CHAPTER 3

PREDICTION OF SOFTWARE DEFECTS

3.1 Software Metrics

Software metrics are used for obtaining quantitative measurements of the software or its specifications. Mostly, value of the metrics is represented as an interval, ordinal or nominal scale. Metric values in ratio scale are preferred for the mathematical equations used in software process as ratio scale allow for easy and meaningful computation. The software metrics is widely researched; the works which stand out in implementation of the metrics program are those of [51]. The first and most extensive report of software metrics program containing key guidelines which formed a base for subsequent metrics program was done by Grady and Caswell. Goal Question Metric (GQM) was proposed, by Basili and Rombach, using ideas from the total quality management to ensure that the metrics activities are always goal-driven. A metrics program without goals or without direction is certain to fail [37]. The GQM ensured that the metrics program should collect only those metrics which are relevant to a particular goal. The works of Briand et al., 1996 highlights the need of scale types to define measurement for software metrics activities. [39]
Defect prediction models are used to find the reliability of the system, estimate the efficiency of the design and testing process over a number of defects. The byproduct of the above process is the reduction of staffing required and reducing the maintenance cost. Defect prediction models uses machine learning methods to learn and predicting potentially defected modules within the software. Cyclomatic complexity and size of the software are the most commonly used metrics for defect models [52]. The testing metrics that are created in the test phase prove to be useful to estimate the sequence of defects [44]. Metrics from all the steps of the software process such as design, implementation, testing should be utilized and connected to specific dependencies. For designing an effective prediction model metrics from a single process will not be sufficient.

The following are the software metrics used for evaluating the new algorithm proposed in this research work:

3.1.1 Lines of Code (LOC)

Lines of Code (LOC) are a size metric, which quantifies the software size. It is the most widely used metric for program size as it is easily and precisely definable. LOC is defined in different ways for a particular program. It includes:

- The total number of lines
• The number of blank lines in module

• The number of lines of comments in a module

• Lines of executable code

• The number of lines which contain both code and comment in a module

3.1.2 Token Count

The software program is considered to be a collection of tokens of either operator or operands. The basic token count measures are:

• The number of operands

• The number of operators

• The number of unique operands

• The number of unique operators

3.1.3 Cyclomatic complexity

Cyclomatic complexity is the measure of logical complexity of software and is useful for predicting modules that are likely to be error prone. If software program is interpreted as a control flow graph, this strongly connected directed graphs contains nodes which represent parts of the source code having no branches and arcs that control flow transfers during program execution as shown Figure 3.1.
The complexity of software can be correlated with the complexity of the graph. McCabe proposed the cyclomatic number $V(G)$ which is equal to the number of linearly independent paths through a program in its graphs representation, to indicate the software complexity. The $V(G)$ for a program control graph $G$, is given by:

$$V(G) = E - N + P$$

Where $E$ is the number of edges in graph $G$, $N$ is number of nodes and $P$ is the number of connected components in graph $G$. The cyclomatic
complexity is used as measure to avoid excessive complexities that
cause reliability problems and is used to quantify the test program
that detects errors.

3.1.4 Design Complexity

Design complexity measures the amount of interaction between
the modules in a system.

3.1.5 Essential Complexity

Essential Complexity (eV(G)) is a measure of the degree to
which a module contains unstructured constructs. This metric
measures the degree of structuredness and the quality of the code. It
is used to predict the maintenance effort and to help in the
modularization process.

3.1.6 Error Count and density

Error count gives the number of errors in a program. Error
density refers to the number of error counts in a module.

3.1.7 Halstead Metrics

Halstead metrics are computed statically from the code and was
introduced by Halstead in 1977 [55]. Halstead proposed a set of
metrics applicable to several aspects of program. The metrics are
defined as follows. The following token counts are used to compute
the various Halstead metrics,
\( n_1 = \) the number of distinct operators

\( n_2 = \) the number of distinct operands

\( N_1 = \) the total number of operators

\( N_2 = \) the total number of operands

**Halstead length content**

The program length is defined as the count of the total number of operators and operands in the program.

\[
N = N_1 + N_2
\]

The estimated value for length \( N' \) can be calculated using distinct number of operators and operands before the final code is actually produced as follows:

\[
N' = n_1 \log_2 n_1 + n_2 \log_2 n_2
\]

**Halstead volume metric**

Volume metric is a measure of the storage volume required to represent the program.

\[
V = N \log_2 n
\]

where \( n = n_1 + n_2 \).

**Halstead difficulty**

The difficult measure is related to the difficulty of the program to write or understand.
\[ D = \frac{n_1 \cdot N_2}{2 \cdot n_2} \]

**Halstead effort of a module**

The effort required to develop the software is given by the Halstead effort as

\[ E = \frac{V}{L} \]

**Estimated Halstead effort**

The estimated Halstead effort is given as

\[ E = \frac{n_1 n_2 \left[ n_1 \log_2 n_1 + n_2 \log_2 n_2 \right] \log_2 n}{2n_2} \]

**Halstead programming time**

The programming time is obtained in seconds as

\[ T = \frac{E}{S} \]

where \( S \) is the Stroud number which is usually taken as 18. If only the length of the program is known, then time is computed as

\[ T = \frac{N^2 \log_2 n}{4S} \]
3.2 Causes of software defects

Most software defects or bugs are due to the human factor; mistakes and errors made in designing or coding by people. They arise from errors made by a software team during specification, design, coding, data entry and documentation. Following are some of the main causes of software defects.

- **Human factor**: Software’s are developed by human and are prone to make mistakes. It is not possible to develop huge software without errors. The errors can be in any stage of software development such as during specification, design or coding.

- **Communication failure**: Lack of communication or erroneous communication among different teams of the development team is a major source of software defect. For example, if the requirements are vague or incomplete, and the programmer’s codes based on incomplete requirements lead to errors.

- **Requirement specification**: When the requirement specifications are not clearly understood, the software becomes prone to defects.

- **Design problems**: Poorer the design, higher the defects in software program. The design problem can happen in
architectural design, conceptual design, database design, and so on.

- **Delivery timeframe**: Unrealistic development timeframe leads to defects. Factors like insufficient resources, project deadlines, and late design changes give rise to defects.

- **Poor coding practices**: Bad coding practices leads to errors in the code.

- **Defective third-party tools**: Software development requires many tools, the software tools used may be defective. Such tools cause defect in the software.

- **Lack of skilled testing**: Poor testing, shortcomings in the testing process leaves the defects undetected and this may result in poor coding and escalate the risk of errors.

### 3.3 Classification tasks for Defect Prediction

An emerging approach for defect prediction is the use of data mining techniques to predict the problematic areas in the software. These techniques extensively rely on software metrics associated with the software. Bayesian inference under simplifying assumptions formed the base of the reliability assessment studies. Software reliability has been studied extensively for the deviation of metrics from the central tendency. However, with the emergence of machine
learning algorithm, since the 80’s a lot of research has focused on data mining techniques for the classification of defects in software through the metrics obtained either in the design stage or in the maintenance stage.

Machine learning algorithms are applied effectively in problem domains with changing values and regularities. The software metric data of modules or software combined with defect data forms the input of the machine learning algorithms. These algorithms help execute a probability distribution and analyze errors [35]. The techniques popularly used for software defect prediction problems are decision trees, neural network and Bayesian belief network [38; 35; 40]. This research investigates the classification performance using Naïve Bayes and Decision tree induction and also proposes to develop a novel NN classifier to improve the classification accuracy.

Data classification using machine learning involves two steps. Firstly, a built model which describes preset classes of data or concepts are called the learning step or training phase. On analysis of database tuples which are expressed by attributes, construction of the model is accomplished. Determined by one of the attributes, it is assumed that each tuple belongs to a predefined class called the class label attribute. Based on the classification, data tuples are also referred to as samples, examples, or objects. The model set built by studying the data tuples forms the training data set. Training samples are the individual tuples present in the training set, which are
randomly selected from the dataset. Since the class label is provided for each training sample, it is known as supervised learning wherein the learning of the model is 'supervised' indicating the class each training sample belongs to. In unsupervised learning, the class labels of the training samples are unknown, and prior to the learning the number of classes may not be known. The unsupervised learning is also known as clustering. Normally, the learned models are represented as mathematical formulae, classification rules, or decision trees.

Simple techniques known as the holdout method uses a test set of randomly selected class-labeled samples and are not governed by the training samples. Depending on the percentage of correctly classified samples of the test set by the model, the accuracy of model on a given test set is determined. The known class label is matched up with the learned model’s class prediction for the test sample. The estimate of accuracy of the model got from the training data set is optimistic as it tends to over fit the data, so a test set is used to get a more realistic estimate.

If the estimated accuracy of the model is deemed suitable, data tuples or objects with unknown class label can be classified using the model. Classification helps in discovering a model that explains and differentiates data classes or concepts to predict the class of objects with unknown class label. The developed model is based on the study
of a set of training data. Some of the most commonly used classification methods are discussed in this section.

### 3.3.1 Naive Bayes Classifier

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. The underlying probability model is an independent feature model. Mathematically classifiers map a discrete or continuous feature space \( X \) to a discrete set of labels \( Y \). Applications of the classifiers are wide spread in various branches of science. Bayes’ theorem (also known as Bayes’ rule or Bayes’ law) is a result in probability theory, which relates the conditional and marginal probability distributions of random variables. The Naïve Bayes classifiers are efficiently trained in supervised learning due to the precise nature of the probability model. Maximum likelihood methodology is used to find the parameter estimates in the Naïve Bayes models. So in practical applications, the Naïve Bayes models work without using any Bayesian methods or Bayesian probability. When the machine learning techniques are used to create functions from the training data, it is called supervised learning. The statistical method through which the underlying probability distribution of the parameters in a dataset is derived is termed Maximum Likelihood Estimation (MLE). Based on the interpretation of Bayesian probability
in accordance with Bayesian theory, a Naïve Bayes classifier states that the occurrence or nonoccurrence of a feature is not linked in any way to the occurrence or nonoccurrence of any other feature. Chotirat worked on identifying the exact classifier in NB algorithm [54].

Even though it is based on over-simplified assumptions, the Naive Bayes classifiers have acceptable accuracy for real world applications. The estimation of the parameter’s mean and variances of the variable with a small set of training data is sufficient when using the Naive Bayes classifier for the classification is viewed as its major advantage. The advantage of Bayesian models is the easy incorporation of different significant but non-quantifiable factors, such as quality of verification and validation activities, software complexity and test coverage in the model. Mingxi et al., utilized a modified approach effectively to solve the error classification problem [14].

3.3.2 Bayesian Logistic Regression

In a supervised learning problem, if an unknown target function \( f : X \rightarrow Y \) or equivalently \( P(Y|X) \) is to be approximated, assume Y is a Boolean-valued random variable, and X is a vector containing n Boolean attributes. In other words, \( X = (X_1, X_2, \ldots, X_n) \), where \( X_i \) is the Boolean random variable denoting the \( i^{th} \) attribute of X.
Applying Bayes rule, it is learnt that \( P(Y = y_i | X) \) can be represented as

\[
P(Y = y_i | X = x_k) = \frac{P(X = x_k | Y = y_i)P(Y = y_i)}{\sum_j P(X = x_k | Y = y_j)P(Y = y_j)}
\]

where \( y_m \) denotes the \( m \)th possible value for \( Y \), \( x_k \) denotes the \( k \)th possible vector value for \( X \).

The Naive Bayes classification algorithm based on Bayes rule that assumes the attributes \((X_1, X_2, \ldots, X_n)\) are all conditionally independent of one another, given \( Y \). The value of this supposition is that it radically simplifies the representation of \( P(X|Y) \), and the problem of approximating it from the training data.

Consider, for example, the case where \( X = (X_1, X_2) \). In this case

\[
P(X/Y) = P(X_1, X_2/Y)
= P(X_1/X_2,Y) P(X_2/Y)
= P(X_1/Y) P(X_2/Y)
\]

Logistic Regression (Georgiana Ifrim, et al., 2008) [8] is an approach to learning functions of the form \( f: X \rightarrow Y \), or \( P(Y|X) \) in the case where \( Y \) is discrete-valued, and \( X = (X_1, X_2, \ldots, X_n) \) is any vector containing discrete or continuous variables. The parametric model assumed by Logistic Regression in the case where \( Y \) is Boolean is:
\[ P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)} \]

and

\[ P(Y = 0|X) = \frac{\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)} \]

One highly convenient property of this form for \( P(X|Y) \) is that it leads to a simple linear expression for classification.

### 3.3.3 Decision Tree Induction

The structure of a decision tree looks like flow-chart; internal nodes representing a test on an attribute, branches symbolize outcome of the test, and the leaf nodes denoting classes or class distributions. The root node forms the topmost node in a tree.

Unknown sample is classified by testing the sample against the decision tree, using values of the attributes. A traceable pathway from the root to a leaf node holds the class prediction for that sample. The classification rules used are easily got from decision trees.

#### 3.3.3.1 Tree pruning

Many of the branches of a decision tree when built will reflect irregularities in the training data. The irregularities occur due to noise
or outliers. Tree pruning methods discovers this problem of over fitting the data. Pruning methods normally use statistical procedures to eliminate the least reliable branches. This elimination usually results in quicker classification and an enhancement in the capability of the tree to appropriately classify independent test data.

In the pre pruning method, a tree is “pruned” by stopping its construction early; thus no further split or partition at a given node. On halting, the node becomes a leaf which may now hold the most recurrent class in the subset samples, or the probability distribution of those samples.

When building a tree, the goodness of the split is assessed by the measures such as statistical significance, information gain, etc. If dividing the samples at a node results in a split that falls below a pre defined threshold, due to which further dividing of the given subset is stopped. Choosing a suitable threshold is crucial as high thresholds could result in oversimplified trees, while low thresholds could result in too little simplification.

The post pruning approach removes branches from a “fully grown” tree. Removing branches helps in pruning tree node. The cost complexity pruning algorithm is an example of the post pruning approach. On removal of branch, the pruned node becomes a leaf. The most common class amid the former branches now forms the leaf at the pruned node. Expected error rate is calculated for each non-leaf
node in the tree and the expected error rate is calculated for each branch when node not pruned. Depending upon the expected error rate calculated, the sub tree is kept or otherwise pruned. After generating a set of increasingly pruned trees, an independent test set is used to evaluate the accuracy of each tree. The decision tree that minimizes the expected error rate is preferred.

Alternatively pre pruning and post pruning may be interleaved for a combined approach. Post pruning entails more calculation than pre pruning.

### 3.3.4 Random Tree

Random tree is a decision tree that considers K randomly chosen attributes at each node and allows class probabilities based on backfitting with no pruning (Frederick Livingston, 2005) [25]. The effects of different variables are generally not found. The steps involved in a random tree are

- A data set [inbag] is created from the training set by sampling with replacement members. The number of examples in the [inbag] data set is equal to that of the training data set. This new data set may contain duplicate examples from the training set. Using the bootstrapping technique, usually one third of the training set data
is not present in the inbag. This left over data is known as the out-of-bag data. This above process is called bootstrapping

- Each tree is made up of a random number of attributes. Nodes and leaves are formed using the attributes with standard tree building algorithms.

- Pruning is not done and tree is grown to maximum extent possible.

Figure 3.2 shows the ensemble of trees that are produced within the random tree. From Figure 3.2 it is seen that the split of the nodes can occur using different logic for the given class labels. Taking the medial value of all the six different trees produces better results.
Fig. 3.2: Six different trees produced within random tree

### 3.3.5 Classification and Regression Tree (CART)

Classification and regression tree (CART) is a non-parametric technique [7], that produces either classification for categorical variable or regression trees for numeric variable. Trees formed depend upon the values of variable in the modeling dataset from which a collection of rules is created. Rules are chosen based on the capability of splits formed on variables’ values can differentiate observations based on the dependent variable.

- ‘Child’ node formed by splitting a node into two, the rule is applied in the case of parent node.
- CART stops splitting when it detects that there is no further splitting. Terminal nodes at end of each branch.

- Each observation falls into one and exactly one terminal node.

- Set of rules define each terminal node uniquely.

Figure 3.3 shows the functioning of CART.

![Diagram](image.png)

**Fig. 3.3: Classification and regression tree**

From Figure 3.3 it is seen that each attribute is taken as a node and a binary split is made on some conditions. The condition for split can be based on statistical value or based on other parameters like
Gini index. The nodes terminate with the required class label. In the figure there a total of 6 attributes and 4 classes.

### 3.3.6 Neural Network

Artificial Neural Networks are a programming model that seeks to imitate the computational analogues of neurons, and are used widely in artificial intelligence problems from simple pattern detection activities to advanced symbolic operations. The Multilayer Perceptron is an example of an artificial neural network that is used extensively for the solution of a number of different problems, including pattern recognition and interpolation.

Artificial Neural Networks endeavors to model the working of the human brain. The human brains consist of millions of individual cells which are called neuron. All experience and knowledge is encoded by the links that are present between neurons. Given that the human brain consists of such a large number of neurons, the quantity and nature of the connections between neurons is, at present levels of understanding, almost impossible to assess.

Using artificial neural networks it is unattainable to model the full intricacy of the brain, and generally Artificial Neural Network will consist of maximum of a few hundred neurons, and very restricted connections between them. In spite of limited number of neurons and
connectivity, even small neural networks can solve quite difficult computational problems. Artificial Neural Networks are basic input and output devices, with the neurons organized into layers. Simple Perceptrons consist of a layer of input neurons, a layer of output neurons, and a single layer of weights between them, as shown in Figure 3.4.

The network in Figure 3.4 has an input layer with three neurons, one hidden layer with three neurons and an output layer with three neurons. Each neuron in the input layer represents one predictor variable, but in the definite variables, A-1 neurons are used to represent the A number of the variable.

Figure 3.4: A Neural network with three layers
**Input Layer** — The input layer is presented with inputs in the form of vector of predictor variable values \((x_1, \ldots, x_p)\). The values of these predictor variables are standardized to values -1 to 1 either at input layer or by processing before input layer. The input layer then distributes the input values to the neurons in the next layer. With the predictor variables, a constant input of 1.0 is fed to each of the hidden layer. The constant 1.0 is called the bias, and the bias is weighted and added to the sum going into the neuron.

**Hidden Layer** — The input values reaching at a neuron in the hidden layer is multiplied by a weight \((w_{ji})\), and the ensuing weighted values are added to produce a combined value \(u_j\). The weighted sum \((u_j)\) in turn is fed into a transfer function, \(\sigma\), which gives outputs as \(h_j\). The outputs from the hidden layer are dispersed to the output layer.

**Output Layer** — Similarly the output from hidden layer is fed to neurons in outer layers which are multiplied by a weight \((w_{kj})\), and the resulting weighted values are added together producing a combined value \(v_j\). The weighted sum \((v_j)\) is fed into a transfer function, \(\sigma\), which outputs a value \(y_k\). The \(y\) values are the outputs of the network.

Neural network is an efficient approach for handling large amounts of dynamic, non-linear and noisy data, especially in situations where the underlying physical relationships are not fully understood. Neural networks are also particularly well suited to modeling systems (Q. Ju, Z. Hao, 2007) [16] on a real-time basis, and
this benefits software defect forecasting systems which aims at predicting the defects based on certain metrics.

### 3.3.7 Fuzzy Logic

Logic deals with true and false. A *proposition* can be right on one instance and false on another. For example, “Rose is red”, if the rose held is yellow, the proposition that rose is a red flower is false or if the rose is of a red variety, then proposition is true. If a proposition is true, it has a truth value of 1; if it is false, its truth value is 0. 0 and 1 are the only possible truth values. Using logical operations, propositions can be combined to generate other propositions.

Naturally there are statements which cannot be made with conviction. A person might be saying it will rain today. If pushed further, he may be able to say with a degree of certainty that it will rain today. The level of certainty is less than 1. Fuzzy logic model was developed to handle this type of situation. Fuzzy logic deals with propositions that can be true to a certain degree—somewhere from 0 to 1. Therefore, the truth value of a proposition’s indicates the degree of certainty about which the proposition is true.

The fuzzy logic is incorporated in neural networks in various ways. Neural networks with a *fuzzifier* function to preprocess or post-process data is the simplest of the setup as shown in Figure 3.5.
In the second method, fuzziness can go into neural networks to define the weights from fuzzy sets. Expert systems are based on crisp rules which are not always available. Expert systems have to reflect on an extensive set of possibilities which are not being known beforehand. When crisp rules are unknown and possibilities extensive, the expert systems method is not a good one. A comparison between expert systems and fuzzy systems is important to understand in the context of neural networks.

### 3.4 Conclusion

In this chapter various software defects and their corresponding metrics were studied. Defect prediction techniques using data mining algorithms available in literature were studied. In the next chapter it is proposed to study the KC1 Dataset and existing data mining algorithms to review their classification accuracy.