Research Journal of Applied Sciences, Engineering and Technology 13(7): 544-554, 2016
DOI:10.19026/rjaset.13.3014
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: March 14, 2016 Accepted: June 25, 2016 Published: October 05, 2016

Research Article

Software Defect Prediction in Class Level Metric Aggregation Using Data Mining Techniques

Reddi Kiran Kumar and S.V. Achuta Rao
Department of Computer Science, Krishna University, Machilipatnam, India

Abstract: Aim of study software defect is a flaw, miscalculation, or failure, in a computer program or framework delivering an inappropriate or surprising result, or making it perform in an unintended way. Software Defect Prediction (SDP) finds defective modules in software. The final product ought to have as few defects as possible to create top notch software. Early software defects discovery prompts diminished development costs and rework effort and better software. Software metrics guarantee quantitative methods to survey software quality. Software metrics are helpful to software process and product metrics. Thus, a defect prediction study is critical to guarantee quality software and software metric aggregation. In this study, the efficiency of classifier for SDP is assessed. Diverse classifiers like Naïve Bayes, K Nearest Neighbor (KNN), C4.5 and Multilayer Perceptrons Neural Network (MLPNN) are assessed for SDP.

Keywords: C4.5 and Multilayer Perceptrons Neural Network (MLPNN), K Nearest Neighbor (KNN), Naïve Bayes, Software Defect Prediction (SDP), software metric

INTRODUCTION

Software metrics is characterized by the measurement of some property of a segment of software or its stipulations. Software metrics give quantitative methods to evaluating the software quality. Software metrics can be characterized as: The ceaseless use of estimation based strategies to the software development process and its products to supply meaningful and convenient Management Information (MI) together with the utilization of those procedures to enhance its products and that process (Verner and Tate, 1992). Software metrics are valuable to the software process and product metrics. Different grouping of software metrics are as per the following:

Software process metrics: Software process metrics includes measuring of properties of the development process and otherwise called administration metrics. These metrics incorporate the cost, exertion, reuse, methodology and advancement metrics. Likewise it decides the magnitude, period and number of errors found amid testing period of the Software Development Life Cycle (SDLC).

Software product metrics: Software product metrics includes measuring the properties of the software or otherwise called quality metrics. These metrics incorporate the trustworthiness, usability, functionality, performance, effectiveness, transportability, reusability, price, magnitude, difficulty and style metrics. These metrics measure the difficulty of the software design, magnitude or documentation made (Debbarma *et al*., 2013).

Software metrics are prized entity in the whole software life cycle. They give estimation to the software development, including software requirement documents, designs, programs and tests. Quick developments of large scaled software have advanced difficulty that makes the quality hard to control. The fruitful implementation of the control over software quality requires software metrics. The ideas of software metrics are lucid, comprehensible and well established and numerous metrics identified with the product quality have been created and utilized.

Process metrics, project metrics and product metrics (Honglei *et al*., 2009) are three kinds of software metrics. Process metrics chief goals are the time taken for the project, the expense to be paid and the kind of methodology utilized. Project metrics are utilised to check project position. It explains the project features and implementation. Product metrics explains the characteristics of the software project at all times of its progress (Rawat *et al*., 2012).

Software metrics has been utilized to depict the difficulty of the program and, to evaluate software development time. "How to predict the quality of software through software metrics, before it is being

Corresponding Author: Reddi Kiran Kumar, Department of Computer Science, Krishna University, Machilipatnam, India

deployed" is an important inquiry, setting off the substantial research endeavours to reveal a response to this inquiry. Many papers support statistical models and metrics which try to answer the inquiry. Commonly, software metrics clarify quantitative estimations of the software product or its specifications (Rawat and Dubey, 2012).

Software Quality Assurance (SQA) is the arrangement of activities that guarantee that a software framework meets a particular quality level. Associations are constantly inspired by approaches to gauge the quality of their software before it is discharged. To encourage these organizational interests, specialists have proposed a huge number of quality measures and constructed statistical models, which influence these measures, to foresee defect-prone areas of software. For instance, some earlier work concentrated on anticipating files that contain one or more post-release or security defects (Fenton and Neil, 1999). The line of examination concerned with building such expectation models is called SDP.

Specifically, SDP can be characterized as the identification of software areas (i.e., subsystems, files or functions) that quality affirmation endeavours ought to concentrate on (i.e., audit or test). Including SDP in the development process offers experts in the decision-making process indicating which areas have a high risk (i.e., high anticipated probability) of having a defect. Now, the restricted validation and verification endeavours can becentred on these areas (Arisholm *et al*., 2010).

It is imperative to take note of that there exist a wide range of ways to accomplish high software quality. SDP is one methodology; it is positively not by any means the only approach. For instance, a lot of exploration work uses model checking and static examination to discover defects in software frameworks. Other work concentrates on fault localization, in which contrasts between the inputs of failing and passing tests are utilized to find errors in the source code. The primary distinction between these different lines of examination and SDP is that they recognize defects in the present code base (i.e., the code base being dissected). SDP warns about future defect-inclined zones.

The aims of SDP are:

- Defect forecast enhances efficiency of the testing stage notwithstanding offering engineers some assistance with evaluating the quality and defect proneness of their software product.
- Help managers in allotting resources, rescheduling, training plans and budget distributions.
- Depending on the forecasted trends, resources can be proficiently increased or decreased and Gaps in Skills and trainings can be stopped.
- Predicts defect spillage into production (Umar, 2013).

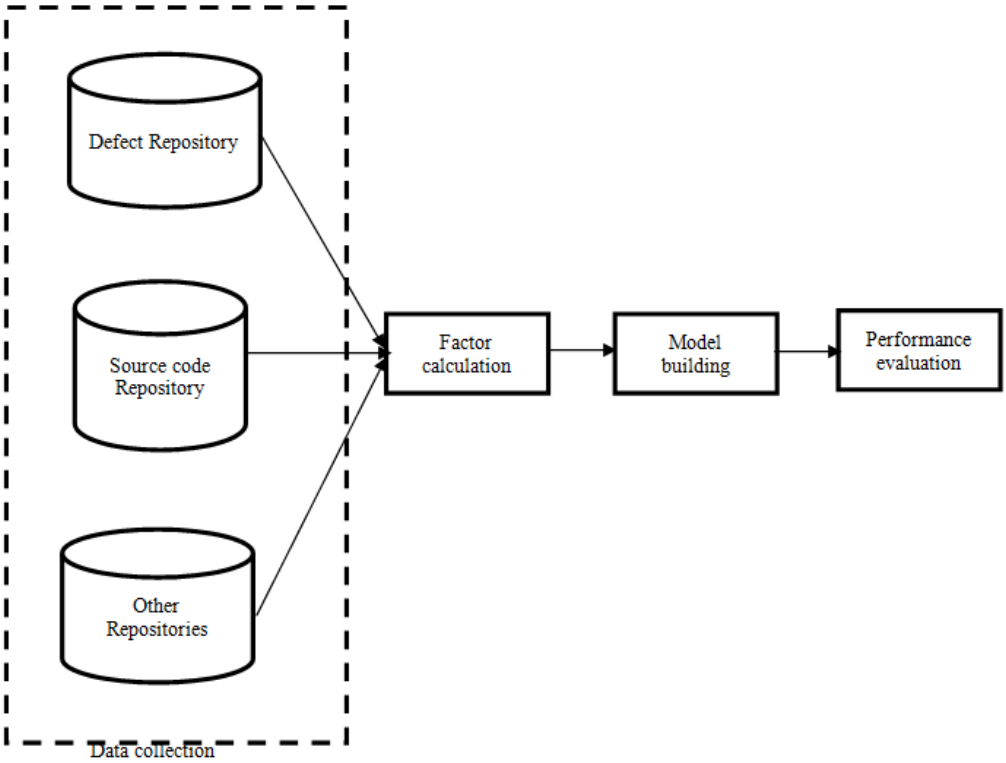

Fig. 1: Overview of software defect prediction (SDP)

Most SDP studies plan to accurately classify software artifacts (e.g., subsystems or files) as being fault-inclined or not. Other SDP studies are occupied with anticipating the quantity of defects that may show up in software objects, so they can be ranked. Figure 1 demonstrates a review of the SDP process. To begin with, project data is gathered from software repositories (e.g., defect and source control repositories). At that point, factors are ascertained from the data. Statistical and machine learning models are assembled to foresee the areas that have a high capability of containing defects. At last, the prediction models are assessed utilizing different measures, for example, precision, recall and explanative power (Shihab, 2012).

There has been a broad body of work that concentrated on SDP. Each of these works utilized its own particular special data, dependent and independent variables and modelling methods and assessed their models in diverse ways. Subsequently, there is a need to thoroughly analyze the former work, to better understand the assumptions and ramifications of the work:

- **Data sources and granularity:** It reports on the sources and the granularity of the data utilized as a part of earlier SDP research.
- **Factors:** It reports on the factors utilized as a part of SDP studies.
- **Models:** It reports on the models utilized as a part of SDP studies.
- **Performance assessment:** It reports on the diverse performance assessment methods used to assess the SDP models.

Software support is a region of software engineering with profound money related ramifications. In fact, it was seen that somewhere around 60% and 90% of the software budgets represent maintenance and evolution costs. Moreover, maintenance and evolution costs were estimated to represent more than half of North American and European software budgets in 2010. Considerably higher figures were accounted for some nations, for example, Norway and Chile. Controlling software support costs requires anticipating how the framework will develop later on, which requires a superior understanding of software evolution (Vasilescu *et al*., 2011).

A prominent way to deal with surveying software viability and foreseeing its advancement includes performing estimations on code artifacts. It begins off by distinguishing various particular properties of the framework under scrutiny and then gathering the relating software metrics and analyzing their evolution.

In any case, metrics are typically characterized at micro level (method, class, package), while the analysis of maintainability and evolution requires knowledge at macro (framework) level. Also, because of security reasons, it may be undesirable to disclose metrics relating to a solitary designer rather than those relating to the whole project. Metrics ought to be aggregated.

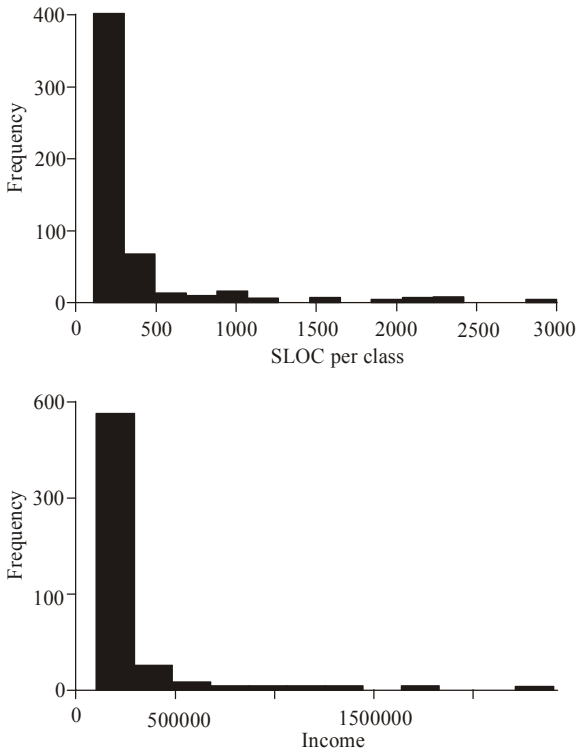

Fig. 2: Software metrics (SLOC) and econometric variables (household income in the Ilocos region, the Philippines) have distributions with similar shapes

Famous aggregation systems incorporate standard summary statistical measures as mean, median, or sum. Their benefit is universality (metrics-independence): whatever metrics are viewed as, the measures ought to be computed similarly. In any case, as the distribution of numerous software metrics is skewed, the interpretation of such measures gets to be inconsistent.

On the other hand, distribution fitting comprises of selecting a known group of distributions (e.g., log-normal or exponential) and fitting its parameters to estimate the metric qualities seen. The fitted parameters can be then seen as totaling these values. Though, the fitting process ought to be repeated at whatever point another metric is being considered. Besides, it is still a matter of contention whether, e.g., software size is distributed log-normally or doubles Pareto. It doesn't consider distribution fitting.

Recently, there is a developing pattern in utilizing more advanced aggregation procedures acquired from econometrics, where they are utilized to study inequality of pay or welfare distributions. The inspiration for applying such systems to software metrics is twofold. In the first place, as various nations have couple of rich and numerous poor, various software frameworks have few major or complex parts and numerous little or simple ones. Subsequently, it is normal both for software metrics and econometric variables to have strongly-skewed distributions (Fig. 2).

Second, the state of these distributions, which show to take after a power law, renders the utilization of

customary aggregation systems, for example, the sample mean and variance flawed. It was seen that numerous vital relationships between software artifacts take after a power law distribution and it is realized that a power law distribution might not have a finite mean and variance. These understanding prompted the utilization of econometric strategies to aggregation of software metrics and to the present interest for these aggregation methods.

In this study, the SDP in class level metric aggregation using data mining techniques and the efficiency of the classifiers to classify defective modules is evaluated. KC1 dataset from the PROMISE software dataset repository is used for evaluation.

LITERATURE SURVEY

Wang and Yao (2013) examined the issue of if and how class imbalance learning methods can be of advantage to SDP for finding better solutions. Distinctive sorts of class imbalance learning methods has been investigated, including resampling systems, threshold moving and ensemble algorithms. Among the methods contemplated, AdaBoost.NC demonstrates the best general performance regarding the measures including balance, G-mean and Area under the Curve (AUC). To further enhance the performance of the algorithm and facilitate its utilization in SDP, it proposed a dynamic rendition of AdaBoost.NC, which modifies its parameter consequently during training. Without the need to pre-define any parameters, it is more successful and effective than the original AdaBoost.NC.

Pelayo and Dick (2012) investigated two noteworthy stratification options (under- and over-sampling) for SDP using Analysis of Variance. The analysis covers a few cutting edge SDP datasets using a factorial design. The main impact of under-sampling is significant at a = 0.05 as the interaction in the middle of under-and over-sampling. In any case, the main impact of over-sampling was not significant.

Rubinić *et al*. (2015) reported preliminary results obtained by using a Matlab variation of NSGA-II in combination with four straightforward voting methodologies on three consequent arrivals of the Eclipse Plug-in Development Environment (PDE) project. Preliminary results indicated that the voting system may influence SDP performances.

Fehlmann and Kranich (2014) concentrated on the predictive property of Exponentially Weighted Moving Average (EWMA) Q control diagrams and investigated whether the predictive property was smart for monitoring and controlling the SDP. Results of initial trials are reported.

Can *et al*. (2013) proposed a SDP model using Particle Swarm Optimization (PSO) and Support Vector Machine (SVM) named P-SVM model. The creators used PSO algorithm to compute best SVM parameters and adopts optimized SVM model to predict software defect. P-SVM model and three other prediction models predict software defects in JM1 data set on a test premise, the outcomes showing that P-SVM has higher prediction accuracy contrasted with BP Neural Network model, SVM and GA-SVM models.

Khan *et al*. (2014) proposed a procedure to choose best attributes set to enhance SDP accuracy. Software quality attributes influence the defect prediction model's performance and effectiveness. The new method is assessed using NASA metric data repository data sets and exhibits good accuracy using basic algorithm.

The best size of feature subset to assemble a prediction model, to be demonstrated that feature selection builds up SDP model was deliberated about by Wang *et al*. (2012b). Mutual information is an outstanding importance indicator among variables and utilized as a part of the new feature selection algorithm. A nonlinear factor for assessment capacity was introduced for feature selection to enhance performance. Consequences of the feature selection algorithm were accepted by varied machine learning methods. Test results have shown that all classifiers accomplished high accuracy.

Wang *et al*. (2012a) proposed three new defect prediction models based on C4.5 model. Spearman's rank correlation coefficient was introduced to pick root node of decision tree which enhances models on defects prediction. An exploratory plan was implemented to check the enhanced models effectiveness and this paper analyzed prediction accuracies of existing and enhanced models. Results demonstrated that enhanced models decreased decision tree size by 49.91% by and large and increased prediction accuracy by 4.58% and 4.87% on two modules in the investigation.

Ma *et al*. (2012) proposed a algorithm called Transfer Naive Bayes (TNB), which utilized information of all training data features. This arrangement estimates test data distribution and transfers cross-company data information to training data weights. The defect prediction model is based on the weighted data. This article exhibits a theoretical analysis of relative methods, showing data sets' test results from different associations. It indicates that TNB is more precise regarding AUC, with lessened runtime than other methods.

Najadat and Alsmadi (2012) proposed a novelr model in view of Ridor algorithm to predict fault in modules. They additionally tested the distinctive classification methods on the data sets given by NASA. The outcomes demonstrated that Ridor algorithm was superior to the existing method as far as accuracy and extraction of number of rules.

Oliveira *et al*. (2014) proposed the idea of relative thresholds for evaluating metrics data following heavy-tailed distributions. The proposed thresholds are relative on the grounds that they expect that metric thresholds ought to be followed by most source code elements, yet that it was additionally normal to have been number of elements in the "long-tail" that don't take after as far as possible. The creators depicted an experimental method for extracting relative thresholds from genuine frameworks. It likewise reports a study on applying this method in a corpus with 106 frameworks.

Finlay *et al*. (2014) investigated the ideas of representing a software development project as a procedure that results in the making of a data stream. Protsenko and Müller (2014) proposed another methodology for the static identification of Android malware by means of machine learning that depends on software complexity metrics, for example, McCabe's Cyclomatic Complexity and the Chidamber and Kemerer Metrics Suite.

Khoshgoftaar *et al*. (2014) proposed an iterative feature selection approach, which repeated data sampling (with a specific end goal to address class imbalance) and then by feature selection (to address high-dimensionality), lastly it performed an accumulation step which combines the ranked feature records from the different cycles of sampling. This methodology was intended to find a ranked feature list which was especially compelling on the more balanced dataset resulting from sampling while minimizing the danger of losing data through the sampling step and missing imperative features.

Xia *et al*. (2013) used different feature selection and dimensionality reduction ways to determine most essential software metrics. Three distinct classifiers, Naive Bayes, SVM and decision tree were utilized. On the NASA data, similar examination results demonstrate that instead of 22 or more metrics, under 10 metrics guarantee better performance.

METHODOLOGY

In this section, KCI dataset are used. The Naïve Bayes, KNN, C4.5 and MLPNN are used for SDP.

KC1 dataset: KC1 dataset is a NASA Metrics Data Program checking/enhancing predictive software engineering models. KC1 is a C++ framework executing storage management for ground data receipt/preparing containing of McCabe and Halstead features code extractors and module based measures (Selvaraj and Thangaraj, 2013).

Defect detectors are computed (Table 1): Accuracy, identification probability (pd) or recall, precision (prec), probability of false alarm (pf) and effort are figured as:

Table 1: Defect detectors

Detection	Prediction of classifier
A	No defects and module has no error
B	No defects and module has error
C	A few defects and module has no error
D	A few defects and module has error

$$Accuracy = \frac{a+d}{a+b+c+d} \quad (1)$$

$$recall = \frac{d}{b+d} \quad (2)$$

$$pf = \frac{c}{a+c} \quad (3)$$

$$prec = \frac{d}{c+d} \quad (4)$$

$$effort = \frac{c.LOC + d.LOC}{TotalLOC} \quad (5)$$

KC1 dataset has 2109 instances and 22 shifted traits including 5 distinctive LOC, 3 McCabe metrics, 12 Halstead metrics, a branch count and 1 objective field. Dataset's characteristic data is: all out operands, outline complexity, McCabe's Line count of Code (LOC), Cyclomatic complexity, program length, effort, Halstead, class and others.
Examples from dataset:

Example 1 - 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, false
Example 2 - 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, true
Example 3 - 83, 11, 1, 11, 171, 927.89, 0.04, 23.04, 40.27, 21378.61, 0.31, 1187.7,65, 10, 6, 0, 18, 25, 107, 64, 21, true

Naïve bayes classifier: Bayesian Classifiers are statistical classifiers. The probability that a given record in the dataset has a place with a specific class can be anticipated with the assistance of Bayesian classifiers, utilizing the class enrollment probabilities. Guileless Bayes classification depends on Baye's theorem. Here the ideal tenets were given as information to the gullible bayes classifier. This sort of classifier has the point of preference that it is anything but difficult to execute and produce great results. Studies looking at classification algorithms have observed innocent Bayesian classifier to be similar in execution with decision tree and chose neural network classifiers (Kumaresh and Baskaran, 2015).

Bayes theorem gives a method for figuring the posterior probability, P (c | x), from P(c), P(x) and P (x | c). Innocent Bayes classifier accepts that the impact of

the estimation of an indicator (x) on a given class (c) is autonomous of the estimations of different indicators. This assumption is called class conditional independence.

The posterior probability can be ascertained by in the first place, building a frequency table for every trait against the objective. At that point, changing the frequency tables to probability tables and at last uses the Naive Bayesian mathematical statement to ascertain the posterior probability for every class. The class with the most elevated posterior probability is the result of expectation.

Consider a supervised learning issue in which it wish to inexact an obscure target function $f : X \to Y$, or proportionally P(Y|X). To start, it will accept Y is a boolean-esteemed random variable and X is a vector containing n Boolean properties. As it were, $X =< X_1, X_2, ..., X_n >$, where Xi is the boolean random variable meaning the ith quality of X (Mitchell, 2006).

Applying Bayes principle, $P(\mathrm{Y} = \mathrm{y}_i \mid \mathrm{X})$ that can be spoken to as:

$$P(Y = y_i \mid X = x_k) = \frac{P(X = x_k \mid Y = y_i)P(Y = y_i)}{\sum_j P(X = x_k \mid Y = y_i)P(Y = y_i)} \quad (6)$$

where, y_m means the m^{th} conceivable quality for Y, x_k signifies the kth conceivable vector esteem for X and where the summation in the denominator is over every lawful estimation of the random variable Y.

One approach to learn P(Y|X) is to utilize the preparation data to gauge P(Y|X) and P(Y). It can then utilize these appraisals, together with Bayes principle above, to decide $P(\mathrm{Y} \mid \mathrm{X} = \mathrm{x}_k)$ for any new instance x_k.

The Naive Bayes algorithm is a classification algorithm in view of Bayes decide that accept the qualities $X_1 .....X_n$ are all conditionally autonomous of each other, given Y. The estimation of this assumption is that it significantly improves the representation of P(X|Y) and the issue of assessing it from the preparation data. Consider, for instance, the situation where $X =< X_1, X_2 >$. For this situation:

$$\begin{aligned} P(X \mid Y) &= P(X_1, X_2 \mid Y) \\ &= P(X_1 \mid X_2, Y)P(X_2 \mid Y) \\ &= P(X_1 \mid Y)P(X_2 \mid Y) \end{aligned} \quad (7)$$

where the second line takes after from a general property of probabilities and the third line takes after straightforwardly from the above meaning of conditional independence.

K-Nearest Neighbor (KNN): The KNN algorithm is a strategy for classifying items taking into account nearest preparing data in the feature space. KNN is a kind of instance-based learning. The KNN algorithm is amongst the least difficult of all machine learning algorithms. Yet, the accuracy of the KNN algorithm can be seriously corrupted by the vicinity of loud or unessential features, or if the feature scales are not reliable with their significance (Soni *et al*., 2011).

Assume every example in the data set has n ascribes which it consolidates to frame a n-dimensional vector:

$$X = (x_1, x_2, ..., x_n)$$

These n credits are thought to be the autonomous variables.

Every specimen additionally has another property, signified by y (the indigent variable), whose quality relies on upon the other n properties x. It accept that y is a straight out variable and there is a scalar function, f, which doles out a class, y = f(x) to each such vectors (Leung, 2007).

The thought in KNN techniques is to distinguish k tests in the preparation set whose free variables x are like u and to utilize these k tests to classify this new example into a class, v. In the event that all it are readied to expect is that f is a smooth function, a sensible thought is to search for tests in the preparation data that are close it (regarding the free variables) and then to process v from the estimations of y for these examples.

The Euclidean distance between the focuses x and u is:

$$d(x,u) = \sqrt{\sum_{i=1}^{n} (x_i - u_i)^2} \quad (8)$$

It will analyze different approaches to quantify distance between focuses in the space of free indicator variables when it talks about bunching strategies.

The easiest case is k = 1 where it discover the example in the preparation set that is nearest (the closest neighbor) to u and set v = y where y is the class of the closest neighbouring specimen.

C4.5 is classification algorithm used to produce a decision tree; this is an augmentation of Quinlan's ID3 algorithm. The decision trees delivered by C4.5 technique can be utilized for forecast, which is the reason C4.5 is often alluded to as a statistical classifier. C4.5 constructs decision trees from a preparation data as utilizing the idea of data entropy. The historical data or preparing data is a set of classified examples. In this strategy for classification at every node of the decision tree, C4.5 technique picks one characteristic of the

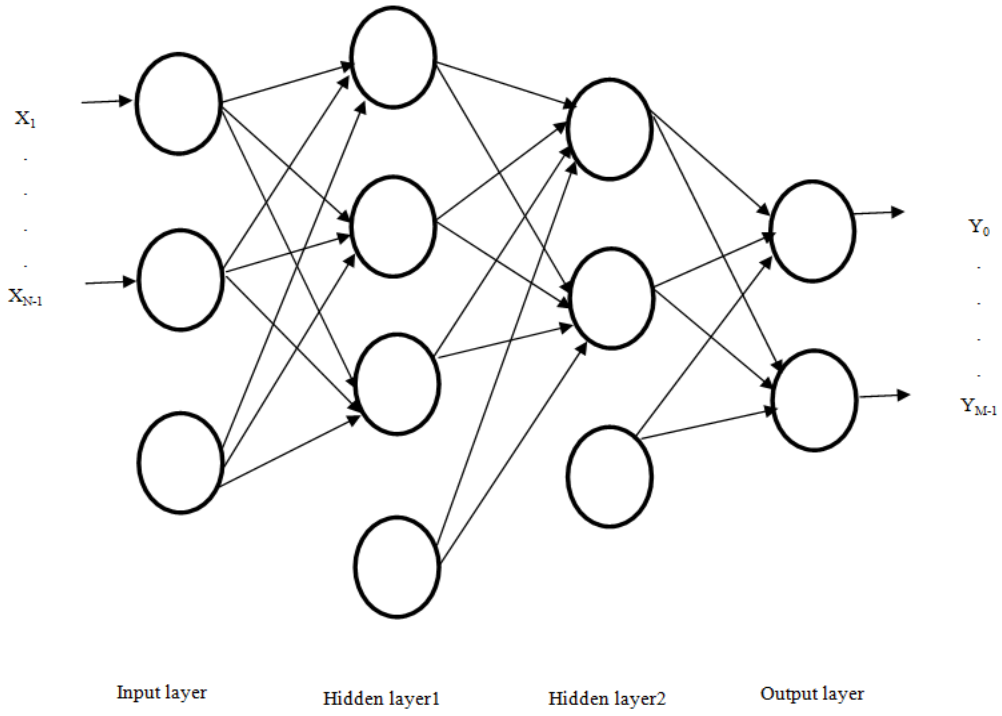

Fig. 3: Multilayer perceptrons architecture

preparation data that most adequately parts the data sets into subsets and this procedure is rehashed until all the datasets are not secured.

Decision tree rule is the normalized data pick up that outcome from picking a most suitable property for part the data sets. The characteristic which have the most astounding data increase is decided for settling on decision. The decision tree algorithm then recourses this procedure on the smaller split sub records. In building or preparing a decision tree it use preparing set or historical data. After decision tree created it can classify new data that have obscure class property estimation by evaluating the different probability values.

Decision trees classify software defective modules by utilizing a progression of tenet. The decision tree has fundamental segments, for example, the decision node, branches and clears out. Information space inside of decision tree is divided into mutually exclusive areas and a quality or an activity or a name is doled out to every district to portray its data focuses. The component of decision tree is straightforward and decision tree structure can be taken after to perceive how the decision is made. A large portion of the decision trees development algorithm comprises of two stages. In the first stage, change huge size tree is developed and then the tree is pruned in the second means to keep away from over fitting issue. At that point the pruned tree is used for classification reason (Han *et al*., 2011).

The algorithm builds a decision tree beginning from a preparation set T S, which is a set of cases, or tuples in the database wording. Every case indicates values for a gathering of characteristics and for a class. Every property may have either discrete or persistent qualities. In addition, the unique quality obscure is permitted, to indicate unspecified qualities. The class may have just discrete qualities. It indicatesC_1, ……., C_{NClass} with the estimations of the class.

A decision tree is a tree data structure comprising of decision nodes and takes off. A leaf determines class esteem. A decision node indicates a test more than one of the traits, which is known as the property chose at the node. For every conceivable result of the test, a youngster node is available. Specifically, the test on a discrete quality A has h conceivable results $A = d_1, ..., A = d_h$, where $d_1, ...d_h$ are the known qualities for property A. The test on a ceaseless property has two conceivable results $A \le t$ *and* $A > t$, where t is a quality decided at the node and called the edge (Ruggieri, 2002).

A decision tree is utilized to classify a case, i.e., to dole out a class worth to a case contingent upon the estimations of the traits of the case. Truth be told, way from the root to a leaf of the decision tree can be taken after in view of the property estimations of the case. The class determined at the leaf is the class anticipated by the decision tree. An execution measure of a decision tree over a set of cases is called classification error. It is characterized as the rate of misclassified cases, i.e., of cases whose anticipated classes vary from the genuine classes.

With a divide and conquers technique, the C4.5 algorithm develops the decision tree. In C4.5, every node in a tree is connected with a set of cases. Additionally, cases are allocated weights to consider obscure property estimations. Toward the starting, just the root is available, with related the entire preparing set T S and with all case weights equivalent to 1.0. At every node the accompanying divide and conquer algorithm (see Program 1) is executed, attempting to abuse the locally best decision, with no backtracking permitted.

Program 1: Pseudo-code of the C4.5 Tree-Construction Algorithm:

```
"FormTree(T)
    (1) ComputeClassFrequency(T);
    (2) if OneClass or FewCases
            return a leaf;
        create a decision node N;
    (3) ForEach Attribute A
            ComputeGain(A);
    (4) N.test = AttributeWithBestGain;
    (5) if N.test is continuous
            find Threshold;
    (6) ForEach T' in the splitting of T
    (7) if T' is Empty
            Child of N is a leaf
        else
    (8) Child of N = FormTree( T' );
    (9) ComputeErrors of N;
        return N"
```

Multi-Layer Perceptron Neural Networks (MLPNN): MLPNN is a kind of feed-forward neural network. The multilayer neural network structural engineering can have any number of layers. Figure 3 show a network structural engineering with 4 layers; first layer is called info layer and last layer is called yield layer; in the middle of first and last layers which are called hidden layers (Askari and Bardsiri, 2014; Joy, 2011). The quantity of cells every layer can have is controlled by experimentation system. In multilayer perceptron neural networks, every neuron in a layer is connected with the past layer's all neurons. Such networks are called 'completely related' networks. The information layer is transformer and an apparatus for setting up the data. The last layer-yield layer-incorporates the qualities anticipated by the network and registers model yield. The center layers-hidden layers-that are framed by number cruncher neurons are the place the data is handled. Network yield is acquired by:

$$Y_i = f_i\left(\sum_{j=1}^{n} X_j W_{ij} + b_i\right) \quad (9)$$

where, Yi speaks to network yield, Xi looks like network information, Wij is the association weights in the middle of data and yield nodes, Bi is the predisposition and Fi is the exchange function.

A MLP is a supervised learning approach and included feed forward artificial NN model. The sets of info data in this methodology map onto a set of fitting yields. A MLP included coordinated diagram of different layers of nodes and they are completely joined with the following one inside of every node. Every data node is called as neuron with a nonlinear enactment function. The sigmoidal units of hidden layer figure out how to inexact the functions. For preparing reason, MLP uses a strategy got back to Propagation (BP) (Alrajeh and Alzohairy, 2012).

MLPNN prepared with BP algorithm have been ended up being helpful in tackling a wide assortment of genuine issues in different spaces. Notwithstanding various augmentations and alterations, for example, the increasing speed of the meeting rate, uncommon learning standards and data representation plans, distinctive error functions, option exchange functions of the neurons, weight appropriation among others, to enhance the outcomes or to accomplish some required properties of the prepared networks, one key component i.e., the BP, still in light of the gradient descent algorithm to minimize the network error, has been hardly changed (Collobert and Bengio, 2004).

Typically a gradient descent algorithm is utilized to modify the NN weight by contrasting the objective and real network results when a set of inputs are presented in the network, yet in spite of its prevalence in the preparation of MLP, BP has a few disadvantages. It relies on upon the state of the error surface, the estimations of the randomly instated weights and some different parameters, that is, BP all that much relies on upon great, issue particular parameter settings. Additionally there is the inclination of the prepared neural network getting stuck in local minima.

RESULTS AND DISCUSSION

KC1 dataset is used for evaluating the efficiency of the various classifiers for identifying defects in the modules. In this section, the classification accuracy, precision, recall and F measure are presented from Table 2 and Fig. 4 to 7.

Table 2: Summary of results

Classifiers used	Classification accuracy %	Precision	Recall	F Measure
Naïve bayes	87.59	0.7862	0.8192	0.8009
K nearest neighbor	86.9	0.7759	0.8000	0.7869
C4.5	90.34	0.8315	0.8510	0.8407
Neural network	92.41	0.8677	0.8787	0.8731

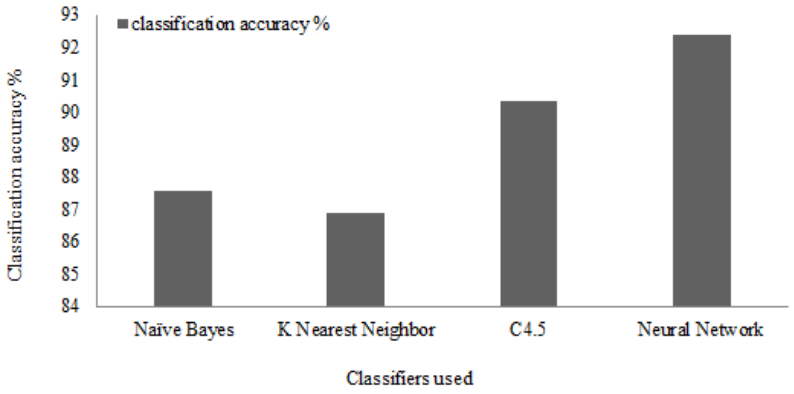

Fig. 4: Classification accuracy

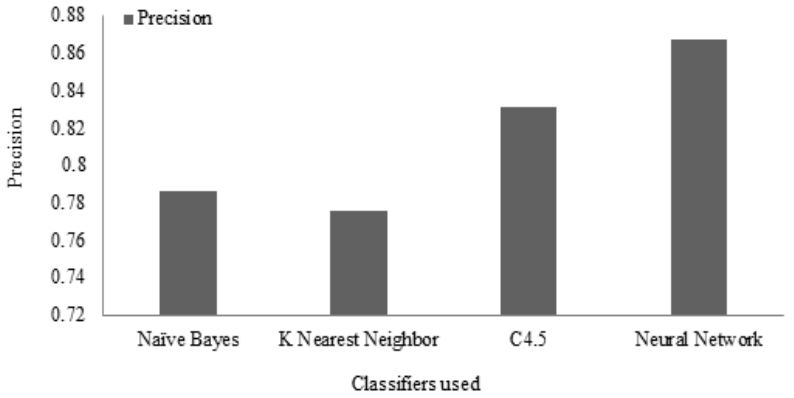

Fig. 5: Precision

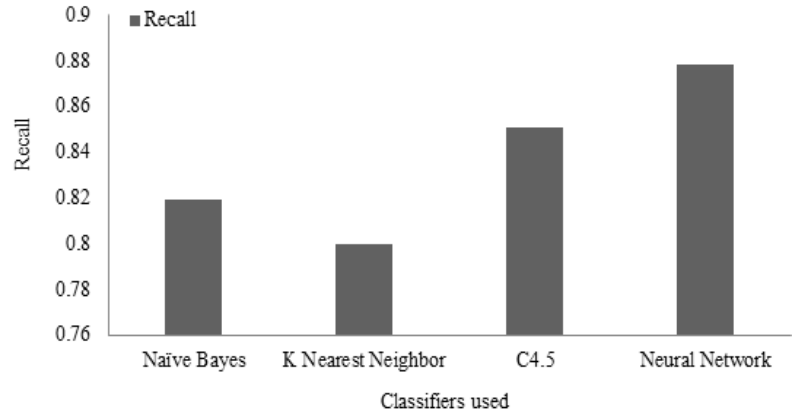

Fig. 6: Recall

From the Table 2 and Fig. 4, it can be observed that the Neural Network method improves classification accuracy by 5.35%, 6.14% and 2.26% when compared with the Naïve Bayes, K Nearest Neighbor and C4.5 methods.

From the Table 2 and Fig. 5, it can be observed that the Neural Network method increased precision by 9.85%, 11.17% and 4.26% when compared with the Naïve Bayes, K Nearest Neighbor and C4.5 methods.

From the Table 2 and Fig. 6, it can be observed that the Neural Network method increased recall by 7.00, 9.37 and 3.20% when compared with the Naïve Bayes, K Nearest Neighbor and C4.5 methods.

From the Table 2 and Fig. 7, it can be observed that the Neural Network method increased F Measure by 8.62, 10.38and 3.71% when compared with the Naïve Bayes, K Nearest Neighbor and C4.5 methods.

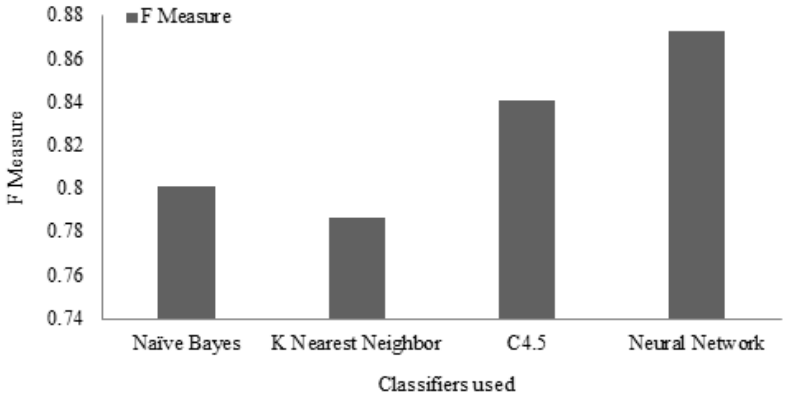

Fig. 7: F Measure

CONCLUSION

Software engineers need to distinguish defective software modules to enhance software metrics framework quality constantly. In this study, examinations are done for assessing the classification accuracy, precision, recall and f measure for classifiers like Naive Bayes, KNN, C4.5 and NN for defect prediction. The classifiers are assessed for KC1 dataset. Experimental results demonstrate that the NN are efficient for the prediction of defects. The classification accuracy of NN classifier performs better by 5.35, 6.14 and 2.26% when compared with the Naïve Bayes, KNN and C4.5 techniques. Likewise precision, recall and f measure is improved.

REFERENCES

Alrajeh, K.M. and T.A.A. Alzohairy, 2012. Date fruits classification using MLP and RBF neural networks. Int. J. Comput. Appl., 41(10): 36-41.

Arisholm, E., L.C. Briand and E.B. Johannessen, 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J. Syst. Software, 83(1): 2-17.

Askari, M.M. and V.K. Bardsiri, 2014. Software defect prediction using a high performance neural network. Int. J. Softw. Eng. Appl., 8(12): 177-188.

Can, H., X. Jianchun, Z. Ruide, L. Juelong, Y. Qiliang and X. Liqiang, 2013. A new model for software defect prediction using particle swarm optimization and support vector machine. Proceeding of the 25th IEEE Chinese Control and Decision Conference (CCDC, 2013), pp: 4106-4110.

Collobert, R. and S. Bengio, 2004. Links between perceptrons, MLPs and SVMs. Proceeding of the 21st International Conference on Machine Learning, pp: 23.

Debbarma, M.K., S. Debbarma, N. Debbarma, K. Chakma and A. Jamatia, 2013. A review and analysis of software complexity metrics in structural testing. Int. J. Comput. Commun. Eng., 2(2): 129-133.

Fehlmann, T. and E. Kranich, 2014. Exponentially Weighted Moving Average (EWMA) prediction in the software development process. Proceeding of the 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), pp: 263-270.

Fenton, N.E. and M. Neil, 1999. A critique of software defect prediction models. IEEE T. Software Eng., 25(2): 675-689.

Finlay, J., R. Pears and A.M. Connor, 2014. Data stream mining for predicting software build outcomes using source code metrics. Inform. Software Tech., 56(2): 183-198.

Han, J., M. Kamber and J. Pei, 2011. Data Mining: Concepts and Techniques. Elsevier, Amsterdam, pp: 1-13.

Honglei, T., S. Wei and Z. Yanan, 2009. The research on software metrics and software complexity metrics. Proceeding of the International Forum on Computer Science-Technology and Applications (IFCSTA'09), 1: 131-136.

Joy, C.U., 2011. Comparing the performance of backpropagation algorithm and genetic algorithms in pattern recognition problems. Int. J. Comput. Inf. Syst., 2(5).

Khan, J.I., A.U. Gias, M. Siddik, M. Rahman, S.M. Khaled and M. Shoyaib, 2014. An attribute selection process for software defect prediction. Proceeding of the International Conference on Informatics, Electronics and Vision (ICIEV, 2014), pp: 1-4.

Khoshgoftaar, T.M., K. Gao, A. Napolitano and R. Wald, 2014. A comparative study of iterative and non-iterative feature selection techniques for software defect prediction. Inform. Syst. Front., 16(5): 801-822.

Kumaresh, S. and R. Baskaran, 2015. Knowledge discovery from unstructured software defect reports using text mining. Int. J. Appl. Eng. Res., 10(2): 1243-1245.

Leung, K.M., 2007. k-Nearest neighbor algorithm for classification. Department of Computer Science/Finance and Risk Engineering, Polytechnic University, pp: 1-17.

Ma, Y., G. Luo, X. Zeng and A. Chen, 2012. Transfer learning for cross-company software defect prediction. Inform. Software Tech., 54(3): 248-256.

Mitchell, T.M., 2006. The discipline of machine learning. Machine Learning Department, School of Computer Science, Carnegie Mellon University, pp: 9.

Najadat, H. and I. Alsmadi, 2012. Enhance rule based detection for software fault prone modules. Int. J. Softw. Eng. Appl., 6(1): 75-86.

Oliveira, P., M.T. Valente and F. Paim Lima, 2014. Extracting relative thresholds for source code metrics. Proceeding of the 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE, 2014), pp: 254-263.

Pelayo, L. and S. Dick, 2012. Evaluating stratification alternatives to improve software defect prediction. IEEE T. Reliab., 61(2): 516-525.

Protsenko, M. and T. Müller, 2014. Android Malware Detection Based on Software Complexity Metrics. In: Eckert, C. *et al*. (Eds.), Trust, Privacy, and Security in Digital Business. Lecture Notes in Computer Science, Springer International Publishing, Switzerland, 8647: 24-35.

Rawat, M.S. and S.K. Dubey, 2012. Software defect prediction models for quality improvement: A literature study. Int. J. Comput. Sci. Issues, 9(5): 288-296.

Rawat, M.S., A. Mittal and S.K. Dubey, 2012. Survey on impact of software metrics on software quality. IJACSA Int. J. Adv. Comput. Sci. Appl., 3(1): 137-141.

Rubinić, E., G. Mauša and T.G. Grbac, 2015. Software Defect Classification with a Variant of NSGA-II and Simple Voting Strategies. In: Barros, M. and Y. Labiche (Eds.), Search-Based Software Engineering. Lecture Notes in Computer Science Springer International Publishing, Switzerland, 9275: 347-353.

Ruggieri, S., 2002. Efficient C4.5 [classification algorithm]. IEEE T. Knowl. Data En., 14(2): 438-444.

Selvaraj, P.A. and P. Thangaraj, 2013. Support vector machine for software defect prediction. Int. J. Eng. Technol. Res., 1(2): 68-76.

Shihab, E., 2012. An exploration of challenges limiting pragmatic software defect prediction. Ph.D. Thesis, Queen's University.

Soni, J., U. Ansari, D. Sharma and S. Soni, 2011. Predictive data mining for medical diagnosis: An overview of heart disease prediction. Int. J. Comput. Appl., 17(8): 43-48.

Umar, S.N., 2013. Software testing defect prediction model-a practical approach. Int. J. Res. Eng. Technol. (IJRET), 2(5): 741-745.

Vasilescu, B., A. Serebrenik and M. van den Brand, 2011. You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics. Proceeding of 27th IEEE International Conference on Software Maintenance (ICSM, 2011), pp: 313-322.

Verner, J. and G. Tate, 1992. A software size model. IEEE T. Software Eng., 18(4): 265-278.

Wang, J., B. Shen and Y. Chen, 2012a. Compressed C4. 5 models for software defect prediction. Proceeding of the 12th International Conference on Quality Software (QSIC), pp: 13-16.

Wang, P., C. Jin and S.W. Jin, 2012b. Software defect prediction scheme based on feature selection. Proceeding of the International Symposium on Information Science and Engineering (ISISE, 2012), pp: 477-480.

Wang, S. and X. Yao, 2013. Using class imbalance learning for software defect prediction. IEEE T. Reliab., 62(2): 434-443.

Xia, Y., G. Yan and Q. Si, 2013. A study on the significance of software metrics in defect prediction. Proceeding of the 6th International Symposium on Computational Intelligence and Design (ISCID, 2013), 2: 343-346.