Defect Prediction in Software Projects-Using Genetic Algorithm based Fuzzy C-Means Clustering and Random Forest Classifier

Pushpavathi T.P, Suma V, Ramaswamy V

Abstract— Software project success is based on prediction of defects at early stages of software development. Aaccurate prediction of defect prone modules in software development process enables effective discovery and identification of the defects. Such prediction approaches are valuable for the large scale systems, where verification experts need to focus their attention and resources to problem areas in the system under development. Identifying and locating defects in software projects to measure the project success is a difficult task. Especially, when project size grows, this task becomes expensive with sophisticated testing and evaluation mechanisms. Context: Software project failures are often caused by mistake made during the project, and such failures make a strong economic impact. It analysed software engineering projects such as product development project of a company, different software engineering projects and the projects were planned to be delivered to customers.Objective: This study reports an analysis for predicting defect prone modules using integrated approach of genetic algorithm based fuzzy c-means clustering with random forest algorithm.Method: This method was developed using Genetic Algorithm based Fuzzy C-means clustering with Random Forest classification applied on empirical data set and analysis was performed. Finally it is validated with five NASA public domain defect data sets. These data sets vary in size, but all typically contain a small number of defect samples in the learning set. For instance, in project PC1, only around 7% of the instances are defects.Results: The overall accuracy maximization is the goal, then learning from such data usually results in a biased classifier, i.e. the majority of samples would be classified into non-defect class.Conclusion: In order to achieve project success, it is better to understanding of the defects. It is reasonable to suggest that a notable portion of project defects are due to mistakes made during the project development in the case of accuracy based software engineering projects.

Index Terms— Data Classification, Defect Prediction, Fuzzy clustering, Genetic algorithm, Software Project Success, Soft Computing, Random forest.

1 INTRODUCTION

EARLY detection of defect-prone software components enables verification experts to concentrate their time and resources on the problem areas of the software system under development. The ability of software quality models to accurately identify critical components allows for the application to focused on verification activities ranging from manual inspection to automated formal analysis methods. Software quality models, hence, help to ensure the reliability of the delivered products. It has become an imperative to develop and apply good software quality models early in the software development life cycle, especially for large-scale development of projects [1].

High assurance and complex mission-critical software systems are heavily dependent on reliability of their underlying software applications. A software defect prediction is a proven technique in achieving high software reliability. Prediction of defect-prone modules provides one way to support software quality engineering through improved scheduling and project control.A fault is a defect, an error in source code that causes failures when executed. A defect prone software module is the one containing more number of expected defects. Accurate prediction of defect prone modules enables the verification and validation activities focused on the critical software components. Clustering is defined as the classification of data or object into different groups. It can also be referred to as partitioning of a data set into different subsets.

Software reliability engineering is one of the most important aspects of software quality [1]. Software metrics represent quantitative description of program attributes and the critical role they play in predicting the quality of the software has been emphasized by Perlis et al [3]. That is, there is a direct relationship between several complexity metrics and the number of changes attributed to defects later found in test and validation [4]. Many researchers have sought to develop a predictive relationship between complexity metrics and defects. Crawford et al [5] suggest that multiple variable models are necessary to find metrics that are important in addition to program size.
Quality of software is increasingly important and testing related issues are becoming crucial for software. Although there is diversity in the definition of software quality, it is widely accepted that a project with many defects lacks quality. Methodologies and techniques for predicting the testing effort, moni-

- *Pushpavathi T.P, JAIN University, Bangalore, India. E-mail: acepushpa@yahoo.co.in*
- *Suma V, Dayanandsagar College of Engineering, Bangalore, India. E-mail: sumavdsce@gmail.com*
- *Ramaswamy V, SASTRA University, Srinivasa Ramanujan Centre,Kumbakonam,Tamilnadu,India, E-mail: researchwork04@gmail.com*

toring process costs and measuring the results can help in increasing efficiency of software testing. Being able to measure the defect-proneness of software can be a key step towards steering the software testing and improving the effectiveness of the whole process [10].

2 Related Review

Mining software engineering data has emerged as one of the successful research directions since few decades. Software repositories contain a wealth of valuable information about software projects. Using the information stored in these repositories which act as predictive pattern for estimating, planning and controlling the future project, the software developing team can now depend less on their intuition and operate more with realistic data. Data mining has emerged as one of the assuring techniques through which valuable information can be mined from the population of empirical projects.
Significant work has been done in the field of defect detection. The complete survey of defect prediction studies till 2011 is provided by C. Catal in [17]. Highlights of selected papers have been discussed in this section, including papers published post 2008. There are various categories of methods to predict defect classes such as machine learning methods, statistical methods, etc. It has been observed that much of the previous work used traditional statistical methods ([18], [20], [21], [16]) to bring out the results but very few studies have used machine learning methods.

Recently, the trend is shifting from traditional statistical methods to modern machine learning methods. The most common statistical methods used are univariate and multivariate logistic regression. A few key points of the papers using statistical methods are discussed by N. Ohlsson et al, in [18] has worked on improving the techniques used by Khosgoftaar [19] (i.e., Principal Component Analysis and Discriminant Analysis). Authors in [18] has discussed problems that were faced while using these methods and thus suggested the remedies to those problems. Another approach to identify defect classes early in the development cycle is to construct prediction models. Authors in [20] have constructed a model to predict defect classes using the metrics that can be collected during the design stage. This model has used only object oriented design metrics.

Tang et al. [21] conducted an empirical study on three industrial real time systems and validated the CK object oriented metric suite [5]. They found that only WMC and RFC are strong predictors of defect classes. They have also proposed a new set of metrics, which are useful indicators of object oriented defect prone classes. It has been seen that most of the empirical studies have ignored the confounding effect of class size while validating the metrics size [16]. El. Emam et al showed a strong size confounding effect in [16] and [23]. Thus they concluded the metrics that were strongly associated with defect proneness were not associated with the size of defect proneness.

Another empirical investigation by M.Cartwright et al. [11] conducted on a real time C++ system discussed the use of object oriented constructs such as inheritance and polymorphism. M. Cartwright et al. [11] have found high defect densities in classes that participated in inheritance as compared to classes. Briand et al. [1] have empirically investigated 49 metrics (28 coupling measures, 10 cohesion measures, and 11 inheritance measures) for predicting defect classes. There were 8 systems being studied (consisting of 180 classes in all), each of which was a medium sized management information system. They used univariate and multivariate analysis to find the individual and the combined effect of object oriented metrics and defect proneness. They did not examine the LCOM metric and found that all the other metrics are strong predictors of defect proneness except for NOC.

Briand et al. [24] has also validated the same 49 metrics. The system used for this study was the multi-agent development system, which consists of three classes. They found NOC metric to be insignificant, while DIT was found to be significant in an inverse manner. WMC, RFC, and CBO were found to be strongly significant. Yu et al. [25] empirically tested 8 metrics in a case study in which the client side of a large network service management system was studied. The system is written in Java and consists of 123 classes. The validation was carried out using regression analysis and discriminate analysis. They found that all the metrics were significant predictors of defect proneness except DIT, which was found to be insignificant.

Recently, researchers have also started using various machine learning techniques to predict the model. Gyimothy et al. [12] calculated CK [5] metrics from an open source web and email suite called Mozilla. To validate the metrics, regression and machine learning methods (decision tree and artificial neural networks) were used. The results concluded NOC to be insignificant, whereas all the other metrics were found to be strongly significant. Zhou et al. [26] have used logistic regression and machine learning methods to show how object oriented metrics and defect proneness are related when defect severity is taken into account. The results were calculated using the CK metrics suite and were based on the public domain NASA dataset. WMC, CBO, and SLOC were found to be strong predictors across all severity levels. Prior to this study, no previous work had assessed severity of defects. The paper by S. Kanmani et al. [13] has introduced two neural network based prediction models. The results were compared with two statistical methods and it was concluded that neural networks performed better as compared to statistical methods.

Fenton et al. [27] introduced the use of Bayesian belief networks (BBN) for the prediction of defect classes. G.J. Pai et al. [2] also built a Bayesian belief network (BN) and showed that the results gave comparable performance with the existing techniques. I. Gondra [14] has performed a comparison between the artificial neural network (ANN) and the support vector machine (SVM) by applying them to the problem of classifying classes as defect or non-defect. Another goal of this paper was to use the sensitivity analysis to select the metrics that are more likely to indicate the errors. After the work of Zhou et al. [26], the severity of defects was taken into account by Shatnawi et al. [28] and Singh et al. [4].
Shatnawi et al. used the subset of CK metrics in [5] and Lorenz & Kidd metrics in [9] to validate the results. The data was collected from three releases of the Eclipse project. They concluded that the accuracy of prediction decreases from releases and

alternative methods are needed to get more accurate prediction. The metrics, which were found to be very good predictors across all versions and across all severity levels, were WMC, RFC, and CBO. Singh et al. [4] used the public domain NASA dataset to determine the effect of metrics on defect proneness at different severity levels of defects.

Machine learning methods (decision tree and artificial neural network networks) and statistical method (logistic regression) were used. The predicted model showed lower accuracy at a higher severity level as compared to medium and low severities. It was also observed that performance of machine learning methods was better than statistical methods. Amongst all the CK metrics used CBO, WMC, RFC, and SLOC showed the best results across all the severity levels of defects. Malhotra et al. [29] have used LR and 7 machine learning techniques (i.e., artificial neural networks, random forest, bagging, boosting techniques [AB, LB], naive bayes, and K-star) to validate the metrics. The predicted model using LB technique showed the best result and the model predicted using LR showed low accuracy. From the review we have made following observations.

Observations:

- The CK metric suite is most widely used even though number of metrics available in literature. It has been seen that most of the studies have also defined their own metric suite and they have used them for carrying out the analysis [5][15][17].
- Diverse categories of methods available to predict the most accurate model such as machine learning methods, statistical methods etc., the trend is shifting from the traditional statistical methods to the machine learning methods[14][25][29].
- It has been observed that machine learning is widely used in new bodies of research to predict defect prone classes. Results of various studies also prove that better results are obtained with machine learning as compared to statistical methods [17][18][25].
- Many researchers have used different types of datasets, which are mostly public datasets, commercial datasets, open source or students/university datasets from the PROMISE and NASA repositories. We have observed that the empirical datasets, which have been rarely used in the studies [2][3][4][11][12][15][21][22][26].

3 Proposed Classification Procedure

The methodology consists of the following steps

3.1 Find the structural code and requirement code attributes

The first step is to find the structural code and requirement code attributes of software systems i.e. software metrics. The real time defect data sets are taken from the NASA's MDP (Metric Data Program) data repository, [online Available: http://mdp.ivv.nasa.gov.in] named as PC1 dataset which is collected from a flight software from an earth orbiting satellite coded in C programming language, containing 1107 modules and only 109 have their requirements specified. PC1 has 320 requirements available and all of them are associated with program modules. All these data sets varied in the percentage of defect modules, with the PC1 dataset containing the least number of defect modules.

The software metric data gives us the values for specific attributes to measure a specific module/function or the whole software. When combined with the weighted error/defect data, this data set becomes the input for a machine learning system. A learning system is defined as a system that is said to learn from experience with respect to some class of tasks and performance measure, such that its performance at these tasks improves with experience. To design a learning system, the data set in this work is divided into two parts: the training data set and the testing data set. Some predictor functions are defined and trained with respect to Multi-Layer Preceptor and Decision Tree algorithms and the results are evaluated with the testing data set.

3.2 Select the suitable metric values as representation

The modelling techniques applied to cover the main classification paradigms, including principal component analysis, discriminate analysis, logistic regression and logical quality will be improved as more defects will be detected. Predicting defects early in the software life cycle can be used to improve software process control and achieve high software reliability. Apt predictions of defects in software modules can be used to direct cost-effective quality enhancement effort modules that are likely to have a high number of defects. Prediction models based on software metrics, can estimate number of defects in software modules. Software metrics are attributes of the software system and may include process, product and execution metrics. Various attributes, which determine the quality of the software, include maintainability, defect density, defect proneness, normalized rework and reusability etc.

The Suitable metric values used are defect and without defect attributes, we set these values in database create in MATLAB R2010 A as 0 and 1.That 0 means data with defect and 1 for data without defect. The metrics in these datasets (NASA MDP dataset) describe projects which vary in size and complexity, programming languages, development processes, etc. When reporting a defect prediction modelling experiment, it is important to describe the characteristics of the datasets. Each data set contains twenty-one software metrics, which describe product's size, complexity and some structural properties. We use only defect and defect free attributes to classify the selected NASA MDP PC1 dataset. Also the product metrics and product module metrics available in dataset which can also be used are,

The product requirement metrics are as follows:

- Module
- Action
- Conditional
- Continuance
- Imperative
- Option
- Risk_Level
- Source
- Weak_Phrase

The product module metrics are as follows:

- Module
- Loc_Blank

- Branch_Count
- Call_Pairs
- LOC_Code_and_Comment
- LOC_Comments
- Condition_Count
- Cyclomatic_complexity
- Cyclomatic_Density
- Decision_Count
- Edge_Count
- Essential_Complexity
- Essential_Density
- LOC_Executable
- Parameter_Count
- Global_Data_Complexity
- Global_Data_Density
- Halstead_Content
- Halstead_Difficulty
- Halstead_Effort
- Halstead_Error_EST
- Halstead_Length
- Halstead_Prog_Time
- Halstead_Volume
- Normalized_Cyclomatic_Complexity
- Num_Operands
- Num_Operators
- Num_Unique_Operands
- Num_Unique_Operators
- Number_Of_Lines
- Pathological_Complexity
- LOC_Total

3.3 Empirical data set attributes

With real-time systems defect prediction becoming more complex and unpredictable, partly due to increasingly sophisticated requirements, traditional software development techniques might face difficulties in satisfying these requirements. Future real-time software systems may need to dynamically adapt themselves based on the run-time mission-specific requirements and operating conditions. This involves dynamic code synthesis that generates modules to provide the functionality required to perform the desired operations in real-time. However, this necessitates the need to develop a real-time assessment technique that classifies these dynamically generated systems as being defect/defect-free. A variety of software defect predictions techniques have been proposed, but none has proven to be consistently accurate. These techniques include statistical method, machine learning methods, parametric models and mixed algorithms [12]

An empirical investigation related to this research is carried on several product based software industries of varying production capabilities. In order to overcome the complexities involved in universe of projects developed in those industries, this research aims to establish hypothesis for the purpose of sources of data collection.The basic hypothesis of software quality prediction is that a module currently under development is defect prone if a module with the similar product or process metrics in an earlier project (or release) developed in the same environment was defect prone. Therefore, the information available early within the current project or from the previous project can be used in making predictions. This methodology is very useful for the large-scale projects or projects with multiple releases [15][21].The following data set attributes are identified from our previous work[30].

- Total project time in hours,
- Inspection time scheduled,
- Number of inspectors involved
- Experience level of inspectors (years),
- Defect count estimation,
- Number of defects detected,
- Defects actually captured,
- Number of defects not captured,
- Defects due to bad fixes,
- Testing time scheduled.
- Testing time scheduled
- Number of testers
- Experience level of testers (years)
- Defect count estimation
- Number of defects detected

3.4 Objective Function Based Fuzzy Clustering

Objective function based fuzzy clustering algorithms such as the fuzzy c-means (FCM) algorithm have been used extensively for different tasks such as pattern recognition, data mining, image processing and fuzzy modelling. Applications have been reported from different fields such as financial engineering, direct marketing and systems modelling. Fuzzy clustering algorithms partition the data set into overlapping groups such that the clusters describe an underlying structure within the data. In order to obtain a good performance from a fuzzy clustering algorithm, a number of issues must be considered. These concern the shape and the volume of the clusters, the initialization of the clustering algorithm, the distribution of the data patterns and the number of clusters in the data.

In general, cluster analysis refers to a broad spectrum of methods which try to subdivide a data set X into c subsets (clusters) which are pairwise disjoint, all nonempty, and reproduce X. via union. The clusters then are termed a hard (i.e., non-fuzzy) c-partition of X. Let $Y = \{y_1, y_2, \cdots y_N\}$ be a sample of N observations in R^n (n-dimensional Euclidean space); y_k is the k-th feature vector; y_{kj} the j-th feature of y_k. If c is an integer, $2 \leq c < n$, a conventional (or "hard") c-partition of Y is a c-tuple $(Y_1, Y_2 \ldots\ldots Y_c)$ of subsets of Y that satisfies three conditions:

$$Y_1 \neq \emptyset, 1 \leq i \leq c \quad (1)$$
$$y_i \cap Y_j = \emptyset; i \neq j \quad (2)$$
$$\cup_{i=1}^{c} Y_i = Y \quad (3)$$

In these equations, $\emptyset$ stands for the empty set, and $(\cup, \cap)$ are respectively, intersection, and union. In the context discussed later, the sets $\{Y_1\}$ are termed "clusters in $'Y'$. Clusters analysis (or simply clustering) in Y refers to the identification of a distinguished c-partition $\{Y_i\}$ of Y whose subsets contain points which have high intracluster resemblance; and, simultaneously, low intercluster similarity. The mathematical criterion of resemblance used to define an "optimal" c-partion is termed a cluster criterion. One hopes that the substructure of Y represented by $\{Y\}_i$ suggests a useful division or relationship between the population variables of the real physical process from whence Y was drawn. One of the first questions one might ask is whether Y was drawn. One of the first questions one might ask is whether Y contains any clusters at all. In many geological analyses, a value for c is known a priori on physical grounds. If c is unknown, then determination of an optimal c becomes an important issue. This question is sometimes termed the "cluster validity" problem.

3.5 Classification of the software components into defect/defect free

The Flow for the classification approach is shown in figure 1 below.

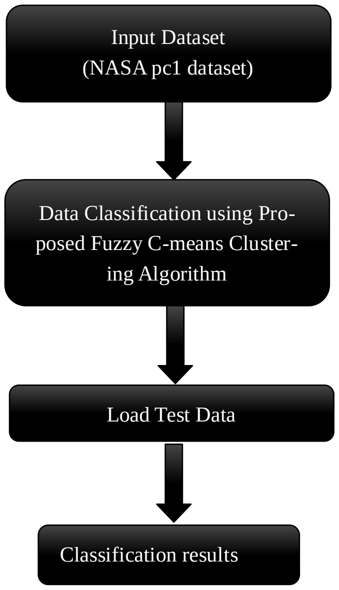

Input Dataset
(NASA pc1 dataset)
Data Classification using Proposed Fuzzy C-means Clustering Algorithm
Load Test Data
Classification results

Fig 1. Block diagram for the proposed classification process system.

3.6 Genetic Algorithm Optimization of Fuzzy C-means Clustering

Genetic Algorithm (GA) is a part of soft computing paradigm known as evolutionary computation. They attempt to arrive at optimal solutions through a process similar to biological evolution. This involves following the principles of survival of the fittest, and crossbreeding and mutation to generate better solutions from a pool of existing solutions.

Genetic algorithms have been found to be capable of finding solutions for a wide variety of problems for which no acceptable algorithmic solutions exist. The GA methodology is particularly suited for optimization, a problem solving technique in which one or more good solutions are searched for in a solution space consisting of a large number of possible solutions. GA reduce the search space by continually evaluating the current generation of candidate solutions, discarding the ones ranked as poor, and producing a new generation through crossbreeding and mutating those ranked as good. The ranking of candidate solutions is done using some pre-determined measure of goodness or fitness.

A genetic algorithm is a probabilistic search technique that computationally simulates the process of biological evolution. It mimics evolution in nature by repeatedly altering a population of candidate solutions until an optimal solution is found.

The GA evolutionary cycle starts with a randomly selected initial population. The changes to the population occur through the processes of selection based on fitness, and alteration using crossover and mutation. The application of selection and alteration leads to a population with a higher proportion of better solutions. The evolutionary cycle continues until an acceptable solution is found in the current generation of population, or control parameter such as the number of generations is exceeded.

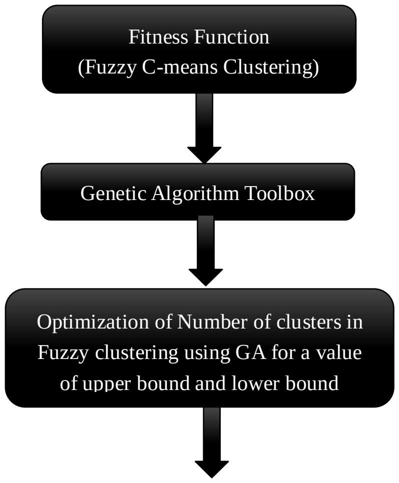

Fitness Function
(Fuzzy C-means Clustering)
Genetic Algorithm Toolbox
Optimization of Number of clusters in Fuzzy clustering using GA for a value of upper bound and lower bound

Figure 2. Block Diagram for the optimization process used in fuzzy c-means clustering.

The steps in the typical genetic algorithm for finding a solution to a problem are listed below:

1. Create an initial solution population of a certain size randomly
2. Evaluate each solution in the current generation and assign it a fitness value.
3. Select “good” solutions based on fitness value and discard the rest.
4. If acceptable solution(s) found in the current generation or maximum number of generations is exceeded then stops.
5. Alter the solution population using crossover and mutation to create a new generation of solutions.
6. Go to step 2.

3.7 Random Forest Classifier

A classification tree is a top-down tree-structured classifier. It is built through a process known as recursive partitioning whereby the measurement space is successively split into subsets, each of which is equivalent to a terminal node in the tree. Starting from the root node (i.e., the top node of the tree) which contains the entire sample, all the candidate splits are evaluated independently, the most appropriate one is selected. The “appropriateness” of a split can be evaluated by different measures. The most popular one is based on impurity functions. Impurity measures are the quantification of how well the classes are being separated.

In general, the value of an impurity measure is the largest when data are split evenly for attribute values and zero when all the data points belonging to a single class. There are various impurity measures used in the literature. The most commonly used are: Entropy-based measure. Purity (or homogeneity) of class labels is measured before and after the split. The split which produces the most discrimination between classes is the most appropriate one. The classification tree is grown to a point where the number of instances in the terminal node is small or the class membership of instances in the node is pure enough. The label of the class which dominates the instances in the terminal node is assigned to this node. The fully grown tree may overfits the training data and may need to be cut back by using criteria which balance the classification performance of the tree and the tree's complexity.

In a tree model, an edge from a parent node to one of its children nodes indicates a rule. This child node is reachable only when the rule is satisfied. A path from the root node to a terminal node of the tree, which consists of at least one edge, specifies a classification rule. Such classification rule is the combination of the functions associated with all of the edges on the path. The decision associated with each classification rule is stated by the label attached to the terminal node. For a new data instance characterized by the input vector x, we would expose it to the root node of the tree and then follow along the path in which x satisfies all the rules. Obtained class label at the terminal node represents the classification decision.

Trees represent an efficient paradigm in generating understandable knowledge structures. But, experience shows that the lack of accuracy is the reason that prevents classification trees from being the ideal tool for predictive learning. One way to improve accuracy of classification trees is to utilize ensemble learning. Ensemble methods learn a large number of models instead of a single model and combine the predictions of all the models with a learning algorithm.

Bagging and random forests are ensemble methods. They construct a collection of base models and simply average their predictions. Bagging generally works for different classifiers, including trees or neural networks, while random forests use only trees as base models. Random forests algorithm stems from bagging and can be considered as a special case of bagging.

The idea behind bagging is to take a bootstrap replicate (a sample collected with replacement) from the original training set and obtain a model f(x). By bootstrapping, K different versions of the learning set can be generated and K models f1 ,..., fk are obtained. Each tree predicts class membership of any test case. For overall classification, the predicated class is determined by plurality voting among the classes C, i.e., the class label most frequently predicted by the K models is selected.

Like bagging, a random forest consists of a collection of K tree classifiers h1(x), h2(x),..., hk(x). Each classifier is built upon a bootstrap replica of the training set and votes for one of the classes in C. A test instance is classified by the label of the wining class. With bootstrap sampling, approximately 36.8% of the training instances are not used in growing each tree (due to sampling with replacement). These data instances are called out-of-bag (OOB) cases. Random forest algorithm uses OOB cases to estimate the classification error and evaluate the performance of the forest.

In many applications including the prediction of defect prone modules in software engineering, the prediction models are faced with class imbalance problem. It occurs when the classes in C have a dramatically different numbers of representatives in the training dataset and/or very different statistical distributions. Learning from imbalanced data can cause the classifier to be biased. Such bias is the result of one class being heavily over-represented in the training data compared to the other classes. Classes containing relatively few cases can be largely ignored by the learning algorithms because the cost of performing well on the large class outweighs the cost of doing poorly on the much smaller classes, provided that the algorithm aims at maximizing overall accuracy.

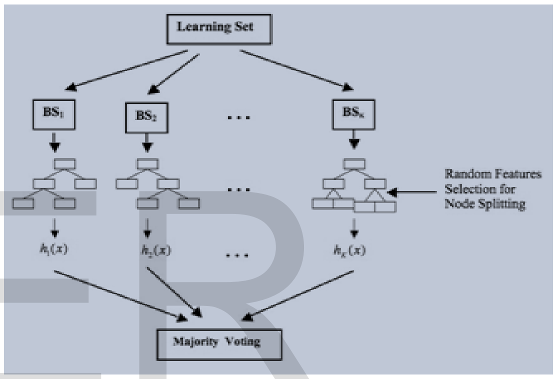


Fig. 3. Construction of a Random Forest

For instance, a binary classification problem such as the identification of defect prone modules may be represented by 1,000 cases, 950 of which are negative cases (majority class, not defect prone modules) and 50 are positive cases (minority class, defect prone modules). Even if the model classified all the cases as negative and misclassified all the positive cases, the overall accuracy could reach 95%, a great result for any machine learning classification algorithm. In many situations, the minority class is the subject of our major interest. In practice, we want to achieve a lower minority classification error even at the cost of a higher majority class error rate. For example, if the goal of identifying defect prone modules early in software development is exposing them to a more rigorous set of verification and validation activities, the imperative is to identify as many potentially defect modules as possible. If we happen to misclassify non-defect modules as defect, the verification process will increase its cost as some modules are unnecessarily analysed. But, the consequence of misclassifying defect software as non-defect may be a system failure, a highly undesirable outcome.

4 Implementing the Model

The proposed methodology is implemented in the MATLAB 7.4.Evaluate both the approaches for the modelling of the reusability data. Implement the model and test the performance of the model using following criteria:

a. Perform the training of the dataset.
b. After training, test it on the basis of error values MAE and RMSE, and efficiency parameters like accuracy and net reliability in percentage.

- **Mean Absolute Error**

Mean absolute error, MAE is the average of the difference between predicted and actual value in all test cases; it is the average prediction error. The formula for calculating MAE is given in equation shown below:

$$\frac{|a_1 - c_1| + |a_2 - c_2| + \cdots + |a_n - c_n|}{n}$$

- **Root Mean Squared Error:**

RMSE is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the thing being modelled or estimated. It is just the square root of the mean square error as shown in equation given below:

$$\sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \cdots + (a_n - c_n)^2}{n}}$$

After calculating MAE and RMSE values for each and every algorithm, the comparisons are made on the basis of the least value of MAE and RMSE error values. The mean absolute error is chosen as the standard error. The technique having lower value of mean absolute error is chosen as the best defect prediction technique.

In general, reliability is the ability of a person or system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances. In the fields of science, engineering, industry, and statistics, the accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity's actual (true) value. The precision of a measurement system, also called reproducibility or repeatability, is the degree to which repeated measurements under unchanged conditions show the same results.

5 Results and Dicussion

In this study, training and testing methodology is being used, wherein a project is chosen for training the system. The NASA MDP dataset named PC1 is used in this.Then the Fuzzy C-means clustering based classification approach is applied on the same project and the finally calculated values are then used to classify the modules of project as defect prone or defect free. The classification is based on values of accuracy, MAE and RMSE.Genetic algorithm consider fuzzy c-means clustering as a fitness function which need to minimize, and give a range (upper bound and lower bound) to genetic algorithm to select a number of clusters for the problem value. In order to get optimum classification (Upper bound is considered to be 8 and lower bound is 2 in this case)

The input data having 1107 modules as we discussed earlier in third section, but in order to reduce complexity in calculations, here it uses only 100 modules for training purpose. The original dataset contains 22 columns, column 1 to 21 shows the data entry values and the last column show the attribute value of dataset, i.e., the data is with defect or without defect. If it plot the data set with 100 modules and all considered columns (22 columns), the figure will look as shown in figure 4.

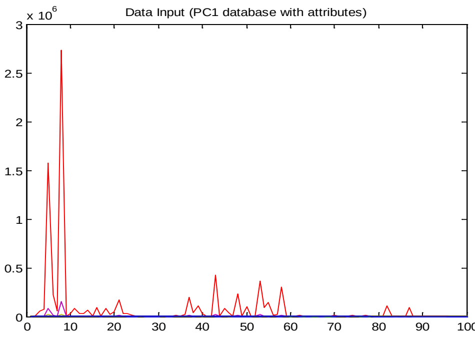


Fig 4. Input NASA pc1 dataset with attributes (defect and without defect)

In original dataset Class Distribution: the class value (defects) is discrete

% false: 77 = 6.94%

% true: 1032 = 93.05%

In training dataset we taken, Class Distribution: the class value (defects) is discrete

% data with positive attribute: 23 = 23%

% data with negative attribute: 77 = 77%

This is a PROMISE Software Engineering Repository data set made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering. In training dataset input, 77 modules have data with negative attributes, if we plot certain defect data, it as shown in figure 5. The X axis shows the data modules with defect, and Y axis shows data value stores in 21 columns of input dataset.

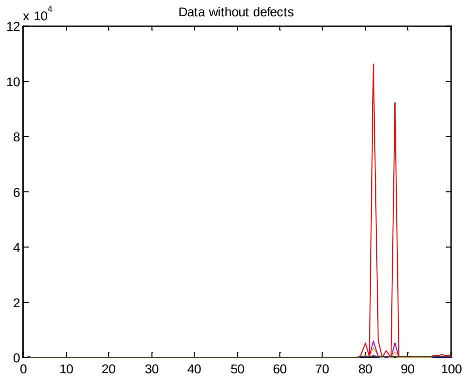


Fig 5. Input NASA pc1 dataset without defect attributes when separating defect attributes from input data

In training dataset input, 77 modules have data with negative attributes, if we plot certain defect data, it as shown in figure 5. The X axis shows the data modules with defect, and Y axis shows data value stores in 21 columns of input dataset.

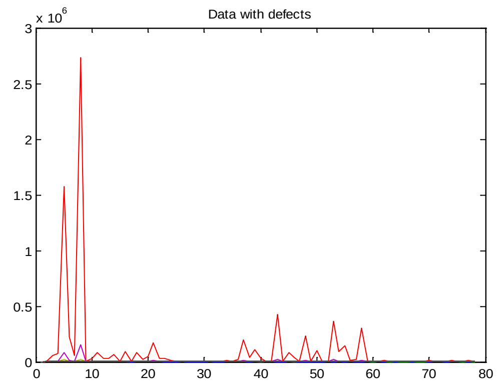


Fig. 6. Input NASA pc1 dataset with defect attributes when separating without defect attributes from input data

In training dataset input, 23 modules has data with positive attributes after the 77 rows of data with negative attributes, if we plot certain defect data, it look like as shown in figure 6. The X axis shows the data modules without defect, and Y axis shows data value stores in 21 columns of input dataset.

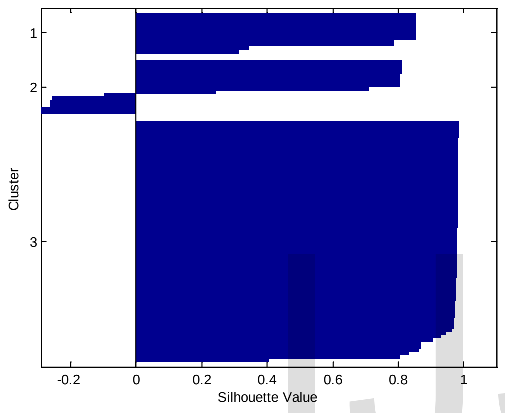


Fig. 7. Cluster separation in silhouette plot for our clustering approach

To get an idea of how well-separated the resulting clusters are, figure 7 infers a silhouette plot using the cluster indices output from fuzzy c-means clustering. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighbouring clusters. This measure ranges from +1, indicating points that are very distant from neighbouring clusters, through 0, indicating points that are not distinctly in one cluster or another, to -1, indicating points that are probably assigned to the wrong cluster.From the silhouette plot, it shows that most points in the third cluster have a large silhouette value, greater than 0.6, indicating that the cluster is somewhat separated from neighbouring clusters. However, the first cluster contains many points with low silhouette values, and the second contains a few points with negative values, indicating that those two clusters are not well separated.Figure 8 shows it can increase the number of clusters to see if this classification can find further grouping structure in the data. Now the optional 'display' parameter to print out information about each iteration in the clustering algorithm.The normal cluster plot does not include the cluster centroids, because a centroid with respect to the cosine distance corresponds to a half-line from the origin in the space of the raw data. However, it can make a parallel coordinate plot of the normalized data points to visualize the differences between cluster centroids as shown in figure 8.

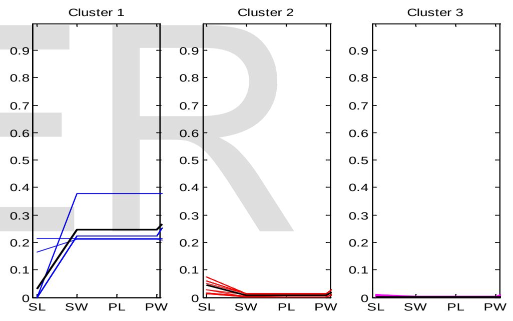
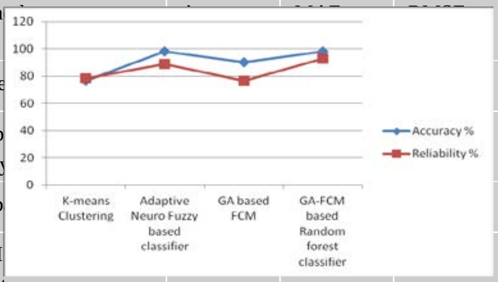


Fig.8. Grouping of different data values in clusters
(3 clusters are considered in this approach)

It's clear from this plot that projects from each of the three clusters have distinctly different relative parameters. The first cluster has parameter that are strictly smaller than the other. The second two clusters' parameters overlap in size, however, those from the third cluster overlap more than the second. it can also observe that the second and third clusters include few projects which are very similar to each other.

Meth				Reliability %
K-me				56.154
Adap Fuzzy				59.75
GA b				60.75
FCM forest				79.852



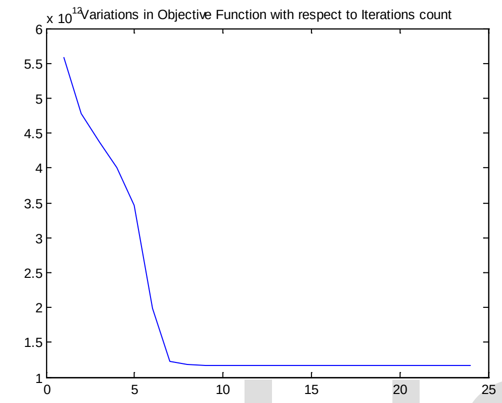


Fig. 9. Variation in objective function with respect to iterations count for random forest classifier

Parent is the vector of parent pointers, with 0 for a root. Post is an optional post-order permutation on the tree nodes. If can leave out post, tree-layout computes it. X and Y are vectors of coordinates in the unit square at which to lay out the nodes of the tree to make it in figure 9.

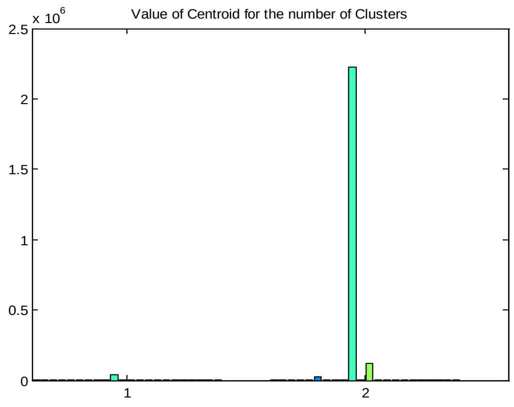


Fig.10. Value of centroid for the number of clusters in genetic algorithm based fuzzy c-means classification

Figure 10 infers genetic algorithm based fuzzy clustering treats observations in this data as objects having locations and distances from each other. It partitions the objects into number of mutually exclusive clusters, such that objects within each cluster are as close to each other as possible, and as far from objects in other clusters as possible. Each cluster is characterized by its centroid or center point. Of course, the distances used in clustering often do not represent spatial distances.The accuracy of a measurement system is the degree of closeness of measurements of a actual quantity (true) value. In our approach, the achieved accuracy is approximately 91.8469 %, if we draw a semilogy plot with this accuracy value. Semilogy plots data with logarithmic scale for the Y-axis. Semilogy of data creates a plot using a base 10 logarithmic scale for the Y-axis and a linear scale for the X-axis. It plots the columns of Y versus their index if Y contains real numbers. "Semilogy (data)" is equivalent to "semilogy (real (data), imag (data))" if Y contains complex numbers. Semilogy ignores the imaginary component in all other uses of this function.

TABLE 1: DEFECT PREDICTION ACCURACY FOR EMPIRICAL DATA SET WITH HYBRID METHODS (WITHOUT GA_FCM_MODIFIED RF)

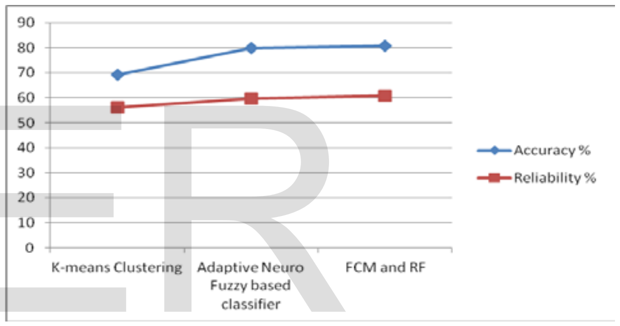


Fig.11. Accuracy and reliability of Hybrid models

TABLE 2: DEFECT PREDICTION ACCURACY FOR EMPIRICAL DATA SET WITH GA_FCM_MODIFIED RF

Method	Accuracy %	MAE	RMSE	Reliability %
K-means Clustering	71.6561	1.0352	0.1200	75.214
Adaptive Neuro Fuzzy based classifier	93.125	0.1200	0.01636	77.55
GA based FCM	90.1253	0.200	0.050	79.855
GA-FCM based Random forest classifier	98.237	0.0300	0.00098	92.625

Fig. 12. Accuracy and reliability of GA_FCM_RF model

Table 1,table 2 and table 3 shows the accuracy and reliability of hybrid models,with integrated (GA_FCM_RF)model and validation results correspondingly.From these results it can detect integrated methods gives the better accuracy and reliability for defect predictions as compared to single and hybrid classifier methods.

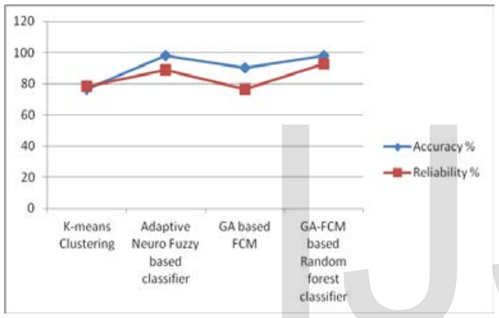


TABLE 3: DEFECT PREDICTION ACCURACY VALIDATION WITH NASA PC1 DATA SET

Technique	Accuracy %	MAE	RMSE	Reliability %
K-means Clustering	76.561	0.10352	0.11200	78.354
Adaptive Neuro Fuzzy based classifier	98.125	0.0210	0.01563	88.751
GA based FCM	90.1253	0.0200	0.04950	76.753
GA-FCM based Random forest classifier	98.237	0.0300	0.00092	92.625

Fig. 13. Accuracy and reliability validation with NASA PC1 data set

6 CONCLUSION

Regarding the overall performance of random forests, two important observations emerge from these experiments. One is that the random forest algorithm is always one of the best classifiers if the user-specific voting thresholds approximate the proportion of defect-prone modules in the project's training set. The second observation relates the classification performance of two variants of the random forest algorithm. Balanced random forests provide a moderate performance increase over the traditional random forest algorithm. Since the improvement is moderate, software quality engineers have to decide whether it is worth additional effort in the design of experiments. Unlike traditional random forests and all other classification algorithms reported in this paper, balanced random forests require some extra effort as their implementation is not immediately available from off-the-shelf software tools. However, if the software project requires that as many as possible modules to be inspected due to a dire consequence of software failures, this extra effort may be warranted.

Prediction of defect-prone modules provides one way to support software quality engineering through improved scheduling and project control. So, there is a need to develop a real-time assessment technique that classifies these dynamically generated systems as being defect/defect-free. A variety of software defect predictions techniques have been proposed, but none has proven to be consistently accurate
Defect prediction concerns the resource allocation problem: Having an accurate estimate of the distribution of bugs across components helps project managers to optimize the available resources by focusing on the problematic system parts. Different approaches have been proposed to predict future defects in software systems, which vary in the data sources they use, in the systems they were validated on, and in the evaluation technique employed; no baseline to compare such approaches exists.

Here it presented a defect prediction model using fuzzy c-means clustering based approach to estimate software reliability and software quality. In order to achieve software quality defects must be known prior to development so that more emphasis can be made on defect prone areas. We used the training and testing methodology. By analyzing the results it is clear that our fuzzy c-means clustering based approach gives the better result with an accuracy of more than 90 %.

REFERENCES

[1] L. Briand, W. Daly and J. Wust, "Exploring the relationships between design measures and software quality," Journal of Systems and Software, Vol.51, No.3, 2000, pp.245-273.

[2] G. Pai, "Empirical analysis of software defect content and defect proneness using Bayesian methods," IEEE Transactions on Software Eng., Vol.33, No.10, 2007, pp.675-686.

[3] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on defect proneness: A replicated case study," Software Process: Improvement and Practice, Vol.16, No.1, 2009, pp.39-62.

[4] Y. Singh, A. Kaur, and R. Malhotra, "Empirical vlidation of object-oriented metrics for predicting defect proneness models," Software Quality Journal, Vol.18, No.1, 2010, pp.3-35.

[5] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design," IEEE Trans. Soft Ware Eng., Vol.20, No.6, 1994, pp.476-493.

[6] L.Briand, P. Devanbu, W. Melo, "An investigation into coupling Measures for C++," In Proceedings of the 19th International Conference on Software Engineering.

[7] J. Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," IEEE Trans. Software Eng., Vol.28, No.1, 2002, pp.4-17.

[8] F. Brito e Abreu and W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality," Proceedings Third Int'l Software Metrics Symposium, 1996, pp.90-99.

[9] M.Lorenz and J. Kidd, "Object-Oriented Software Metrics," Prentice-Hall, 1994.

[10] W. Li and W. Henry, "Object-Oriented Metrics that Predict Maintainability," In Journal of Software and Systems, 1993, Vol.23, pp.111-122.

[11] M.Cartwright and M. Shepperd, "An empirical investigation of an object-oriented software system," IEEE Transactions on Software Engineering, Vol.26, No.8, 1999, pp.786-796.

[12] T.Gyimothy, R. Ferenc, and I.Siket, "Empirical validation of object-oriented metrics on open source software for defect prediction," IEEE Transactions on Software Engineering, Vol.31, No.10, 2005, pp.897-910.

[13] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, P. Thambidurai, "Object-oriented software prediction using neural networks," Information and Software Technology, Vol.49, 2007, pp.482-492.

[14] I. Gondra, "Applying machine learning to software defect-proneness prediction," The Journal of Systems and Software," Vol.81, 2008, pp.186-195.

[15] Promise. http://promisedata.org/repository/.

[16] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, "A validation of object-oriented metrics," NRC Technical report ERB-1063, 1999.

[17] C. Catal and B. Diri, "A systematic review of software defect prediction studies," Expert Systems with Applications Vol.36, 2009, pp 7346-7354.

[18] N. Ohlsson, M. Zhao and M. Helander, M, "Application of multivariate analysis for software defect prediction," Software Quality Journal, Vol.7, 1998, pp.51-66.

[19] T.M. Khoshgoftaar, E.B. Allen, K.S. Kalaichelvan and N. Goel, "Early quality prediction: a case study in telecommunications," IEEE Software, Vol.13, No.1, 1996, pp.65-71.

[20] K.E. Emam and W. Melo, "The Prediction of Defect Classes Using Object-Oriented Design Metrics," Technical report: NRC 43609, 1999.

[21] M.H. Tang, M.H. Kao, and M.H. Chen, "An empirical study on object-oriented metrics," In Proceedings of Metrics, 242-249.

[22] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, "A comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study," International Software Engineering Research Network, technical report ISERN-98-29, 1998.

[23] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, "The confounding effect of class size on the validity of object oriented metrics," IEEE Transactions on Software Engineering, Vol.27, No.7, 2001, pp.630-650.

[24] L. Briand, J. Wu¨st, J and H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object Oriented Designs," Empirical Software Engineering. International Journal (Toronto, Ont.), Vol.6, No.1, 2001, pp.11-58.

[25] P. Yu, T. Systa, and H. Muller, "Predicting defect-proneness using OO metrics: An industrial case study," In Proceedings of Sixth European Conference on Software Maintenance and Reengineering,Budapest, Hungary, 2002, pp.99-107.

[26] Y. Zhou, and H. Leung, H, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Defects," IEEE Transactions on Software Engineering, Vol.32, No.10, 2006, pp.771-789.

[27] N. Fenton and N. Ohlsson, "Quantitative analysis of defects and failures in a complex software system," IEEE Transactions on Software Engineering, Vol.26, No.8, 2000, pp.797-814.

[28] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post release software evolution process," The Journal of Systems and Software, Vol.81, 2008, pp.1868-1882.

[29] R. Malhotra and Y. Singh, "On the Applicability of Machine Learning Techniques for Object Oriented Software Defect Prediction," Software Engineering: An International Journal, Vol.1, No.1, 2011, pp.24-37.

[30] V. Ramaswamy , V. Suma , T.P. Pushphavathi, "AN APPROACH TO PREDICT SOFTWARE PROJECT SUCCESS BY CASCADING CLUSTERING AND CLASSIFICATION", Journal of Parallel and Distributed Computing. ISBN: 978-1-84919-736-6, 2012, Inspec.http://digital-library.theiet.org/content/conferences/2012/004, DOI:10.1049/ic.2012.0137 ,Publisher: IEEE.