An Intuitionistic Fuzzy K-Medoids Based Similar Pattern Analysis in Software Defect Prediction

M. Jaikumar[1], V. Kathiresan[2]
[1]Assistant Professor & Head, Department of Computer Applications(UG)
[2]Associate Professor & Head, Department of Computer Applications(PG)
[1]Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore, Tamilnadu, India
[2]Dr.SNS Rajalakshmi College of Arts & Science, Coimbatore, Tamilnadu, India

Abstract

In the software field the quality and the reliability are the important factors which have to be greatly handled with the help of software defect prediction. During the period of development and the maintenance of the software detecting and rectifying the software defects is really more expensive. By designing prediction model which accurately determines the occurrence of defect in software greatly assist in efficient software testing, reducing the cost and considerably improvising testing process of software by focusing on fault prone modules. Machine Learning Clustering has emerged as a way to predict the fault in the software system by grouping the similar patterns. This paper focuses on predicting the software defect contributed by NASA repository dataset. This work uses Intuitionistic fuzzy K-medoids based clustering for finding the similar pattern among the software defect dataset and design the rule based on it.

Keywords

Software, defect, intuitionistic fuzzy, K-medoids, clustering, fault

I. Introduction

With the help of software metrics and the software fault dataset which was collected from the earlier developed software or projects based on the prediction model is trained and developed for software defect detection. This well trained model can then be applied to unknown defect data of any software module. The performance of the defect prediction is greatly influenced by the attributes of the software metrics which increases the efficiency of the software considerably. As sovereign testing team, it is significant to map and administer the test implementation behavior in order to convene the tight limit for releasing the software to end-users. Since the aspire of test carrying out is to determine as many fault as possible, testing team is typically put into encumber to guarantee all defects are establish and set by the developers inside the system testing stage.

Extra number of days has to be added to the timeline to contain testing team in effecting their test with the trust that all defects have been originate and set. On the other hand, the stakeholders would also ask the difficult team on the forecasted defects in the software so that they could choose whether the software is reasonable and robust for release. This is owing to the environment that system testing is the last gate before the software is made visible to end-users, thus as the custodian of executing system testing, the autonomous testing team has to take liability to ensure software to be unrestricted is of high excellence.

Therefore, the ability to predict how many defects that can be found at the start of system testing shall be a good way to tackle this issue. This becomes the reason for conducting this study. Besides serving as a target on how many defects to capture in system testing, defect prediction can also become an early quality indicator for any software entering the testing phase. Testing team can use the predicted defects to plan, manage and control test execution activities. This could be in the form aligning the test execution time and number of test engineers assigned to particular testing project. Having defect prediction as part of the testing process allows testing team to strengthen their test Strategies by adding more exploratory testing and user experience testing to ensure known defects are not escaped and re-introduced to end-users.

II. Related Work

Software defect prediction is not a new thing in software engineering domain. To come out with the right defect prediction model various related studies and approaches have been conducted. Understanding what defect really means is important so that the term defect is not confused with error, mistake or failure. In the event the defect have taken place, when the software or system fails to perform its desired function [1]. Defect is also observed as the

deviation from its specification [2] as well as any imperfection related to software itself and its related work product [3]. Consequently, defect can be referred as its work product and something that is not according to requirement for software. Since, the defects means it is the structure the prediction model for defects, it is used to know how defects are introduced as part of verification and validation (V&V) activities [3].

Defects predicting can be characterized in the proactive process of many types of defects that can be found in software's content, design and codes in producing high quality product [4]. To predict defect density Rayleigh model was also used for different phases of project life cycle [5]. In [6] product and project metrics collected from design review, code testing, code peer review as well as product release usage and defect validation can be constructed using the model to predict defects. Linear regression was applied to these metrics via product metrics only, project metrics only and both. As the result, both product and project metrics provided better correlation between defects and the predictors using linear regression. It demonstrated the feasibility of using regression analysis to build defect prediction model at the same time. To predict defects an approach was carried out using mathematical distributions that serve as quality prediction model [7].

In order to identify and predict the highest defects in the large software systems will prone to more defect is investigated was performed in it. The important factor for the prediction and its impact to the model quality is development information will be the result of the investigation, which focuses on three metrics: number of developers who modified the file during the prior release; the number of new developers who modified the file during the prior release; and the cumulative number of distinct developers who modified the file during all releases through the prior release [8].

We also study to investigate on how to defect fault-proneness in the source code of the open source Web and e-mail suite called Mozilla. To conduct the investigation it used object-oriented metrics proposed by Chidamber and Kemerer [9]. On the other hand, [10] to build defect prediction model was proposed several inputs to simulate the system test phase, in which those inputs could be considered as potential predictors. The defect prediction was based on simple Bayesian Network in a form of Defect Type Model (DTM) that predicts defects based on severity minor, major and minor was the another approach to defects prediction [11]. To come out with defect inflow prediction for large software projects either short-term defect inflow prediction or long-term defect inflow prediction [12] is used by Multivariate linear regression. [13] To predict defect density statistical approach in Six Sigma methodology is applied. In this case, Statistical method was used against the function point as the base metrics to predict defect density before releasing software to production. Defect prediction can also be observed from different perspective which is by predicting remaining total number of defects while the testing activities are still on-going [14], which is called as defect decay model. This model depends on on-going test execution data instead of historical data. [15] Case studies can be presented on building and assisting their organization to assess testing effectiveness and predict the quantity of post release defects and enables quantitative decision about production go-live readiness the defect prediction model was used.

Their model was mostly focused on predicting defects in receiving test or manufacture which involves estimate total possible defects based on defined thorough requirements, applying defect elimination efficiency and finally estimates the defects per phase as well as post discharge defects. It display a 1% defect removal efficiency improvement which equals to $20,000 for implementing this model, The defect prediction would be difficult However, if past data is not available. Sample-based defect prediction was proposed to overcome this difficulty by using a small sample of modules to construct cost-effective defect forecast models for large scale systems, in which Co Forest, a semi supervised learning method was applied [16]. For defect prediction testing resources portion could be optimized, [17] on predicting defects of cross-project when chronological data is not in place possibility study must be conducted.

The training data is very significant for machine learning based defect prediction provided that the data is carefully selected from the projects was demonstrated as results. Building of defect prediction system, it is necessary to couple with the technique to find its success. In [18] the authors proposed to compute the percent of faults establish in the recognized files as one of the ways to review the efficiency of the prediction Systems. In addition that, the model is said to be a good if it can help in the resource planning in order to maintain the software and insure based on the software system itself is insured [19]. However, it is firm to discover an recognized standard specific for defect prediction. An attempt was taken by given that an all-embracing contrast of well-known bug prediction approaches, jointly with narrative approaches using openly available dataset consisting of numerous software systems [20]. The findings showed that there is still a

difficulty with observe to exterior soundness in defect prediction. It necessitate larger mutual data set towards having a noteworthy target of defect prediction

K-Medoids Clustering

The k-medoids algorithm is a clustering algorithm related to the k-means algorithm and the medoids shift algorithm. Both the k-means and k-medoids algorithms are partitioned (breaking the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids chooses data points as centers (medoids or exemplars) and works with a generalization of the Manhattan Norm to define distance between data points instead of this method was proposed for the work with norm and other distances.

K-medoids is a classical partitioning technique of clustering that clusters the data set of n objects into k clusters known a priori. A useful tool for determining k is the silhouette. It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances. A medoids can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster.

Input:

k: The number of clusters

D: A data set containing n objects

Output: A set of k clusters that minimizes the sum of the dissimilarities of all the objects to their nearest medoid.

Method:

Arbitrarily choose k objects in D as the initial representative objects;

Repeat

Assign each remaining object to the cluster with the nearest medoid;

Randomly select a non medoid object Orandom;

Compute the total points S of swapping object Oj with Orandom; if S < 0 then swap Oj with Orandom to form the new set of k medoid; Until no change;

It works as follows:

1. Initialize: select k of the n data points as the medoids
2. Associate each data point to the closest medoid.
3. While the cost of the configuration decreases:
    1. For each medoid m, for each non-medoid data point o:
        1. Swap m and o, recomputed the cost (sum of distances of points to their medoid)
        2. If the total cost of the configuration increased in the previous step, undo the swap

Proposed K-medoids algorithm

4. Suppose that n objects having p variables each should be grouped into k (k < n) clusters, where k is assumed to be given. Let us define j^{th} variable of object i as Xij (i = 1,. . .,n; j = 1,. . .,p). The Euclidean distance will be used as a dissimilarity measure in this study although other measures can be adopted. The Euclidean distance between object i and object j is given by

$$d_{ij} = \sqrt{\sum_{a=1}^{P} (X_{ia} - X_{ja})^2} \qquad (1)$$

$$I = 1,\ldots, n \; ; j = 1,\ldots n \qquad (2)$$

It should be noted that the above Euclidean distance will be adopted in K-means and PAM algorithms in this study. The proposed algorithm is composed of the following three steps.

Step 1: (Select initial medoids)

1-1. Calculate the distance between every pair of all objects based on the chosen dissimilarity measure (Euclidean distance in our case).

1-2. Calculate vj for object j as follows:

$$V_j = \sum_{i=1}^{n} \frac{d_{ij}}{\sum_{l1}^{n} d_{il}} \quad , \qquad j = 1 \ ,\ldots, \ n \qquad (3)$$

1-3. Sort vj's in ascending order. Select k objects having the first k smallest values as initial medoids.

1-4. Obtain the initial cluster result by assigning each object to the nearest medoid. 1-5. Calculate the sum of distances from all objects to their medoids.

Step 2: (Update medoids)

Find a new medoid of each cluster, which is the object minimizing the total distance to other objects in its cluster. Update the current medoid in each cluster by replacing with the new medoid.

Step 3: (Assign objects to medoids)

3-1. Assign each object to the nearest medoid and obtain the cluster result.

3-2. Calculate the sum of distance from all objects to their medoids. If the sum is equal to the previous one, then stop the algorithm. Otherwise, go back to the Step 2.

The above algorithm is a local heuristic that runs just like K-means clustering when updating the medoids. In Step 1, we proposed a method of choosing the initial medoids.

5. This method tends to select k most middle objects as Initial medoids. The performance of the algorithm may vary according to the method of selecting the initial medoids

Fuzzy C-medoids Clustering

Fuzzy C – Medoids Algorithm Kaufman et al. in 1987 developed a k-medoids-based clustering called PAM. A medoid is defined as the object of a cluster, whose average dissimilarity to all the objects in the cluster is minimal i.e. it is a most centrally located point in the given data set. The k-medoids [3] approach also produces a data set partition with k clusters in order to minimize the total intra-cluster dissimilarity, just like k-means algorithm. But the main difference between k-means and K-Medoids lies in the selection of center of cluster that represents the cluster. In k-medoids, the center is a real object from the dataset while in k-means the center may be a non-real object calculated as mean of all the data elements. K – Medoids algorithm avoids calculating means of clusters in which extremely large values may affect the membership computations substantially. K-medoids can handle outliers well by selecting the most centrally located object in a cluster as a reference point, namely, medoid.

The basic idea of k-medoids is that it first arbitrarily finds k objects amongst n objects in the dataset as the initial medoids. Then the remaining objects are partitioned into k clusters by computing the minimum Euclidian distances that can be maintained for the members in each of the clusters. An iterative process then starts to consider objects Pi, i = 1,…,n if a medoid oj , j = 1,…,k, can be replaced by a candidate object oc, c = 1,…,n, c not equal to i.

There are four situations to be considered in this process:

1. Shift-out membership: an object Pi may need to be shifted from currently considered cluster oj to another cluster

2. Update the current medoid: a new medoid oc is found to replace the current medoid oj .

3. No change: objects in the current cluster result have the same or even smaller SEC (square error criterion) for all possible redistributions considered

4. Shift in membership: an outside object pi is assigned to the current cluster with the new (replaced) medoid oc.

The Fuzzy c-Medoids Algorithm

The fuzzy c-Medoids Algorithm (FCMdd)

Fix the number of clusters c; Set *iter* = 0;

Randomly pick the initial set of metdoids:

V = {v_1, $v_{1,\ldots,}$ v_c } from X_c ;

Repeat

for *i=1* **to** *c* **do** /*compute membership:*/

for j=1 **to** n **do**

Compute u_{ij} by using (2),(3),(4) or (5).

endfor

endfor

store the current medoids: $\mathbf{V}^{old} = \mathbf{V}$;

Compute the new medoids:

for *i=1* to *c* **do**

$$q = \text{argmin} \sum_{j-1}^{n} u_{ij}^{m} r(X_k, X_j)$$

$$1 \le k \le n$$

$v_i = x_{ij}$;

endfor

iter = iter +1;

Until ($V^{odd} - V$ *or iter* $-$ MAX_ITER).

The Fuzzy Medoids Algorithm minimizes its objective function as

$$J_m(V; X) = \sum_{i=1}^{n}\sum_{i=1}^{c} u_{ij}^{m} r(x_j, v_i), \quad (4)$$

Where the minimization is performed over all V in Xc. In (1), u_{ij} represents the fuzzy [18] or possibilistic [19] [20] membership of xj in clusters i. The membership uij can be defined heuristically in many different ways. For example, we can use the FCM [18] membership model given by:

$$u_{ij} = \frac{\left(\frac{1}{r(x_j, v_i)}\right)^{1/(m-1)}}{\sum_{k=1}^{c}\left(\frac{1}{r(x_j, v_k)}\right)^{1/(m-1)}} \quad (5)$$

where m2[1;∞)is the "fuzzifier".

Intuitionist Fuzzy Logic System

Preliminaries

In this section, some basic definitions, which are prerequisites for the study, are outlined.

Definition 1 [4] Let the universal set X be fixed. An intuitionist fuzzy set Ain X is defined as an object of the form A = {_x,μA(x), νA(x)_ : x ∈ X} where the functions μA : X → [0, 1] and νA : X → [0, 1] define the degrees of membership and non-membership of the element x ∈ X respectively, and for every x ∈ X in A, $0 \le \mu A(x) + \nu A(x) \le 1$ holds.

Definition 2. [4] For every common intuitionistic fuzzy subset A on X, we have $\pi A(x) = 1 - \mu A(x) - \nu A(x)$ called the intuitionistic fuzzy index or hesitancy index of x in A. πA(x) is the degree of indeterminacy of x ∈ X to the IFS A. πA(x) expresses the degree of lack of knowledge of every x ∈ X belongs to IFS or not. Obviously, for every x ∈ X and $0 \le \pi A(x) \le 1$.

Definition 3. [9] Membership function for an intuitionistic fuzzy set A on the universe of discourse X is defined as μA : X → [0, 1], where each element X is mapped to a value between 0 and 1. The value μA(x), x ∈ X is called the membership value or degree of membership.

Definition 4. [9] Non-membership function for an intuitionistic fuzzy set A on the universe of discourse X is defined as νA : X → [0, 1], where each element X is mapped to a value between 0 and 1. The value νA(x), x ∈ Xis called the non-membership value or degree of non-membership.

Proposed Method

This proposed performs the predictions of software defect using the dataset collected NASA repository. The Dataset is normalized to fall under the range of 0 to 1. The min-max normalization is followed. The dataset is voluminous to handle so to overcome that the feature subset selection strategy is applied for finding the optimal features. The features selected using the dempster shafer theory for handling uncertainty in selection of feature selection. From the reduced feature set the similarity among the dataset instances are determined using the intuitionistic fuzzy K-Medoids based clustering and the resultant cluster set are used to generated the useful pattern of rule using fuzzy logic to classify whether the given instance is defect or defect free.

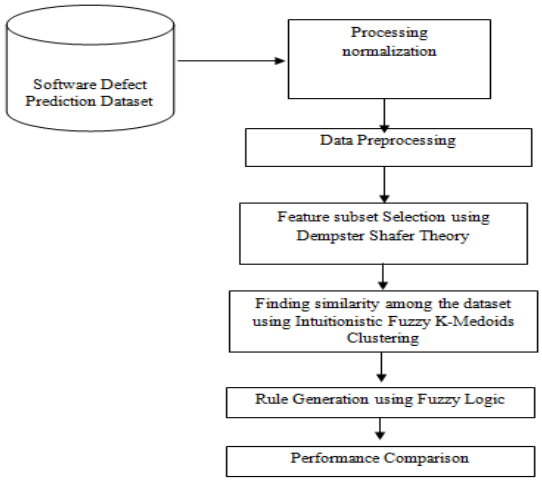

Fig 1: Overall framework of proposed Intuitionistic Fuzzy k-medoids for software defect predicition.

Intuitionistic Fuzzy C-Means Clustering

IFCM algorithm assigns pixels to each category by using membership, non-membership and hesitancy values. Let X= (x_1, x_2, … x_{MxN}) be an image with M x

N pixels to be partitioned into c clusters where *x*i represents multispectral (features) data. The IFCM clustering algorithm is an iterative function and aimed at minimizing the inter cluster similarity and IFE. The objective function of IFCM clustering is defined as follows.

$$J = \sum_{i=1}^{C}\sum_{k=1}^{n} u_{ik}^{*m} d(x_k, v_i)^2 + \sum_{i=1}^{C} \pi_i^* e^{1-\pi_i^*}$$

with m = 2

(6)

where c is the number of clusters, n is the number of data points, u^*_{ik} is the intuitionistic fuzzy membership matrix, v_i is the cluster center; is $d(x_k, v_i)$ the distance measure between data points and cluster center; and is $\pi_i^* e^{1-\pi^*_i}$ the fuzzy entropy.

$$u_{ik}^* = \frac{1}{\sum_{j=1}^{c}\left[[d_{ik}^2 / [d_{jk}^2 \right]^{1/m-1}} + \pi_{ik}$$

(7)

$$v_i^* = \frac{\sum_{k=1}^{n} u_{ik}^* x_{ik}}{\sum_{k=1}^{n} u_{ik}^*}$$

(8)

Normalized Euclidian distance measure is used to calculate the distance between the data points (A) and cluster center (B) and defined as

qIFS(A,B) =

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mu_A(x_i) - \mu_B(x_i))^2 + (v_A(xi) - v_B(x_i))^2}$$

(9)

During implementation, matrix v^* is randomly initialized, and then u^* and v^* are updated through an iterative process using Equations for membership , non-membership and distance calculation.

Design of Fuzzy Rules and FIS

Fuzzy rules are modeled by making use if the expert knowledge in software engineering. In our model the needed fuzzy rules for the prediction of defects of software projects are obtained by us. The obtained fuzzy rules are generated by considering all the selected software metrics one at a time for the prediction of software defects. Consequently, software metrics involved from phase to phase are considered in our model that lessens the required number of fuzzy rules.

Fuzzy inference system evaluates and combines the result of each fuzzy rule. Fuzzy inference engine maps fuzzy set into a fuzzy set. A fuzzy max–min operator is used for this step. In many applications, crisp value needs to be obtained as an output. The centroid defuzzification method is used in this research work in order to calculate the value of *z*′ in the proposed model. This method is the most common and physically appealing of all the defuzzification methods as found in [13].

Defect density indicator value is obtained using fuzzy inference tool of MATLAB at the end of requirement analysis phase, design phase coding phase and testing phase. There exist an approximately linear relationship between software size and number of defects [15,16].

1. If (Loc is L) and (iv(G) is L) and (I is L) and (IOComment is L) and (IOBlank is L) and (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is NON-DEFECT) (1)

2. If (Loc is M) and (iv(G) is L) and (I is L) and (IOComment is L) and (IOBlank is L) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

3. If (Loc is H) and (iv(G) is L) and (I is L) and (IOComment is L) and (IOBlank is L) (Unique_OP is L) and (Unique_OPND)then (SOFTWARE_DEFECT is DEFECT) (1)

4. If (Loc is L) and (iv(G) is M) and (I is M) and (IOComment is M) and (IOBlank is M) (Unique_OP is L) and (Unique_OPND)then (SOFTWARE_DEFECT is DEFECT) (1)

5. If (Loc is M) and (iv(G) is M) and (I is M) and (IOComment is M) and (IOBlank is M) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

6. If (Loc is H) and (iv(G) is H) and (I is H) and (IOComment is H) and (IOBlank is H) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

7. If (Loc is M) and (iv(G) is H) and (I is H) and (IOComment is H) and (IOBlank is H) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

8. If (Loc is M) and (iv(G) is M) and (I is H) and (IOComment is M) and (IOBlank is H) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

9. If (Loc is L) and (iv(G) is H) and (I is H) and (IOComment is H) and (IOBlank is M) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

10. If (Loc is M) and (iv(G) is M) and (I is L) and (IOComment is M) and (IOBlank is L) (Unique_OP is L) and (Unique_OPND) then (SOFTWARE_DEFECT is DEFECT) (1)

Experimental Result

This proposed method is implemented using matlab. The dataset is collected form the NASA repository. This work used four different NASA dataset namely CM1, JM1, KC1, PC1. The similarity among the reduced features is find using the intuitionistic fuzzy K-Medoids and from the generated clusters of data the rules are generated.

Evaluation Metric

For the comparison result three parameters are used and they are accuracy, precision and recall and their calculations are as follows

- Accuracy = (TP + TN) / (TP + TN + FP + FN) = #correct / #all_instances
- Precision = TP / (TP + FP) = #correct_positive / #classified_as_positive
- Recall = TP/ (TP + FP) = # correct_positive / #classified_as_correct_positive

Table 1: Attribute Description of the four Dataset

S.No	Variables	Description
1	Loc	McCabe's line count of code
2	v(g)	McCabe "cyclomatic complexity"
3	ev(g)	McCabe "essential complexity"
4	iv(g)	McCabe "design complexity"
5	N	Halstead total operators + operands
6	V	Halstead "volume"
7	L	Halstead "program length"
8	D	Halstead "difficulty"
9	I	Halstead "intelligence"
10	E	Halstead "effort"
11	B	Halstead
12	T	Halstead's time estimator
13	lOCode	Halstead's line count
14	lOComment	Halstead's count of lines of comments
15	lOBlank	Halstead's count of blank lines
16	lOCodeAndComment	
17	uniq_Op	unique operators
18	uniq_Opnd	unique operands
19	total_Op	total operators
20	total_Opnd	total operands
21	branchCount	% of the flow graph
22	Defects	Yes/No module has/has not one or more

Table 2: Performance Evaluation of the proposed Fuzzy K-Mediods with K-means and K-Medoids

	K-Medoids	Fuzzy K-Medoids	Intuitionistic Fuzzy K-medoids
Correctly Clustered Instance	300	305	420
Incorrectly Clustered Instance	198	193	78
Accuracy	69%	81%	90%
Precision	75%	87%	94%
Recall	72%	86%	92%

The table shows the performance of the proposed intuitionistic fuzzy k-medoids clustering method for finding the similarity pattern of the software defect analysis with the existing approaches namely K-Medoids and Fuzzy K-Medoids. The simulation result analysis are done based on the metrics like correctly clustered instances, incorrectly clustered instances, accuracy, precision and recall of the each methods.

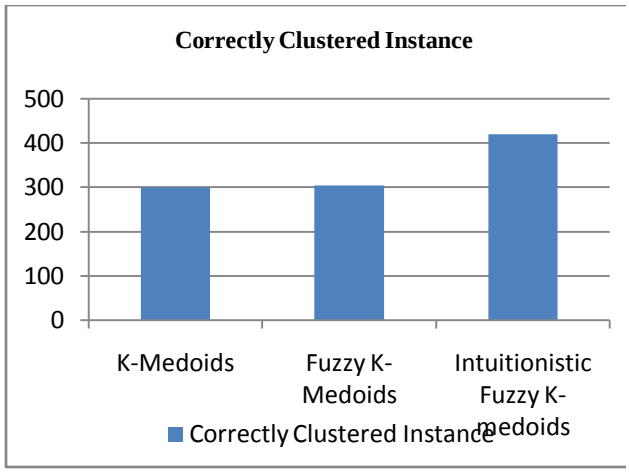

Fig 2: Performance comparison of the clustering methods based on correctly clustered instances

The chart depicts the performance of the methods k-medoids, Fuzzy K-Medoids, Intuitionistic fuzzy K – medoids based on how they determine the number of instances classified in correct cluster. The performance of the intuitionistic fuzzy based K-Medoids is exhibiting high correctly classified instances because of its ability to handle the uncertainty by using the degree of hesitation.

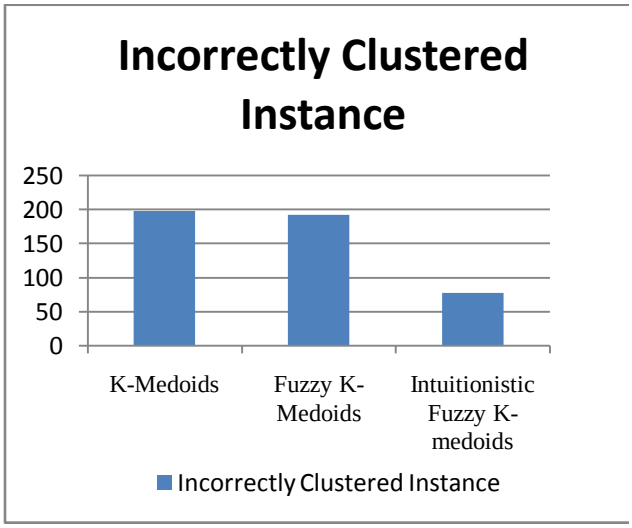

Fig 3: Performance comparison of the clustering methods based on incorrectly clustered instances

The chart shows the incorrectly clustered instances the performance of the intuitionistic fuzzy k-medoids produces less number of false clustering of instances while comparing the fuzzy K-Medoids and K-Medoids. Because both the existing approaches fails to handle the instances while they hold vague information or lie on the boundary of the cluster.

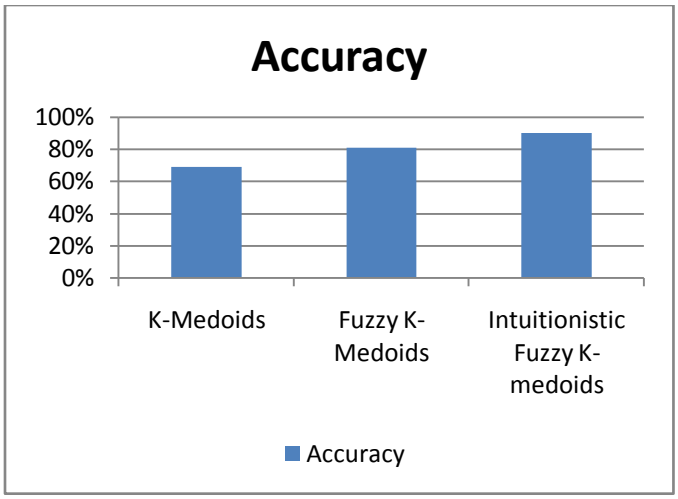

Fig 4: Performance comparison of the clustering methods based on accuracy

The figure shows the performance of the three methods K-Medoids, Fuzzy-K Medoids, Intuitionistic Fuzzy K-Medoids in which the proposed work accuracy is much better than other two existing approaches this is due to the fact that Intuitionistic fuzzy Medoids well-handled the problem of uncertainty in clustering the instances with the help of degree of non-membership and hesitation while it is failed by the other two methods.

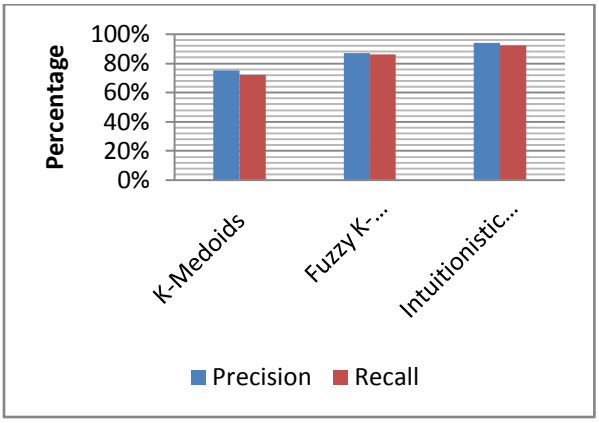

Fig 5: Performance Analysis of the clustering methods based on Precision and Recall

The precision and recall value of each methods are shown in the figure where the performance of the intuitionistic fuzzy logic is better than the other two existing approaches namely K-Medoids, Fuzzy K-Medoids.

Conclusion

In this paper, the performance of the Intuitionistic Fuzzy K-Medoids based Clustering of Applications is

explored for level of software defect prediction. The result shows that the performance of proposed method is better than the existing fuzzy K-medoids for the four different dataset collected from the NASA repository. The reduced features are used for clustering and the fuzzy logic based rule generation is done for performing the classification accuracy. Most of the cases the problem of uncertainty or vagueness in deciding about the instances to which they belong is common. Such kind of instances has to be handled in a special way because the misleading of such instances to wrong decision may increase the rate of false alarm so this proposed Intuitionistic fuzzy logic overwhelms this problem by introducing the degree of hesitation for handling the special case of instances very precisely.

References

1. G. Graham, E.V. Veenendaal, I. Evans, R. Black, "Foundations of Software Testing: ISTQB Certification", Thomson Learning, United Kingdom, 2007.
2. N.E. Fenton, M. Neil, "A Critique of Software Defect Prediction Models", IEEE Transactions on Software Engineering, vol. 25, no.5, pp.675-689, 1999.
3. B. Clark, D. Zubrow, "How Good is the Software: A Review of Defect Prediction Techniques", Carnegie Mellon University, USA, 2001.
4. V. Nayak, D. Naidya, "Defect Estimation Strategies", Patni Computer Systems Limited, Mumbai, 2003.
5. M. Thangarajan, B. Biswas, "Software Reliability Prediction Model", Tata Elxsi Whitepaper, 2002.
6. D. Wahyudin, A. Schatten, D. Winkler, A.M. Tjoa, S. Biffl, "Defect Prediction using Combined Product and Project Metrics: A Case Study from the Open Source "Apache" MyFaces Project Family" In Proceedings of Software Engineering and Advanced Applications (SEAA '08), 34th Euromicro Conference, pp. 207-215, 2008.
7. Sinovcic, L. Hribar, "How to Improve Software Development Process using Mathematical Models for Quality Prediction and Element of Six Sigma Methodology", In Proceedings of the 33rd International Conventionions 2010 (MIPRO 2010), pp. 388-395, 2010.
8. E.J. Weyuker, T.J. Ostrand, R.M. Bell, "Using Developer Information as a Factor for Fault Prediction", In Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), pp.8, 2007.
9. T. Gyimothy, R. Ferenc, I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction", IEEE Transactions on Software Engineering, vol. 31, no.10, pp. 897-910, 2005.
10. J.S. Collofello, "Simulating the System Test Phase of the Software Development Life Cycle", In Proceedings of the 2002 Summer Software Computer Simulation Conference, 2002.
11. L. RadliRski, "Predicting Defect Type in Software Projects", Polish Journal of Environmental Studies, vol.18, no. 3B, pp. 311-315, 2009.
12. M. Staron, W. Meding, "Defect Inflow Prediction in Large Software Projects", e-Informatica Software Engineering Journal, vol. 4, no. 1, pp. 1-23, 2010.
13. T. Fehlmann, "Defect Density Prediction with Six Sigma", Presentation in Software Measurement European Forum, 2009.
14. S.W. Haider, J.W. Cangussu, K.M.L. Cooper, R. Dantu, "Estimation of Defects Based on Defect Decay Model: ED3M", IEEE Transactions on Software Engineering, vol. 34, no. 3, pp. 336-356, 2008.
15. L. Zawadski, T. Orlova, "Building and Using a Defect Prediction Model", Presentation in Chicago Software Process Improvement Network, 2012.
16. M. Li, H. Zhang, R. Wu, Z.H. Zhou, "Sample-based Software Defect Prediction with Active and Semi-supervised Learning", Journal of Automated Software Engineering, vol. 19, no. 2, pp. 201-230, 2012.
17. Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, "An Investigation on the Feasibility of Cross-Project Defect Prediction", Journal of Automated Software Engineering, vol. 19, no. 2, pp. 167-199, 2012.
18. T.J. Ostrand, E.J. Weyuker, "How to Measure Success of Fault Prediction Models", In Proceedings of Fourth International Workshop on Software Quality Assurance 2007 (SOQUA '07), pp. 25-30, 2007.
19. L.P. Li, M. Shaw, J. Herbsleb, "Selecting a Defect Prediction Model for Maintenance Resource Planning and Software Insurance", In Proceedings of 5th Workshop on Economics-Driven Software Engineering Research (EDSER '03), pp. 32-37, 2003.
20. M. D'Ambros, M. Lanza, R. Robbes, "Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison, Journal of Empirical Software Engineering, vol. 17, no. 4-5, pp. 531-577, 2012.
21. Kaufman, L. and Rousseeuw, P.J. (1987), Clustering by means of Medoids, in Statistical Data Analysis Based on the Norm and Related Methods, edited, North-Holland, 405–416.