International Journal of Future Generation Communication and Networking
Vol. 14, No. 1, (2021), pp. 1105-1119

Software Defect Prediction based on Random Forest Classifier with Artificial Neural Networks

R.Janarthanan
Research Scholar
Department of BCA
Kongunadu College of Arts and Science
janarthanan.raju@gmail.com

Dr.A.Hema
Associate Professor & Head
Department of BCA
Kongunadu College of Arts and Science
hemasrgm@yahoo.com

Abstract

Software is playing an increasingly essential role in many industries. However, defects are not only inconvenient and aggravating, but can also have serious cost for software systems, especially for mission-critical systems. Therefore, software defect prediction models are useful for understanding, evaluating and improving the quality of a software system. Machine learning techniques have been working to make predictions about the defectiveness of software components by exploiting historical data of software components and their defects. In order to predict software defects, many studies using Random forest classifier with Artificial Neural networks (RF-ANN) have been proposed. The effectiveness of our proposed method is evaluated using historical data from the NASA and PROMISE software engineering repository, by comparing it with a k-nearest neighbor, SVM and Random Forest baseline. Our evaluation on a widely used data set shows that our method significantly improves the performance of the proposed classifier.

Keywords: *Software defect, Random forest, Artificial Neural networks, code metrics, complexity*.

I. INTRODUCTION

Improving unwavering quality of the ideal programming is quite possibly the most searched out exploration territories in computer programming. Programming engineers lay accentuation on planning dependable programming, so that inadequately planned programming can be recognized in the starter phases of the Software Development life cycle (SDLC) to try not to convey inferior quality programming item to the partner. In this manner, programming quality goes about as a significant factor in deciding the dependability of programming. In this way, there is a requirement for plan of forecast models to anticipate shortcoming inclined modules or classes in programming created dependent on item situated advancement system**.**

In writing it is seen that, few quality models have been proposed also, concentrated, for example, McCall's quality model [1], Boehm's quality model [2], Dromey's quality model [3], and so on to assess the nature of a product item. A huge programming comprises of enormous number of lines of code in go prompting the presence of a colossal number of modules. It is very difficult to do unit testing of every single module. To check the usefulness and to guarantee

ISSN: 2233-7857 IJFGCN

unwavering quality of the product, a set number of significant intelligent ways in a module ought to be chosen and testing ought to be practiced on those modules, where likelihood of deficiencies are high [4].Software measurements assume an essential job in foreseeing the nature of the product. They give a quantitative premise, and a cycle for approving the models during SDLC [5]. The convenience of these measurements lies in their capacity to foresee the unwavering quality of the created programming. Practically defined, programming nature of a product framework can be best decided dependent on the FURPS model, which describes boundaries, for example, Functionality, Usability, Reliability, Performance and Acceptability [6]. Nature of any item is generally settled based on a significant boundary like unwavering quality. Dependability is for the most part estimated by the number of flaws identified in the created programming during a time frame. Engineers expect to foresee issues in modules apriori in order to convey a product with least number of shortcomings. Various models have been created for issue forecast as accessible in writing. In any case, issue expectation stays as a testing task in programming. There is a requirement for planning efficient models to anticipate programming inclined modules all the more precisely.

II. SOFTWARE DEFECT PREDICTION PROCESS WITH METRICS

The exceptionally basic cycle of foreseeing programming deserts is to utilize AI strategies that give PC frameworks the capacity to gain from information without being expressly customized. Right off the strike, informational collections are created from programming stores including deformity global positioning frameworks, source code changes, mail chronicles, information extraction and performance control frameworks. Those informational indexes comprise of examples, which can be programming segments, documents, classes, capacity and modules. In view of specific measurements like static code describe extricated from the product storehouses, an occasion is marked as deficient or imperfection free. The gathered informational collections are then cleaned utilizing preprocessing techniques, for example, commotion recognition and decrease, information standardization, and characteristic choice .After that, the preprocessed informational indexes are utilized for building an imperfection expectation model that is to foresee if new occasions contain absconds. Aside from the parallel order, this model can assess the quantity of imperfections in each occurrence. As far as AI, this assessment is additionally called regression.

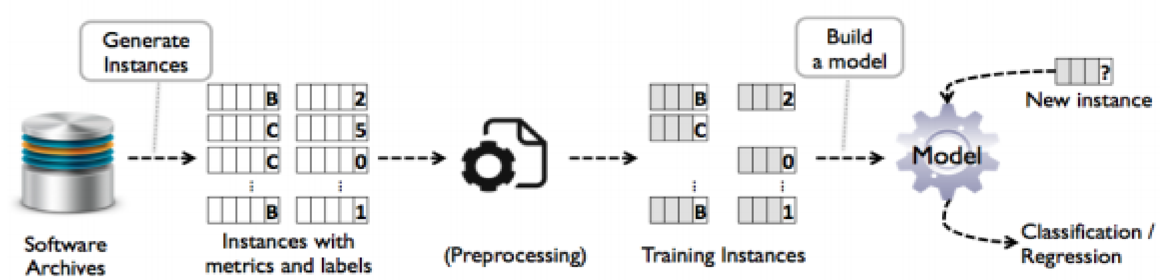


Fig-1 Software Defect Prediction process

ISSN: 2233-7857 IJFGCN

Software metrics can be considered as a quantitative measurement that assigns symbols or numbers to features of predicted instances. In fact, they are features, or 13 attributes, that describe many properties such as reliability, effort, complexity and quality of software products. These metrics play a key role in building an effective software defect predictor. They can be divided into two main categories: code metrics and process metrics.

Symbol	Description
μ_1	Number of distinct operators
μ_2	Number of distinct operands
N_1	Total number of operators
N_2	Total number of operands
μ_1^*	Minimum possible number of operators

Figure 1 – Halstead basic measurements

Name	Description
$Length: N = N_1 + N_2$	The program length
$Vocabulary: \mu = \mu_1 + \mu_2$	The vocabulary size
$Volume: V = N * \log_2 \mu$	The information content of a program
$Potential\ volume: V^* = (2 + \mu_2^*) \log_2(2 + \mu_2^*)$	The volume of the minimal size implementation of a program
$Level: L = V^*/V$	The program level
$Difficulty: D = 1/L$	The difficulty level of a program
$Error\ estimate: \hat{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$	Error estimate for a program
$Content: I = \hat{L} * V$	The intelligence content of a program
$Effort: E = \frac{V}{L} = \frac{\mu_1 N_2 N \log_2 \mu}{2\mu_2}$	The effort required to generate a program
$Programming\ time: T = E/18\ (seconds)$	The programming time required for a program

Figure 2 – Halstead Metrics

McCabe features are Cyclomatic measurements representing to the unpredictability of a product item. The features proposed dependent on the presumption that "the intricacy of pathways between

ISSN: 2233-7857 IJFGCN

module images is more smart that simply a tally of the images". Varying from Halstead ascribes, McCabe credits measure the unpredictability of source code structure. They are gotten by processing the quantity of associated parts, circular segments and hubs in control stream graphs of source code. Every hub of the stream outline speaks to a program proclamation while a circular segment is the progression of 15 controls from an assertion to another. The basic features are Cyclomatic complexity, essential complexity and design complexity. The object oriented metrics are represented figure 3.

Name	Description
Fan-in	The number of other classes that reference the measured class
Fan-out	The number of classes referenced by the measured class
NOA	The number of attributes
NOPA	The number of public attributes
NOPRA	The number of private attributes
NOAI	The number of attributes inherited
LOC	The number of lines of code in a class
NOM	The number of methods
NOPM	The number of public methods
NOPRM	The number of private methods
NOMI	The number of methods inherited

Figure 3- Object oriented metrics

McCabe features are used to predict the quality prediction based on Cyclomatic Complexity metrics the general form

$$V\ (G) = E - N + 2p$$

Where N–Nodes, E–Edges, P–connected procedures Extended Cyclomatic complexity (ECC): McCabe measures the program complexity based on conditional statement. Extended Cyclomatic complexity that may be defined as:

$$ECC=eV(G)=Pe+1$$

Where, Pe=number of predicate nodes in flow graph G weighted by number of compound statements. Information flow metrics may be finding by count the number of local information flows input (fan-in) and flows output (fan-out). The procedure may be defined as:

$$C=[\text{procedure length}]*[(\text{fan-in})*(\text{fan-out})]2$$

ISSN: 2233-7857 IJFGCN

III. METHODOLOGY

The proposed methodology is systematically presented in figure4. The methodology consists of data collection, feature selection & Extraction, classification and performance evaluation. For this software defect process approach, ANN is used for train the data without imbalance class criteria and get the optimal values for the weights. The main aim of this process is to minimize the error. Random Forest algorithm will select the small subset of available attributes at random. It splits the node with the best variable among the available features. The embedded classifier based on Random forest with the help of artificial neural networks. The performance evaluation is carried out to evaluate and distinguish classes namely defectiveness and non-defective. The proposed algorithm is described in figure 5.

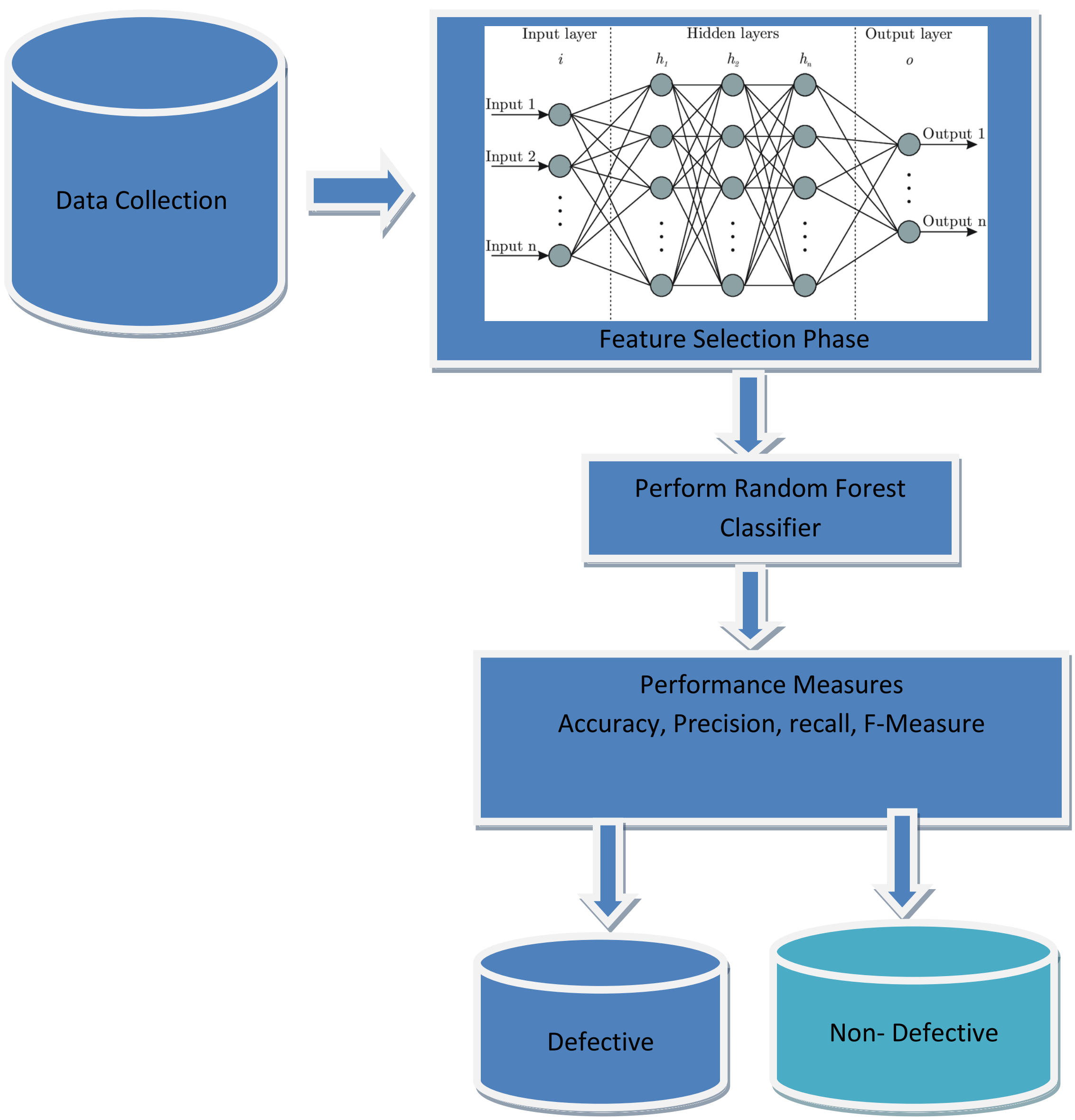


Figure 4 Proposed frameworks

ISSN: 2233-7857 IJFGCN

Pseudo code of RF-ANN Algorithm

Define the ANN Architecture – number of input, hidden and output neurons.

Identify the fitness function which returns the error as difference of actual and predicted output for the ANN.
Initialize a neuron of x particles with random weights of n dimension where n dimension where n is the total number of weights that needs to be optimized for the ANN
To generate c classifiers
For i=1 to c do
Randomly sample the training data D with replacement to produce D_i
Create a root node, N_i containing D_i
Call Build Tree (N_i)
End for
Build Tree (N):
If N Contains instances of only one class then
Return
else
Randomly select x% of the possible splitting features in N
Select the feature F with the highest information gain to split on
Create f child nodes and f possible values (F_1……….F_n)
For i=1 to f do
Set the contents of N_i to D_i, where D_i is all instances in N that match
F_i
Call Build Tree (Ni)
End for
End IF

Figure 5 – Pseudo code of Proposed Algorithm

IV. RESULTS AND DISCUSSION

In this section, we design experiments to verify the effectiveness of RF-ANN. two research questions are needed to be answered as follows

1. Do the machine learning methods improve the performance of defect prediction compared to traditional methods based on static code metrics?
2. Compared with features generated by the classical unsupervised learning methods, do features learned by the machine learning methods better represent syntax and semantics of programs?

4.1 Experimental Datasets

In order to evaluate the software quality we consider the following dataset for empirical study. In this paper we are discuss four open datasets in NASA, PROMISE which are used for defect prediction based quality assessment.

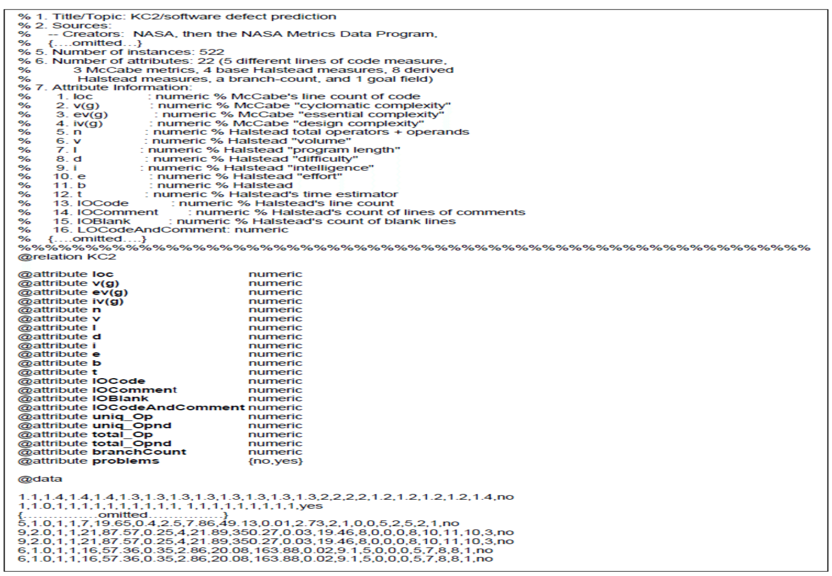
ISSN: 2233-7857 IJFGCN

a) NASA Dataset- this dataset was collected by NASA metrics data program (Table 1). This dataset contain 40 features may contain both Hallstead features and McCabe features.

Data	Modules/ instances	Language	Description
CM1	498	C	Space craft instrument
PC1	1109	C	Earth orbiting satellite
KC1	2109	C++	Storage management for ground data
KC2	522	C++	Science data processing
PC4	1458	C	Flight software for earth orbiting satellite

Table 1 –NASA Dataset

PROMISE Dataset – it is open source java projects which contain different metrics such as lines of code, Response for class, Average method complexity, coupling between object classes etc. this metrics is used for evaluation of software quality effectiveness. The fig 6 describes the metrics and attributes of the PROMISE Data set



```
% 1. Title/Topic: KC2/software defect prediction
% 2. Sources:
%    -- Creators:  NASA, then the NASA Metrics Data Program,
%    {....omitted...}
% 5. Number of instances: 522
% 6. Number of attributes: 22 (5 different lines of code measure,
%         3 McCabe metrics, 4 base Halstead measures, 8 derived
%          Halstead measures, a branch-count, and 1 goal field)
% 7. Attribute Information:
%      1. loc             : numeric % McCabe's line count of code
%      2. v(g)            : numeric % McCabe "cyclomatic complexity"
%      3. ev(g)           : numeric % McCabe "essential complexity"
%      4. iv(g)           : numeric % McCabe "design complexity"
%      5. n               : numeric % Halstead total operators + operands
%      6. v               : numeric % Halstead "volume"
%      7. l               : numeric % Halstead "program length"
%      8. d               : numeric % Halstead "difficulty"
%      9. i               : numeric % Halstead "intelligence"
%     10. e               : numeric % Halstead "effort"
%     11. b               : numeric % Halstead
%     12. t               : numeric % Halstead's time estimator
%     13. lOCode          : numeric % Halstead's line count
%     14. lOComment       : numeric % Halstead's count of lines of comments
%     15. lOBlank         : numeric % Halstead's count of blank lines
%     16. LOCodeAndComment: numeric
%    {....omitted....}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
@relation KC2

@attribute loc                 numeric
@attribute v(g)                numeric
@attribute ev(g)               numeric
@attribute iv(g)               numeric
@attribute n                   numeric
@attribute v                   numeric
@attribute l                   numeric
@attribute d                   numeric
@attribute i                   numeric
@attribute e                   numeric
@attribute b                   numeric
@attribute t                   numeric
@attribute lOCode              numeric
@attribute lOComment           numeric
@attribute lOBlank             numeric
@attribute lOCodeAndComment    numeric
@attribute uniq_Op             numeric
@attribute uniq_Opnd           numeric
@attribute total_Op            numeric
@attribute total_Opnd          numeric
@attribute branchCount         numeric
@attribute problems            {no,yes}

@data

1.1,1.4,1.4,1.4,1.3,1.3,1.3,1.3,1.3,1.3,1.3,1.3,2,2,2,2,1.2,1.2,1.2,1.2,1.4,no
1,1.0,1,1,1,1,1,1,1,1,1,1,1, 1,1,1,1,1,1,1,1,1,yes
{................omitted...............}
5,1.0,1,1,7,19.65,0.4,2.5,7.86,49.13,0.01,2.73,2,1,0,0,5,2,5,2,1,no
9,2.0,1,1,21,87.57,0.25,4,21.89,350.27,0.03,19.46,8,0,0,0,8,10,11,10,3,no
9,2.0,1,1,21,87.57,0.25,4,21.89,350.27,0.03,19.46,8,0,0,0,8,10,11,10,3,no
6,1.0,1,1,16,57.36,0.35,2.86,20.08,163.88,0.02,9.1,5,0,0,0,5,7,8,8,1,no
6,1.0,1,1,16,57.36,0.35,2.86,20.08,163.88,0.02,9.1,5,0,0,0,5,7,8,8,1,no
```

Figure 6 PROMISE Data set

4.2 Evaluation Metrics

The proposed system is evaluated several performance metrics such as true positive rate, false positive rate, precision, recall, F-Measure and accuracy.

True positive rate: This measure is projected by the modules that are predicted positively as the results specified at the end. The general for that is represented below equation.

ISSN: 2233-7857 IJFGCN

True positive rate = true positive rate / (true positive rate + false negative rate

False Positive rate: This measure is projected by the modules that are predicted incorrectly categorized ad class x/ actual total of all classes, except x.

False positive rate = false positive rate / (true negative + true negative rate

Precision: precision gives positive predicate values and it process values or product quality or exactness.

Precision = True positive / (True Positive + False positive)

Recall: recall gives sensitive of problem and it process values or product quantity or completeness. This measure is used to recognize total number of modules.

Recall = true positive / (true positive + false negative)

F-Measure: it is one of the quality measures of the modules. The general formula is represented as given below

F-Measure = 2* Precision * recall / (precision + recall)

Accuracy: it is calculated as a number of instances predicted positively divided by total number of instances

Accuracy = (true positive + true negative) / (P+N)

Table 1: classified Instances for CM1

Method	Approximately Classified Instances	Inaccurately classified instances	Total instances
K-Means	425	73	498
Proposed	445	53	498

Fig- 4 Performance Analysis for CM1 Data set

Table 2: Performance analysis for CM1

Method/ Performance measures	Random Forest	Proposed
TP Rate	0.85	0.90
FP Rate	0.61	0.70
Precision	0.83	0.85
Recall	0.85	0.87
F-Measure	0.85	0.86
Accuracy	0.83	0.88

ISSN: 2233-7857 IJFGCN

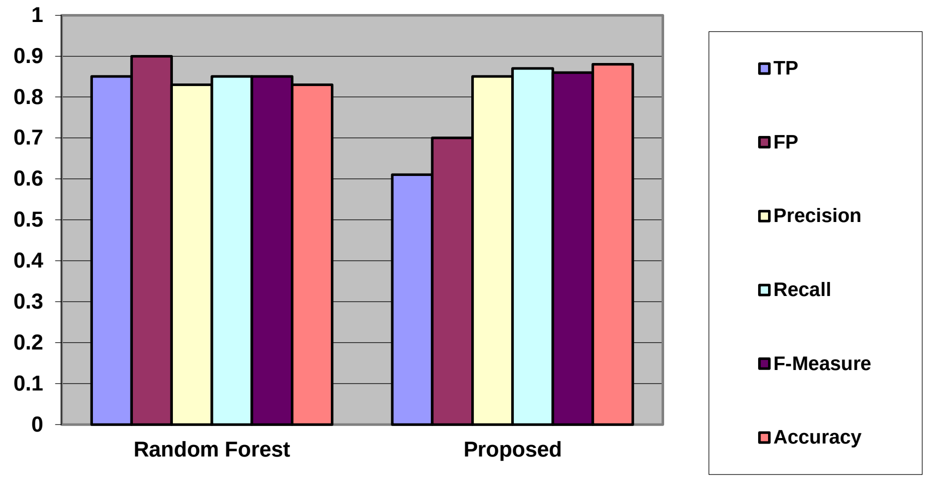


Fig- 4 Performance Analysis for CM1 Data set

Table 3: classified Instances for PC1

Method	Approximately Classified Instances	Inaccurately classified instances	Total instances
K-Means	965	144	1109
Proposed	1025	84	1109

Table 4: Performance analysis for PC1

Method/ Performance measures	Random Forest	Proposed
TP Rate	0.92	0.96
FP Rate	0.65	0.69
Precision	0.88	0.90
Recall	0.89	0.90
F-Measure	0.89	0.91
Accuracy	0.89	0.92

ISSN: 2233-7857 IJFGCN

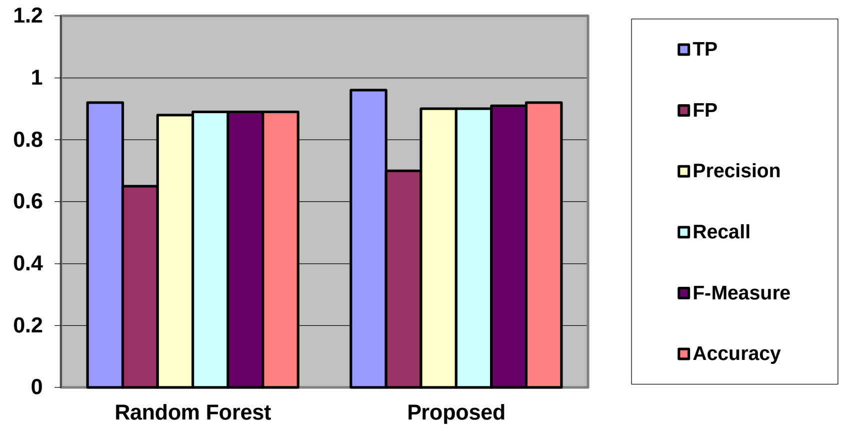


Fig- 5 Performance Analysis for PC1 Data set

Table 5: classified Instances for KC1

Method	Approximately Classified Instances	Inaccurately classified instances	Total instances
K-Means	965	144	1109
Proposed	1025	84	1109

Table 6: Performance analysis for KC1

Method/ Performance measures	Random Forest	Proposed
TP Rate	0.92	0.96
FP Rate	0.65	0.69
Precision	0.88	0.90
Recall	0.89	0.90
F-Measure	0.89	0.91
Accuracy	0.89	0.92

ISSN: 2233-7857 IJFGCN

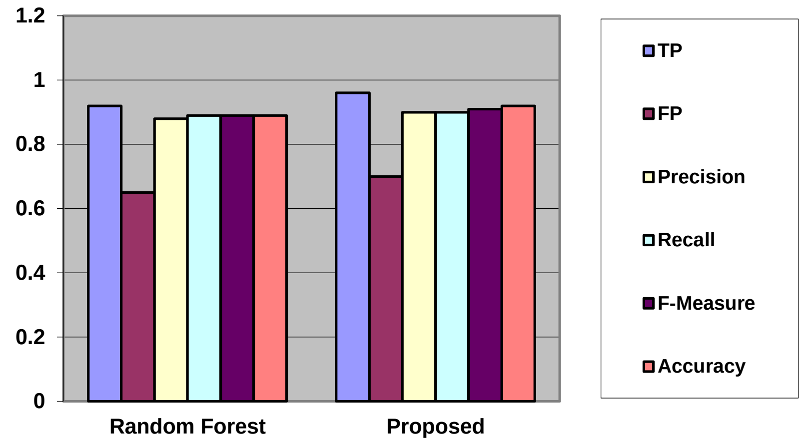


Fig- 6 Performance Analysis for KC1 Data set

Table 7: classified Instances for KC2

Method	Approximately Classified Instances	Inaccurately classified instances	Total instances
K-Means	470	52	522
Proposed	501	21	522

Table 8: Performance analysis for KC2

Method/ Performance measures	Random Forest	Proposed
TP Rate	0.90	0.94
FP Rate	0.63	0.67
Precision	0.80	0.90
Recall	0.87	0.90
F-Measure	0.88	0.91
Accuracy	0.86	0.94

ISSN: 2233-7857 IJFGCN

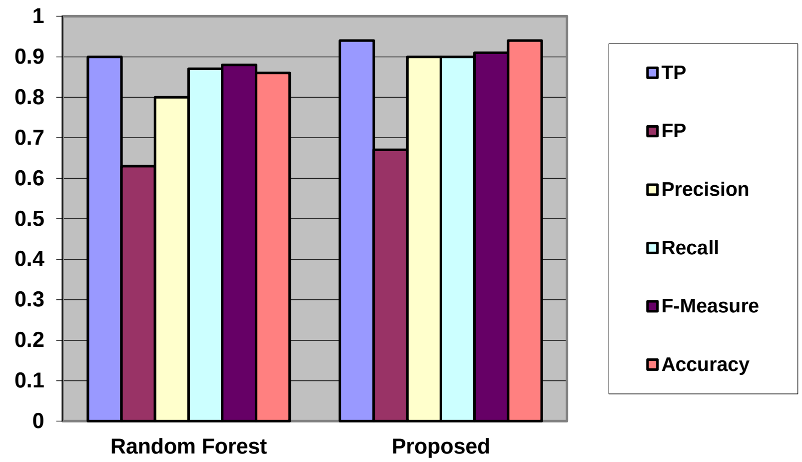


Fig- 7 Performance Analysis for KC2 Data set

Table 9: classified Instances for PC4

Method	Approximately Classified Instances	Inaccurately classified instances	Total instances
K-Means	1280	178	1458
Proposed	1325	155	1458

Table 10: Performance analysis for PC4

Method/ Performance measures	Random Forest	Proposed
TP Rate	0.91	0.95
FP Rate	0.70	0.75
Precision	0.89	0.91
Recall	0.87	0.93
F-Measure	0.85	0.95
Accuracy	0.90	0.98

ISSN: 2233-7857 IJFGCN

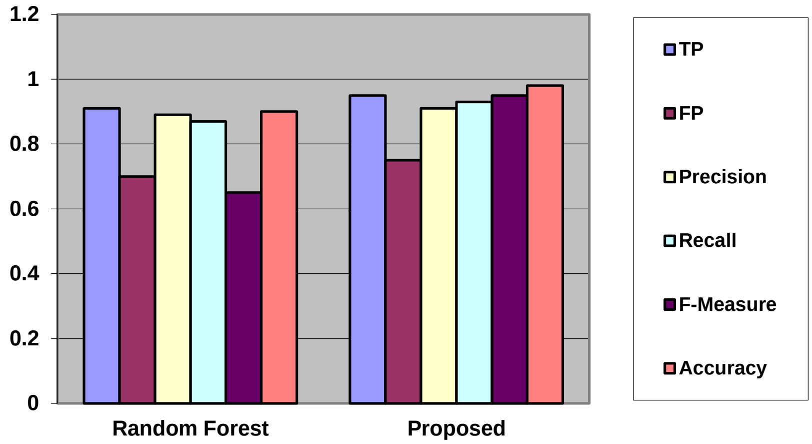


Fig- 8 Performance Analysis for PC4 Data set

V. CONCLUSION

The main intention of this work is to analyze the performance of Random Forest classifier with RF-ANN algorithm using different metrics of NASA datasets. Based on this performance analysis we conclude that our proposed approach is suitable for small and large data set. The complexity factor is low when compared to the existing approach. The future enhancement of this work is planned to measure different similarity measures with Fuzzy logic approach based on Equivalence and Composite relations.

References

[1] A Kaur, et. al. (2009),|| Early software fault prediction using real time defect data|| , 2009 Second International Conference on Machine Vision, pp 243-245

[2] Rashid, Ekbal, Patnayak S, Bhattacherjee V. Estimation and evaluation of change in software quality at a particular stage of software development. Indian Journal of Science and Technology. 2013; 6(10):5370-9.

[3] Sathyaraj R, Prabu S. A survey-quality based object oriented software fault prediction. International Journal of Engineering and Technology. 2013 Jun–Jul; 5(3): 2349-51.

[4] Catal C, Diri B. Investigating the effect of dataset size, metrics sets and feature selection techniques on software fault prediction problem. Information Sciences. 2009; 170(8):1040-58.

[5] Jiang Y, Cukic B, Ma Y. Techniques for evaluating fault prediction models. Empirical Software Eng. 2008; 13(5):561– 95.

[6] Kaur S, Kumar D. Software fault prediction in object oriented software systems using density based clustering approach. International Journal of Research in Engineering and Technology (IJRET). 2012 Mar; 1(2):111-7.

[7] Moeyersoms J, Fortuny EJ, Dejaeger K, Baesens B. Comprehensible software fault and effort prediction: A datamining approach. The Journal of Systems and Software. 2015 Feb; 100:80-90.

ISSN: 2233-7857 IJFGCN

[8] Boetticher G. Improving credibility of machine learner models in software engineering. Advanced Machine Learner Applications in Software Engineering. Hershey, PA, USA: Idea Group Publishing; 2006.

[9] NASA Metrics Data Program. 2015 Apr 15. Availablefrom:http://promise.site.uottawa.ca/SERepository/datase ts-page.html

[10] Catal C, Sevim U, Diri D. Practical development of an Eclipsebased software fault prediction tool using Naive Bayes algorithm. Expert Systems with Applications. 2011; 38(3):2347-53.

[11] Archana Singh et. al. International Journal of Computer Applications (0975 – 8887) Volume 67– No.10, April 2013

[12] Aditi Sanyal, Balraj Singh, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 1, January 2014 ,ISSN: 2277 128X

[13]Teknomo, Kardi, Similarity Measurement Available fromhttp:\\people.revoledu.com\kardi\tutorial\Similarity\

[14] Bray J. R., Curtis J. T., 1957. An ordination of the upland forest of the southern Winsconsin. Ecological Monographies, 27, 325-349.

[15] G.Gan,C. Ma,J.Wu,―Data clustering: theory,algorithms, and applications‖ , Society for Industrial and Applied Mathematics, Philadelphia, 2007.

[16] Jiang Y. et. al., ―Fault Prediction Using Early Lifecycle Data‖ . ISSRE 2007, the 18th IEEE Symposium on Software Reliability Engineering, IEEE Computer Society, Sweden, pp. 237-246.

[17] Seliya N., Khoshgoftaar T.M. (2007), ―Software quality with limited fault-proneness defect data: A semi supervised learning perspective‖ , published online pp.327-324.

[18] Jiang Y, Cukic B, Menzies T,‖ Cost curve Evaluation of fault prediction models‖ , Proceedings of the 2008 19th International Symposium on Software Reliability Engineering, 2008,pg 197- 206

[19] Basili, V.R., Calidiera, G., Rombach, H.D.: Goal Question Metric Paradigm.In: Marciniak, J.J. (ed.): Encyclopaedia of Software Engineering, pp. 528-532, Wiley, New York, 1994.

[20] Catal, Cagatay, and Banu Diri. “A systematic review of software fault prediction studies.” Expert systems with applications 36.4 (2009): 7346-7354.

[21] Dubelaar, Chris, Amrik Sohal, and Vedrana Savic. “Benefits, impediments and critical success factors in B2C E-business adoption.” Technovation25.11 (2005): 1251-1262.

[22] Graves, Todd L., et al. “Predicting fault incidence using software change history.” IEEE Transactions on software engineering 26.7 (2000): 653-661. IEEE Standard Classification for Software Anomalies,” in IEEE Std 1044-2009 (Revision of IEEE Std 1044- 1993) , vol., no., pp.1-23, Jan. 7 2010, doi: 10.1109/IEEESTD. 2010. 5399061.

[23] Jiang, Yue, Bojan Cukic, and Tim Menzies. “Fault prediction using early lifecycle data.” The 18th IEEE International Symposium on Software Reliability (ISSRE’07). IEEE, 2007. Sommerville, Ian. “Integrated requirements engineering: A tutorial.” IEEE software 22.1 (2005): 16-23.

[24] Todd L. Graves, Alan F. Karr, J.S. Marron, and Harvey Siy, “Predicting Fault Incidence Using Software Change History” IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 26(7), 653-661, JULY 2000.

[25] Zelkowitz, M.V., Wallace, D.R.: Experimental models for validating technology. IEEE Computer, (31)5, pp. 23-31, May 1998.

[26] Ma, Y., Guo, L. (2006), “A Statistical Framework for the Prediction of Fault-Proneness”, West Virginia University, Morgantown.

ISSN: 2233-7857 IJFGCN

[27] Thomas Zimmermann, Nachiappan Nagappan, " Predicting Defects Using Social Network Analysis on Dependency Graphs", International Conference on Software Engineering (ICSE 2008), Leipzig, Germany.

[28] Audris Mockus, Nachiappan Nagappan and Trung T.Dinh-Trong "Test Coverage and Post-Verification Defects: A Multiple Case Study," ACM-IEEE Empirical Software Engineering and Measurement Conference (ESEM), Orlando, FL, 2009

ISSN: 2233-7857 IJFGCN