ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20151219112129

SAMPLING IMBALANCE DATASET FOR SOFTWARE DEFECT PREDICTION USING HYBRID NEURO-FUZZY SYSTEMS WITH NAIVE BAYES CLASSIFIER

K. Punitha, B. Latha

Original scientific paper
Software defect prediction (SDP) is a process with difficult tasks in the case of software projects. The SDP process is useful for the identification and location of defects from the modules. This task will tend to become more costly with the addition of complex testing and evaluation mechanisms, when the software project modules size increases. Further measurement of software in a consistent and disciplined manner offers several advantages like accuracy in the estimation of project costs and schedules, and improving product and process qualities. Detailed analysis of software metric data also gives significant clues about the locations of possible defects in a programming code. The main goal of this proposed work is to introduce software defects detection and prevention methods for identifying defects from software using machine learning approaches. This proposed work used imbalanced datasets from NASA's Metrics Data Program (MDP) and software metrics of datasets are selected by using Genetic algorithm with Ant Colony Optimization (GACO) method. The sampling process with semi supervised learning Modified Co Forest method generates the balanced labelled using imbalanced datasets, which is used for efficient software defect detection process with machine learning Hybrid Neuro-Fuzzy Systems with Naive Bayes methods. The experimental results of this proposed method proves that this defect detecting machine learning method yields more efficiency and better performance in defect prediction result of software in comparison with the other available methods.

Keywords: *Genetic algorithm with Ant Colony Optimization (GACO); NASA's Metrics Data Program (MDP); Semi supervised learning Modified Co Forest method; Software defect prediction (SDP)*

Izbor neuravnoteženog niza podataka za predviđanje grešaka u računalnom programu primjenom hibridnih neuro-fuzzy sustava s Naive Bayes klasifikatorom

Izvorni znanstveni članak
Predviđanje grešaka u računalnom programu (SDP-software defect prediction) je težak zadatak kad se radi o projektima računalnog programa. Taj je postupak koristan za identifikaciju i lokaciju neispravnosti iz modula. Taj će zadatak postati skuplji uz dodatak složenih mehanizama za ispitivanje i ocjenjivanje kad se poveća veličina modula programa. Daljnje konsistentne i disciplinirane provjere programa nude nekoliko prednosti, na pr. točnost u procjeni troškova i programiranja projekta, povećanje kvalitete postupka i proizvoda. Detaljna analiza metričkih podataka programa također može značajno pomoći u lociranju mogućih grešaka u programskom kodiranju. Osnovni je cilj ovoga rada predstaviti metode za detekciju i otkrivanje grešaka u programu primjenom postupaka strojnog učenja. U radu su korišteni nebalansirani nizovi podaka iz NASA-inog Metrics Data Programa (MDP) i programska metrika niza podataka izabrana je primjenom Genetičkog algoritma metodom Optimizacije kolonije mrava (Ant Colony Optimization - GACO). Postupak uzorkovanja metodom Modified Co Forest - polu-nadgledanog učenja, generira balansirano označene nizove podataka koristeći nebalansirane nizove, a primjenjuje se za učinkoviti postupak otkrivanja greške u programu s Hibridnim Neuro-Fuzzy sustavima za strojno učenje po Naive Bayes metodama. Eksperimentalni rezultati predložene metode dokazuju da je ova metoda za otkrivanje greške u računalnom program učinkovitija od drugih postojećih metoda, s boljim rezultatima u predviđanju greške.

Ključneriječi: *Genetički algoritam s optimizacijom kolonije mrava (GACO); NASA-in Metrics Data Program (MDP); modificirana Co Forest metoda polu-nadgledanog učenja; predviđanje greške u računalnom programu (SDP)*

1 Introduction

The software defect prediction is considered as a crucial activity in the process of decision support in the field of software quality assurance [1÷4].

In a view to apply defect prediction schemes in the process of assuring quality software products, several types of machine learning classification algorithms have been adopted for predicting the software defect [5]. The process of defect prediction has been utilizing various machine learning approaches including Logistic Regression [6], Decision Trees [7], Neural Networks [8] and Naive-Bayes [9]. The two important data quality aspects such as class imbalance and noisy data set attributes [10] generally influence the performance of classification.

The imbalanced aspects of the datasets holding software defect have lesser defective modules than the defect-free datasets [11]. As many of the datasets are found to be prone to non-defect type, prediction of software defect using models based on the imbalance characteristics becomes impractical. There is a decreased performance of software defect prediction owing to the noisy attributes of the datasets [12]. The presumption of noise prone data points having malicious aspects present in the datasets is impossible as the data modelling is based on the simple method. In the field of machine learning techniques, feature selection plays a key role by involving the learning task that enables the process of predicting datasets possessing high dimensional and noisy attributes. Most feature selection algorithms involve local search rather than the global search throughout the process. This is due to the reason that the issues in the feature selection methods are found in the regions ranging from sub-optimal and near optimal ends. Hence, finding the solutions in the regions ranging between near-optimal and optimal solutions becomes very difficult by means of feature selection techniques.

In contrast to feature selection, the method of genetic algorithm enables the capability of global by considering the entire region of search space. This aspect eventually results in the substantial increase in the capacity to obtain high-quality solutions in a given time period. The present work deals with the method of modified co-forest possessing multi-class classification datasets. This method helps in resolving the problem of imbalance dataset during the process of predicting the software defects by means of employing semi-supervised learning. This

effective method of final defect prediction involves the process of converting the unlabelled data into balanced labelled data. By combining Genetic Algorithm method with Ant Colony Optimization (GACO) and Bagging technique helps in improving the overall accuracy of predicting the software defects. The method of GACO offers the task of region selection as that of the feature selection technique whereas the bagging technique solves the problem of handling the class imbalance. The reason for selecting the bagging technique is attributed to the efficient handling of class imbalance problem while occurring in the process of dataset selection. The hybrid method of Fuzzy Naive Bayes classifier learning helps in prediction of the noisy defects present in the software modules. This is done by utilizing the selected modules of software data matrices with the help of the GACO as important technique in the method of software detection and prediction.

The research strategy as reported earlier [13] suggested the need for utilizing several machine learning algorithms for the performance stability in spite of employing various imbalance levels for identifying the prediction datasets having the software defect. The primary case study was performed on the NASA MDP dataset from open repository by means of the multivariate binary logistic regression for both forward and backward feature selection. The performance analysis results also have shown the instability with the imbalance of about 80%. The method of feature selection approach integrated with data sampling technique has been proposed [14] which enables to resolve the problems such as high dimensional defects and class imbalance existing in data repositories in the process of software quality modelling. The process of choosing a data subset having required aspects so as to either maintain or enhance the quality of prediction models is termed the feature selection. The data sampling task aims to identify the well balanced dataset with instances of either adding or removing the data. The integration of both these techniques would result in incorporating three various approaches as given below:

- Performing Sampling before Feature Selection, however Retaining the Unsampled Data Instances.
- Performing Sampling before Feature Selection, Retaining the Sampled Data Instances,
- Performing Sampling after Feature Selection.

By considering nearly 6 real-world software datasets, the empirical study has been performed. This is conducted by employing the selection approach of hybrid correlation based feature selection with the technique of random under sampling method which incorporates the filter-based (there is an absence of learning algorithm in the process of selection) feature subset for performing the study. The study results have deduced that it follows the selection Approach 1 in which the sampling is performed before feature selection in which the unsampled data instances are retained. Further, this approach is observed to perform well when compared to the rest of the other approaches.

The software defect prediction using the method of dictionary learning has also been described earlier [15]. This technique utilizes the metric features which have been derived out of the open source software by learning the sparse representation coefficients from the multiple dictionaries (including total dictionary and defective module and defective-free module sub-dictionaries). Moreover, this technique has also considered the problem of misclassification cost of defective modules as it would lead to incur higher risk cost which is relatively critical than the defective-free modules. This shortcoming has necessitated the introduction of a cost-sensitive discriminative dictionary learning (CDDL) approach to classify and predict the software defect systems. The performance evaluation is done by comparing the techniques in resolving the test data which is most commonly derived from the datasets of NASA projects. The performance evaluation results have also confirmed the efficiency of CDDL approach outperforming other related state-of-the-art techniques for defect prediction.

In the area of Software Engineering, the method of software defect prediction (SDP) is still emerging as a potential field of research as it aims to identify the software modules which are prone to defects. The efficient allocation of restricted testing resources helps in SDP to identify the defect-prone modules. It is known that SDP requires acceptable and sufficient local data within a company, however there are few instances in which local data has not been available as in case of pilot projects. Such projects or companies having no local data can utilize the approach of the cross-project defect prediction (CPDP) by making use of the external data for enabling the performance of the classifiers.

The critical challenge in employing CPDP approach lies on variation in the distributions considered between test and training data. This can be overcome by choosing the source data instances which are almost like target data for dealing with the classifiers' building. It is usual that the software datasets possess the problem of class imbalance that specifies the lesser defective class to clean class ratio. This lesser ratio would generally affect the classifier performance. Hence, a novel method of hybrid classification called Hybrid Instance Selection [16] was proposed in which the Nearest-Neighbor (HISNN) was introduced by selective learning of both local knowledge (via *k*-nearest neighbour) and global knowledge (via Naıve Bayes). There are also examples showing strong local knowledge that is identified through nearest - neighbours carrying the label of the same class. The previous reported works had employed either low PD (probability of detection) or high PF (probability of false alarm) which are considered impractical in real-world software systems. The experimental results had specified the higher performance of HISNN generates when compared to the overall performance of high PD and low PF.

The prediction of software quality has been given by the new approach [17] in which the Support Vector Machine (SVM) is adopted for classifying the software modules in accordance to the complexity metrics. The conventional models for predicting software quality have not performed well in SDP as only limited information pertaining to software complexity metrics was available during the life cycle of early software systems. It is well-known that SVM typically performs well ahead than other models as it can resolve even high-dimensional spaces

under the situation having small insignificant training samples. This drives the need for incorporating the SVM-based software classification model as its features are more suitable for early prediction of the software quality even under the condition of only a small number of sample datasets available in the search space. The performance results based on experiments based on data corresponding to a Medical Imaging System software metrics confirmed that the proposed SVM prediction model facilitates better prediction of software quality when compared to other commonly used software quality prediction models.

Input	The labelled set s_j, the unlabeled set u_j within class $j$the confident threshold θ, the number of random trees N
Step 1	Construct a random forest with N random trees $R = \{r_1, r_2, \ldots, r_N\}$ in class j
Step 2	Repeat the step 3 to 9 until none of the random trees of Rchanges
Step 3	Update the number of $iteration, i(i = 1, 2, \ldots)$ of each labeled and unlabelled dataset in class j
Step 4	For each random tree $r_n in\ R$, do step 5 to 9
Step 5	Construct associated ensemble R_{-n} within class j
Step 6	Use $R_{-n}$to label all the unlabeled data, and estimate the labeling confidence
Step 7	Add the unlabelled data whose labelling confidences are above threshold θ to a newly labeled set $s'_{j,i}$
Step 8	Under sample $s'_{j,i}$to make (1) holds. If it does not hold, skip step 9
Step 9	Update $r_n$by learning a random tree using s_j cup $s'_{j,i}$
Output	A balanced labelled dataset is generated by the majority voting from all the component trees within the class j

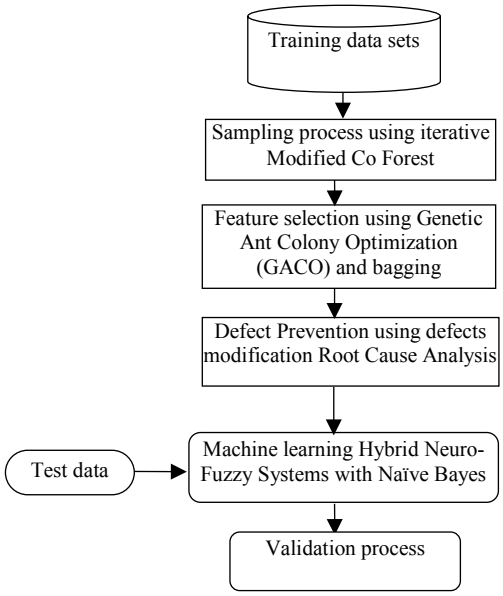

Figure 1 Block diagram of the proposed methodology

2 Proposed methodology

The effectiveness of Software defect prediction models is largely shaped by the class distribution of the training Datasets. Class distribution is defined as the number of instances of every class in the training dataset. If the number of instances that belongs to one class is much higher than the number of instances belonging to another class, then the problem is called class imbalance problem. The class with more instances is known as majority class and the one with lesser instances is known as minority class.

The problem widens when the class under consideration, i.e. the faulty class is denoted by fewer instances. Here this work has been introduced for addressing this imbalance problem defect detection and prevention method and the flow diagram is given in Fig. 1.

2.1 Iterative Modified Co Forest method for sampling

For solving the software defects in case of real-world applications, the sampling of the labelled datasets is done with respect to the method of normalization distribution. It is easier to obtain the sample labels in few classes whereas other classes are difficult to get in spite of the classes being in similar significant level. In a view to deal with the problem of imbalance classification in approaches related to semi-supervised learning machine, the space having low-dimensional features is assumed to present several unlabelled samples around a labelled sample. Therefore, in this work, some unlabelled samples are selected in case of minority class for creating a balanced dataset. This is explained by the description of the sampling model employed in the proposed work as given below:

Considering the multiclass classification scenarios; let $s = \{s_1, s_2, ..., s_n\}$ represent the size set of labelled samples in labelled dataset, where s_j (j = 1, ..., n) is the number of labelled samples in class j. In this work, a standard variance $var(s)$ is used for representing the dispersion degree of the quantity of labelled samples in every class, and the imbalance ratio $var(s)$ can be defined as follows:

$$var(s) = \left(\frac{1}{n}\sum_{j=1}^{n}(s_j - \bar{s})\text{^}2\right)^{\frac{1}{2}} \quad (1)$$

where $\bar{s} = \left(\frac{1}{n}\right) s_{sum}$, $s_{sum} = \sum_{j=1}^{n} s_j$. In this work Co Forest [18] is applied in order to limit the adverse effect of imbalanced labelled dataset in the defect prediction process. This proposed work is based on a popular ensemble learning algorithm called Random Forest for tackling the issues pertaining to the determination of the most confident examples for labelling and producing the final classification.

The Algorithm of Modified Co Forest is proposed in Algorithm 1. The proposed algorithm works as follows. Let s_j represent the labelled data set in class j and u_j represent the unlabelled data set. First, N random trees are initialized from the training sets bootstrap-sampled from the labelled data set s_j in class j for the creation of a random forest. Afterwards, in each learning iteration each random tree is fine-tuned with the original labelled examples s_j and the newly labelled examples s_j chosen by its associated ensemble (i.e., the ensemble of the other random trees with the exception of the current tree within the class j). The learning process repeats all the labelled samples in class j with labelled samples in other classes till a certain stopping criterion is reached. At the end, the prediction is made on the basis of the majority voting

from the ensemble of random trees. It is to be noted that in this way, Modified Co Forest is capable of exploiting the advantages of both semi-supervised learning and ensemble learning at the same time, as said in [19].

Algorithm 1: Sampling Modified Co Forest

The semi-supervised learning process will be useful only if the following condition is satisfied.

$$\frac{\hat{e}_{i,j}}{\hat{e}_{i,j-1}} < \frac{w_{i,j-1}}{w_{i,j}} < 1, \quad (2)$$

where $\hat{e}_{i,j}$ and $\hat{e}_{i,j-1}$ represent the estimated multi class classification error of the i^{th} random tree in the j^{th} and $(j-1)^{\text{th}}$ class, respectively, and $w_{i,j}$ and $w_{i,j-1}$ represent the total weights of its newly labelled sets $s_{i,j}$ and $(L'_{i,j-1})$ in the j^{th} and $(j-1)^{\text{th}}$ class, respectively, and $i \in \{1, 2, \ldots, N\}$.

2 .2 Feature selection using hybrid Genetic algorithm with Ant Colony Optimization (GACO) and bagging Technique

From the datasets having balanced and labelled samples, the method of feature selection is employed for performing the software defect prediction using the Bagging technique and Genetic Algorithm (GA) combined with ACO (GACO). The aim of utilizing the GACO is to identify the optimum solution among the set showing likely potential solution. The probabilistic technique inspired by means of the natural foraging behaviour of ants enables the performance of cross validation approach in ant population. This technique by incorporating the genetic algorithm for the process of feature selection as similar to ant population is termed the genetic ant colony optimization. The reason for highly efficient cross validation process depends on the nature of the real ants identifying the shortest route by pheromone deposition along the routing path which subsequently offers improved fitness value in choosing the shortest path.

2.2.1 Genetic Algorithm

Accuracy of the final result of the defect prediction process is improved on the basis of the number of selected features and the feature cost, which is useful for constructing a fitness function with the help of the GA. Every chromosome is assessed by the following fitness function equation.

$$Fitnessvalue = W_a * a + W_f \left(P + \left(\sum_{i=1}^{n_f} c_i * f_i\right)\right)^{-1} \quad (3)$$

where ii is classification accuracy, W_a is weight of classification accuracy, f_i is feature value, W_f is feature weight, c_i is feature cost, P is constant.

2.2.2 Ant Colony Optimization (ACO)

ACO Graph representation. The problem pertaining to the process of feature selection can also be mentioned and resolved in the ACO based problem solving method. It is known that graphical representation of the problem is required by the ACO in which the features are represented by the nodes. The edges between the nodes represent the method of choosing the subsequent feature. Then, the process of searching subset possessing the optimal feature takes the form of an ant traversed along the graphical representation where in the traversal stopping criterion can be satisfied by the lesser number of nodes which have spanned the path.

Heuristic attractiveness. In the proposed work, the performance of the classifier algorithm is considered with respect to the available heuristic information for selecting the feature datasets. The heuristic attractiveness tends to build the probabilistic transition rule by merging the pheromone levels in traversal and nodes. This enables the representation of the probability of ant k including its feature i in the resulting solution at the given time step of t:

$$P_i^k(t) = \begin{cases} \frac{[\tau_i(t)^\alpha . [\eta_i]^\beta]}{\sum_{u \epsilon j^k} \tau_u(t)^\alpha . [\eta_u]^\beta} & \text{if } i \epsilon J^k \\ 0 & otherwise \end{cases} \quad (4)$$

where J^k denotes possible feature datasets that can be included in the partial solution; τ_i and η_i are respectively the pheromone value and heuristic desirability that is associated with feature i. α and β are two parameters which help in determining the relative importance between pheromone value and heuristic information.

Pheromone update rule. Once all the ants have finished their solutions, triggering of pheromone evaporation on all nodes is done, and then in accordance with Eq. (3) each ant k deposits a quantity of pheromone, $\Delta(ii)$, on each node that it has utilized,

$$\Delta\tau_i^j(t) = \begin{cases} \phi . \gamma . (V^j(t)) + \frac{\varphi(n - [V^j(t)])}{n} & if i \epsilon V^j(t) \\ 0 & otherwise \end{cases} \quad (5)$$

where $V_j(t)$ the feature subset is discovered by ant j at iteration t, and $|V^j(t)|$ is feature subset length. The pheromone is updated based on both the measure of the classifier performance, $((t))$, and feature subset length. φ and ϕ are two parameters that do the control of the relative weight of classifier performance and feature subset length $\phi \in [0,1]$ and $\varphi = 1-\varphi$.

2.2.3 Bagging approach

The Bagging approach, otherwise named as bootstrap aggregating method has been reported [20] which has aimed at combining the classifications of stochastically generated training sets for overall improvement of the classification performance. This is done by the bagging classifier by means of differentiating the training set into several other training sets using the random sampling method. This in turn leads to the construction of sampling models with respect to the newly identified training sets. Finally, the classification result is achieved by electing the models based on votes. This tends to limit the variance thus offering to avoid the problem of over-fitting. The detailed description of the process of bagging technique is explained as given below.

Given a standard training set D of size n, bagging builds m new training sets D_i, each of size $n' < ii$, by sampling from D uniformly and with replacement. By doing the sampling with replacement, few observations may be iterated in each Di. If $n' = n$, then for large n the set D_i is anticipated to have the fraction $(1 - 1/e)$ of the unique examples of D, the rest being duplicates. This kind of sample is referred to as a bootstrap sample. The m models are fitted by making use of the above m bootstrap samples and merged by doing the average of the output (for regression) or voting (for classification). Bagging results in improvements for unstable procedures [20], which consist of neural network, classification and regression trees, and subset selection in linear regression.

Steps for proposed feature selection GACO Algorithm 2:

Step 1:	Initialization process. Determine the population of ants. Cross validate the GA with population of ants to find the fitness value for best path in graph representation. Set the intensity of pheromone trial associated with any feature. Determine the maximum of allowed iterations.
Step 2:	Solution generation and evaluation of ants. Assign ant to features and evaluate the mean square error (MSE) of the classifier. If an ant is not able to decrease the MSE of the classifier in ten successive steps, it will finish its work and exit.
Step 3:	Evaluation of the selected subsets. Sort selected subsets according to classifier performance and their length. Then, select the best subset.
Step 4:	Check the stop criterion. Exit, if the number of iterations is more than the maximum allowed iteration, otherwise continue.
Step 5:	Pheromone updating.
Step 6:	Generation of new ants.
Step 7:	Go to 2 and continue.

2.3 Defect prevention

The process of defect prevention plays an important, crucial role in any of the software projects. It must be noted that the project team used to concentrate much on defect detection and prevention in most of the present software organizations. This implies the need for considering the defect prevention as an important component requiring attention. Hence, it is necessary to bring and establish the measures for defect prevention even at the initial stages of the project till its completion and execution. On account of cost reduction to minimal due to the necessary prevention measures, the advantages owing to the entire cost saving strategy become remarkably higher when compared to the expenses sustained during defect solving at the final stage. This makes it clear that the prevention of the defects at an initial stage is considerably better in terms of saving cost, time and the required resources. The basic information on techniques and processes related to defect injection offers the efficient prevention of the defect. On practical application of the awareness on defect prevention, the quality of software products will be substantially improved which subsequently would improve the efficiency of overall production.

2.3.1 Defect prevention using defects Modification Requests Root Cause Analysis Scheme

The prime challenge in conducting any method of software defect measurement remains on identifying the defect properties for a minimal set as it offers the facile classification approach. While performing the complete mapping of the overall activities during the development process, the overhead which is being added becomes minimal in the development process. In order to establish the instruments for measuring the casual relationship between the software defects, root cause analysis (RCA) utilizes the modification request based two main classification methods such as Defect Type and Defect Trigger. The process of defect type characterizes the defect dataset based on the nature of the variation to properly solve the defect. This parameter measures the overall progress of the product during the entire process of development. The aspects of defect trigger follow the defect characterization based on the catalyst which is responsible for causing the defect and hence leads to the failure. This parameter helps in yielding the measure of the verification process. The present work proposes the method of RCA that plays a significant role during the prediction analysis of defects in the software systems. The RCA deals with finding the basic reason for the defect so as to initiate remedial action for removing the entire root cause of the defect. This is possible by analysing all known defects at the time specified [21]. This requires qualitative analysis which can be restricted only by the investigation carried out by the human range of abilities. The results of the qualitative analysis become the reliable feedback source to the next upcoming software developers as it offers the improvement in both quality and the productivity of the real-world software applications [22].

2.4 Software defect prediction

The process of predicting the software defects involves identifying the defective modules present in the software. The high quality software can be developed only when the end product is found to have only few defects. The method of early prediction and detection of software defects would ultimately enable the minimization of the development costs, resolving load and bring about the software having high reliability. Thus, the approach of analysing the defect prediction becomes significant in reaching the best quality in software systems. The well-known crucial problem lies on the software quality and reliability in the filed of the software defect prediction schemes.

2.4.1 Software defect prediction using Hybrid Fuzzy Naïve Bayes Classifiers

The approach related to fuzzy classifiers utilizes the classifier neuro-fuzzy systems for getting fuzzy classifiers from the datasets used for the learning techniques working based on the neural network. These classifiers

are widely applied for predicting the software defects owing to some similarities between a neuro-fuzzy classifier and a Naive Bayes classifier in terms of structure. The underlying principle behind this classifier further reinforces the idea of mapping the features of the software code for further improving the potential of the software defect prediction.

2.4.1.1 Fuzzy classifier

Fuzzy rules are appropriate for representation of the classification knowledge. It is chiefly the abstraction from numbers to linguistic attributes which makes them easy to be read and to be interpreted. In addition to this, fuzzy rules are employed in a very intuitive and comprehensible fashion for classifying new datasets. The fundamental idea of fuzzy classification systems is the description of the areas of the input space, to which various class labels are allocated, by invariable cluster prototypes. These prototypes are defined by several numbers of fuzzy sets which indicate them in the different dimensions of the domain taken under consideration. That is, a specific cluster β labelled with class c is defined by a fuzzy classification rule r of the form [23]: if A_1 is μ_1 and A_2 is μ_2 and . . . and A_n is μ_n then pattern (A_1, A_2, ..., A_n) belongs to class c, where the μ_j are fuzzy sets that describes the cluster β write to attribute A_j. Additionally, few approaches present so-called rule weights W_r, whose intention is indicating the "importance" or "reliability" of a rule. The degree of fulfilment of a rule is computed from the membership degrees of the antecedents with a t-norm, usually $T_{\min}$ or $>T_{\text{prod}}$. For instance, if the t-norm T_{prod} is applied, the degree of fulfilment or activation $o(\omega)$ of rule r at $\omega \in \Omega$ is defined as:

$$Or(\omega) = \\ = or(A_1 = a_{i1}^1, ..., A_n = a_{in}^n = \prod_{k=1}^n \mu_j)(A_j = a_{ij}^{(j)}). \quad (6)$$

For computing the output of a fuzzy classification system, as a first step, the degree of fulfilment of each rule in the rule base is computed. Later, for each class, the sum, the maximum or the average of the (possibly weighted) rule activations is calculated. The output class is decided by a winner-takes-all principle, i.e., the class with the highest accumulated (weighted) activation is predicted.

2.4.1.2 Neuro-fuzzy method

Rule induction in systems such as Neuro Fuzzy Classification, which begin with a fixed number p_j of manually, defined or equally spaced fuzzy sets as a partitioning of the domain of attribute, is simple: First the rule antecedents are built. After this ends, the sample cases are investigated in turn. For each case the fuzzy sets are analysed and for each dimension that a fuzzy set is chosen, which provides the highest membership degree. Then a rule is developed for each distinct choice of fuzzy sets, which in turn becomes the antecedent of the rule. The consequent of a rule is decided from the classes of all sample cases that are covered by the rule. In Neuro Fuzzy Classification the activations of the rule are summed per class over the (covered) sample cases and then the class with the highest activation sum is selected as the consequent. In the learning phase the fuzzy partitioning of the input dimensions is brought into use (i.e., the parameters of the fuzzy sets are altered) for optimizing the location and extension of the clusters. Generally observed issues of neuro-fuzzy classifiers are either a huge number of rules (that are difficult to be read and interpreted) or a sparse covering of the input space (at times resulting in inadequate generalization).

2.4.1.3 Naive Bayes classifier

Naive Bayes classifiers [24] are an old and popular kind of classifiers, i.e., of programs which assign a class from a predetermined set to an object or case which is under consideration on the basis of the values of attributes utilized for describing this object or case. They do so by making use of a probabilistic approach, i.e., they attempt to calculate conditional class probabilities and then do the prediction of the most probable class. To be more specific, let C represent a class attribute with a finite domain of m classes, i.e., $dom(C) = \{c_1, ..., c_m\}$, and let A_1, ..., A_n be a set of (other) attributes that is used for describing a case or an object of the domain considered. These other attributes may be symbolic, i.e., $dom(A) = \{a_1^{(j)}, ..., a_{mj}^{(j)}\}$, or numeric, i.e., $dom(A_j) = \mathbb{R}$. For simplicity, the notation a_{ij}^j for a value of an attribute A_j is used, with no dependence on whether it is a symbolic or a numeric one. With this notation, a case or an object can be defined by an instantiation $\omega = (a_{i1}^1, ..., a_{in}^n)$ of the attributes A_1, ..., A_n and thus the universe of discourse is $\Omega = dom(A_1) \times ... \times dom(A_n)$. For a given instantiation ω, a Naive Bayes classifier seeks to calculate the conditional probability,

$$(C = c_i \mid \omega) = (C = c_i \mid A_1 = a_{i1}^1, ..., A_n = a_{in}^n) \quad (7)$$

for all c_i and then predicts the class c_i for which this probability is the highest. Of course, it is generally impossible to save all of these conditional probabilities in an explicit manner, such that a simple lookup would be the only thing necessary for finding the most probable class.

2.4.1.4 Hybrid Neuro-Fuzzy Systems with Naive Bayes classifiers for defect prediction

The mapping of a Naive Bayes classifier can be done to a neuro-fuzzy classification system, in case the $ii -$ prod is utilized and few limitations are laid on the fuzzy sets and the rule weights. Particularly, the fact that probability distributions/density functions are normalized to 1 is to be taken care of, i.e.,

$$\sum_x(x) = 1 \text{ or } \int_x^1(x) = 1.$$

For simplifying the following explanation, let us first assume that there is an analogy to a Naive Bayes classifier with only one cluster β_i per class, defined by a (fuzzy)

rule. With this limitation, the membership functions $\mu_{ri,}$ can be used for defining the probability density functions f for each attribute A_j given the class:

$$\mu_{ri}\left(x_j = a_{ij}^j := f(A_j = a_{ij}^j \mid C = c_i)\right) \quad (8)$$

In this proposed system it is assumed that a class is defined by more than one rule. With this the limitations of Naive Bayes classifiers are taken over and hence gains flexibility in order to describe the conditional distributions.

Intuitively, each class is split into a set of subclasses, each of which is defined by a separate (fuzzy) rule. Nonetheless, the fact has to be admitted that for ensuring interpretability, neuro-fuzzy systems also pose restrictions which are not observed in Naive Bayes classifiers. On the contrary, in a Naive Bayes classifier there are always as many (independent) distribution functions per dimension of the input space as there are classes available, in a neuro-fuzzy system the number of fuzzy sets per dimension is determined by the selected fuzzy partition. If there is only one rule per class available, this is no actual restriction, as in most of the classification issues the number of classes is not very large.

With the presence of more than one rule per class, this restricts the degrees of freedom. It should not be taken as a setback, though, since too many attributes of freedom intend to result in overfitting and therefore poor generalization abilities. With more than one rule per class present, the requirements that the fuzzy sets have to meet are, of course, similar as above. Only the derivation of the rule weights is little more complicated, as this work has to take the prior probability for each class c_i into consideration and the conditional probability that a sample case for this class belongs to the cluster β_{ik}, i.e., the k^{th} cluster defining class c_i which is denoted by rule r_{ik}. For the sake of simplicity this proposed work uses two rule weights: $w_{rik}{}^{\text{class}}$, which indicates the prior probability of class c_i, and $W_{rik}{}^{\text{cluster}}$, which again states the conditional probability of cluster β_{ik} given that the case belongs to class c .

A Naive Bayes classifier can be mapped onto a neuro-fuzzy classifier. Additionally, it is possible to make use of more than one cluster (i.e., more than one rule) per class for describing more complicated distributions and so this work may receive a more powerful classifier for predicting the defects from the software product module. With this mapping, neuro-fuzzy learning techniques can be utilized for learning and optimizing a Naive Bayes classifier. As obvious, the learned probability distribution functions need not match with the standard maximum likelihood prediction results, as the aim of the employed learning algorithms is the minimization of the number of misclassifications and not finding the maximum likelihood prediction.

3 Experimental results

This section explains the data sets, learning algorithms, and evaluation criteria that have been used in this proposed work. The data sets selected differ in data sizes, and programming languages. The selected learning algorithm does the defect prediction in the various kinds of datasets used in this work.

3.1 Data sets

The data that is used in this study was provided by the NASA MDP repository. This repository currently includes 13 data sets that are intended for software metrics research. The given 4 of these data sets were utilized in this work: the brief details of them are illustrated in Tab. 1. Each of the MDP data sets denotes a NASA software system/subsystem and consists of the static code metrics and the respective fault data for each of the comprising modules.

Table 1 Datasets from NASA MDP used in proposed software defect detection process

Data	Language	Examples	No. of attributes	Defect percentage
PC1	C	1107	21	6,940
PC2	C	5589	26	0,423
PC3	C	1563	37	10,230
PC4	C	1458	37	12,200

The data that is used in this research are gathered from the NASA MDP repository. NASA MDP repository is a database which is a storage of problem, product, and metrics data [25]. The primary aim of this data repository is providing project data to the software community [26÷28]. While doing so, the Metrics Data Program gathers artefacts from a huge NASA dataset, does the generation of metrics on the artefacts, and then again generates reports which are made available to the public free of cost. The data which are made available to general public have been sanitized and authorized for publication by the Metrics Data Program Web site by officials who represent the projects from where the data originated.

Table 2 Software Metrics in PC1, PC2, PC3 and PC4 dataset used in proposed methodology

Code Attributes		NASA MDP dataset			
		PC1	PC2	PC3	PC4
LOC counts	LOC total	√	√	√	√
	LOC blank	√	√	√	√
	LOC code and comment	√	√	√	√
	LOC comments	√	√	√	√
	LOC executable	√	√	√	√
Halstead	content	√	√	√	√
	difficulty	√	√	√	√
	effort	√	√	√	√
	Error est	√	√	√	√
	length	√	√	√	√
	level	√	√	√	√
	Prog time	√	√	√	√
	volume	√	√	√	√
	Num operands	√	√	√	√
	Num operators	√	√	√	√
	Num unique operands	√	√	√	√
	Num unique operators	√	√	√	√
McCabe	Cyclomatic complexity	√	√	√	√
	Cyclomatic density	√	√	√	√
	Design complexity	√	√	√	√
	Essential complexity	√	√	√	√
Programming language		C	C	C	C
Number of code attributes		37	77	37	37
Number of modules		1059	4505	1511	1347
Number of fault-prone modules		76	23	160	178
Percentage of fault-prone modules		7,18	0,51	10,59	13,21

In this work four software defect prediction data sets from NASA MDP are used. Individual attributes per data set, along with few general statistics and descriptions, are given in Tab. 2. These data sets have large number scales of line of code (LOC), various software modules coded attributes chosen by programming language C.

3.2 Performance measurement

In this work, the accuracy of predicting the number of defects by making use of machine learning prediction systems is evaluated. Here two levels of data like training and testing data for the final prediction are used. The input attributes (input data) are considered as contiguous values, whereas the output takes discrete or continuous values based on the classifier employed. Software defect predictor performance of the proposed scheme based on Accuracy, precision and recall, is defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} = \\ = \text{the precentage of prediction that are correct.} \quad (9)$$

Where: *TP* – TruePositive, *TN* – TrueNegative, *FP* – FalsePositive and *FN* – FalseNegative.

3.3 Prediction accuracy

From the accuracy graph, the novel hybrid Neuro-Fuzzy Systems with Naive Bayes algorithm yielding different accuracy on PC1, PC2, PC3 and PC4 datasets is shown.

Fig. 2 shows the proposed defect prediction method has high accuracy value in all of the datasets which have been used in this work rather than the other available prediction techniques.

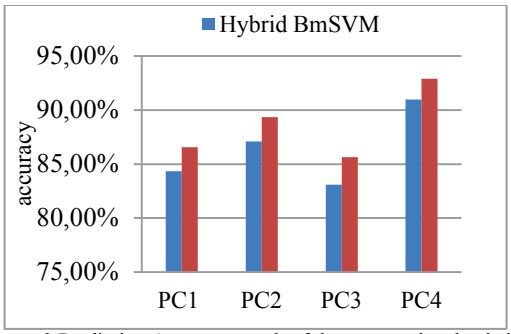

Figure 2 Prediction Accuracy result of the proposed and existing method

3.4 Precision

Precision and recall are then defined as

$$\text{Prescision} = \frac{TP}{TP + FP}. \quad (10)$$

From Figs. 3, 4, and 5, the result indicates the novel hybrid machine learning algorithm precision, recall and *F*- measure values from the 4 different datasets utilized in this work. The proposed hybrid machine learning defect prediction method gives better precision, recall and *F*-measure value compared to other existing BmSVM methods.

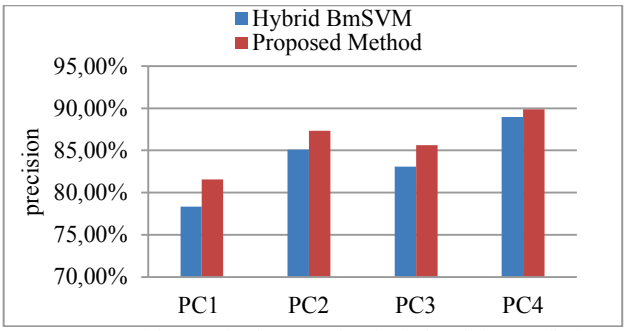

Figure 3 Precision result of proposed and existing defect prediction methods

3.5 Recall

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (11)$$

Recall in this context is also known as the true positive rate or sensitivity, and precision is also called as positive predictive value (PPV); other related measures that are used in classification are true negative rate and accuracy. True negative rate is also known as specificity.

$$\text{True negative rate} = \frac{TN}{TN + FP}. \quad (12)$$

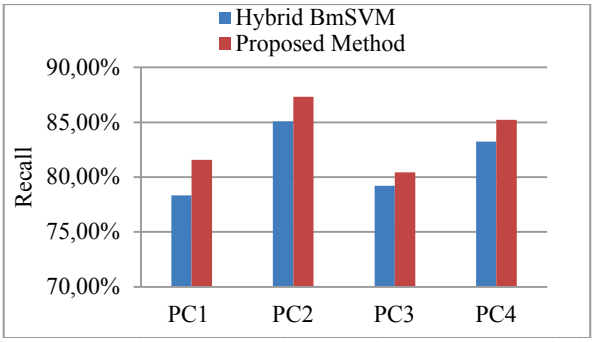

Figure 4 Recall of Proposed and Existing defect prediction methods

3.6 *F*-measure

The *F*-measure calculates some average of the information retrieval precision and recall metrics.

$$F = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Precision} + \text{Recall}} \quad (13)$$

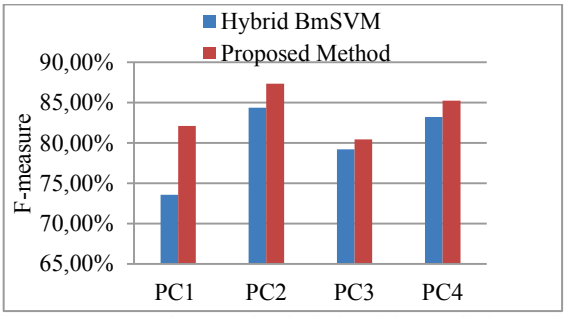

Figure 5 *F*-measure of proposed and existing defect prediction methods

4 Conclusion and future work

In this proposed work, a novel defect prediction model on the basis of the machine learning methods is used. Software defect prediction is useful for developers for the identification of defects in accordance with the current software metrics with novel hybrid machine learning Hybrid Neuro-Fuzzy Systems with Naive Bayes technique. The goal of this work is the analysis of the classification performance of novel hybrid for Neuro-Fuzzy Systems with Naive Bayes classifiers. PC1, PC2, PC3 and PC4 datasets were employed for the evaluation of the machine learning algorithm that is utilized in this software defect prediction process. The experimental results of the proposed method are analysed by making use of performance parameters like precision, recall and f-measure. These performance parameters are also helpful in evaluating the defect prediction efficiency of the proposed methods applying software metrics of the balanced dataset in software defect prediction process and this method's performance results are also compared with the other existing methods like hybrid BmSVM. The comparison result proves that this proposed hybrid learning method for defect prediction is far more efficient in comparison to the other available methods. The future work is aimed at establishing an improved method for predicting the software quality by means of combining various classifiers on the basis of different software measures and different voting schemes and then the effect of classifier with different feature selection methods is analysed. It is also aimed at finding out whether the cost sensitive learning algorithms can be utilized in order to build better defect prediction models.

5 References

[1] Naik, K.; Tripathy, P. Software Testing and Quality Assurance. John Wiley & Sons, Inc. (2008). DOI: 10.1002/9780470382844

[2] McDonald, M.; Musson, R.; Smith, R. The practical guide to defect prevention. // Control. (2007), pp. 260-272.

[3] Catal, C. Software fault prediction: A literature review and current trends. // Expert systems with applications. 38, 4(2011), pp. 4626-4636. DOI: 10.1016/j.eswa.2010.10.024

[4] Menzies, T.; Zach, M.; Turhan, B.; Cukic, B.; Jiang, Y.; Bener, A. Defect prediction from static code features: current results, limitations, new approaches. // Automated Software Engineering. 17, 4(2010), pp. 375-407. DOI: 10.1007/s10515-010-0069-5

[5] Mues, C.; Baesens, B.; Lessmann, S.; Pietsch, S. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. // IEEE Transactions on Software Engineering. 34, 4(2008), pp. 485-496. DOI: 10.1109/TSE.2008.35

[6] Denaro, G., Estimating software fault-proneness for tuning testing activities. // Proceedings of the 22nd International Conference on Software engineering (ICSE '00), (2000), pp. 704-706. DOI: 10.1145/337180.337592

[7] Khoshgoftaar, T. M.; Seliya, N. Tree-based software quality estimation models for fault prediction. // Proceedings Eighth IEEE Symposium in Software Metrics, (2002), pp. 203-214. DOI: 10.1109/METRIC.2002.1011339

[8] Park, B.-J.; Oh, S.-K.; Pedrycz, W. The design of polynomial function-based neural network predictors for detection of software defects. // Information Sciences. 229, (2013), pp. 40-57. DOI: 10.1016/j.ins.2011.01.026

[9] Menzies, T.; Greenwald, J.; Frank, A. Data Mining Static Code Attributes to Learn Defect Predictors. // IEEE Transactions on Software Engineering. 33, 1(2007), pp. 2-13. DOI: 10.1109/TSE.2007.256941

[10] Khoshgoftaar, T. M.; Van Hulse, J.; Napolitano, A. Comparing Boosting and Bagging Techniques with Noisy and Imbalanced Data. // IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans. 41, 3 (2011), pp. 552-568. DOI: 10.1109/TSMCA.2010.2084081

[11] Tosun, A.; Bener, A.; Turhan, B.; Menzies, T. Practical considerations in deploying statistical methods for defect prediction: A case study within the Turkish telecommunications industry. // Information and Software Technology. 52, 11(2010), pp. 1242-1257. DOI: 10.1016/j.infsof.2010.06.006

[12] Kim, S.; Zhang, H.; Wu, R.; Gong, L. Dealing with noise in defect prediction. // Proceeding of the 33rd International Conference on Software Engineering, (2011), pp. 481-490. DOI: 10.1145/1985793.1985859

[13] Grbac, T. G.; Mausa, G.; Basic, B. D. Stability of Software Defect Prediction in Relation to Levels of Data Imbalance. // SQAMIA. (2013), pp. 1-10.

[14] Gao, K.; Khoshgoftaar, T. M.; Napolitano, A. Combining Feature Subset Selection and Data Sampling for Coping with Highly Imbalanced Software Data. // The 27th International Conference on Software Engineering and Knowledge Engineering (SEKE 2015). Wyndham Pittsburgh University Center, Pittsburgh, USA, July 6 - July 8, 2015. DOI: 10.18293/SEKE2015-182

[15] Jing, X. Y.; Ying, S.; Zhang, Z. W.; Wu, S. S.; Liu, J. Dictionary learning based software defect prediction. // Proceedings of the 36th International Conference on Software Engineering, (2014), pp. 414-423. DOI: 10.1145/2568225.2568320

[16] Ryu, D.; Jang, J. I.; Baik, J. A Hybrid Instance Selection Using Nearest-Neighbor for Cross-Project Defect Prediction. // Journal of Computer Science and Technology. 30, 5(2015), pp. 969-980. DOI: 10.1007/s11390-015-1575-5

[17] Xing, F.; Guo, P.; Lyu, M. R. A novel method for early software quality prediction based on support vector machine. // ISSRE 2005. 16th IEEE International Symposium on Software Engineering. 31, 4(2005), pp. 340-355.

[18] Li, M.; Zhou, Z. H. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. // IEEE Trans. Syst. Man Cybern., Part A, Syst. Hum. 37, 6(2007), pp. 1088-1098. DOI: 10.1109/TSMCA.2007.904745

[19] Zhou, Z.-H. When semi-supervised learning meets ensemble learning. // Proceedings of 8th International Workshop on Multiple Classifier Systems, Reykjavik, Iceland, (2009), pp. 529-538. DOI: 10.1007/978-3-642-02326-2_53

[20] Breiman, L. Bagging predictors. // Machine Learning. 24, 2(1996), pp. 123-140. DOI: 10.1007/BF00058655

[21] Yu, W. D. A software prevention approach in coding and root cause analysis. // Bell Labs Technical Journal. 3, 2(1998), pp. 3-21. DOI: 10.1002/bltj.2101

[22] Laitenberger, O.; Leszak, M.; Stoll, D.; El-Amam, K. Causal analysis of review success factors in an industrial setting. // Proceedings of the 6th IEEE International Symposium on Software Metrics, West Palm Beach, FL, 1999.

[23] Höppner, F.; Klawonn, F.; Kruse, R.; Runkler, T. Fuzzy Cluster Analysis. Kluwer, Amsterdam, Netherlands (1998).

[24] Langley, P.; Sage, S. Induction of Selective Bayesian Classifiers. // Proc. 10th Conf. on Uncertainty in Artificial Intelligence (UAI'94, Seattle, WA, USA), Morgan Kaufman, San Mateo, CA, USA. (1994), pp. 399-406. DOI: 10.1016/b978-1-55860-332-5.50055-9

[25] Gray, D.; Bowes, D.; Davey, N.; Sun, Y.; Christianson, B. Reflections on the NASA MDP data sets. // IET Software. 6, 6(2012), pp. 549-558. DOI: 10.1049/iet-sen.2011.0132
[26] Crawford, B.; Soto, R.; Johnson, F.; Misra, S.; Paredes, F.; Olguín, E. Software Project Scheduling using the Hyper-Cube Ant Colony Optimization algorithm. // Tehnicki vjesnik-Technical Gazette. 22, 5(2015), pp. 1171-1178, DOI: 10.17559/TV-20140519212813
[27] Popovic, T. Getting ISO 9001 certified for software development using scrum and open source tools: a case study. // Tehnicki Vjesnik-Technical Gazette. 22, 6(2015), pp. 1633-1640, DOI: 10.17559/TV-20140704180948
[28] Varajao, J; Dominguez, C.; Ribeiro, P.; Paiva, A. Critical success aspects in project management: similarities and differences between the construction and software industry. // Tehnicki Vjesnik-Technical Gazette. 21, 3(2014), pp. 583-589.

Authors' addresses

K. Punitha, Research scholar
Anna University,
Sardar Patel Road, Chennai 600025, Tamil Nadu, India
E-mail: researchpunitha@gmail.com
Mobile: +91 9443916174

Dr. B. Latha, Professor & Head
Sri Sai Ram Engineering College,
Sai Leo Nagar, West Tambaram,
Chennai 600 044, Tamil Nadu, India
E-mail: latha.it@sairam.edu.in
E-mail: sivasoorya2003@yahoo.com
Mobile: +91 984028435