International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Improve the Performance of Cross Project Defect Prediction Using an Pattern Based Modified Hidden Markova Fault Tree

V.Ruckmani[a*], ***Dr. S.PRAKASAM***[b]

[a]Research Scholar, Computer Science and Applications, SCSVMV University , Enathur, Kanchipuram

[b]Associate Professor Computer Science and Applications, SCSVMV University , Enathur, Kanchipuram

ABSTRACT

Defect prediction results provide a list of source code artifacts that are prone to defects. Quality assurance teams can effectively devote more energy and allocate limited resources to defect-prone source code verification software products. A module that identifies defect prediction methods for frequent defects before the start of the testing phase. Measurement-based defect-prone modules improve software quality and reduce costs, leading to effective resource allocation. The previous method doesn't analyze the defect pattern, and it has less performance during software development. This work introduces a Pattern-based Modified Hidden Markova Fault Tree (PMHMFT) framework to extract the hidden fault analysis during cross-project validation. Evidential sequence feature selection method to random feature subset selection to estimate the fitness value. This feature selection method reduces the dimensionality of the features to improve the performance of the prediction model. Density-based clustering method to extract the interesting pattern from a large set of defect data based on defect density. The proposed Modified Hidden Markova Fault Tree algorithm constructs the defect fault tree to analyze the cross-project code defect. Using a Levy flight, optimize the method to search the fault classes efficiently compared to another method. The proposed PMHMFT to implement evaluate the performance using k-fold validation. Thus, the proposed work on software defect prediction achieves higher accuracy in true classification and prediction with less error rate.

Keywords: Defect prediction, Pattern-based Modified Hidden Markova Fault Tree (PMHMFT), cross-project validation, Levy flight

1. Introduction

Software metrics and software defect datasets can train and develop a potential clustering model to enhance the process of software defect prediction. After training, the model is used for finding unknown defect modules in the selected dataset. The attributes involved in software defect prediction produce a great impact on the effectiveness of the process. During testing, the testing team is responsible for providing a guarantee on all defects have been rectified and fixed by the software developers during the system testing stage. On the other side, the end-users will expect to predict defects in the software to select whether the software is robust and reasonable for release. Henceforth the capability to forecast how many defects are identified at the initial time of system testing could be an ideal way to overwhelm this issue. Software Defect Prediction [SDP] plays a significant role in active research in software engineering. Software malfunction is an error, defect, malfunction, software malfunction or bug that can result in a false or unexpected result. The main risk factors related to not detecting software flaws in the early stages of software development are wasting time, quality, cost, energy and resources. These are flaws that can appear at various stages of software development. Growing up, software companies focus on software quality, especially in the early stages of software development.

An example of this approach is incorporating functional information and process artifacts into the source code, such as information representing version control concepts behind the developer's functional changes. The integration of such source code artifacts and software processing artifacts in software engineering, such as automatically predicting where failures in source code may occur during software processing, allows for more complex automated learning and rationalization. The proposed idea is that assistance should not be included in the sample of activities that do not improve the forecast software. Interestingly, the reduced feature set helps developers provide information about each block of code's functionality, which is still vulnerable to vulnerabilities to improve model description capabilities.

SDP identifies defective modules and requires extensive testing. An error can lead to premature detection of an efficient allocation of resources, reducing the time and cost of developing software and high-quality software. Therefore, the SDP model plays an important role in understanding, evaluating, and improving software systems' quality.

2. Related work

Possible loss identification information of random forest algorithms, deficiencies due to aggregation, and accumulation may adversely affect the forecast model's performance [1]. According to Micro interaction metrics (MIM), although it is acknowledged that a developer's behavior may affect software quality, this widely used deficiency predictor does not take into account developer behavior [2]. Deficiency prognosis after minimizing the strong interactions' impact, we found that the research team had a lower impact than the indicator series [3]. However, the test is central in selecting the factor that dominates the forecasting system (especially the classifier) in influencing the forecasting performance, the classification method for detection [4].

The deep belief network (DBN) learns to use itself, extracting token vector semantic features (file-level defect prediction models) and source abstract modifications (transition level defect prediction models) from the abstract syntax trees (AST) project [5]. The proposed structure's performance was verified using multivariate exponentially weighted moving average (MEWMA) using statistical multidimensional quality control [6]. Features subcommittee selection and feature ranking programs are closely related to these two feature subcommittee selections. The feature sequence system performance [7], CPTP (cross-project defect prediction), feature selection, capability analysis are involved. The Semi-supervised transfer component analysis (SSTCA) explores class inequality learning's relevance under cross-item defect prediction [8].

Cross-project data distribution is typically different from the target volume to find the most closely related training data and the time-to-project Within-project semi-supervised defect prediction (WSDP) [9]. The Clustering-based Multi-Version Classifier (CMVC) can repeatedly select the most appropriate and quietest version of the training data by assigning more weight than others [10]. The best first search algorithm always focuses on the continuous programs; no work has been considered for simultaneous program error prediction, which differs from program characteristics by successive programs [11]. Therefore, to minimize vulnerabilities, the software manager can focus on the transition rate from one project to another over time [12]. The chromosomal theory forms our understanding of inherited laws of various traits from humans and other organisms [13].

To obtain a mathematical model, the defect prognosis model must be satisfied so that there are limitable conditions that can be demonstrated when using the defect prediction model [14]. Vector sequences and their labels (defective or non-defective) are used to generate project semantic information that the Long Short Term Memory (LSTM) can automatically learn and make defect predictions [15]. Intuitively, the main type is likely to be off-road vehicles because they participate in more activities or have more contact and dependency. However, there is nothing wrong with using K-nucleus decay analysis software [16]. Also, to suggest a large-scale deficiency prognosis solution for inexperienced software practitioners and novices [17]. Disabled diagnostic solutions can help disabled software practitioners and newcomers identify/work with defective documentation improve their coded analysis performance [18].

The data imbalance problem helps make an integrated modeling method without changing the adaptive synthetic sampling (ADASYN) and random model. This classification is established based on attributes and classes, which reduce the dependency and variance association rules using multi boost to minimize classification errors [19]. Gated hierarchical long short-term memory networks (GH-LSTMs) are based on the development of defect prediction technology that exposes outdated software's inability to capture semantic information, and economic features are used to develop error prediction models [20]. Over the literature show, the cross-project verification and difficulty in predicting defect pattern are provided unnecessary result. The proposed method to handle the uncertainty in determining the similar patterns among the dataset using clustering.

3. Implementation of the proposed method

The proposed Pattern-based Modified Hidden Markova Fault Tree (PMHMFT) framework is adapted to provide higher performance classification. Features, when they operate individually, will give the best classification output while combining the attributes. As a cleaning process, the dataset is normalized using the min-max process to convert the dataset values to lie under the range of 0 to 1. The dataset consists of a huge volume of data, so to overwhelm this problem, the Evidentialsequence feature selection is performed. TheRandom features, software error predictions are used to determine the similarity between the data set events.

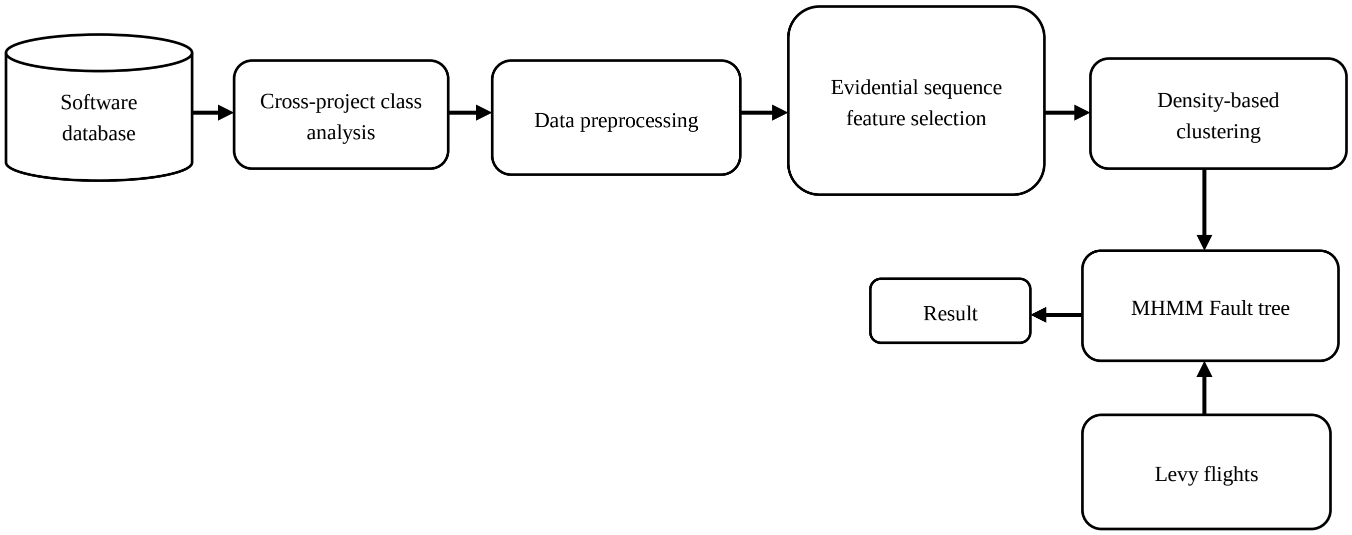

Fig.1Proposed method block diagram

The random features of each instance are clustered using density-based clustering. The resulting cluster set is used to generate useful and interesting patterns to classify whether the given instance is defect-free using MHMM.

3.1 Evidential sequence feature selection

All-new features support the distributing propositions to determine software defects. The "frame-of-discernment" scheme computes the system. All the mutual exclusive situation clarification are computed and represented by sign φ. Significance is denoted as 1 for "defect", 2 for "no defect," and 3 for "Either defect or no defect," actually a sign of witlessness or "neither defect nor no defect" is a sign of the exceptional condition. To analyze the feature subset value of each parameter (p), Select the instance IR and find the related hits (h)

To read the dataset to analysis the value

```
For each i in sd do
    Labeled for each class ← Sd
    Neighbor value estimation $e_n = \sum_{i=1}^{Sd} \frac{sd(i)\in n}{Sd}$
    While n to $e_n$
        $\varphi \leftarrow \max_n(e_n) > Sd_i$
    End
End
```

$$f_x(\varphi) \leftarrow \sum_{C\neq class\,(IR)} \left[\frac{p(x_i^{\lambda})}{1-p\left(class(x_i^{\lambda})\right)} \sum_{i=1}^{k} diff\left(\sigma(i), \mu\,(i)\right) \right]$$

The first check for the existence of outliers is performed in these two variables and their dependent variables.

3.2 Density based clustering for pattern extraction

Density-based clustering basically operates by associating related items contained in the sample space. The association is performed by maintaining maximum interclass similarity and minimum interclass similarity. However, the major downside of such an approach is that it is time-consuming in huge datasets. The basic idea of a density-based clustering algorithm is that it operates based on neighbor density points. A node is considered as a part of a cluster if it has a group of $neighbors >= min_{pts}$, satisfying the distance threshold defined bymax_{th}. This neighborhood-based analysis acts as the basis for identifying varied shaped clusters without providing the initial cluster count. To estimate each cross-code defect weightage

$$f(\varphi) = \sum_{i=0}^{d} sin\sqrt{\frac{l_i+cost_i}{processing\ \ time}}(l) \qquad (1)$$

Where the l_i number of logic.

```
Algorithm steps
Input: preprocessed dataset (PD)
Step 1: initialize the dataset and cluster classes(C), $min_{pts}$ and number of cluster points (k).
Step 2: where the cluster list is taken $C_1, C_2, C_3\ and\ C_n$
Step 3: mark the $min_{pts}$ based on vector classes.
    For each i from PD
        Neighbor cluster class ($NC_c$) = fault class ($min_{pts}, C_i$)
        To estimate the fault weight based on above equation
        If size of (Neighbor point ($\varphi$)<$min_{pts}$)
            Mark the fault point $min_{pts}(i)$
        Else C= next cluster
            Expand cluster ($min_{pts}, cluster\ point, C_i, PD(i)$) followed step 4.
        End if
    End for
Step 4: Expand cluster $min_{pts}, cluster\ point, C_i, PD(i)$
        Add $min_{pts}$ to cluster $C_i$
    Repeat step 3
    If the $min_{pts}(i)$is not visited
        Update the cluster value
    If size of (Neighbor point ($\varphi$)>=$min_{pts}$) then
        $\varphi =$ Neighbor cluster (fault value estimate)
        Add $min_{pts}$current cluster
        Return all object with the$C_i$.
    End if
Step 5: Until each fault class within $min_{pts}$ in Neighbor Cluster
    It can also find clusters surrounded by a different cluster that is not connected to another cluster.
```

3.3 Modified Hidden Markova Fault Tree

MHMM statistical framework, a set of elementary probabilistic models of basic linguistic units (e.g., phonemes), is used to build speech representation. MHMM training algorithm creates a statistical representation model of the fault datasets. The proposed is designed to solve any problem that can be considered a sequence of defect states d1, d2, d3, dn. Markov process is a random model that describes the case's sequence, the possible conditions that depend on only the current fault state to a previous fault state. Our model will be useful in using fault trees to identify the cause of failures, repair rates and analyze defect prediction performance metrics. After construct, the fault tree applies the Lavy flight optimization model to predict the cross-code defect module. Lévy flight has a global exploration approach solution by tracking in large steps in the long run. It uses most of the optimization problems to look for the best new solution with Lévy flight efficiently. Levy flights select the random fault feature to evaluate the fitness value and generate the new solution.

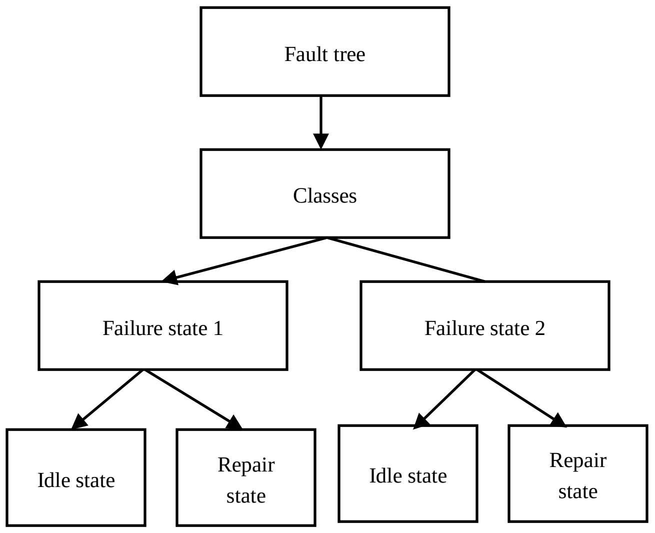

Fig. 2 fault tree model

The occurrence of an event on this node is represented by logic 1; otherwise, the node has a logical value of 0. The calculation set to the probability of n steps is used to evaluate the probability (p_{ij}(n)) of positive steps.

$p_{ij}(n) = p_{ij}$ (2)

$p_{ij}(n) = \Pr(X_{n+m} = k)|(X_n = i)$ (3)

To compute the transition probability for each chain

$p_{ij}(\mathrm{n}) = \Pr(X_m = k)|(X_{m-1} = i)$ (4)

Following equation 3 and 4 represent a transition probability if the state $n \geq 1$, and 0 state-transition probability fault tree,

$p_{ij}(0) = \{0, 1\}$ (5)

In this probability, the state is changed based on time. To assume the n^{th} state $X_n = i$ and the next state probability j following equation.

$X_n + 1 = j$ (6)

In this Markov chainmodel, to predict the fault class and validate time taken long iteration. To reduce the random selection and long-run prediction shorten using a Monte Carlo. This method approximates the posterior distribution of sampling interest parameters in probability space to identify the user's corresponding policy value randomly.Levy flights estimate the fitness value using the following equation. The fitness value of f_i is greater than thef_j. The objective function $f(x)$is defined on the search space (S),

$x_i(f) = x_i(f) + \alpha \otimes levy(x, \lambda)$ (7)

It has considered four post-placement metrics to select the fault class from the software module's solution space.

4. Result and discussion

The simulation analysis for validating this process uses the publicly available dataset from the NASA MSP repository. This experimental result uses k-fold cross-validation to determine the proposed work's performance by training and testing the given dataset. The dataset was trained, and test sets were split into different ratios like 70/30, 75/25, and 80/20 (A, B, and C).

Table 1 proposed method simulation parameter

Parameters	**Value**
Tool	Visual studio
Dataset name	NASA MSP repository
Number of data	1000

The implementation of the proposed Pattern-based Modified Hidden Markova Fault Tree (PMHMFT) method simulation is present in table 1. In this PMHMFT method compared to existing method Kernel Supervised Sub-Code Prediction (KSSP) method compare to existing method Support Conventional neural network (CNN), Defect prediction via an attention-based recurrent neural network (DP-ARNN), Random Forest (RF) and Improved Subclass Discriminant Analysis (ISDA) method.

Table 2performance analysis and k-fold cross-validation

No of fold	precision	recall	f1-score	Accuracy
1	0.89	0.75	0.70	0.92
2	0.82	0.75	0.78	0.91
3	0.83	0.70	0.72	0.86
4	0.79	0.75	0.80	0.92
5	0.71	0.85	0.71	0.88
6	0.78	0.70	0.73	0.88
7	0.89	0.65	0.73	0.91
8	0.85	0.85	0.85	0.89
9	0.83	0.74	0.88	0.89
10	0.82	0.78	0.80	0.91

Our Proposed method evaluation result provides better precision, recall, f1-score and accuracy rate, as shown in table 2. This analysis of the proposed method provides better accuracy, f1-score, recall and precision results compare to another method.

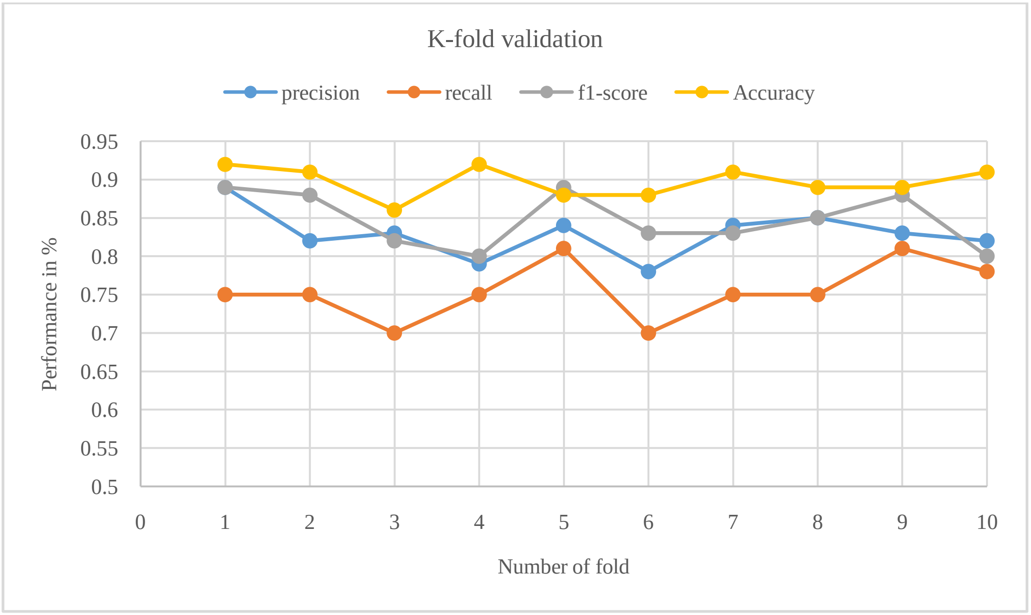

Fig.3 proposed method k fold validation performance analysis

The k-fold validation for split the data into the different datasets to analyze the system performance. The proposed PMHMFT method has a 91% accuracy for 10-folds and 80% of f1-score, 78% of recall rate, and 82% Precision rate for 10-fold data validation. In this analysis, the result comparison is illustrated in fig. 4.

Table 3 performance analysis comparison based Test sets

Test sets	F1-score	recall	Precision	Accuracy
A	93.4	78.9	84.2	93.5
B	92.0	79.3	81.9	90.1
C	91.8	80.2	83.6	91.3

Table 3 presents a comparison of proposed PMHMFT method test sets software defect prediction performance of f1-score, recall, precision and accuracy. The result comparison shows the proposed PMHMFT method has a 91.3% accuracy for the C dataset and 83.6%precision rate, 91.8% of f1 score better result.

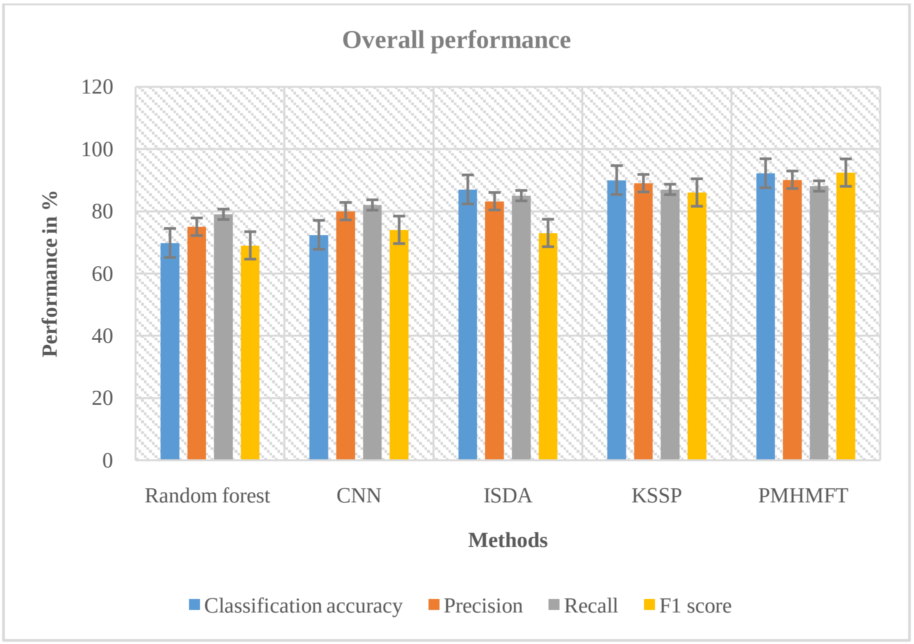

Fig. 4 overall performance of the proposed and existing method

The fig. 4 compares the proposed PMHMFT method test sets software defect prediction performance of f1-score, recall, precision and accuracy. The comparison result shows the overall average performance of the existing CNN, Random Forest, ISDA, KSSP method and proposed PMHMFT method. It provides a 91.8% average more accuracy compared to another existing method.

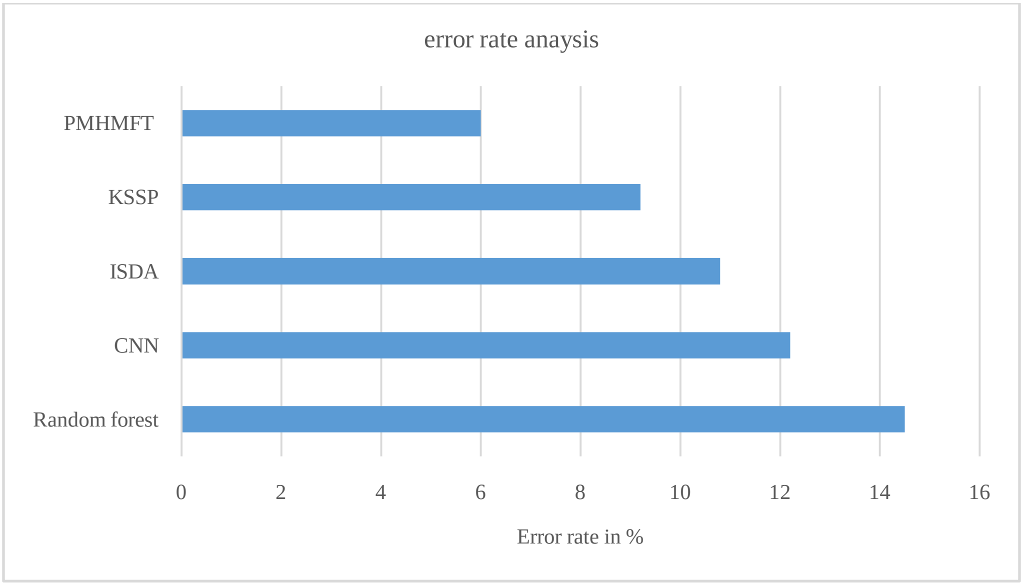

Fig.5 Error rate analysis of proposed method

The classification error rate comparison of the proposed PMHMFT method has a 6% of low error rate, and it is shown in the figure. In this error rate, the existing Random forest method has 14.5%, CCN has a 12.2%, ISDA has 10.8% and KSSP 9.2% higher error rate than the proposed PMHMFT method.

5. Conclusion

The software's quality entirely depends on fault-free reliable software; therefore, there is a need to find the solution to overcome software defects. The importance of feature selection in software defect prediction when the dataset is very vague and imprecise to handle. Several faults that occurred can also be included in software defect prediction to understand the defect pattern more precisely and improve the measures accordingly. The proposed Pattern-based Modified Hidden Markova Fault Tree (PMHMFT) framework method analyzes the fault tree to predict the defect prediction. The Density-based clustering method to extract the fault pattern and estimate the fault wait to improve classification accuracy. The proposed PMHMFT successfully handles the class imbalance problem by giving a good performance compared to existing models.

Reference

[1]. Zhang, Feng; Hassan, Ahmed E.; McIntosh, Shane; Zou, Ying (2016). The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models. IEEE Transactions on Software Engineering, 1–1. doi:10.1109/tse.2016.2599161.

[2]. Lee, Taek; Nam, Jaechang; Han, DongGyun; Kim, Sunghun; In, Hoh (2016). Developer Micro Interaction Metrics for Software Defect Prediction. IEEE Transactions on Software Engineering, 1–1. doi:10.1109/tse.2016.2550458.

[3]. Tantithamthavorn, Chakkrit; McIntosh, Shane; Hassan, Ahmed E.; Matsumoto, Kenichi (2016). Comments on "Researcher Bias: The Use of Machine Learning in Software Defect Prediction". IEEE Transactions on Software Engineering, 1–1. doi:10.1109/TSE.2016.2553030.

[4]. Shepperd, Martin; Hall, Tracy; Bowes, David (2017). Authors' Reply to "Comments on 'Researcher Bias: The Use of Machine Learning in Software Defect Prediction". IEEE Transactions on Software Engineering, 1–1. doi:10.1109/TSE.2017.2731308.

[5]. Wang, Song; Liu, Taiyue; Nam, Jaechang; Tan, Lin (2018). Deep Semantic Feature Learning for Software Defect Prediction. IEEE Transactions on Software Engineering, 1–1. doi:10.1109/tse.2018.2877612.

[6]. Huda, Shamsul; Alyahya, Sultan; Ali, MdMohsin; Ahmad, Shafiq; Abawajy, Jemal; Al-Dossari, Hmood; Yearwood, John (2018). A Framework for Software Defect Prediction and Metric Selection. IEEE Access, 1–1. doi:10.1109/ACCESS.2017.2785445.

[7]. Yu, Qiao; Qian, Junyan; Jiang, Shujuan; Wu, Zhenhua; Zhang, Gongjie (2019). An Empirical Study on the Effectiveness of Feature Selection for Cross-Project Defect Prediction. IEEE Access, 1–1. doi:10.1109/ACCESS.2019.2895614.

[8]. Jing, Xiao-Yuan; Wu, Fei; Dong, Xiwei; Xu, Baowen (2016). An Improved SDA-based Defect Prediction Framework for both Within-project and Cross-project Class-imbalance Problems. IEEE Transactions on Software Engineering, 1–1. doi:10.1109/TSE.2016.2597849.

[9]. Wu, Fei; Jing, Xiao-Yuan; Sun, Ying; Sun, Jing; Huang, Lin; Cui, Fangyi; Sun, Yanfei (2018). Cross-Project and Within-Project Semisupervised Software Defect Prediction: A Unified Approach. IEEE Transactions on Reliability, 1–17. doi:10.1109/TR.2018.2804922.

[10]. Zhang, Jie; Wu, Jiajing; Chen, Chuan; Zhang, Zibin; Lyu, Michael R. (2020). CDS: A Cross -Version Software Defect Prediction Model with Data Selection. IEEE Access, 8 110059–110072. doi:10.1109/ACCESS.2020.3001440.

[11]. Yu, Tingting; Wen, Wei; Han, Xue; Hayes, Jane (2018). ConPredictor: Concurrency Defect Prediction in Real-World Applications. IEEE Transactions on Software Engineering, 1–1. doi:10.1109/TSE.2018.2791521.

[12]. Felix, EbubeoguAmarachukwu; Lee, Sai Peck (2017). Integrated Approach to Software Defect Prediction. IEEE Access, 1–1. doi:10.1109/ACCESS.2017.2759180.

[13]. EboBennin, Kwabena; Keung, Jacky; Phannachitta, Passakorn; Monden, Akito; Mensah, Solomon (2017). MAHAKIL: Diversity-based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction. IEEE Transactions on Software Engineering, 1–1. doi:10.1109/TSE.2017.2731766.

[14]. Herbold, Steffen (2019). On the costs and profit of software defect prediction. IEEE Transactions on Software Engineering, (), 1–1. doi:10.1109/TSE.2019.2957794.

[15]. Liang, Hongliang; Yu, Yue; Jiang, Lin; Xie, Zhuosi (2019). Seml: A Semantic LSTM Model for Software Defect Prediction. IEEE Access, 7(), 83812–83824. doi:10.1109/ACCESS.2019.2925313.

[16]. Qu, Yu; Zheng, Qinghua; Chi, Jianlei; Jin, Yangxu; He, Ancheng; Cui, Di; Zhang, Hengshan; Liu, Ting (2019). Using k-core Decomposition on Class Dependency Networks to Improve Bug Prediction Model's Practical Performance. IEEE Transactions on Software Engineering, (), 1–1. doi:10.1109/TSE.2019.2892959.

[17]. Wan, Zhiyuan; Xia, Xin; Hassan, Ahmed E.; Lo, David; Yin, Jianwei; Yang, Xiaohu (2018). Perceptions, Expectations, and Challenges in Defect Prediction. IEEE Transactions on Software Engineering, (), 1–1. doi:10.1109/TSE.2018.2877678.

[18]. Y. Zhao, Y. Yang, H. Lu, J. Liu, H. Leung, Y. Wu, Y. Zhou, and B. Xu. Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. Automated Software Engineering, 24(2):393–453, Jun 2017.

[19]. He, Haitao; Zhang, Xu; Wang, Qian; Ren, Jiadong; Liu, Jiaxin; Zhao, Xiaolin; Cheng, Yongqiang (2019). Ensemble MultiBoost Based on RIPPER Classifier for Prediction of Imbalanced Software Defect Data. IEEE Access, 7(), 110333–110343. doi:10.1109/ACCESS.2019.2934128.

[20]. H. Wang, W. Zhuang and X. Zhang (2020) Software Defect Prediction Based on Gated Hierarchical LSTMs, in IEEE Transactions on Reliability, doi: 10.1109/TR.2020.3047396.