Semi-Supervised Machine Learning and Adaptive Data Clustering Approach for Software Defect Prediction

Sumangala Patil [1], A.Nagaraja Rao [2], C. Shoba Bindu [3]

[1, 3] *Department of CSE*, JNTU, Anantapur, Andhra Pradesh, India.
[2] *Department of CSE*, SJT, Vellore Institute of Technology, Tamil Nadu, India.
[1] sumangala14jan@gmail.com; [2] nagarajaa@vit.ac.in, [3] shobabindhu@gmail.com

***Abstract* - Software defect prediction is an important area of research due to its significant utilization in various real-time applications. The ever-increasing demand of software application has resulted in rapid development of software modules development that further has elevated the chances of faulty software production. In order to deal with this issue software testing solutions are recommended. As, manual testing require more human effort and time hence, automated testing has gained attention from industries and researchers. In this work, we present an automated software defect prediction strategy using semi-supervised machine learning approach. The complete process is divided into four main sections: pre-processing where we apply KNN based missing value imputation, feature selection where we apply PCA based optimal feature selection method, clustering where we apply SOM based clustering model and finally Naive Bayes classifier is implemented. Experimental study shows that proposed approach achieves promising performance when compared with the existing software defect prediction techniques.**

Keywords - Software Defect Prediction; KNN; PCA; SOM Clustering; Machine Learning

I. INTRODUCTION

Software engineering field has gained huge attraction due to their various real-time applications. However, due to increased demand of software-based applications, the complexity of software development and functionality of software also increases which may raise software failure issues. Software failure can cause huge loss in terms of economic conditions and business reliability to the software industries [1]. A study presented in [2] revealed that 80% of defects in software application are caused due to the 20% defective modules or sub-modules hence there is a need to evaluate the entire software module to identify the bug which can help to reduce the software development cost and optimized resource utilization during the complete development of the application. For small application-based software modules, manual inspection systems can be utilized but manual inspection systems are not recommended for huge software applications due to error-prone human nature, more time requirement and cost constraints. Hence, automated systems for software module analysis and bug identification is highly recommended for software industries.

According to the software engineering, software defect prediction is a promising and important aspect for the improving the software development quality. Software defect is defined as a bug or fault in the software which can cause a faulty or unexpected outcome. In software industry, early identification of software bugs is a crucial task which can affect the software quality, cost and resource utilization. In general, software defects can occur at any phase of the software development process. Nowadays, software industries are focusing on the improving the software quality and development of reliable software application by applying software defect identification technique during early phase of software development model. in these industries, identification of software bugs is a main objective in the early phase of Software Development Life Cycle (SDLC) [3]. In SDLC, various stages are considered such as software application requirement collection, analysis, designing, coding, testing, implementation and maintenance phase. In software defect management process, the software defect management is an important stage which is very useful for defect management. The complete process of software defect management includes several stages which are: defect identification, defect categorization, defect analysis, and defect removing. However, this complete process can be categorized under software defect prediction technique where software quality can be analyzed using machine learning processes and defects can be identified based on the severity of quality.

Software defect prediction (SDP) plays an important role in the field of software industries to identify the bugs. In his work, our main aim is to develop an automated process for software defect prediction using machine learning based approaches. Several techniques have been introduced in this field which are based on the different type of machine learning and classification algorithms. According to the machine learning algorithms approach, several number of feature attributes are extracted from the software data and feed to the suitable classification algorithms such as Support vector machine, Naïve Bayes and Random forest etc. These features are extracted from the software data such as Derived Halsted [5] where time complexity, program length, difficulty etc. parameters are included, McCabe [4] where different type of

complexity parameters such as design complexity, cyclomatic complexity etc. are considered, similarly, Basic Halstead [5] attributes are considered where operator counts, comment lines, blank line etc. parameters are considered as attributes. Based on these parameters, a trained model is developed which is used for classifying the defects in the software module. Generally, machine learning classification algorithms are divided into two main categories which are known as supervised [6] and unsupervised learning scheme [7].

A. Supervised Learning Scheme

According to these machine learning tasks, the software data is inferred from labeled data which is called as a training data. This training data contains various number of training labels which are labeled according to the class of defect as defective or non-defective. Each training data contains input attributes and their corresponding desired value. The supervised learning scheme performs two tasks such as classification which is used for generating a predictive model with a discrete range and regression where continuous range model is generated to identify the class of data. Several techniques have been introduced for classification and regression. According to the classification techniques, Naïve Bayes [8], Support Vector Machine [9], Random Forest [10], and Neural Networks [11] etc. are widely used for software defect prediction. Similarly, regression-based schemes use linear, non-linear, logistic [12], and multivariate regression [13].

B. Unsupervised Learning Scheme

Unlike supervised scheme, the unsupervised schemes do not contain any previous information about the dataset and it performs the desired task dynamically where database is trained using only based on the inputs. This approach analyzes the inputs and identifies the various relationship between different types of inputs. Clustering is considered as a most efficient solution for the unsupervised learning scheme where maximum similarity data are grouped into one cluster and then it can be labelled. Several clustering techniques have been introduced for software defect prediction such as K-Means, Hierarchical, Probabilistic and Density – based Clustering.

Significant amount of work has been carried out in this field of software defect prediction using machine learning techniques. Moreover, supervised learning schemes are also adopted widely. The use of unsupervised scheme is a challenging task for the data which doesn't contain label information. In this work, we focus on the unsupervised learning scheme for software defect prediction using clustering-based approaches.

C. Contribution

The man contribution of this work are as follows:

- Study about semi-supervised learning schemes
- Development of semi-supervised learning scheme for software defect prediction
- Implementation of KNN based missing value imputation technique
- Optimal Feature selection
- Data clustering which is used for grouping the similar data into different clusters
- Classifier construction

D. Organization

The complete article is arranged as follows: section II briefly describes recent techniques for software defect prediction using supervised and unsupervised schemes. Problem formulation and solution for unsupervised scheme is presented in section III, section IV presents experimental study and comparative performance analysis and finally, section V presents concluding remarks.

II. LITERATURE SURVEY

This section presents study about recent techniques in this field of software defect prediction using machine learning and pattern learning scheme. The complete section is divided into two main sub section where we present brief discussion about SDP techniques using supervised and unsupervised learning schemes.

A. Supervised Techniques for Software Defect Prediction

In this section we present a brief discussion about recent techniques of software defect prediction supervised machine learning approaches. In this field, Arar et al [14] presented Naïve Bayes classifier approach for software defect prediction. In this work, authors discussed that Naïve Bayes approach is a probabilistic model where it is considered that the features of given input set are independent and weights of the given input are equally important for each instance. Sometimes features of the given set are interrelated where this process may lead towards the degrade performance of the system. Hence, authors presented feature dependent naïve Bayes classifier approach. Based on supervised learning scheme, Liu et al. [15] developed SDP and classification scheme. According to this approach, irrelevant and redundant features affect the performance of classification model. Hence, in this work authors suggested that the efficient feature selection can efficiently improve the system performance and developed feature selection approach as feature clustering and feature ranking approach. According to this approach, original features are partitioned into k clusters based on the correlation measurement. Later, feature and cluster relevancy-based features are considered for selecting the optimal features. These features are measured using information gain, chi-square and relief. This feature selection strategy is implemented for NASA software defect dataset. Performance of software defect prediction technique gets affected due to

data redundancy, feature irrelevancy, correlation and missing data attributes. Moreover, inappropriate data distribution also can cause performance issues in SDP. Based on these assumptions, to overcome the performance issue of SDP, Laradji et al. [16] presented a combined scheme using feature selection and ensemble learning scheme for defect classification. In order to show the robustness of the classification, with and without feature selection-based algorithms are implemented which shows that proposed approach efficiently mitigates the issue of class imbalance.

Generally, the software defect prediction techniques suffer from the issue of data imbalance which can lead towards the degraded performance of SDP system. In order to deal with this issue, Mauša et al. [17] presented a novel approach for software defect prediction using evolutionary computation optimization strategy using multi-objective functionalities. In this work, authors also addressed the convergence issue of optimization problem also and presented a fast convergence approach for the reduced computation time and complexities. Similarly, the class imbalance problem issue is addressed in [18] where authors discussed about the within –project defect prediction and cross-project defect related issues in the SDP modeling. In order to overcome the issue of cross-project defect prediction, data resampling scheme is developed using asymmetric misclassification and similarity weights obtained using dataset. The data resampling helps to associate the characteristics of the imbalanced data. To accomplish this task of SDP, feature selection is also considered as a promising technique which helps to reduce the complexity and time, provides better performance for classification. Based on these assumptions, recently, Manjula et al. [19] presented a hybrid approach using Deep Neural Network and genetic algorithm based optimal feature selection approach. In this work, performance of the system is improved by applying a newly designed scheme of fitness function computation. Similarly, Jayanthi et al. [20] also presented a new scheme for feature selection base software defect prediction using PCA (Principal Component Analysis) which is improved by applying maximum-likelihood based technique for error reduction in the optimal solution identification. Finally, neural network-based classification technique is implemented to classify the software defects.

B. Unsupervised Learning schemes

In this section we discuss about the recent unsupervised learning technique for defect prediction and classification of software defects. Generally, unsupervised schemes include data clustering-based approaches where similarity is measured and the similar data can be grouped together to label them as defective and non-defective.

Initially, Zhong et al. [22] presented unsupervised learning scheme for developing the software quality estimation system. Authors discussed about the current software defect prediction techniques which are based on the supervised models of classification and quality of software is measured in by identifying the faulty and non-faulty software module. On the other hand, cross-defect project defect prediction requires unsupervised clustering scheme. Hence, authors developed unsupervised clustering scheme. According to the proposed approach, software modules are clustered into smaller coherent groups and processed by the software quality expert which labels the software module as defective and non-defective based on the knowledge and statistics. Catal et al. [23] analyzed quantitative and qualitative models for the software defect prediction and concluded that quantitative models are more reliable for decision making process. These methods divide the decision-making process into two types such as generalized model and product specific models. However, in the absence of labeled data, it becomes a tedious task for these techniques to identify and classify the faulty software module. Hence, in this work, authors have focused on the development of unsupervised approach for defect prediction using clustering scheme. In this work, first of all total number of software modules are processed and clustered together with the help of K-means clustering scheme and later software metrics threshold is analyzed with the help of expert in domain. Abaei et al. [24] also presented a study for software defect prediction using unsupervised learning scheme and suggested that unsupervised learning scheme can be used for limiting the issue in SDP such as cost, time and complexity etc. In this work, clustering based threshold scheme is developed for classifying the defects. In order to build an automated SDP model, self-organizing map (SOM) along with the threshold is developed to improve the prediction performance.

Zhang et al. [21] focused on cross-project defect prediction for classifying the software bugs. In this process, the classifiers are used which are obtained from the other software projects. In this work, authors have discussed that conventional techniques require specific degree of homogeneity i.e. similar distribution between training and testing software projects. Hence, the homogeneity criteria requirements are significantly important for developing the unsupervised software defect prediction technique. However, unsupervised clustering scheme doesn't require any training data, in this work authors presented distance based and connectivity based unsupervised classification-based schemes. This study shows that distance-based classifier (i.e. k-means clustering) shows poor performance whereas connectivity-based techniques are not applied hence, authors presented connectivity-based classifier (spectral clustering) scheme for within and cross-project defect prediction. This study shows that spectral classifier is ranked as second tier classifier whereas supervised random forest classifier is ranked and first tier classifier. Moreover, it is also concluded that this approach achieves promising results for within and cross defect prediction.

III. THE PROPOSED MODEL

In this section we present the proposed approach for software defect prediction which is based on the semi-supervised learning approach. According to the proposed approach, compete process is divided as follows:

- Data pre-processing which is used for missing value imputation
- Optimal Feature selection
- Data clustering which is used for grouping the similar data into different clusters
- Classifier construction.

A. Data Pre-Processing

In this module, we present KNN (K-Nearest Neighborhood) based modeling for missing value imputation for software datasets. The desired objective of missing value imputation is obtained using benchmark KNN imputation algorithms.

B. Optimal Feature Selection

In order to reduce the complexity, we apply feature selection scheme which helps to improve the clustering and classification performance by removing the redundant and unwanted features. In this work, we consider PCA (Principal Component Analysis) technique and applied it in improved manner for feature selection and reduction in the given attribute list.

According to PCA, let us consider that total n number of feature vector are represented as $(x_1, x_2, \dots, x_i, \dots x_n)$ from d diemsional space to n vectors as $(x_1', x_2', \dots, x_i', \dots, x_n')$ in new space as:

$$x_i' = \sum_{k=1}^{d'} a_{k,i} e_k \, , d' \leq d \tag{1}$$

Where e_k represents the Eigen vector corresponding to the scatter matrix S which contains the largest Eigen values and $a_{k,i}$ denotes the projection of x_i vector over Eigenvectors which are called as principal components of the original data. Here, it is known that d and d'; are the two positive integers where dimesion of d can not be greater that the d'. Based on these assumptions, the scatter matrix S can be represented as:

$$S = E[x_i x_i^T], i = 1{:}n \tag{1}$$

This helps to reduce the error between original and transformed feature vectors which can be represented by taking the variance of the principle component, expressed as:

$$\sigma^2(e_k) = E\left[\sigma_{k,i}^2\right] = e_k^T S e_k \tag{2}$$

Where e_k denotes the $d \times 1$ dimensional vector as $e_k = \left[e_{1,k}, e_{2,k}, \dots e_{d,k}\right]^T$. Local minima and maxima are also computed for the given vector where it presents a relationship as

$$\sigma^2(e_k + \delta e_k) = \sigma^2(e_k) \tag{3}$$

With the help of scatter matrix, it can be re-written as: $(\delta e_k)^T S e_k - \lambda (\delta e_k)^T e_k = 0$ where λ denotes the scaling factor which leads to Eigen problem formulation, given as $S e_k = \lambda e_k$.

The error induced in the input vector due to data reduction, is given as:

$$E_{d'} = \frac{1}{2} \sum_{d'+1}^{d} \lambda_k \tag{4}$$

λ_k denotes the Eigen values of the computed scatter matrix. From Eq. (5) it can be concluded that large Eigen values will provide less error in the data representation. Hence, the Eigen values can be presented in the decreasing order as:

$$\lambda_1 > \lambda_2 > \dots > \lambda_k > \dots > \lambda_d \tag{5}$$

Hence the final arrangement of these Eigen values represents the selected features which can be extracted based on the required number of features.

C. Data Clustering

Here we present a clustering-based model for reducing the computational complexity and this clustering also can be helpful for the unlabeled data. Here, we use Self-organizing map (SOM) technique to perform the clustering on the data without knowing the class this technique maintains the data mapping along with the distance between the data points. The topological related information about data of the input space is preserved using neighborhood function. According to this process, p number of units are which can have n number of optimal selected features such as $\{X_{p,1}, X_{p,2}, \dots, X_{p,n}\}$ and output of this clustering approach is a vector which represents the total number of clusters in the data as $\{y_1, y_2, \dots, y_m\}$ where m and n are not dependent to each other and can be different. In this, a weight vector is considered which has the similar size according to the given input vector. Any vector p is grouped into n number of clusters. Weights of each input unit are computed and connected to the output neurons. During each iteration, a random input is selected and its distance from weight vector is computed using Euclidean distance. This process is repeated for all input nodes and the minimum distance vector is selected which is called as winning node. After finishing this process, the nearby nodes and their weights are updated. This process is stopped as it achieves the specified number of iterations are accomplished or if there is no significant change in the neuron weights. The distance between neurons can be computed as

$$\forall_{x,w}, d(x,w) = \left(\sum_{t-1}^{n} |x_t - w_t|^p \right)^{\frac{1}{p}} \tag{1}$$

Where x denotes the input vectors, w denotes the corresponding vector and d represents the distance measurement outcome between x and w. Initially, random weights are assigned to each input vector and the attributes are non-linear since authors in [25] have reported that the

random weight initialization performs well for the non-linear datasets.

In the next phase, we apply weight updating process for each iteration, which is given as:

$$w_j(t+1) = w_j(t) + R(t)\left(x_i - w_j(t)\right) \quad (2)$$

Where R denotes the learning rate which decreases as number of iterations are increasing. The learning rate at t_0 iteration can be computed as:

$$R(t) = R_0 \exp\left(\frac{t}{\lambda}\right) \quad (3)$$

At this stage, distance of the output node from the winning node also affects the learning rate which results in the weight updating process. Hence, the weight updating process can be re-written as:

$$w_j(t+1) = w_j(t) + \phi(t)R(t)\left(x_i - w_j(t)\right) \quad (4)$$

The winning node has influence on the neighboring node which is denoted by $\phi(t)$ and can be computed as:

$$\phi(t) = \exp\left(-\frac{d^2}{2\alpha^2(t)}\right) \quad (5)$$

Where, d is the distance and α is the width of neighborhood area. The neighborhood area decreases as the number of iterations increase.

D. Classifier Construction

At this stage, we have final clusters with different attributes which need to be classified to obtain the defective and non-defective modules. In order to classify these attribute clusters, we present Naïve Bayes classifier model to predict the software bugs. Naïve Bayes classifier is a probabilistic method for classification which can classify the data based on the probabilities. The main advantage of this classifier is that it requires only one scan for training the complete data and moreover it can handle the missing values by computing the likelihood of membership probabilities.

Let us consider that D is the training data which is associated with the different training class labels. Each training instance is represented as n dimensional attribute vector which is represented as $X = \{x_1, x_2, .., x_n\}$ where n measurements are presented based on the attributes such as $\{A_1, A_2, A_3, .. A_n\}$ with m number of classes as $\{C_1, C_2, C_3, .., C_m\}$. According to the naïve Bayes classification scheme, a test instance X is provided where it predicts the highest probability of this instance to identify that the given instance belongs to the which type of class. In our case, we have instance in the form of different type of attributes and their corresponding classes are defective or non-defective software modules. The probability prediction of software instances can be expressed as:

$$\mathcal{P}(C_i|X) > \mathcal{P}(C_i|X) \; for \; 1 \le j \le m, j \ne i \quad (6)$$

The probability of predicted class C_i can be maximized is called maximum posteriori hypothesis which can be expressed as:

$$\mathcal{P}(C_i|X) = \frac{\mathcal{P}(X|C_i)\mathcal{P}(C_i)}{\mathcal{P}(X)} \quad (7)$$

Eq. (7) shows Bayesian theorem where $\mathcal{P}(X)$ remains constant for all classes and $\mathcal{P}(X|C_i)\mathcal{P}(C_i)$ need to be maximized for improving the classification performance. In general, if the class probabilities are not known then probabilities of all classes are not known then we consider that probability of all classes is equal as $P(C_1) = P(C_2) == \cdots P(C_m)$ and hence $P(X|C_i)$ need to be maximized. However, the class probabilities are computed as $P(C_i) = \frac{|C_{i,D}|}{|D|}$ where $|C_{i,D}|$ denotes the total number of available training instances in dataset D. In this approach it is assumed that there is no dependency between attributes hence the $P(X|C_i)$ can be computed as:

$$\mathcal{P}(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \quad (8)$$

$$\mathcal{P}(X|C_i) = P(x_i|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

where x_k represents the attributes of A_k for the given instance X hence the probabilities of each class $P(x_1|C_i), P(x_2|C_i) \ldots, P(x_n|C_i)$ can be estimated using training instances. In order to predict the class label of instance$X$, we evaluate $\mathcal{P}(X|C_i)\mathcal{P}(C_i)$ for each class C_i from the given dataset D. With the help of Naïve Bayes classifier, the class labels can be predicted as:

$$\mathcal{P}(X|C_i)P(C_i) > P(X|C_i)P(C_j) \quad for \; 1 \le j \le m \; and \; j \ne i \quad (9)$$

IV. RESULTS AND DISCUSSION

In this section we present experimental results and discussion using proposed approach for software defect prediction. The complete experimental study is implemented using MATLAB simulation tool running on Windows 10 platform with an Intel Core I5 CPU (3.20GHz) and 8GB memory. In order to evaluate the performance, we used opensource dataset obtained from PROMISE repository [26].

A. Dataset Description

This section briefly describes the datasets obtained from the PROMISE repository named as CM1, JM1, KC3, KC1, PC1, PC3, PC4 and PC5 and Eclipse dataset Eclipse dataset (Eclipse 2.0, Eclipse 2.0 and Eclipse 3.0). These datasets are labeled with the rate of 10%, 20% and 30% with two classes named as true and false which denotes the defective and non-defective software modules. Table 1 presents a brief description about these software dataset modules. The below given table denotes the total number of modules, defective measurement, language used for software, total line of codes and number of defects.

TABLE I. TABLE I. PROMISE AND ECLIPSE SOFTWARE DEFECT PREDICTION DATASET DETAILS

Name of Dataset	Total Number of modules	Features	Faulty Modules	Non-Faulty Modules	Class
CM1	327	37	42	285	2 {true, false}
KC1	2109	20	326	1783	2 {true, false}
KC3	194	39	36	158	2 {true, false}
JM1	7782	21	1672	6110	2 {true, false}
PC1	705	37	61	644	2 {true, false}
PC3	1077	37	134	943	2 {true, false}
PC4	1458	37	178	1280	2 {true, false}
PC5	17186	38	516	16670	2 {true, false}
Eclipse 2.0	6729	155	975	5714	2 {true, false}
Eclipse 2.1	7888	155	854	7034	2 {true, false}
Eclipse 3.0	10593	155	1568	9025	2 {true, false}

These software databases contain several attributes which are divided into three main categories such as derived Halsted, McCabe and Basic Halstead. The complete information about these metrics is presented in table II.

TABLE II. PROMISE REPOSITORY ATTRIBUTE DESCRIPTION

Type	Metrics	Definition
Derived Halsted	t	Time Estimator
	n	Number of operators and operands
	d	Difficulty
	e	Effort to write code
	l	Program length
	v	Volume
	i	Intelligence
McCabe	$iv(g)$	Design Complexity
	$v(g)$	Cyclomatic complexity
	LOC	Number of codes lines
	$ev(g)$	Essential complexity
Basic Halstead	$branchcount$	Total count of branches
	$loComment$	Comment count
	$lOCode$	Total line count
	$uniq\_op$	Unique operator count
	$uniq\_opnd$	Unique operand count
	$total\_op$	Count of total operators
	$loBlank$	Blank line count
	$lOCodeAndComment$	Count of total lines of code and comments
	$total\_op$	Number of total operators
Class	Defect	Defective or Non-Defective

B. Performance Measurement Parameters

In this section, we present some performance measurement parameters which are used for measuring the performance of classification studies. Confusion matrix is a widely adopted technique is this field of classification which represents the true positive, false positive, true negative and false negative classes and these parameters are used for classification accuracy computation. Table 3 shows a basic representation of confusion matrix for software defect prediction task.

TABLE III. CONFUSION MATRIX

Actual class	Predicted class	
	Non-defective	Defective
Non-Defective	False negative (F_N)	True Positive (T_P)
Defective	True Negative (T_N)	False Positive (F_P)

With the help of confusion matrix, we compute probability of detection (pd), probability of false alarm (pf), F-measure, and precision for comparative performance analysis. These parameters can be computed as:

$$pd = \frac{T_P}{T_{P+F_N}}$$
$$pf = \frac{F_P}{F_P + T_N}$$
$$pr = \frac{T_P}{T_{P+F_P}} \quad (10)$$
$$FMeasure = \frac{2 * pd * pr}{pd + pr}$$

C. Comparative Analysis

Here we present a comparative performance analysis using proposed semi-supervised learning approach for software defect prediction. In order to compare the performance, we considered different rate of labels.

C1. Performance Measurement for 10% Label Rate

In the first experiment, we considered 10% data label rate and compared the performance with state-of-art technique in terms of the various parameters which are discussed in the previous sub-section.

TABLE IV. COMPARATIVE ANALYSIS FOR 10% LABEL RATE

DB	Measure	FTF	ROCUS	LDS	CMN	GKSLP	NSGLP	DFCM	Proposed
CM1	pd	0.37	0.54	0.50	0.46	0.63	0.71	0.74	0.81
	pf	0.16	0.26	0.28	0.21	0.33	0.35	x	0.40
	F-Measure	0.30	0.33	0.34	0.32	0.34	0.37	0.38	0.41
KC1	pd	0.42	0.51	0.60	0.45	0.56	0.68	0.69	0.71
	pf	0.11	0.19	0.27	0.14	0.23	0.26	x	0.28
	F-Measure	0.37	0.41	0.42	0.39	0.41	0.44	0.43	0.45
JM1	pd	0.44	0.52	0.58	0.46	0.59	0.69	0.73	0.80
	pf	0.13	0.26	0.25	0.14	0.25	0.27	x	0.32
	F-Measure	0.38	0.44	0.39	0.39	0.40	0.43	0.44	0.46
KC3	pd	0.40	0.47	0.43	0.44	0.51	0.66	0.69	0.73
	pf	0.17	0.25	0.18	0.19	0.23	0.28	X	0.33
	F-Measure	0.33	0.36	0.35	0.35	0.36	00.40	0.41	0.48
PC1	pd	0.28	0.30	0.40	0.33	0.45	0.62	0.63	0.77
	pf	0.12	0.13	0.23	0.15	0.26	0.26	X	0.31
	F-Measure	0.23	0.24	0.28	0.26	0.30	0.35	0.34	0.36
PC3	pd	0.33	0.37	0.42	0.36	0.48	0.67	0.67	0.73
	pf	0.15	0.18	0.23	0.17	0.27	0.25	X	0.28
	F-Measure	0.25	0.26	0.29	0.26	0.30	0.34	0.36	0.39
PC4	pd	0.46	0.53	0.58	0.51	0.63	0.70	0.72	0.86
	pf	0.17	0.20	0.25	0.19	0.26	0.24	X	0.35
	F-Measure	0.36	0.39	0.41	0.37	0.42	0.49	0.51	0.56
PC5	pd	0.53	0.55	0.64	0.54	0.63	0.75	0.78	0.83
	pf	0.19	0.21	0.23	0.20	0.22	0.23	X	0.29
	F-Measure	0.45	0.46	0.50	0.46	0.50	0.59	0.61	0.63

The above given table shows a comparative performance in terms of pd, pf and f-measure for 10%label rate experiment. The complete experimental study shows that proposed approach achieves better performance when compared with the other existing techniques such as FTF, ROCUS, LDS, CMN, GKSLP, NSGL and DFCM.

C2. Performance Measurement for 20% Label Rate

In this experiment, we have considered labeling rate 20% and evaluated the performance using similar setup as experiment 1. Obtained performance and comparative measurement is depicted in table IV.

TABLE V. COMPARATIVE PERFORMANCE FOR 20% LABELING RATE

DB	Measure	FTF	ROCUS	LDS	CMN	GKSLP	NSGLP	DFCM	Proposed
CM1	pd	0.39	0.55	0.61	0.49	0.64	0.72	0.76	0.83
	pf	0.18	0.29	0.30	0.25	0.35	0.36	x	0.42
	F-Measure	0.32	0.33	0.35	0.32	0.33	0.37	0.39	0.40
KC1	pd	0.46	0.54	0.65	0.48	0.60	0.71	0.73	0.85
	pf	0.14	0.22	0.29	0.16	0.24	0.30	x	0.36
	F-Measure	0.39	0.42	0.42	0.41	0.41	0.45	0.44	0.50
JM1	pd	0.45	0.62	0.66	0.51	0.62	0.70	0.74	0.82
	pf	0.17	0.31	0.27	0.18	0.26	0.31	x	0.32
	F-Measure	0.39	0.42	0.45	0.42	0.41	0.44	0.45	0.46
KC3	pd	0.43	0.52	0.46	0.49	0.54	0.69	0.70	0.74
	pf	0.18	0.28	0.22	0.21	0.33	0.30	x	0.36
	F-Measure	0.36	0.38	0.39	0.38	0.38	0.41	0.43	0.45
PC1	pd	0.31	0.34	0.44	0.35	0.46	0.66	0.65	0.73
	pf	0.13	0.14	0.26	0.16	0.28	0.27	x	0.36
	F-Measure	0.25	0.30	0.32	0.27	0.31	0.39	0.40	0.45
PC3	pd	0.35	0.41	0.46	0.39	0.52	0.72	0.73	0.76
	pf	0.16	0.21	0.27	0.20	0.29	0.28	x	0.36
	F-Measure	0.28	0.32	0.33	0.30	0.33	0.39	0.41	0.45
PC4	pd	0.49	0.57	0.61	0.54	0.65	0.74	0.75	0.85
	pf	0.19	0.23	0.26	0.21	0.27	0.27	x	0.29
	F-Measure	0.39	0.47	0.45	0.40	0.46	0.53	0.55	0.63
PC5	pd	0.55	0.56	0.66	0.61	0.67	0.78	0.78	0.87
	pf	0.22	0.23	0.25	0.22	0.25	0.27	x	0.39
	F-Measure	0.45	0.46	0.48	0.47	0.51	0.58	0.60	0.63

C3. Performance Measurement for 30% Label Rate

In this experiment, we have considered labeling rate 30% and evaluated the performance using similar setup as experiment 1. Obtained performance and comparative measurement is depicted in table VI.

TABLE VI. COMPARATIVE PERFORMANCE FOR 30% LABELING RATE

DB	Measure	FTF	ROCUS	LDS	CMN	GKSLP	NSGLP	DFCM	Proposed
CM1	*pd*	0.42	0.59	0.51	0.54	0.67	0.76	0.78	0.83
	pf	0.22	0.28	0.33	0.27	0.36	0.36	x	0.44
	F-Measure	0.32	x	0.35	0.34	x	0.38	0.40	0.63
KC1	*pd*	0.49	0.54	0.62	0.52	0.60	0.75	0.79	0.86
	pf	0.14	0.22	0.33	0.20	0.24	0.32	x	0.41
	F-Measure	0.41	x	0.45	0.43	x	0.47	0.48	0.66
JM1	*pd*	0.52	0.67	0.70	0.55	0.65	0.73	0.77	0.85
	pf	0.26	0.32	0.34	0.21	0.29	0.33	x	0.45
	F-Measure	0.40	x	0.45	0.42	x	0.46	0.40	0.66
KC3	*pd*	0.47	0.55	0.51	0.48	0.57	0.70	0.72	0.78
	pf	0.23	0.31	0.23		0.36	0.35	x	0.45
	F-Measure	0.37	x	0.41	0.37	x	0.43	0.45	0.55
PC1	*pd*	0.35	0.38	0.48	0.38	0.49	0.69	0.69	0.58
	pf	0.12	0.17	0.29	0.20	0.32	0.31	x	0.35
	F-Measure	0.27	x	0.35	0.30	x	0.41	0.40	0.45
PC3	*pd*	0.38	0.44	0.51	0.43	0.55	0.74	0.74	0.81
	pf	0.19	0.23	0.31	0.22	0.33	0.31	x	0.36
	F-Measure	0.31	x	0.37	0.33	x	0.42	0.43	0.52
PC4	*pd*	0.54	0.61	0.56	0.58	0.69	0.78	0.81	0.89
	pf	0.21	0.27	0.29	0.23	0.32	0.31	x	0.35
	F-Measure	0.42	x	0.48	0.43	x	0.55	0.56	0.61
PC5	*pd*	0.57	0.59	0.70	0.63	0.72	0.82	0.83	0.92
	pf	0.23	0.24	0.31	0.26	0.29	0.30	x	0.35
	F-Measure	0.46	x	0.52	0.46	x	0.59	0.62	0.70

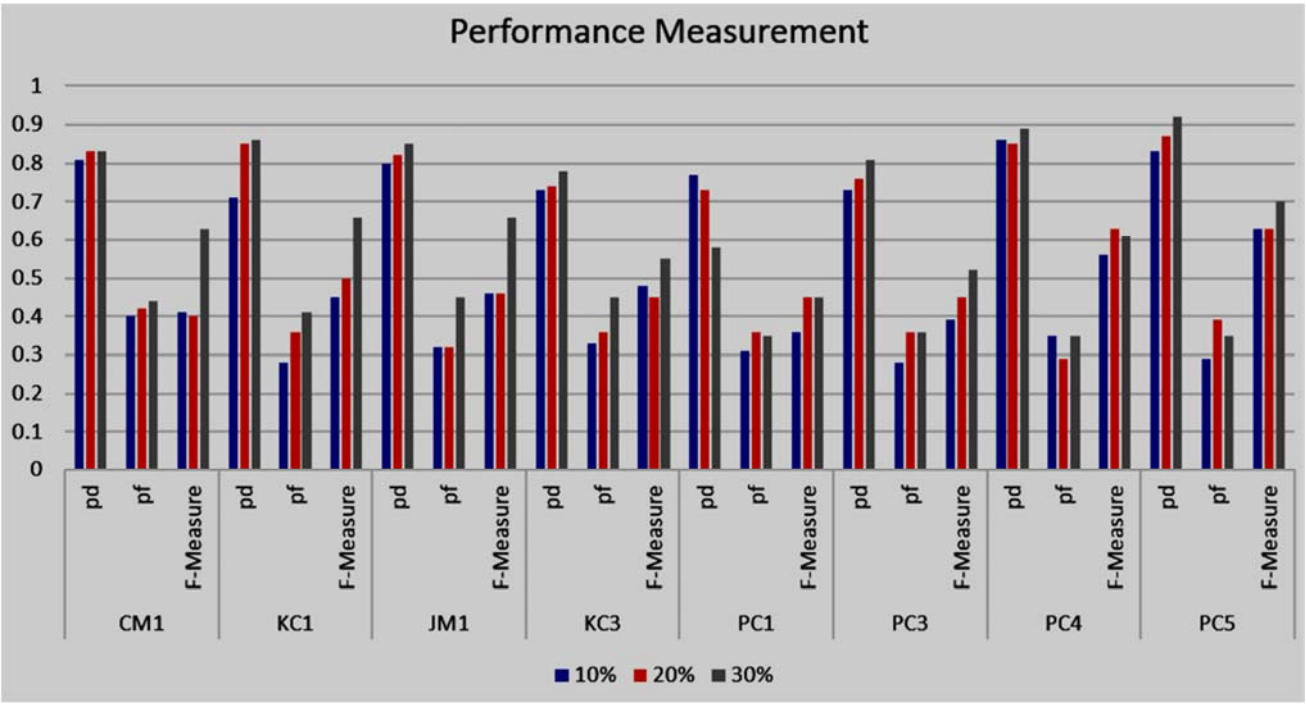

Figure 1. Overall Performance Measurement

The above figure 1 combines the results and illustrates the performance of proposed scheme on various datasets with 10%, 20% and 30% labeling rate. The complete experimental study shows that the proposed semi-supervised learning scheme achieves better performance when compared with the other existing techniques of software defect prediction.

V. CONCLUSION

In this work, we have focused on software defect prediction using machine learning techniques. Several techniques have been introduced for software defect prediction using supervised techniques but these techniques can be implemented for the data where labels are already present and fails to be implemented for unlabeled data. In order to overcome this issue, here we present a semi-supervised learning scheme where unlabeled data are processed and software defects are classified. The complete process is as follows: first of all, we apply, KNN based approach for missing value imputation, in the next phase we apply feature selection approach using principal component analysis (PCA) method, later we apply, SOM clustering approach and finally the obtained clusters are classified using Naïve Bayes classifier. The experimental study shows improved performance when compared with the existing techniques.

REFERENCES

[1] I. Gondra, "Applying machine learning to software fault-proneness prediction", Journal of Systems and Software, vol. 81, no. 2, pp. 186-195, 2008.

[2] E. Weyuker, T. Ostrand and R. Bell, "Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models", Empirical Software Engineering, vol. 13, no. 5, pp. 539-559, 2008.

[3] Kalaivani, N. and Beena, R., "Overview of Software Defect Prediction using Machine Learning Algorithms", International Journal of Pure and Applied Mathematics, vol. 118, no. 20, pp. 3863-3873, 2018.

[4] N. Fenton and J. Bieman, Software metrics: a rigorous and practical approach. CRC Press, 2014.

[5] Ö. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network", Applied Soft Computing, vol. 33, pp. 263-277, 2015..

[6] D. Karaboga and C. Ozturk, "Neural networks training by artificial bee colony algorithm on pattern classification", Neural Network World, vol. 19, no. 3, pp. 279–292, 2009.

[7] A.S. Nickerson, N. Japkowicz and E. Milios, "Using unsupervised learning to guide resampling in imbalanced data sets", In Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, FL, USA, pp. 261–265, January 4–7, 2001.

[8] T. Wang and W. Li, "Naive Bayes Software Defect Prediction Model," 2010 International Conference on Computational Intelligence and Software Engineering, Wuhan, pp. 1-4, 2010.

[9] K. Elish and M. Elish, "Predicting defect-prone software modules using support vector machines", Journal of Systems and Software, vol. 81, no. 5, pp. 649-660, 2008.

[10] L. Guo, Y. Ma, B. Cukic and Harshinder Singh, "Robust prediction of fault-proneness by random forests," 15th International Symposium on Software Reliability Engineering, Saint-Malo, Bretagne, 2004, pp. 417-428.

[11] S. Kanmani, V. Uthariaraj, V. Sankaranarayanan and P. Thambidurai, "Object-oriented software fault prediction using neural networks", Information and Software Technology, vol. 49, no. 5, pp. 483-492, 2007.

[12] S. Wang, T. Liu and L. Tan, "Automatically Learning Semantic Features for Defect Prediction," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, 2016, pp. 297-308.

[13] B. Ghotra, S. McIntosh and A. E. Hassan, "Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 789-800.

[14] Ö. Arar and K. Ayan, "A feature dependent Naive Bayes approach and its application to the software defect prediction problem", Applied Soft Computing, vol. 59, pp. 197-209, 2017.

[15] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu and D. Chen, "FECAR: A Feature Selection Framework for Software Defect Prediction," 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, 2014, pp. 426-435.

[16] I. Laradji, M. Alshayeb and L. Ghouti, "Software defect prediction using ensemble learning on selected features", Information and Software Technology, vol. 58, pp. 388-402, 2015.

[17] G. Mauša and T. Galinac Grbac, "Co-evolutionary multi-population genetic programming for classification in software defect prediction: An empirical case study", Applied Soft Computing, vol. 55, pp. 331-351, 2017.

[18] D. Ryu, O. Choi and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction", Empirical Software Engineering, vol. 21, no. 1, pp. 43-71, 2014.

[19] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics", Cluster Computing, 2018.

[20] [9]R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier", Cluster Computing, 2018.

[21] F. Zhang, Q. Zheng, Y. Zou and A. E. Hassan, "Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier," 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, pp. 309-320, 2016.

[22] Shi Zhong, T. M. Khoshgoftaar and N. Seliya, "Unsupervised learning for expert-based software quality estimation," Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings., Tampa, FL, USA, 2004, pp. 149-155.

[23] C. Catal, U. Sevim, and B. Diri. "Metrics-driven software quality prediction without prior fault data", In S.-I. Ao and L. Gelman, editors, Electronic Engineering and Computing Technology, volume 60 of Lecture Notes in Electrical Engineering, Springer Netherlands, pp. 189–199, 2010.

[24] G. Abaei, Z. Rezaei and A. Selamat, "Fault prediction by utilizing self-organizing Map and Threshold," 2013 IEEE International Conference on Control System, Computing and Engineering, Mindeb, 2013, pp. 465-470.

[25] A.A. Akinduko, E.M. Mirkes, Initialization of Self-Organizing Maps: Principal Components versus Random Initialization. A Case Study, arXiv preprint arXiv:1210.5873, Oct 2012.

[26] "PROMISE DATASETS PAGE", Promise.site.uottawa.ca, 2018. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets-page.html. [Accessed: 13- Nov- 2018].