equal to the old one.

(e) Mutation

Crossover generates offspring which may be very similar to the parents. This may result in low diversity in the process of mutation. Mutation operator randomly changes some feature parameters of offspring to avoid the diversity issues.

These steps are performed until the best optimal solution is achieved for given problem. In this work we have considered, software metrics as input parameter. These metrics are processed through the above mentioned process and optimal solution is obtained. Here, it is assumed that *a priori* information of potential areas of the given feature set is already known based on the assumption initial population can be generated such that it shall cover all points and areas of given input set. At this stage, our main aim is to reduce the search space during computation. The process of search space reduction is divided into two main categories: (a) initial solution generation and (b) reduced feature subset generation. In order to improve the performance, initial solutions are generated using filtering techniques as information gain, Gini index, correlation and gain ratio. This process of filtering and feature subset reduction is presented in Fig 2.

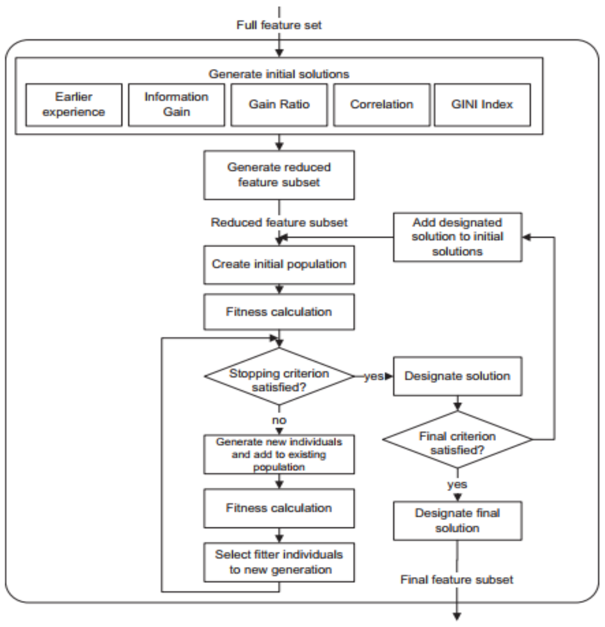
Full feature set
Generate initial solutions
Earlier experience
Information Gain
Gain Ratio
Correlation
GINI Index
Generate reduced feature subset
Reduced feature subset
Add designated solution to initial solutions
Create initial population
Fitness calculation
Stopping criterion satisfied?
yes
Designate solution
no
Generate new individuals and add to existing population
Final criterion satisfied?
yes
Fitness calculation
Designate final solution
Select fitter individuals to new generation
Final feature subset

Fig.2 Filtering and Feature Subset Reduction

3.2 Deep belief network for learning

After feature selection and dimension reduction, we apply Deep Belief Network (DBN) for pattern learning. In this section, we discuss about DBN along with Restricted Boltzmann Machines (RBMs).

3.2.1 Restricted Boltzmann Machines

RBM are considered as a part of neural networks which is composed by two neuron layers as visible and hidden layer. In these layers, learning phase is conducted by using unsupervised learning. In RBM, connections between neurons and same layer are not allowed unlike classical Boltzmann Machine. The architecture of RBM contains a visible layer v with m units and a hidden layer h with n number of units. The input data matrix W is $m \times n$ which models the weight between hidden and visible layer where w_{ij} denotes the weights between layers h_j and v_i, where it is assumed that hidden layers are denoted by h and visible layers denoted by v. For this machine, energy function can be given as:

$$E(v,h) = \sum_{i=1}^{m}\sum_{j=1}^{n} v_i h_j w_{ij} - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m} a_i v_i \qquad (2)$$

Where a and b denotes biases for visible and hidden layers. The probability of visible and hidden layer is computed as:

$$P(v,h) = \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \qquad (3)$$

With the help of this modeling, deep belief network learning is presented. RBM parameters can be optimized by applying gradient scaling on the training pattern's log-likelihood data. Let us consider that a training sample is given in the form of visible unit, it probability can be computed as follows:

$$P(v) = \frac{\sum_h e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \qquad (4)$$

Weights and biases varies for each iteration, hence it is necessary to compute the updated weights and biases with the help of following derivatives:

$$\begin{aligned} \frac{\partial \log P(v)}{\partial w_{ij}} &= E[h_j v_i]^{data} - E[h_j v_i]^{model} \\ \frac{\partial \log P(v)}{\partial a_i} &= v_i - E[v_i]^{model} \\ \frac{\partial \log P(v)}{\partial b_j} &= E[h_j]^{data} - E[h_j]^{model} \end{aligned} \qquad (5)$$

Where $E[.]$ Denotes the expectation operations, $E[.]^{data}$ denotes data driven probability and $E[.]^{model}$ denotes reconstructed data driven probability. In general, $E[h_j v_i]^{data}$ can be computed as:

$$E[hv]^{data} = P(h|v)v^T \qquad (6)$$

$P(h|v)$ denotes the probability of training data vector v as $P(h_j = 1|v) = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + b_j\right)$ this denotes logistic sigmoid function . Similarly, testing data can be denoted as $P(h_j = 1|h) = \sigma(\sum_{i=1}^{n} w_{ij} v_i + a_j)$. Various studies show that DBN suffer from overfitting error which may lead to faulty prediction during classification. In order to deal with this issue, here we include L1-regularization model to improve the learning by reducing overfitting error. Let us consider that a learning task is given with M training sequences whose training instances are $\{(x^{(i)}, y^{(i)}), i = 1, \ldots, M\}$ where $x^{(i)} \in \mathbb{R}^N$ denotes the N dimensional feature vector and $y^{(i)} \in \{0,1\}$ denotes class label. For these feature vectors and classes, logistic regression model can be expressed as:

$$p(y = 1|x; \theta) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)} \qquad (7)$$

θ denotes the l1 regularized regression model parameters and $\sigma(.)$denotes sigmoid function. Furthermore, maximum a posteriori (MAP) estimate of the parameters of θ can be given as:

$$\min_{\theta} \sum_{i=1}^{M} -log\, p(y^{(i)}|x^{(i)}; \theta) + \beta\|\theta\|_1 \qquad (8)$$

This optimization problem can be referred as L1 regularization which can be further solved by using following L1 regularization solution:

$$\min_{\theta} \sum_{i=1}^{M} -log\, p(y^{(i)}|x^{(i)}; \theta)$$

$$sub\ to\ \|\theta\|_1 \leq C$$

The above given optimization problem provides the minimal error in training by reducing overfitting error. The complete proposed model can be used for better classification performance which can be obtained by using genetic algorithm for dimensionality reduction, DBN and L1 regularization for better learning and reduced error.

Input: total number of attributes, number of hidden layer, Boltzmann machine configuration parameters
Output: optimal weights, learned attribute set, probability model and logistic regression optimal function set.
Step 1: input the visible and hidden layer units with initial weights corresponding to the hidden and visible layer unit. Step 2: Construct an initial energy function using Bernoulli's RBM given in (2) Step 3: Based on this energy function, find the probability of hidden and visible layers which can be used for weight estimation given in (3) Step 4: Initiate the learning process by for given input samples by computing the probability function of hidden units. Step 5: apply weight update computation by applying logarithmic derivative computation for hidden layer and visible layer units Step 6: establish the logistic regression model using sigmoid function as (7) Step 7: maximum a posteriori (MAP) computation for best probability function
Step 8: construct the optimization problem as (8) and iterate until the desired value is obtained

4. EXPERIMENTAL STUDY

In this section we present complete experimental study for software defect prediction using proposed approach. In order to perform this operation, we have considered dataset obtained from SEIP Lab [23] where eclipse dataset is provided with subsequent release of the data. These two datasets are named as (a) Eclipse JDT (release 2.0,2.1,3.0,3.1 and 3.2) and Eclipse PDE (release 2.0, 2.1, 3.0, 3.1, 3.2) b) Private Dataset ABC.

4.1 Dataset description

These datasets are obtained in .csv format with comma and points as decimal mark. First row of the csv data file contains the description of metrics: column 1 denotes file path and columns 2-49 denotes the independent software metrics variable which are given with the software metric abbreviations and final column number 50 denotes the total number of defects. A brief description about these datasets [24] is provided in the Table 1given below.

Table.1. Database Description

Database	Total Class	Total versions	Transactions	Post release defect
Eclipse JDT core	997	91	9135	463
Eclipse PDE UI	1562	97	5026	401
Equinox framework	439	91 (9)	1616	279
Mylyn	2196	98	9189	677
ABC private Dataset	779	99	1915	403

4.*2 Performance measurement*

Performance of proposed approach is measured in terms of classification accuracy which can be computed by using confusion matrix. Table 2 shows a sample representation of confusion matrix.

Table 2. Confusion Matrix

Actual class	Predicted class	
	Non defective	Defective
Non Defective	False negative (FN)	True Positive (TP)
Defective	True Negative (TN)	False Positive (FP)

With the help of this, accuracy of the proposed model is computed and compared with other state of art models. Moreover, first experiment is conduced only using DBN, next experiment contains DBN and genetic algorithm and finally, DBN, genetic algorithm and L1 regularization scheme is also incorporated.

4.3 Comparative analysis

Based on this approach, we present a comparative analysis where various parameters such as precision, recall, false positive rate, F-measure and AUC (Area under curve) have been computed and compared.

4.3 Comparative Analysis

Measurement Parameter	Approaches	Eclipse JDT core	Equinox framework	Eclipse PDE UI	Mylyn	ABC Private Dataset
Precision	DBN	0.47	0.63	0.28	0.21	0.21
	GA+DBN	0.8	0.89	0.62	0.75	0.66
	GA+DBN+L1	0.86	0.93	0.78	0.83	0.77
False Positive rate	DBN	0.17	0.24	0.2	0.17	0.13
	GA+DBN	0.69	0.88	0.83	0.89	0.81
	GA+DBN+L1	0.76	0.87	0.91	0.93	0.85
F-Measure	DBN	0.52	0.62	0.35	0.34	0.26
	GA+DBN	0.91	0.9	0.77	0.81	0.85
	GA+DBN+L1	0.93	0.92	0.81	0.83	0.92
AUC	DBN	0.76	0.78	0.7	0.68	0.69
	GA+DBN	0.91	0.86	0.88	0.94	0.81
	GA+DBN+L1	0.94	0.92	0.9	0.94	0.85

This comparative analysis is presented in Table 3 where existing system performance are compared with proposed approach. This comparative analysis shows that proposed approach achieves better performance when compared with conventional software defect technique.

5 ACKNOWLEDGEMENT

I would like to thank the organization who has provided me the ABC private data set for my research work.

6 CONCLUSION

In this work, we have focused on software defect prediction techniques using deep learning technique and presented a novel approach for early bug prediction. In order to perform this task, we have considered deep learning techniques along with feature selection and overfitting error reduction in deep belief networks. First of all, genetic algorithm is implemented for dimensionality reduction from considered feature subset. In next stage, deep belief network is applied for pattern learning which is further improved by solving an optimization problem using L1-regularization model. Performance study of the proposed model is carried out using SPIE lab dataset repository and private dataset ABC. Proposed approach performance is compared with existing models in terms of classification accuracy. This experimental analysis shows that proposed approach is capable of achieving enhanced classification performance.

REFERENCES

[1] Hryszko, Jaroslaw, and Lech Madeyski. "Assessment of the Software Defect Prediction Cost Effectiveness in an Industrial Project." In Software Engineering: Challenges and Solutions, pp. 77-90. Springer International Publishing, 2017.

[2]

[3] He, P., Li, B., Liu, X., Chen, J. and Ma, Y., 2015. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59, pp.170-190.

[4] Arora, I. and Saha, A., 2018. Software Defect Prediction: A Comparison Between Artificial Neural Network and Support Vector Machine. In Advanced Computing and Communication Technologies (pp. 51-61). Springer, Singapore.

[5] Mohanty, R. and Ravi, V., 2017. Machine Learning Techniques to Predict Software Defect. In Artificial Intelligence: Concepts, Methodologies, Tools, and Applications (pp. 1473-1487). IGI Global.

[6] Moussa, R. and Azar, D., 2017. A PSO-GA approach targeting fault-prone software modules. Journal of Systems and Software, 132, pp.41-49.

[7] Liu, M., Miao, L. and Zhang, D., 2014. Two-stage cost-sensitive learning for software defect prediction. IEEE Transactions on Reliability, 63(2), pp.676-686.

[8] Ryu, D., Jang, J.I. and Baik, J., 2015. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. Journal of Computer Science and Technology, 30(5), pp.969-980.

[9] Shepperd, M., Song, Q., Sun, Z. and Mair, C., 2013. Data quality: Some comments on the nasa software defect datasets. IEEE Transactions on Software Engineering, 39(9), pp.1208-1215.

[10] Arar, Ö.F. and Ayan, K., 2017. A Feature Dependent Naive Bayes Approach and Its Application to the

Software Defect Prediction Problem. Applied Soft Computing.

[11] Quah, T.S. and Thwin, M.M.T., 2003, September. Application of neural networks for software quality prediction using object-oriented metrics. In Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on (pp. 116-125). IEEE.

[12] Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V. and Thambidurai, P., 2007. Object-oriented software fault prediction using neural networks. Information and software technology, 49(5), pp.483-492.

[13] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," in IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485-496, July-Aug. 2008.

[14] Chen, X., Shen, Y., Cui, Z. and Ju, X., 2017, July. Applying Feature Selection to Software Defect Prediction Using Multi-objective Optimization. In Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual (Vol. 2, pp. 54-59). IEEE.

[15] Hosseini, S., Turhan, B. and Mäntylä, M., 2017. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. Information and Software Technology.

[16] Sabharwal, S., Nagpal, S., Malhotra, N., Singh, P. and Seth, K., 2018. Analysis of Feature Ranking Techniques for Defect Prediction in Software Systems. In Quality, IT and Business Operations (pp. 45-56). Springer, Singapore.

[17] Maua, G. and Galinac Grbac, T., 2017. Co-evolutionary multi-population genetic programming for classification in software defect prediction. Applied Soft Computing, 55(C), pp.331-351.

[18] Afzal, W. and Torkar, R., 2016. Towards benchmarking feature subset selection methods for software fault prediction. In Computational Intelligence and Quantitative Software Engineering (pp. 33-58). Springer International Publishing.

[19] Li, J., He, P., Zhu, J. and Lyu, M.R., 2017, July. Software Defect Prediction via Convolutional Neural Network. In Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on (pp. 318-328). IEEE.

[20] Wang, T., Zhang, Z., Jing, X. and Zhang, L., 2016. Multiple kernel ensemble learning for software defect prediction. *Automated Software Engineering*, *23*(4), pp.569-590.

[21] Wang, S., Ping, H. and Zelin, L., 2016. An enhanced software defect prediction model with multiple metrics and learners. *International Journal of Industrial and Systems Engineering*, *22*(3), pp.358-371.

[22] Tomar, D. and Agarwal, S., 2016. Prediction of defective software modules using class imbalance learning. Applied Computational Intelligence and Soft Computing, 2016, p.6.

[23] Hryszko, J. and Madeyski, L., 2017. Assessment of the Software Defect Prediction Cost Effectiveness in an Industrial Project. In Software Engineering: Challenges and Solutions (pp. 77-90). Springer International Publishing.

[24] http://www.seiplab.riteh.uniri.hr/?page_id=834

[25] D'Ambros, M., Lanza, M. and Robbes, R., 2010, May. An extensive comparison of bug prediction approaches. In Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on (pp. 31-41). IEEE.

[26] Zhang, F., Mockus, A., Keivanloo, I. and Zou, Y., 2014, May. Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (pp. 182-191). ACM.