Journal of Theoretical and Applied Information Technology

15th September 2023. Vol.101. No 17
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN EFFICIENT FILTER BASED FEATURE RANKING AND NON-LINEAR ENSEMBLE LEARNING FRAMEWORK FOR SOFTWARE RELIABILITY ESTIMATION

ANUSHA MERUGU[1], DR. M. CHANDRA MOHAN[2]

[1]Research Scholar, Department of Computer Science and Engineering, JNTUH, Hyderabad, india
[2]Professor , Department of Computer Science and Engineering, JNTUH, Hyderabad, india
.

ABSTRACT

As the size and complexity of the software systems are increasing day-by-day, the software reliability is an essential research concern for both software developers and clients. Software reliability ensures that the software products are reliable and error-free during the product deployment and software testing phases. Most of the conventional reliability estimation models are independent of heterogeneous datatype with homogeneous non-homogeneous poison process measures. In this work, an efficient reliability feature ranking based multi-class ensemble classification framework is implemented for reliability estimation process. In comparison to the traditional software reliability models, experimental results showed that the proposed model has a high software reliability prediction rate with less error and high accuracy.

***Keywords**: Software Reliability, Feature Ranking, Ensemble Learning Model, Classification.*

1. INTRODUCTION

Machine learning based reliability prediction is an essential area for extracting significant patterns and relationships between the information stored on reliability databases. These results are recycled for different objectives and for different areas, ranging from organizational management, product marketing, research and development, fault detection, data analysis and MIS decision-making. Such choices are defined by enormous amounts of data that are related in various ways to each other. Because of the massive data and high data growth rate in the software reliability data, human analysis skills are inadequate. Because biotechnology is developing at a high rate of growth, more and more biological data is being collected and made available for analysis. The importance of developing new techniques to extract knowledge from it also increases when bio-molecular data grows significantly. The healthcare environment is generally seen as rich in information, but unfortunate in the distribution of knowledge [1]. One of the heuristically approaches of optimization for action extraction is colony optimization (ACO) along with particle swarm optimization. PSO optimizes a problem by preserving a particle population and transferring these particles into the search area. Both the ACO and PSO algorithms use one-classification rules sequence of covering patterns. The construction in machine learning models was an active research topic. The main learning machine models to learn group classifiers in high-dimensional datasets are boosting, bagging or stacking. Boemh stated that 80% of software product bugs are covered by approximately 20 percent of modules [2]. In particular in minority class (default modules), the uneven distribution of defeat modules and non-default modules is leading to bad performance[3]. This was addressed at the data or algorithm level[4]. Different methods of over-sampling and under-sampling at the data level were used to balance output of the class. The methodology at the algorithm level, on the other hand, tackles the problem of class inequality by changing your training mechanism to improve the accuracy of the minority classes[5]. As a reliability benchmarking data set, NASA MDP datasets are often use. The study uses five of this repository's most frequent datasets. Each dataset consists of multiple input quality software modules. The building algorithm is comprised of two phases for the majority of decision trees. The first phase constructs a large tree and cuts the tree in the latter phase to avoid problems of adaptation. Then the tailed tree is used to classify. The bootstrap[6] is one group classification system and constructs individual members with different data sets. The Bootstrap Aggregating is a group classification system. The projections will then be made by combining an average or voting on a class label. To build a single tree, a simple algorithm is used. At each decision tree node the tapping process is not performed and attributes are randomly sampled. The example was classified by a majority[7]. The important advantage of random forestry is that it is fast and can handle numerous input properties. A nonlinear function neuron is called every input node. The sigmoidal units of the cloaked layer learn about the functions. MLP uses a back-propagation technique for training purposes. Process measures are time-based measurements that are based on historical source code changes. Those metrics can be extracted from the SCM and include, for example, the number of code additions and deletions, the number of separate committers and the number of lines modified. [8] compared the performance of changing metrics (process) to static code metrics for prediction of defects. They concluded that process data includes more discrimination in the distribution of reliability than the source code itself, i.e. source code metrics. Their explaining is that source code metrics concern the human comprehension of the code, for example, many lines of code or complexity are not necessarily software reliability indicators. [9] used some method metrics to explain whether or not a file was new and whether it was changed. The authors used these metrics in conjunction with other metric families and found that 20 percent of the files, which are identified as most susceptible to reliability in the prediction model, have an average of 83 percent of faults. Wrappers select attributes using the ML algorithm to evaluate the usefulness of different functions. Filters use heuristics based on data characteristics to evaluate the characteristics. The positive effects of filters over wrappers are that they work faster[10] and are thus more suitable in large feature sets for selection.

Software metrics are used as the major variable for software prediction modeling, and reliability data like 1 or 0 is used as the minor variable. The SDP uses software metrics that also include software attributes or features such as changing information and code lines (LOC) to predict that software testing activities are prone to failure. For the instruction of the latest software modules the most examined SDP models are able to rely on their reliability and categorize whether or not the latest software modules have reliability[11]. Another positive effect of filter use is that it can be used with any ML algorithm. In most cases, however, the problem of classification is discreet, while the wrappers with any problem of classification can be used. Because wrappers use the same algorithm for selecting and classifying features, the function selection process should be performed for any prediction algorithm. SE tasks are currently employed in order to extend premium software systems and minimize the number of software defects. The software reliability are recognized that the costs for solving the software reliability often use the development method after software is released. Software reliability can not be considered directly because they can be estimated by software metrics. The recognition of software modules is linked to the faults that are achieved by binary classifiers and predict even if the software is defective by means of various software metrics . ST uses time at SDLC levels and it is very costly to check the software modules. The complexity and size of the software is gradually increased for several purposes, including the safety requirement, reliability and combination of customers ' innovative technologies. Consequently, it is not possible to produce defect-free and consistent software due to a time and budget limitation. Since the need for well organized software systems and maximum performance is progressively developed, their complications are also constantly increased. The greatest complication is the task of quality assurance. Conversely, checking the premium software is a costly task and requires enough resources. Moreover, most of the information is recommended for detailed verification of the software project in the given limited test resources[12].

Software reliability has received much attention because reliability has always had obvious effect on highly visible aspects of software development. The classification scheme should be comprehensive allowing distinct classification of each failure. For latest software systems, the higher level categorization would be to group failures such as configuration failures, planned events and unplanned events. Conventional failures like incorrect input, system crash etc. are unplanned events triggered by software defects. Planned events are cases where there is a planned shutdown of the software for performing any task. Issues in configuration setting lead to configuration failure. Though configuration failures and planned events are generally not caused by software faults, they do affect the reliability experience of the user. Based on what is important to their users, specific types of failures may be focused for different products. Estimation of current reliability and prediction of the future failures is carried out using the reliability growth models using extrapolation techniques. It is important for the test managers to make decisions on the release of the software during the testing phase[13].

Reliability models are used to estimate the product's reliability as a function of test time based on the failure data that is collected during testing. These statistical processes help in determining the best estimates of reliability during testing and on projecting the reliability during field operation. The process is iterated by executing additional testing cycles as required, until the reliability objective is met. When many parameters are involved in a model and its probability density function (PDF) is non-linear, determining an analytical solution for the MLE estimate becomes very difficult. In such cases, using nonlinear optimization algorithms, the numerical MLE estimate is done. In LSE, a parameter value that minimizes SSE is generally determined using a non-linear optimization algorithm. By using the method of least squares in curve fitting methodology the error is minimized in the SSE between the predicted values and the ones actually obtained. In cases where normal distribution is followed for the errors, the LSEs are also the maximum likelihood estimates. A number of software tools are available today for the purpose of estimating parameters of the SRGM. There are two models, static and dynamic that exists to ensure that the quality attributes of the application are assessed. These models help on taking the release related decisions of the software product. A static model estimates the number of reliability in the software using software metrics. Cumulative failure profile or the rate at which failures are discovered in the past is used by dynamic model for estimating the failures. The failure process itself is measured and modeled by the dynamic models. So depending on the cumulative time of the failure, a time component is included in it. Time is recorded either as calendar time or execution time. It focuses on the history of software failure. These models generally assume that the software should be executed based on the operational profile. The basic assumption of most of the SRGMs is that as the reliability that cause failures are removed from the product instantaneously, the reliability of the product increases. Another assumption is that the failure rate of a software product improves over time irrespective of whether or not the reliability are fixed[14-15].

Research Gaps:

1 One research gap in the field of software reliability estimation is the need to incorporate the dynamic nature of user experience and the learning effect over time. While existing models consider the fault detection rate and assume reliability growth with the fixing of underlying faults, they often overlook the impact of user familiarity and their ability to adapt to the software. As users gain experience, they tend to develop workarounds to handle situations that previously caused failures, resulting in an increase in reliability over time. However, current models do not explicitly capture this phenomenon.

2. Moreover, software reliability models make various assumptions related to fault detection rates, the location of faults in the software and data space, and the testing environment. These assumptions may not always hold true in practical scenarios, leading to limitations in the accuracy and applicability of the models.

3. Another research gap lies in the classification and characterization of software reliability models. While some models treat the software as a black box without considering its internal structure, others, known as white box models, explicitly incorporate the software's architecture and module interactions. There is a need to explore and develop more comprehensive and flexible models that can effectively handle diverse software systems and testing scenarios.

4. Additionally, there is a distinction between parametric and non-parametric models in terms of the interpretation and limitations of model parameters. Parametric models assume that the parameters have physical meanings and explicit ranges, while non-parametric models lack such restrictions. Exploring and comparing the strengths and weaknesses of these different modeling approaches can provide valuable insights for improving software reliability estimation.

2. RELATED WORK

The software system's reliability depends on the time frame, when the product has been used. Over time, users learn to use the product properly and identify some workaround to handle situations that causes failure. Thereby increase in the period of usage leads to reliability growth. Any software reliability model has several assumptions. The assumptions are mainly based on the fault detection rate[16]. The reliability in the software and the data space locations where the faults are hidden are also considered by the models. The test environment is one more factor that drives the model assumptions. A main attribute of Markovian model is that, number of reliability remaining in a software system is the countable states that the system has at any specific point of time. In cases where a process is at a particular state, past history will not influence the future development. Depending on the current state of the system the transition probabilities among the states would vary. A discontinuous function is considered as the failure intensity of the system and it relies on the present status of the software. For studying the fault removal process of software, Markovian models are considered to be very helpful. They are considered to significantly support the testing phase of the software development life cycle. Software reliability are identified and fixed in the course of testing. The number of reliability is represented by the state of process at a time t. Reliability growth is assumed by most reliability models at instances when a failure happens and the underlying fault is fixed. Bayesian models take a different approach to reliability growth. Observation of no failure is viewed as an indication of reliability growth, reflecting the user's growing confidence on the software. Bayesian models are described by prior and posterior distributions. It reflects the view of parameters of the model based on the past data. This turns out to be the important consideration in this method. Based on the information from past projects combined with current data, parameter of the model is assumed to follow different distribution patterns. The Littlewood-Verall (L-V) model is probably the best example of the Bayesian class of models. The distribution of failure times is assumed to be exponential with a certain failure rate but it is assumed to be a random variable rather than a constant as in other models [17]. The inter-failure interval is treated as a random variable in TBF models. The number of failures in a specific period is treated as a random variable in FC models. As the testing cycle continue the parameters of inter-failure distribution changes in the case of TBF models. In FC models, the software reliability evolution is illustrated by defining the distribution parameters such as mean value function as appropriate functions of time. Both the models have a generic assumption that when the reliability are detected, the failures are independent. The curve fitting of the identified time-based failure data is performed by a pre-defined model formula in all the SRGMs that depend on time domains. Input domain models and fault seeding are time-independent models. In the fault seeding models, the program is tested after "seeding" numbers of faults. The failures that arise due to the seeded faults are recorded. This data helps on estimating the number of native faults in the program. In input-domain based models, a set of equivalence classes are derived by partitioning the input domain of the program. Every equivalence class is related to a particular program path. The software is validated for every test case. Based on the failures recorded by executing the sample tests, reliability is determined[18]. In 2001, a new classification scheme was arrived in which software reliability models are classified into black box and white box models. The whole software is considered as a single system by the black box model. The reliability estimation does not consider the structure of the model. White box models are based on the architecture of the software system and the way different modules interact with each other. It explicitly incorporates the methodologies involved in the testing cycle and the structure of the system that is tested. There are many models that are classified based on the characteristic of the testing activity as parametric and non-parametric models. The basic assumption of the parametric model is that the parameters have some meaning and have a range. The model parameters have a physical interpretation and are defined explicitly. On the contrary, the parameters in the non-parametric models have no limits. There is no physical interpretation of the parameters and it involves analytical formulation. The Artificial Neural Network (ANN) is an example of non-parametric models[19]. The software system's reliability depends on the time frame, when the product has been used. Over time, users learn to use the product properly and identify some workaround to handle situations that causes failure. Thereby increase in the period of usage leads to reliability growth. Any software reliability model has several assumptions. The assumptions are mainly based on the fault detection rate[20]. The reliability in the software and the data space locations where the faults are hidden are also considered by the models. The test environment is one more factor that drives the model assumptions. A main attribute of Markovian model is that, number of reliability remaining in a software system is the countable states that the system has at any specific point of time. In cases where a process is at a particular state, past history will not influence the future development. Depending on the current state of the system the transition probabilities among the states would vary. A discontinuous function is considered as the failure intensity of the system and it relies on the present status of the software. For studying the fault removal process of software, Markovian models are considered to be very helpful. They are considered to significantly support the testing phase of the software development life cycle. Software reliability are identified and fixed in the course of testing. The number of reliability is represented by the state of process at a time t. Reliability growth is assumed by most reliability models at instances when a failure happens and the underlying fault is fixed. Bayesian models take a different approach to reliability growth. Observation of no failure is viewed as an indication of reliability growth, reflecting the user's growing confidence on the software. Bayesian models are described by prior and posterior distributions. It reflects the view of parameters of the model based on the past data. This turns out to be the important consideration in this method. Based on the information from past projects combined with current data, parameter of the model is assumed to follow different distribution patterns. The Littlewood-Verall (L-V) model is probably the best example of the Bayesian class of models. The distribution of failure times is assumed to be exponential with a certain failure rate but it is assumed to be a random variable rather than a constant as in other models [21]. The inter-failure interval is treated as a random variable in TBF models. The number of failures in a specific period is treated as a random variable in FC models. As the testing cycle continue the parameters of inter-failure distribution changes in the case of TBF models. In FC models, the software reliability evolution is illustrated by defining the distribution parameters such as mean value function as appropriate functions of time. Both the models have a generic assumption that when the reliability are detected, the failures are independent. The curve fitting of the identified time-based failure data is performed by a pre-defined model formula in all the SRGMs that depend on time domains. Input domain models and fault seeding are time-independent models. In the fault seeding models, the program is tested after "seeding" numbers of faults. The failures that arise due to the seeded faults are recorded. This data helps on estimating the number of native faults in the program. In input-domain based models, a set of equivalence classes are derived by partitioning the input domain of the program. Every equivalence class is related to a particular program path. The software is validated for every test case. Based on the failures recorded by executing the sample tests, reliability is determined[22]. In 2001, a new classification scheme was arrived in which software reliability models are classified into black box and white box models. The whole software is considered as a single system by the black box model. The reliability estimation does not consider the structure of the model. White box models are based on the architecture of the software system and the way different modules interact with each other. It explicitly incorporates the methodologies involved in the testing cycle and the structure of the system that is tested. There are many models that are classified based on the characteristic of the testing activity as parametric and non-parametric models. The basic assumption of the parametric model is that the parameters have some meaning and have a range. The model parameters have a physical interpretation and are defined explicitly. On the contrary, the parameters in the non-parametric models have no limits. The number of reliability is represented by the state of process at a time t. Reliability growth is assumed by most reliability models at instances when a failure happens and the underlying fault is fixed. Bayesian models take a different approach to reliability growth. Observation of no failure is viewed as an indication of reliability growth, reflecting the user's growing confidence on the software. Bayesian models are described by prior and posterior distributions. It reflects the view of parameters of the model based on the past data. This turns out to be the important consideration in this method. Based on the information from past projects combined with current data, parameter of the model is assumed to follow different distribution patterns.

Research Objectives and contributions

1. To implement an efficient software reliability estimation model on large datasets.
2. To implement a filter based data feature ranking and classification using optimized kernel function.

Pros of Using SVR for Software Reliability Estimation

Non-Linear Modeling: SVR can capture complex, non-linear relationships, which could be beneficial for modeling user experience and learning effects.

Generalization: SVR is known for its good generalization performance, making it potentially more accurate in practical scenarios.

Flexibility: SVR doesn't make strong assumptions about the underlying data, which could make it more adaptable to different kinds of software systems and testing environments.

Parameter Tuning: SVR allows for the tuning of parameters like the regularization term, which could be adjusted to better fit the specific characteristics of a software system.

Cons of Using SVR for Software Reliability Estimation

Computational Complexity: SVR can be computationally expensive, especially for large datasets, which could be a limitation in real-time or resource-constrained environments.

Interpretability: SVR models are often considered "black-box" models, making them less interpretable than some other methods. This could be a drawback when you need to understand the model's decision-making process.

Sensitivity to Outliers: SVR can be sensitive to outliers, which could distort the reliability estimation.

Feature Scaling Required: SVR requires feature scaling for optimal performance, adding an additional preprocessing step.

3. PROPOSED MODEL

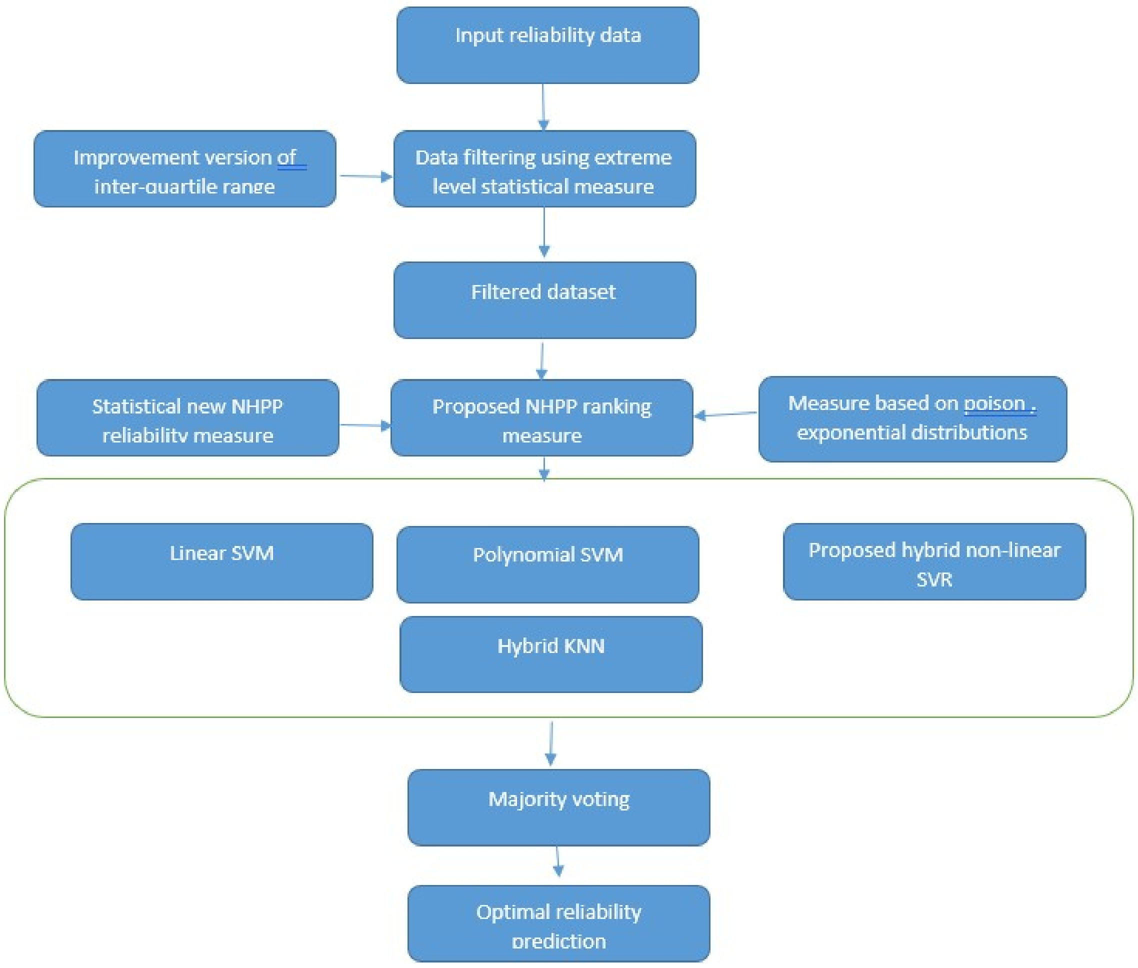
Input reliability data
Improvement version of inter-quartile range
Data filtering using extreme level statistical measure
Filtered dataset
Statistical new NHPP reliability measure
Proposed NHPP ranking measure
Measure based on poison , exponential distributions
Linear SVM
Polynomial SVM
Proposed hybrid non-linear SVR
Hybrid KNN
Majority voting
Optimal reliability prediction

Figure 1: Proposed Model

In the context of software reliability estimation, the research gap discussed above highlights the need for new or profound information and best practices, rather than incremental knowledge creation. While incremental knowledge creation involves building upon existing models and improving their accuracy or applicability within certain boundaries, addressing the research gap requires going beyond incremental advancements.

To bridge this gap, researchers should aim to introduce novel concepts and methodologies that capture the dynamic nature of user experience and the learning effect over time. This may involve exploring new data sources, such as user feedback or usage patterns, to quantify and incorporate the impact of user familiarity and adaptation into the reliability estimation process. By incorporating these aspects, software reliability models can better reflect the real-world behavior of software systems and provide more accurate predictions.
Furthermore, addressing the assumptions made in existing models is crucial. Researchers should critically evaluate the validity of these assumptions and develop approaches that relax or adapt them to better align with practical scenarios. This could involve conducting empirical studies to validate or refine the assumptions and identify any potential biases or limitations associated with them.
In addition to addressing the research gap, it is essential to discuss best practices in software reliability estimation. This includes the development of standardized evaluation metrics, benchmark datasets, and guidelines for model selection and validation. By establishing best practices, researchers and practitioners can ensure the reproducibility and comparability of results, enabling better decision-making in software development and maintenance.It is important to emphasize that tackling the research gap and promoting best practices in software reliability estimation requires collaboration and knowledge sharing among researchers, industry professionals, and relevant stakeholders. Open discussions, collaborative research efforts, and interdisciplinary approaches can facilitate the exchange of ideas, promote innovation, and drive the field forward.
Overall, by focusing on new or profound information and best practices, rather than solely incremental knowledge creation, researchers can contribute to advancing the field of software reliability estimation and provide valuable insights for improving software development processes and decision-making.The provided excerpt introduces an efficient filter-based feature ranking and non-linear ensemble classification framework for software reliability estimation. The objective of this work is to address the increasing size and complexity of software systems by improving reliability prediction with high accuracy and reduced errors.

The introduction emphasizes the significance of machine learning in extracting valuable patterns and relationships from reliability databases. The extracted knowledge can be utilized in various domains such as organizational management, product marketing, research and development, fault detection, data analysis, and decision-making. The exponential growth of software reliability data necessitates the development of advanced techniques to effectively analyze and extract insights from this vast amount of data.

Algorithm 1: Proposed Statistical Quartile deviation based Outlier detection

In the model creation phase, we implement a hybrid outlier detection framework based on the interquartile range for numerical heterogeneous databases. This framework aims to address the sparsity issue in data visualization and prediction by analyzing and testing different levels of outliers on the training data.The model utilizes the lower outlier and lower extreme levels to filter out numerical attributes in the lower range, and the upper outlier and upper extreme levels to filter out values in the upper range. This filtering process facilitates effective data visualization as depicted in Figure 1. By incorporating these techniques, we aim to enhance the accuracy and reliability of the data visualization and prediction tasks.

```
1. Input: Dataset D, feature space F.
2. Output: Input data objects A, Non-anomaly objects N.
3. Read the input dataset D.
4. Calculate the proposed Interquartile Range (IQR) measure for each feature in the feature space F.
5. Define the hybrid IQR measure as follows:
     $F[] = AttIndices();$
     $P1 = V(|F|/4));$
     $P2 = (V(|F|/2) + V(|F|/2+1))/2;$
     $P3 = (V(|F| - |F|/4 - 1) + V(|F| - |F|/4))/2;$
     $U\_E[] = P3 + \eta.(P3 - P1).\log((P3 - P1))$
     $L\_E[] = P1 - \eta.(P3 - P1).\log((P3 - P1))$
     $U\_Outlier = P3 + \Gamma\eta.(P3 - P1)))$
     $L\_Outlier = P1 - \Gamma\eta.(P3 - P1)))$
6. Perform proposed feature ranking for
```

classification process.

2: Feature ranking using optimal NHPP

```
1.  To each feature in the features-list F[]
2.  Do
3.  Compute maxgainratio( ) using the filtered data features.
4.  Computing alpha and beta measure for the feature ranking process.
5.  Let pi be the probability of occurrence of the ith feature value in a class.
6.  Let MaxGainRatio (θ) be the sum of pi multiplied by the exponential of pi.
7.  Define a function chisqr(D) that computes the chi-square value on the data D.
8.  Let |D| be the size of the data.
9.  Compute α as the logarithm of θ divided by |T|^3 multiplied by the square root of the chi-square value of D, multiplied by the square root of |D|.
10. Define a function E(D) that calculates the entropy of D.
11. Let |T| be the total sum of the features.
12. Compute β as θ multiplied by E(D) divided by |T|^(1/3).
13. Done
14. Sort feature according to ranks.
```

Proposed decision tree feature ranking measure:

```
# Conditional Entropy of alpha on beta
CE(alpha[] / beta[]) = Σ P(beta[], alpha[]) * log(P(alpha[]) / P(beta[], alpha[]))
# Conditional Entropy of beta on alpha
CE(beta[] / alpha[]) = Σ P(beta[], alpha[]) * log(P(alpha[]) / P(beta[], alpha[]))
ρ1 = (-CE(beta[/ alpha[]))^3) / ((Σ alpha[]_i)^3 * Corr(D_i)^3)
ρ2 = (-CE(alpha[/ beta[]))) / ((Σ beta[]_i)^3 * sqrt(Corr))
N = total observations
m = minimum(#rows, #columns)
# Proposed software reliability learning model
PE = cbrt(entropy(data) * N * Hellinger(data)) * E(D) / chiVal(data)
# ProposedHER
```

ProposedHER = max(cbrt(Σ(D=1)^(Dp |) Σ(n=1)^(Dp |)(cbrt(Dp / |Dp|) - cbrt(Dn / |Dn|))^2), GainRatio(D), PE)

$$s = D.\log(D);$$

$$p1 = -s/((\sqrt{\sum D[i]})^3 * \sqrt{chisqr(D) * \sum D[i]}$$

$$p2 = -(s * condentropy(D))/(chisqr(D) * \sum D[i])^3$$

$$Rank(A[i]) = Max\{p1, p2\}$$

In the proposed decision tree approach, an optimized random forest approach is implemented in order to improve the feature ranking process of the tree construction process.

In this approach, a hybrid kernel based SVM classifier is implemented in order to improve the accuracy of the traditional SVM classifier. Proposed polynomial kernel approach is optimized in order to improve the true positive rate of the heterogeneous datasets.

4. EXPERIMENTAL RESULTS

In this section, we have executed our proposed model on NASA software reliability datasets and compared the results with traditional reliability prediction models. NASA Metrics Data Program, it is publicly available for verifying, refuting and improving predictive models of software engineering. KC1 is a C++ system implementing storage management for receiving and processing ground data. The dataset consists of the McCabe and Halstead features extractors of the code. The measures are module based. The probability of detection is proportional to the effort; thus, higher rate of detection, more effort is required. Probability of false alarm decreases with increase in detection. This linkage can be observed in receiver operating curve (ROC).

Generated Patterns:

```
Column Total 1783.0
Column Total 326.0
0 th attribute          Feature Ranking Score
3.3850476690526947
Column Total 1783.0
Column Total 326.0
1 th attribute          Feature Ranking Score
1.515598532717133
Column Total 1783.0
Column Total 326.0
2 th attribute          Feature Ranking Score
0.35556767061942784
Column Total 1783.0
Column Total 326.0
3 th attribute          Feature Ranking Score
1.4973976686596235
Column Total 1783.0
Column Total 326.0
4 th attribute          Feature Ranking Score
4.012868998943865
Column Total 1783.0
Column Total 326.0
5 th attribute          Feature Ranking Score
4.37912237350255
Column Total 1783.0
Column Total 326.0
6 th attribute Feature Ranking Score 3.328230276307007
Column Total 1783.0
Column Total 326.0
7 th attribute Feature Ranking Score 4.217648685280728
Column Total 1783.0
Column Total 326.0
8 th attribute Feature Ranking Score 4.715521483001722
Column Total 1783.0
Column Total 326.0
9 th attribute Feature Ranking Score 4.5343361506687705
Column Total 1783.0
Column Total 326.0
10 th attribute Feature Ranking Score 2.608257904809534
Column Total 1783.0
Column Total 326.0
11 th attribute Feature Ranking Score 4.540634725823738
Column Total 1783.0
Column Total 326.0
12 th attribute Feature Ranking Score 3.067330013255967
Column Total 1783.0
Column Total 326.0
13 th attribute Feature Ranking Score 0.9716000250450556
Column Total 1783.0
Column Total 326.0
14 th attribute Feature Ranking Score 1.4806198436250742
Column Total 1783.0
Column Total 326.0
15 th attribute Feature Ranking Score 0.3094963794762421
Column Total 1783.0
Column Total 326.0
16 th attribute Feature Ranking Score 2.614202751287529
Column Total 1783.0
Column Total 326.0
17 th attribute Feature Ranking Score 3.0864948195661555
Column Total 1783.0
Column Total 326.0
18 th attribute Feature Ranking Score 3.4362885054443124
Column Total 1783.0
Column Total 326.0
19 th attribute Feature Ranking Score 3.0569699242858226
Column Total 1783.0
Column Total 326.0
20 th attribute Feature Ranking Score 1.5118092747549445
21 th attribute Feature Ranking Score 0.0


=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.
        Threshold for discarding attributes: -0.0198

Attribute Evaluator (supervised, Class (nominal): 22 CLASS):


Ranked attributes:
 4.7155    9 I
 4.5406   12 L
 4.5343   10 J
 4.3791    6 F
 4.2176    8 H
 4.0129    5 E
 3.4363   19 S
 3.385     1 A
 3.3282    7 G
 3.0865   18 R
 3.0673   13 M
 3.057    20 T
 2.6142   17 Q
 2.6083   11 K
 1.5156    2 B
 1.5118   21 U
 1.4974    4 D
 1.4806   15 O
 0.9716   14 N
 0.3556    3 C
 0.3095   16 P

Selected attributes: 9,12,10,6,8,5,19,1,7,18,13,20,17,11,2,21,4,15,14,3,16 : 21M < 3.5
|  A < 39.5
|  |  M < 0.5
|  |  |  H < 1.75
|  |  |  |  S < 2.5
|  |  |  |  |  L < 0.13 : FALSE (75/0)
|  |  |  |  |  L >= 0.13
|  |  |  |  |  |  A < 3 : FALSE (22/1)
|  |  |  |  |  |  A >= 3 : FALSE (20/0)
|  |  |  |  S >= 2.5 : FALSE (339/0)
|  |  |  H >= 1.75
|  |  |  |  J < 54.56
| | | | | A < 3
| | | | | | I < 7.03
| | | | | | | I < 6
| | | | | | | | A < 1.5 : FALSE (1/0)
| | | | | | | | A >= 1.5 : FALSE (13/1)
| | | | | | | I >= 6 : FALSE (7/1)
| | | | | | I >= 7.03 : FALSE (8/0)
| | | | | A >= 3
| | | | | | G < 0.45
| | | | | | | A < 4.5 : TRUE (1/0)
| | | | | | | A >= 4.5 : FALSE (1/0)
| | | | | | G >= 0.45 : FALSE (2/0)
| | | | J >= 54.56 : FALSE (44/0)
| | M >= 0.5
| | | J < 120.55
| | | | F < 38.2
| | | | | F < 28.25
| | | | | | A < 1.05 : TRUE (1/0)
| | | | | | A >= 1.05
| | | | | | | F < 27.54
| | | | | | | | I < 10.09
| | | | | | | | | O < 0.5
| | | | | | | | | | Q < 5.5
| | | | | | | | | | | E < 6.5
| | | | | | | | | | | | H < 2.25
| | | | | | | | | | | | | A < 4.5
| | | | | | | | | | | | | | A < 3.5
| | | | | | | | | | | | | | | G < 0.83
| | | | | | | | | | | | | | | | F < 4
| | | | | | | | | | | | | | | | | S < 0.5 : FALSE (2/0)
| | | | | | | | | | | | | | | | | S >= 0.5 : FALSE (73/2)
| | | | | | | | | | | | | | | | F >= 4 : FALSE (11/0)
| | | | | | | | | | | | | | | G >= 0.83
| | | | | | | | | | | | | | | | A < 2.5 : FALSE (4/0)
| | | | | | | | | | | | | | | | A >= 2.5
| | | | | | | | | | | | | | | | | H < 0.75 : FALSE (1/0)
| | | | | | | | | | | | | | | | | H >= 0.75 : FALSE (8/1)
| | | | | | | | | | | | | | A >= 3.5
| | | | | | | | | | | | | | | K < 0.01
| | | | | | | | | | | | | | | | H < 0.25
| | | | | | | | | | | | | | | | | E < 3 : FALSE (3/1)
| | | | | | | | | | | | | | | | | E >= 3 : FALSE (1/0)
| | | | | | | | | | | | | | | | H >= 0.25
| | | | | | | | | | | | | | | | | E < 3.5 : FALSE (7/0)
| | | | | | | | | | | | | | | | | E >= 3.5
| | | | | | | | | | | | | | | | | | L < 0.56 : FALSE (5/1)
| | | | | | | | | | | | | | | | | | L >= 0.56
| | | | | | | | | | | | | | | | | | | M < 1.5 : FALSE (1/0)
| | | | | | | | | | | | | | | | | | | M >= 1.5
| | | | | | | | | | | | | | | | | | | | G < 0.58 : FALSE (20/1)
| | | | | | | | | | | | | | | | | | | | G >= 0.58
| | | | | | | | | | | | | | | | | | | | | E < 4.5 : FALSE (59/4)
| | | | | | | | | | | | | | | | | | | | | E >= 4.5 : FALSE (13/1)
| | | | | | | | | | | | | | | K >= 0.01 : FALSE (12/0)
| | | | | | | | | | | | | A >= 4.5 : FALSE (20/0)
| | | | | | | | | | | | H >= 2.25 : FALSE (2/1)
| | | | | | | | | | | E >= 6.5 : FALSE (30/0)
| | | | | | | | | | Q >= 5.5
| | | | | | | | | | | I < 8.5
| | | | | | | | | | | | M < 2.5
| | | | | | | | | | | | | I < 6.71
| | | | | | | | | | | | | | G < 0.17 : FALSE (1/0)
| | | | | | | | | | | | | | G >= 0.17 : TRUE (1/0)
| | | | | | | | | | | | | I >= 6.71 : FALSE (7/0)
| | | | | | | | | | | | M >= 2.5 : TRUE (1/0)
| | | | | | | | | | | I >= 8.5 : FALSE (7/0)
| | | | | | | | | O >= 0.5
| | | | | | | | | | I < 8.8 : FALSE (9/0)
| | | | | | | | | | I >= 8.8
| | | | | | | | | | | E < 7.5 : TRUE (1/0)
| | | | | | | | | | | E >= 7.5
| | | | | | | | | | | | O < 1.5
| | | | | | | | | | | | | A < 7 : TRUE (1/0)
| | | | | | | | | | | | | A >= 7 : FALSE (2/0)
| | | | | | | | | | | | O >= 1.5 : FALSE (1/0)
| | | | | | | | I >= 10.09 : FALSE (24/0)
| | | | | | | F >= 27.54 : TRUE (1/0)
| | | | | F >= 28.25 : FALSE (45/0)
| | | | F >= 38.2
| | | | | A < 7 : TRUE (1/0)
| | | | | A >= 7 : FALSE (1/0)
| | | J >= 120.55 : FALSE (35/0)
| A >= 39.5
| | B < 2 : TRUE (2/0)
| | B >= 2 : FALSE (1/0)
```

```
M >= 3.5
|  T < 67.5
|  |  S < 40.5
|  |  |  C < 2
|  |  |  |  G < 0.08
|  |  |  |  |  E < 61.5
|  |  |  |  |  |  R < 4.5 : TRUE (3/0)
|  |  |  |  |  |  R >= 4.5
|  |  |  |  |  |  |  O < 2.5
|  |  |  |  |  |  |  |  I < 14.54 : FALSE (8/0)
|  |  |  |  |  |  |  |  I >= 14.54
|  |  |  |  |  |  |  |  |  J < 3414.18 : TRUE (2/0)
|  |  |  |  |  |  |  |  |  J >= 3414.18
|  |  |  |  |  |  |  |  |  |  A < 28.5 : FALSE (2/0)
|  |  |  |  |  |  |  |  |  |  A >= 28.5 : TRUE (1/0)
|  |  |  |  |  |  |  O >= 2.5 : TRUE (3/0)
|  |  |  |  |  E >= 61.5 : FALSE (7/0)
|  |  |  |  G >= 0.08
|  |  |  |  |  S < 30.5
|  |  |  |  |  |  I < 20.22
|  |  |  |  |  |  |  R < 2.5
|  |  |  |  |  |  |  |  G < 0.24
|  |  |  |  |  |  |  |  |  I < 6.98
|  |  |  |  |  |  |  |  |  |  I < 6.32 : FALSE (6/0)
|  |  |  |  |  |  |  |  |  |  I >= 6.32
|  |  |  |  |  |  |  |  |  |  |  Q < 6.5 : FALSE (3/0)
|  |  |  |  |  |  |  |  |  |  |  Q >= 6.5 : TRUE (1/0)
|  |  |  |  |  |  |  |  |  I >= 6.98 : TRUE (1/0)
|  |  |  |  |  |  |  |  G >= 0.24 : TRUE (3/0)
|  |  |  |  |  |  |  R >= 2.5
|  |  |  |  |  |  |  |  J < 109.3 : FALSE (36/0)
|  |  |  |  |  |  |  |  J >= 109.3
|  |  |  |  |  |  |  |  |  G < 0.39
|  |  |  |  |  |  |  |  |  |  Q < 5.5 : FALSE (20/0)
|  |  |  |  |  |  |  |  |  |  Q >= 5.5
|  |  |  |  |  |  |  |  |  |  |  N < 7.5
|  |  |  |  |  |  |  |  |  |  |  |  P < 1.5
|  |  |  |  |  |  |  |  |  |  |  |  |  H < 6.71
|  |  |  |  |  |  |  |  |  |  |  |  |  |  J < 389.72
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  L < 21.35
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  I < 12.06 :
FALSE (23/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  I >= 12.06
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  J < 316.13
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  I < 16.54
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  L <
16.38
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  F <
51.85
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  L
< 10.95
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
A < 8 : FALSE (12/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
A >= 8
```

```
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  E < 13.5
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  O < 0.5 : TRUE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  O >= 0.5
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  I < 12.75
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  A < 10 : FALSE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  A >= 10 : FALSE (2/1)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  I >= 12.75 : FALSE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  E >= 13.5 : FALSE (4/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  L
>= 10.95 : TRUE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  F
>= 51.85 : FALSE (16/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  L >=
16.38 : TRUE (3/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  I >=
16.54
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  S <
12.5
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  E <
16
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  A
< 7.5 : TRUE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  A
>= 7.5 : FALSE (2/1)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  E
>= 16 : FALSE (4/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  S >=
12.5 : TRUE (2/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  J >=
316.13
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  F <
76.08 : FALSE (12/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  F >=
76.08
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  J <
367.63
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  I <
17.55
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  A
< 8.5 : TRUE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  A
>= 8.5 : FALSE (6/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  I >=
17.55 : FALSE (1/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  J >=
367.63 : FALSE (2/0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  L >= 21.35 :
```

```
TRUE (1/0)
| | | | | | | | | | | | | |  J >= 389.72
| | | | | | | | | | | | | | |  L < 32.75 :
FALSE (25/0)
| | | | | | | | | | | | | | |  L >= 32.75
| | | | | | | | | | | | | | | |  L < 33.06
| | | | | | | | | | | | | | | | |  H < 6.2 :
TRUE (1/0)
| | | | | | | | | | | | | | | | |  H >= 6.2
| | | | | | | | | | | | | | | | | |  A < 16.5
: FALSE (7/0)
| | | | | | | | | | | | | | | | | |  A >=
16.5 : FALSE (7/1)
| | | | | | | | | | | | | | | |  L >= 33.06 :
FALSE (24/0)
| | | | | | | | | | | | |  H >= 6.71
| | | | | | | | | | | | | |  H < 9.82
| | | | | | | | | | | | | | |  F < 111.2
| | | | | | | | | | | | | | | |  F < 81.72
| | | | | | | | | | | | | | | | |  I < 9.59 :
TRUE (1/0)
| | | | | | | | | | | | | | | | |  I >= 9.59
| | | | | | | | | | | | | | | | | |  F <
79.56 : FALSE (4/0)
| | | | | | | | | | | | | | | | | |  F >=
79.56 : TRUE (1/0)
| | | | | | | | | | | | | | | |  F >= 81.72 :
FALSE (15/0)
| | | | | | | | | | | | | | |  F >= 111.2
```

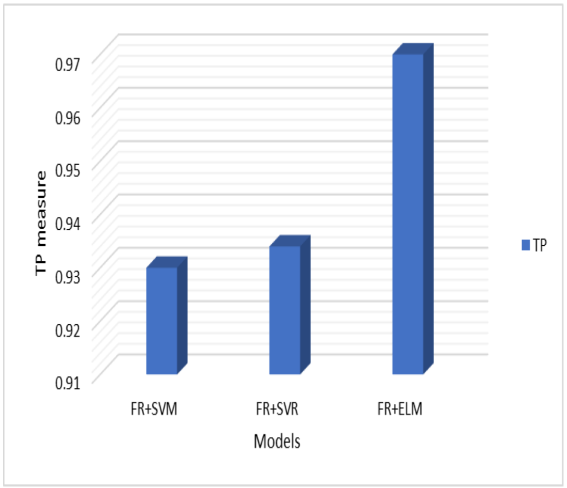

Figure 2: Comparative Analysis Of Proposed Ensemble Software Reliability Model To The Conventional Models For True Positive Rate.

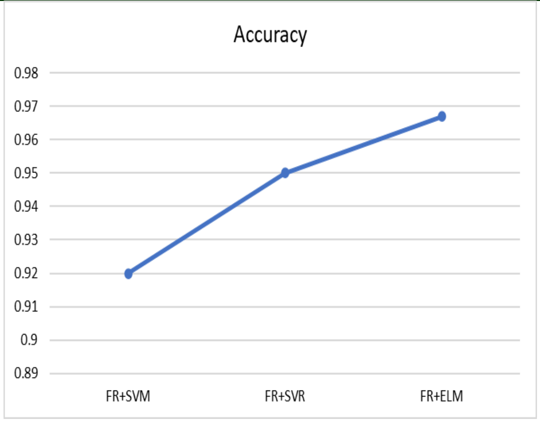

*Figure 3: Comparative Analysis Of Proposed Ensemble Software Reliability Model To The Conventional Models For Accuracy Rate***.**

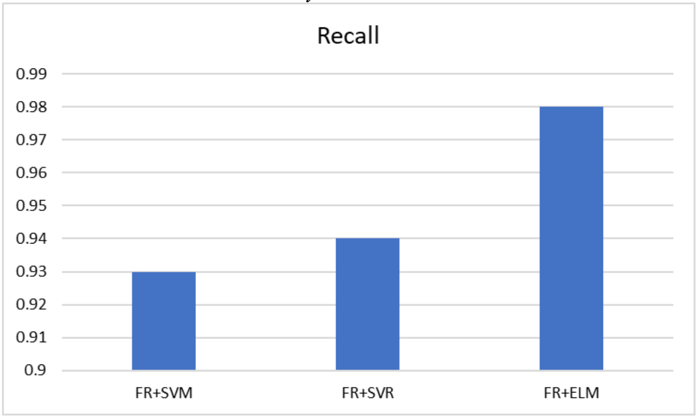

Figure 4: Comparative Analysis Of Proposed Ensemble Software Reliability Model To The Conventional Models For Recall Rate.

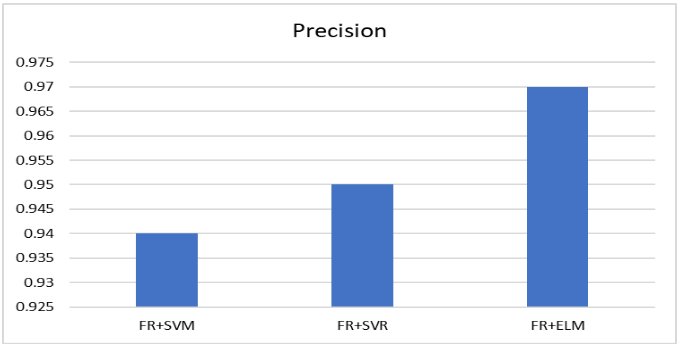

Figure 5: Comparative Analysis Of Proposed Ensemble Software Reliability Model To The Conventional Models For Precision Rate.

5. CONCLUSION

Due to the high true positive rate of conventional support vector regression models, the majority of conventional software reliability estimation models are unable to predict the test samples. For reliability severity classification, the majority of conventional machine learning-based

fault prediction models integrate common software reliability growth measures. However, these models are employed to forecast the binary class reliability level with lower standard errors. The training fault datasets are used to implement a hybrid support vector regression-based quartile deviation growth measure in the third contribution. On various reliability datasets with various configuration parameters for fault prediction, experimental results are simulated. In this work, an efficient reliability feature ranking based multi-class ensemble classification framework is implemented for reliability estimation process. In comparison to the traditional software reliability models, experimental results showed that the proposed model has a high software reliability prediction rate with less error and high accuracy.
Compare research findings and analyze the strengths and weaknesses of this work:
Literature Review: Conduct a thorough literature review to identify current relevant work in the field of software reliability estimation. Look for studies, papers, or articles that address similar or related research questions, methodologies, or objectives.
Identify Key Findings: Compare the key findings and results of your work with those of the current relevant work. Look for similarities, differences, and trends in the outcomes.
Methodological Analysis: Evaluate the methodologies employed in your work and the current relevant work. Assess the appropriateness, rigor, and limitations of the methodologies used. Consider factors such as data collection, sample size, statistical analyses, and experimental design.
Strengths and Weaknesses: Identify the strengths and weaknesses of your work and the current relevant work. Consider aspects such as novelty of approach, clarity of presentation, validity of results, generalizability, limitations, and potential biases.
Impact and Contribution: Assess the impact and contribution of your work in comparison to the current relevant work. Evaluate how your findings build upon existing knowledge, address research gaps, or introduce novel insights. Consider the implications and potential applications of your work.
Future Directions: Discuss the potential avenues for further research and improvement based on the findings of your work and the current relevant work. Identify areas where further investigation is needed and propose potential solutions or approaches. While traditional methods like Bayesian and Markovian models offer some insights, they often fall short in capturing the multifaceted nature of software reliability. The exploration of Support Vector Regression (SVR) as an alternative has shown promise, offering flexibility and the ability to model complex, non-linear relationships. However, it also comes with its own set of limitations, such as computational complexity and reduced interpretability.

6. FUTURE WORK

User Experience Modeling: Future research should focus on developing models that incorporate the dynamic nature of user experience and learning effects. This could involve the integration of machine learning techniques with traditional reliability models.
Validation of Assumptions: A critical evaluation of the assumptions underlying existing models is necessary. Empirical studies could be conducted to validate or challenge these assumptions, leading to more robust and practical models.
Hybrid Models: The development of hybrid models that combine the strengths of both parametric and non-parametric approaches could offer a more comprehensive solution to software reliability estimation.
Computational Efficiency: Given the computational complexity associated with methods like SVR, future work should also focus on optimizing algorithms for better scalability and real-time applicability.

REFERENCES

[1] S. Chatterjee and A. Shukla, “Software reliability modeling with different type of faults incorporating both imperfect debugging and change point,” in 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), Sep. 2015, pp. 1–5. doi: 10.1109/ICRITO.2015.7359361.
[2] K.-C. Chiu, “A study of software reliability growth model for time-dependent learning effects,” in 2012 IEEE International Conference on Industrial Engineering and Engineering Management, Dec. 2012, pp. 1015–1019. doi: 10.1109/IEEM.2012.6837894.
[3] J. Dhar, Anamika, S. Ingle, and Y. Sheshker, “Software reliability growth model with logistic-exponential TEF in imperfect debugging environment,” in International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014),

May 2014, pp. 1–4. doi: 10.1109/ICRAIE.2014.6909281.

[4] Md. A. Haque and N. Ahmad, "An Imperfect SRGM based on NHPP," in 2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA), Sep. 2021, pp. 1574–1577. doi: 10.1109/ICIRCA51532.2021.9544959.

[5] Y. He, "NHPP software reliability growth model incorporating fault detection and debugging," in 2013 IEEE 4th International Conference on Software Engineering and Service Science, May 2013, pp. 225–228. doi: 10.1109/ICSESS.2013.6615293.

[6] R.-H. Hou, S.-Y. Kuo, and Y.-P. Chang, "On a unified theory of some nonhomogeneous Poisson process models for software reliability," in Proceedings. 1998 International Conference Software Engineering: Education and Practice (Cat. No.98EX220), Jan. 1998, pp. 60–67. doi: 10.1109/SEEP.1998.707634.

[7] Y.-F. Hou, C.-Y. Huang, and C.-C. Fang, "Using the Methods of Statistical Data Analysis to Improve the Trustworthiness of Software Reliability Modeling," IEEE Access, vol. 10, pp. 25358–25375, 2022, doi: 10.1109/ACCESS.2022.3154103.

[8] C. Y. Htoon and N. L. Thein, "Model-based Testing Considering Cost, Reliability and Software Quality," in 6th Asia-Pacific Symposium on Information and Telecommunication Technologies, Nov. 2005, pp. 160–164. doi: 10.1109/APSITT.2005.203649.

[9] C.-Y. Huang and W.-C. Huang, "Software Reliability Analysis and Measurement Using Finite and Infinite Server Queueing Models," IEEE Transactions on Reliability, vol. 57, no. 1, pp. 192–203, Mar. 2008, doi: 10.1109/TR.2007.909777.

[10] C.-Y. Huang, J.-H. Lo, and S.-Y. Kuo, "Pragmatic study of parametric decomposition models for estimating software reliability growth," in Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257), Nov. 1998, pp. 111–123. doi: 10.1109/ISSRE.1998.730861.

[11] S. Hwang and H. Pham, "Quasi-Renewal Time-Delay Fault-Removal Consideration in Software Reliability Modeling," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no. 1, pp. 200–209, Jan. 2009, doi: 10.1109/TSMCA.2008.2007982.

[12] S. Inoue and S. Yamada, "Two-Dimensional Software Reliability Assessment with Testing-Coverage," in 2008 Second International Conference on Secure System Integration and Reliability Improvement, Jul. 2008, pp. 150–157. doi: 10.1109/SSIRI.2008.21.

[13] T. Ishii and T. Dohi, "A New Paradigm for Software Reliability Modeling – From NHPP to NHGP," in 2008 14th IEEE Pacific Rim International Symposium on Dependable Computing, Dec. 2008, pp. 224–231. doi: 10.1109/PRDC.2008.24.

[14] G. Kumar and R. Sharma, "Analysis of software reliability growth model under two types of fault and warranty cost," in 2017 2nd International Conference on System Reliability and Safety (ICSRS), Dec. 2017, pp. 465–468. doi: 10.1109/ICSRS.2017.8272866.

[15] N. Kurishima, H. Oikawa, J.-I. Nakamura, K. Amari, and M. Fujioka, "Quantitative analysis of errors in telecommunications software," in 1993 Conference on Software Maintenance, Sep. 1993, pp. 190–198. doi: 10.1109/ICSM.1993.366943.

[16] H. Li, Q. Li, and M. Lu, "Software Reliability Modeling with Logistic Test Coverage Function," in 2008 19th International Symposium on Software Reliability Engineering (ISSRE), Nov. 2008, pp. 319–320. doi: 10.1109/ISSRE.2008.51.

[17] Q. Li and C. Mao, "Considering Testing-Coverage and Fault Removal Efficiency Subject to the Random Field Environments with Imperfect Debugging in Software Reliability Assessment," in 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Oct. 2016, pp. 257–263. doi: 10.1109/ISSREW.2016.13.

[18] Q. Li and H. Pham, "A Generalized Software Reliability Growth Model With Consideration of the Uncertainty of Operating Environments," IEEE Access, vol. 7, pp. 84253–84267, 2019, doi: 10.1109/ACCESS.2019.2924084.

[19] Q. Li, C. Zhang, and H. Zhang, "A New Software Reliability Model for Open Stochastic System Based on NHPP," in 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Jul. 2017, pp. 624–625. doi: 10.1109/QRS-C.2017.120.

[20] S. Li, "A Useful Parametric Family to Characterize NHPP-based Software Reliability Models," in 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S), Jun. 2021, pp. 23–24. doi: 10.1109/DSN-S52858.2021.00022.

[21] S. Li, T. Dohi, and H. Okamura, "A Comprehensive Evaluation for Burr-Type NHPP-based Software Reliability Models," in 2021 8th International Conference on Dependable Systems and Their Applications (DSA), Aug. 2021, pp. 1–11. doi: 10.1109/DSA52907.2021.00010.

[22] C.-T. Lin, K.-W. Tang, J.-R. Chang, and C.-Y. Huang, "An investigation into whether the NHPP framework is suitable for software reliability prediction and estimation," in 2010 IEEE International Conference on Industrial Engineering and Engineering Management, Dec. 2010, pp. 626–630. doi: 10.1109/IEEM.2010.5674524.