Evaluating Performance of Grid Partitioned based Neuro-Fuzzy System for Prediction of Level of Severity of Faults in Software Systems

Gagandeep Singh, Prof. Amanpreet Singh Brar and Dr. Parvinder S. Sandhu

Abstract— This paper evaluates the Prediction of Level of Severity of Faults in Software Systems using Grid Partitioned based Neuro-Fuzzy System. The objective is to find the best algorithms that can be used to model software Maintenance Severity i.e. to predict the level of impact of the faults in the software system. WEKA (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in java, developed at the University of Waikato. The results are measured in terms of Accuracy, MAE and RMSE values.

Keywords— Fault Prediction, Neuro-Fuzzy System, Software Fault.

I. INTRODUCTION

FAULTS in software systems continue to be a major problem. A software bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from behaving as intended (e.g., producing an incorrect result). A software fault is a defect that causes software failure in an executable product. In software engineering, the non-conformance of software to its requirements is commonly called a bug. Most bugs arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code. Knowing the causes of possible defects as well as identifying general software process areas that may need attention from the initialization of a project could save money, time and work. The possibility of early estimating the potential faultiness of software could help on planning, controlling and executing software development activities.

When a software system is developed, the majority of faults are found in a few of its modules. In most of the cases, 55% of faults exist within 20% of source code. It is, therefore, much of interest to find out fault-prone software modules at early stage of a project [14]. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. Quality will be improved as more faults will be detected. Predicting faults early in the software life cycle can be used to improve software process control and achieve high software reliability. Timely predictions of faults in software modules can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults. Prediction models based on software metrics, can estimate number of faults in software modules.

Machine learning algorithms have proven to be of great practical value in a variety of application domains. Not surprisingly, the field of software engineering turns out to be a fertile ground where many software development and maintenance tasks could be formulated as learning problems and approached in terms of learning algorithms[1-4],[12]. WEKA (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in java, developed at the University of Waikato. WEKA is free software available under the GNU General Public License.

In this present work, the Grid Partitioning based Neuro-Fuzzy Based techniques is explored and comparative analysis is performed for the prediction of faults in software systems.

The paper is organized as follows: section II explains about the methodology followed and section III the result of the study. Finally conclusions of the research are presented in section IV.

II. METHODOLOGY

The first step is to find the structural code and design attributes of software systems i.e. software metrics. The real-time defect data sets are taken from the NASA’s MDP (Metric Data Program) data repository. The dataset is related to the safety critical software systems being developed by NASA.

The suitable metrics like product module metrics out of these data sets are considered. The term product is used referring to module level data. The term metrics data applies to any finite numeric values, which describe measured qualities and characteristics of a product. The term product refers to anything to which defect data and metrics data can be associated. In most cases products will be synonymous with code related items such as functions and systems/sub-systems.

The metrics are as follows:

- MODULE – This metric describes the unique numeric identifier of the product.
• **LOC_BLANK** - This metric describes the number of blank lines in a module i.e. the number of lines that have nothing on them at all.

• **BRANCH_COUNT** - This metric describes the branch count metrics i.e. the number of branches for each module. Branches are defined those edges that exit from a decision node. The greater the number of branches in program modules, the more testing resources required.

• **CALL_PAIRS** - This metric describes the number of calls to other functions in a module.

• **LOC_CODE_AND_COMMENT** - This metric describes the number of lines which contain both code & comment in a module.

• **LOC_COMMENTS** - This metric describes the number of lines of comments in a module. This includes all lines of code that are completely commented and therefore, are completely ignored by the compiler.

• **CONDITION_COUNT** - This metric describes the number of conditionals in a given module.

• **CYCLOMATIC_COMPLEXITY** - This metric describes the cyclomatic complexity of a module. It is the number of linearly independent paths. Given a control flow graph G of a program, the cyclomatic complexity V(G) can be computed as: V (G) = E – N + 2.

• **CYCLOMATIC_DENSITY** - This metric describes the ratio of the module's cyclomatic complexity to its length. The intent is to factor out the size component of complexity. It has the effect of normalizing the complexity of a module, and therefore its maintenance difficulty.

• **DECISION_COUNT** - It describes the number of decision points in a given module. Decisions are caused by conditionals statements.

• **DECISION_DENSITY** - This metric is calculated as CC / LLOC, where CC is cyclomatic complexity and LLOC is logical line of code. This metric shows the average cyclomatic density of the code lines within the procedures of your project. Single-line procedure declarations aren't counted since cyclomatic complexity isn't defined for them. The denominator is the logical lines of code metric. A logical line of code is one that contains actual source code. An empty line or a comment line is not counted in LLOC.

• **DESIGN_COMPLEXITY** - This metric describes the design complexity of a module. Design complexity is a measure of a module’s decision structure as it relates to calls to other modules. This quantifies the testing effort related to integration.

• **DESIGN_DENSITY** - It describes the design density of a module and is calculated as design complexity divided by cyclomatic complexity.

• **EDGE_COUNT** - This metric describes the number of edges found in a given module. It represents the transfer of control from one module to another.

• **ERROR_COUNT** - This metric describes the number of defects associated with a module.

• **ERROR_DENSITY** - This metric describes the number of defects per 1000 lines of code for a module. It is given by, $\frac{\text{ERROR_COUNT}}{1000*\text{LOC_TOTAL}}$.

• **ESSENTIAL_COMPLEXITY** - It describes the essential complexity of a module. It quantifies the extent to which software is unstructured, providing a continuous range of structural quality assessments applicable to all software rather than the "all or nothing" approach of pure structured programming.

• **ESSENTIAL_DENSITY** – The Essential density is given by, $\frac{\text{ERROR_COUNT}}{\text{LOC_TOTAL}}$.

• **HALSTEAD_EFFORT** - This metric describes the Halstead effort metric of a module. Effort is the number of mental discriminations required to implement the program and also the effort required to read and understand the program. The effort to implement (E) or understand a program is proportional to the number of unique operators in the program. It is given by, $E = D * V$, where V stands for volume.

• **HALSTEAD_DIFFICULTY** - The difficulty level or error proneness (D) of the program is proportional to the number of unique operators in the program.

• **HALSTEADCONTENT** - This metric describes the Halstead length content of a module. The Halstead measures are based on four scalar numbers derived directly from a program's source code.

\[ n1 \] is the number of distinct operators,
\[ n2 \] is the number of distinct operands,
\[ N1 \] is the total number of operators and
\[ N2 \] is the total number of operands.

Therefore, Halstead length content, \[ n = n1 + n2 \].

• **HALSTEADLEVEL** - This metric describes the halstead level metric of a module. The halstead level metric i.e. level at which the program can be understood. The program level (L) is the
The NF system is trained using a hybrid learning algorithm using both least squares method and back propagation. In the forward pass the consequent parameters are identified using the NF system is found to be the best out of all the hybrid NF systems [15-16] and the extra complexity in structure and computation of Mamdami based Adaptive NF Inference system with max-min composition does not necessarily imply better learning capability or approximation power [18-19]. Hence, in MATLAB 7.4, the Sugeno based Adaptive Neuro-fuzzy Inference System is used for modeling of software maintenance severity. The inference system, which is already trained, will get the metric values from the earlier stages and estimate the software maintenance severity value of the software components or modules.

In the next step the metrics are analyzed, refined and normalized. Thereafter, it is tried to evaluate the Grid Partitioning based Neuro-Fuzzy Inference System for modeling of the software maintenance severity in software systems. The comparisons are made on the basis of the least value of MAE and RMSE values. Accuracy value of the prediction model is also used for comparison. The mean absolute error is chosen as the standard error. The technique having lower value of mean absolute error is chosen as the best fault prediction technique.

III. RESULTS

The performance of Adaptive Neuro Fuzzy Inference System is found to be the best out of all the hybrid NF systems [15-16] and the extra complexity in structure and computation of Mamdami based Adaptive NF Inference system with max-min composition does not necessarily imply better learning capability or approximation power [18-19]. Hence, in MATLAB 7.4, the Sugeno based Adaptive Neuro-fuzzy Inference System is used for modeling of software maintenance severity. The inference system, which is already trained, will get the metric values from the earlier stages and estimate the software maintenance severity value of the software components or modules.

The Structure of Adaptive Neuro-Fuzzy Inference System is shown in fig. 1.

![Fig.1: Structure of Adaptive Neuro-Fuzzy Inference System](image-url)
least squares and in the backward pass the premise parameters are identified using back propagation. The trained NF system is then tested for the fifteen inputs and it shows 0.1571, 0.2140 and 93.3333 as MAE, RMSE and Accuracy values respectively.

The plot of the expected and the output of the NF system for the different inputs is shown in fig. 2

![Plot of Testing Data V/S FIS Output.](image)

### IV. CONCLUSION

The Grid partitioning based Neuro-fuzzy based Modeling technique has outperformed the other technique on the basis of the testing data with 0.1571, 0.2140 and 93.3333 as Mean Absolute Error, Root Mean Square Error and Accuracy values respectively. It is therefore, concluded the model is implemented and the best algorithm for classification of the software components into different level of severity of impact of the fault is found to be Neuro-Fuzzy based technique. The algorithm can be used to develop model that can be used for identifying modules that are heavily affected by the faults and those need immediate attention for debugging.

The future work can be extended in following directions:

- This work can be extended to other programming languages.
- More algorithms can be evaluated and then we can find the best algorithm.
- Further investigation can be done and the impact of attributes on the fault tolerance can be found.
- Other dimensions of quality of software can be considered for mapping the relation of attributes and fault tolerance.

### REFERENCES


