INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume3, Issue1)
Available online at: www.ijariit.com

Software Defect Prediction using Support Vector Machine

Er. Ramandeep Kaur
Bahra Group of Institutes, Patiala.
ramanpurewal04@gmail.com

Er. Harpreet Kaur
Bahra Group of Institutes, Patiala.
preet.harry11@gmail.com

Abstract— ***developing a defect free software system is very difficult and most of the time there are some unknown bugs or unforeseen deficiencies even in software projects where the principles of the software development methodologies were applied care-fully. Due to some defective software modules, the maintenance phase of software projects could become really painful for the users and costly for the enterprises. In previous work , original data was taken with 21 features and 21 features are having high dimension features which increases the complexity of processing. Ignore the boundary decision for software default predictor because boundary condition is not detected by previous used classifier. Features of compaction were not considered because of that information is overlapped and prediction error is increased. They are not able to trained the component based classifier which results in more prediction error.***

Keywords— ***Software, Defect, Prediction, Feature Selection.***

I. INTRODUCTION

If you want to start a debate among your engineering friends, ask the question, "Is software engineering real engineering?" Unfortunately, I suspect that if your friends are from one of the "hard" engineering disciplines such as mechanical, civil, chemical, and electrical, then their answers will be "no." This is unfortunate because software engineers have been trying for many years to elevate their profession to a level of respect granted to the hard engineering disciplines. There are strong feelings around many aspects of the practice of software engineering — licensure, standards, minimum education, and so forth. Therefore, it is appropriate to start a book about software engineering by focusing on these fundamental issues. Software engineering is "a systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software. In the software engineering approach, several models for the software life cycles are defined, and many methodologies for the definition and assessment of the different phases of a life-cycle model" [9]. Progress in any discipline depends on our ability to understand the basic units necessary to solve a problem. It involves the building of models1 of the application domain, e.g., domain specific primitives in the form of specifications and application domain algorithms, and models of the problem solving processes, e.g., what techniques are available for using the models to help address the problems. In order to understand the effects of problem solving on the environment, we need to be able to model various product characteristics, such as reliability, portability, efficiency, as well as model various project characteristics such as cost and schedule [1]. Software defect prediction is a quality assurance technique in software engineering, where sophisticated methods (including machine learning) are used to predict future defects in computer programs. Such information can be used to support optimal efforts and resources allocation in the software development projects (e.g. to focus quality assurance activities on software classes which are predicted to be defect-prone) [2]. As large software systems are developed over a period of several years, their structure tends to degrade and it becomes more difficult to understand and change them. Difficult changes are excessively costly or require an excessively long interval to complete. A measure of the average age of the lines in a module can also help predict numbers of future faults: In our data, roughly two-thirds as many faults will have been found in a module which is a year older than an otherwise similar younger module. In addition to size, other variables that do not improve predictions are the number of different developers who have worked on a module and a measure of the extent to which a module is connected to other modules [3].

II. LITERATURE REVIEW

Barry Boehm (2005) proposes the approach towards developing an experimental component of such a paradigm. The approach is based upon a quality improvement paradigm that addresses the role of experimentation and process improvement in the context of industrial development. The paper outlines a classification scheme for characterizing such experiments. Progress in any discipline depends on our ability to understand the basic units necessary to solve a problem. It involves the building of models1 of the application domain, e.g., domain specific primitives in the form of specifications and application domain algorithms, and models of the problem solving processes, e.g., what techniques are available for using the models to help address the problems. In order to understand the effects of problem solving on the environment, we need to be able to model various product characteristics, such as reliability, portability, efficiency, as well as model various project characteristics such as cost and schedule [1].

Jarosław HRYSZKO (2015) studies focused on software defect prediction in real, industrial software development projects are extremely rare. We report on dedicated R&D project established in cooperation between Wroclaw University of Technology and one of the leading automotive software development companies to research possibilities of introduction of software defect prediction using an open source, extensible software measurement and defect prediction framework called DePress (Defect Prediction in Software Systems) the authors are involved in. In the first stage of the R&D project, verified what kind of problems can be encountered. This work summarizes results of that phase. Software defect prediction is a quality assurance technique in software engineering, where sophisticated methods (including machine learning) are used to predict future defects in computer programs. Such information can be used to support optimal efforts and resources allocation in the software development projects (e.g. to focus quality assurance activities on software classes which are predicted to be defect-prone) [2].

Todd L. Graves (2000) S large software systems are developed over a period of several years, their structure tends to degrade and it becomes more difficult to understand and change them.
Difficult changes are excessively costly or require an excessively long interval to complete. In this paper, we concentrate on a third manifestation of code decay: when changes are difficult in the sense that excessive numbers of faults are introduced when the code is changed. As the system grows in size and complexity, it may reach a point such that any additional change to the system causes, on the average, one further fault, at which point, the system has become unstable or unmanageable This paper is devoted to identifying those aspects of the code and its change history that are most closely related to the numbers of faults that appear in modules of code. (In this paper, the term module is used to refer to a collection of related files.) Our most successful model computes the fault potential of a module by summing contributions from the changes (deltas) to the module, where large and or recent deltas contribute the most to fault potential [3].

Xiaoxing Yang(2014) Software defect prediction can help to allocate testing resources efficiently through ranking software modules according to their defects. Existing software defect prediction models that are optimized to predict explicitly the number of defects in a software module might fail to give an accurate order because it is very difficult to predict the exact number of defects in a software module due to noisy data. This paper introduces a learning-to-rank approach to construct software defect prediction models by directly optimizing the ranking performance. In this paper, we build on our previous work and further study whether the idea of directly optimizing the model performance measure can benefit software defect prediction model construction. The work includes two aspects: one is a novel application of the learning-to-rank approach to real-world data sets for software defect prediction, and the other is a comprehensive evaluation and comparison of the learning-to-rank method against other algorithms that have been used for predicting the order of software modules according to the predicted number of defects. Our empirical studies demonstrate the effectiveness of directly optimizing the model performance measure for the learning-to-rank approach to construct defect prediction models for the ranking task [4].

Romi Satria Wahono (2015) recent studies of software defect prediction typically produce datasets, methods and frameworks which allow software engineers to focus on development activities in terms of defect-prone code, thereby improving software quality and making better use of resources. Many software defect prediction datasets, methods and frameworks are published disparate and complex, thus a comprehensive picture of the current state of defect prediction research that exists is missing [5].

Jun Zheng (2010) in the process of software defect prediction, the misclassification of defect-prone modules generally incurs much higher cost than the misclassification of not-defect-prone ones. Most of the previously developed predication models do not consider this cost issue. In this paper, three cost-sensitive boosting algorithms are studied to boost neural networks for software defect prediction. The first algorithm based on threshold- moving tries to move the classification threshold towards the not-fault-prone modules such that more fault-prone modules can be classified correctly. The other two weight-updating based algorithms incorporate the misclassification costs into the weight-update rule of boosting procedure such that the algorithms boost more weights on the samples associated with misclassified defect-prone modules. The performances of the three algorithms are evaluated by using four datasets from NASA projects in terms of a singular measure, the Normalized Expected Cost of Misclassification (NECM). The experimental

results suggest that threshold-moving is the best choice to build cost-sensitive software defect prediction models with boosted neural networks among the three algorithms studied, especially for the datasets from projects developed by object-oriented language [6].

David Gray automated software defect prediction is a process where classification and/or regression algorithms are used to predict the presence of non-syntactic implementation errors (henceforth; defects) in software source code. To make these predictions such algorithms attempt to generalize upon software fault data; observations of software product and or process metrics coupled with a level of defectiveness value. This value typically takes the form of a number of faults reported metric, for a given software unit after a given amount of time (post either code development or system deployment) [7].

Pradeep Kumar Singh (2015) this paper explains how to find the defects in the software using various techniques. We have analyzed different data sets which have been used in finding faults in various research papers. The main aim of this paper is to study various methods that can be used to predict the defects in software. The methods to estimate the software defects are regression, genetic programming, clustering, neural network, statistical technique of discriminate analysis, dictionary learning approach, hybrid attribute selection approach, classification, attribute selection and instance filtering, Bayesian Belief Networks, K-means clustering, Association Rule Mining [8].

III. OBJECTIVE

Improved default prediction in Software module by using feature extraction and adaptive boost learning approach. To study of machine learning approaches and learns WEKA Tool. Implementation of PCA for feature extraction and to implement SVM- RBF Kernel. To implement Hybrid Adaptive Boost with SVM- RBF. To analyse our approach by precision, recall and accuracy then compare them with existing methods.

IV. PROPOSED METHODOLOGY

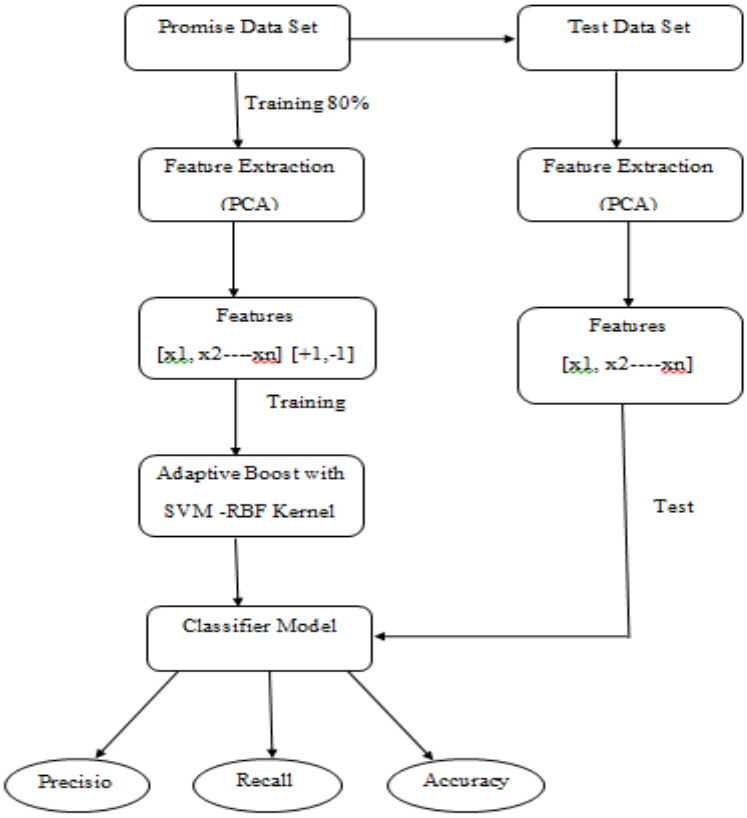
Step 1: Take the promise data set with 21 different features like cyclomatic complexity, design complexity, effort, time estimator, line count etc for defect prediction in software module.

Step 2: Implement feature extraction on promise data set by using Principle component Analysis (PCA). Feature Extraction is used to merge the data set. In feature extraction merging process is based on eigenvalues , having high eigen value means contain more information.

Step 3: Take the different features x1, x2, x3.....xn and find out the status that whether they are default or not default [+1, -1]. If the value is +1 that means its 'default' and if -1 then it is 'not default'.

Step 4: Implement Hybrid Adaptive Boost with SVM -RBF Kernel for component learning and to remove compaction and boundary error condition.

Step 5: Apply Classifier model to find out precision, recall and accuracy of the software training 20%.



V. RESULT

Table 1: Comparison of Classifier

Classifier	Accuracy	Recall	Precision
Linear	34	35	35
Polynomial	42	69	63
Quadratic	48	79	71
RBF	46	55	68
Multilayer perception	38	62	58
Adaptive boost	88	73	85

Graph 1

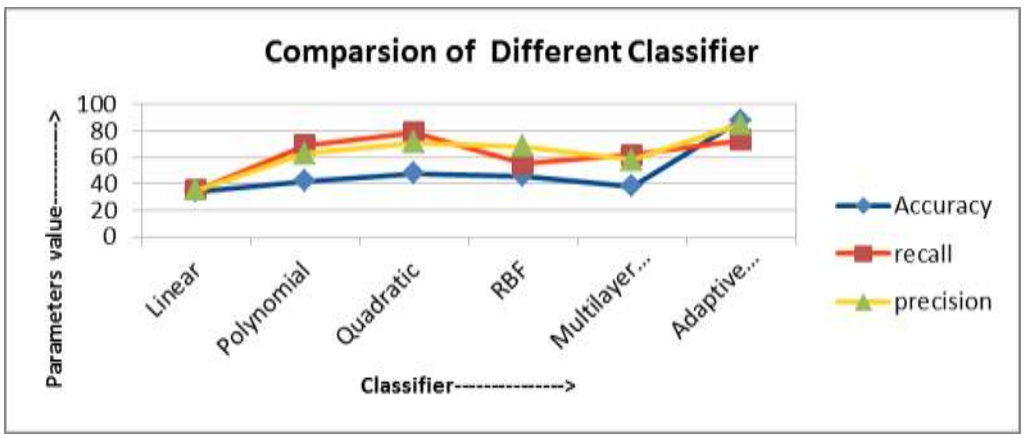


Figure 1: Comparison of different classifier

Table 2: Comparison of distinct classifier

Classifier	Accuracy	Recall	Precision
Linear	48	50	50
Polynomial	50	55	54
Quadratic	54	55	55
RBF	44	52	50
Multilayer perception	50	53	53
Adaptive boost	89	77	87

Graph 2

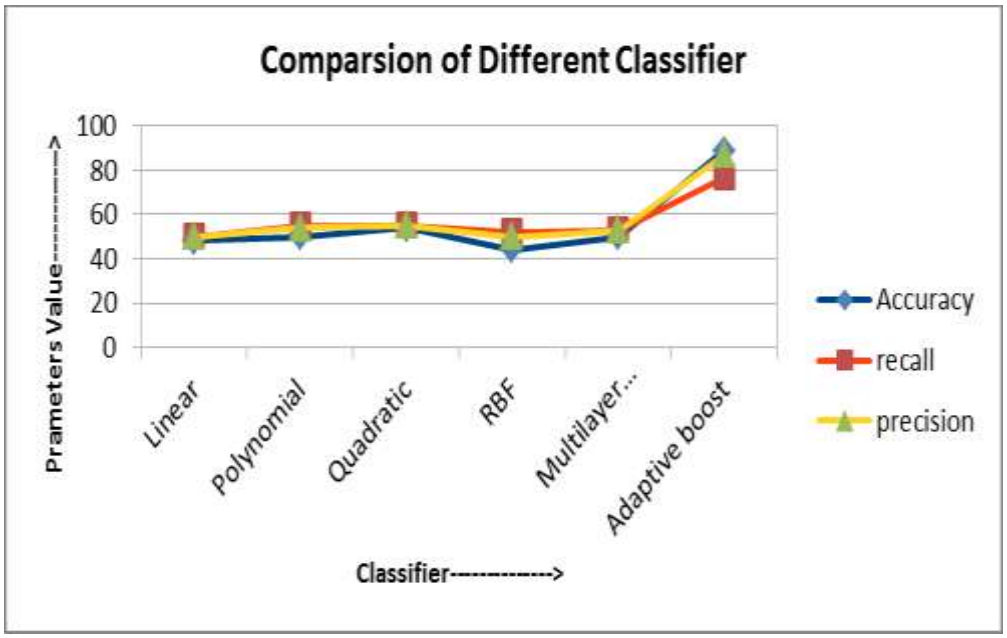


Figure 2: Comparison of different classifier

Table 3: Comparison of different classifier

Classifier	Accuracy	Recall	Precision
Linear	48	50	50
Polynomial	54	57	57
Quadratic	58	59	59
RBF	48	56	54
Multilayer perception	50	52	52
Adaptive boost	89	77	87

Graph 3

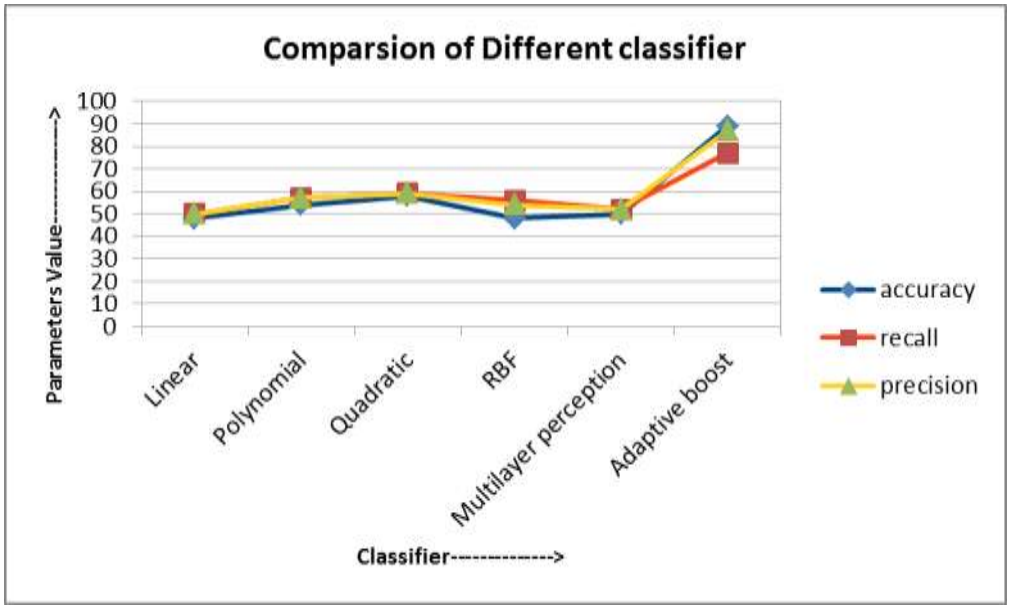


Figure 3: Comparison of different classifier

REFERNCES

[1] Boehm, Barry, Hans Dieter Rombach, and Marvin V. Zelkowitz, eds. *Foundations of empirical software engineering: the legacy of Victor R. Basili*. Springer Science & Business Media, 2005.

[2]Hryszko, Jarosław, and Lech Madeyski. "Bottlenecks in software defect prediction implementation in industrial projects." *Foundations of Computing and Decision Sciences* 40.1 (2015): 17-33.

[3]Graves, Todd L., et al. "Predicting fault incidence using software change history." *IEEE Transactions on software engineering* 26.7 (2000): 653-661.

[4]Yang, Xiaoxing, Ke Tang, and Xin Yao. "A learning-to-rank approach to software defect prediction." *IEEE Transactions on Reliability* 64.1 (2015): 234-246.

[5]Wahono, Romi Satria. "A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks." *Journal of Software Engineering* 1.1 (2015): 1-16.

[6]Zheng, Jun. "Cost-sensitive boosting neural networks for software defect prediction." *Expert Systems with Applications* 37.6 (2010): 4537-4543.

[7]Gray, David, et al. "The misuse of the NASA metrics data program data sets for automated software defect prediction." *Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on*. IET, 2011.

[8]Singh, Pradeep Kumar, Dishti Agarwal, and Aakriti Gupta. "A Systematic Review on Software Defect Prediction." *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*. IEEE, 2015.

[9] Laplante, Philip A. *What every engineer should know about software engineering*. CRC Press, 2007.

