A hybrid approach based on k-nearest neighbors and decision tree for software fault prediction

Manpreet, Jitender Kumar Chhabra
Dept. of Computer Engineering
National Institute of Technology, Kurukshetra-136119 INDIA
Corresponding author: manibhangu92@gmail.com

Abstract

Software testing is a very important part of the software development life cycle to develop reliable and bug-free software but it consumes a lot of resources like development time, cost, and effort. Researchers have developed many techniques to get prior knowledge of fault-prone modules so that testing time and cost can be reduced. In this research article, a hybrid approach of distance-based pruned classification and regression tree (CART) and k- nearest neighbors is proposed to improve the performance of software fault prediction. The proposed technique is tested on eleven medium to large scale software fault prediction datasets and performance is compared with decision tree classifier, SVM and its three variations, random forest, KNN, and classification and regression tree. Four performance metrics are used for comparison purposes that are accuracy, precision, recall, and f1-score. Results show that our proposed approach gives better performance for accuracy, precision, and f1-score performance metrics. The second experiment shows a significant amount of running time improvement over the standard k-nearest neighbor algorithm.

Keywords: Decision Tree; k- nearest neighbors; machine learning; pruning; software fault prediction.

1. Introduction

The 21st century has seen an unprecedented growth of automation and software with more emphasis on security, interactive graphical user interface, and faster development with more features (Singh *et al.*, 2016). But all this necessitates reliable and bug-free software which can be achieved by effective software testing and maintenance.

Software testing is an essential part of the software development life cycle and is used to identify fault-prone and complex modules so that faults can be removed from fault-prone modules and refactoring of complex modules can be done during the software development process. But software testing and maintenance phase consume almost fifty percent of software development resources such as time, effort, and cost (Aziz *et al.*, 2019). It is necessary to reduce the testing time and cost to develop reliable software within a limited budget and resources. If a somehow testing team manages to get prior knowledge about fault-prone and complex modules that need more attention, then the team can directly focus on those modules and a significant amount of testing time and effort can be reduced. Hence early identification of the faulty modules has caught the attention of researchers.

Many approaches have been developed in the recent past to detect and predict fault-prone modules. Machine learning is one of these approaches which have gained popularity in the past few years in this area. Decision trees (C4.5 and CART) (Quinlan *et al.*, 1986), support vector machine and its variations (Noble *et al.*, 2006), multi-layer perception (Gardner *et al.*, 1998), k-nearest neighbors (Kozma *et al.*, 2008), and random forest (Biau *et al.*, 2016) are some standard machine learning approaches that are the most commonly used for software fault prediction(Beygelzimer *et al.*, 2008). But standard machine learning approaches usually give an average performance in most cases (Cheng *et al.*, 2014). Researchers are developing new and hybrid approaches by combining existing approaches to get better performance.

In this research article, we propose a hybrid approach by combining distance-based pre-pruned classification and regression tree with weighted k-nearest neighbors. The next section explains previous works and research gaps. The third section explains the working of our proposed approach. The fourth section discusses the experimentation and performance evaluation of our proposed approach.

2. Related works

Studies related to software fault prediction area are summarized in this section. Saravanan et al. (Saravanan *et al.*, 2021) proposed an African buffalo optimizer based convolution neural network for fast training in the software fault prediction field. Kassaymeh et al. (Kassaymeh *et al.*, 2021) used a salp swarm optimizer for neural network training instead of backpropagation. Singh et al. (Singh *et al.*, 2021) proposed a new node splitting method for decision tree generation. Haouari et al. (Haouari *et al.*, 2020) presented an application of AIRS for inter-release software fault prediction. Yucalar et al. (Yucalar *et al.*, 2020) compared different ensemble learning approaches like voting, bagging, and boosting in the software fault prediction field. Alsghaier et al. (Alsghaier *et al.*, 2020) in 2020 applied genetic algorithm, PSO algorithm, and GA-PSO integrated algorithm to train support vector machine on twelve software fault prediction datasets and results show that GA-PSO integrated approach gives the best results. Abuassba et al. (Abuassba *et al.*, 2022) in 2022 developed a general plateform for ensembles in classification context. proposed framwork is applied on twelve datasets to prove the diversity and efficiency of ensemble learning approaches. Khan et al. (Khan *et al.*, 2016) in 2016 explained various machine learning approaches in their survey. Rajkumar et al. (Rajkumar *et al.*, 2015) in 2015 applied various machine learning approaches for thyroid problem diagnosis.

The decision tree was initially developed by Quinlan in 1986 (Quinlan *et al.*, 1986). The initial version of the decision tree is called ID3 and it can handle only categorical attributes. C4.5 is an extended version of ID3 that can handle continuous attributes also was developed by Ross Quinlan (Quinlan *et al.*, 1986). Both of these decision tree generation algorithms use information as a node splitting criterion. Ruggieri et al. (Ruggieri *et al.*, 2002) in 2002 developed an efficient C4.5 classifier based on the quicksort and counting sort algorithms to efficiently calculate information gain of continuous attributes. Safavian et al. (Safavian *et al.*, 1991) explained different types of decision tree classifiers and their building methods in detail in their survey. k-NN is a non-parametric classifier initially developed by Evelyn Fix and Joseph Hodges in 1951 (Fix *et al.*, 1989). In this classifier value of k is fixed and for the prediction of the class label of the testing sample, it checks the labels of k-nearest neighbors and assigns a label to the testing sample based on the majority labels of nearest neighbors.Zhang et al. (Zhang *et al.*, 2007) in 2007 developed a lazy learning approach called ML-KNN based on a standard KNN algorithm for a multi-label classification problem like test classification. Cheng et al. (Zhang *et al.*, 2014) in 2014 developed a new k nearest neighbors algorithm based on sparse learning with data-driven k values and neglecting the correlation of samples.

After studying previous literature, we develop a hybrid approach based on k-nearest neighbors and decision tree. The main contributions of the projected work are listed below:

1. A new decision tree pruning approach called distance-based pruning is proposed to prune decision tree nodes. Detailed steps of the distance-based decision tree pruning approach are explained in section 3.2.

2. Standard k nearest neighbor algorithm has $\mathcal{O}(n)$ running cost which is reduced to $\mathcal{O}(\log n) + c$ in our proposed approach. KNNs are added at leaf nodes of decision tree in training phase to reduce the running cost.

3. The CART decision tree is generated using distance based pruning approach and instead of storing class labels, k nearest neighbors are stored on leaf nodes of the decision tree.

4. The concept of weights is introduced based on the sigmoid function in the prediction phase to make

standard k nearest neighbors more effective. Weights are inversely proportional to the distance of nearest neighbors from the point under consideration.

3. Proposed approach

This section of the research article explains the notations used to formulate our proposed approach, working of proposed approach and running time cost of our proposed approach.

3.1 Notations

A variable $X \in \mathbb{R}^{n*m}$ represents training data. Where n represents the total number of training samples and m represents the total number of independent attributes (dimensions of the dataset). Symbol $K - Matrix$ is used to describe k-nearest neighbors matrix where each element of $K - Matrix$ is represented by symbol e_{ij}. Variable i means i^{th} row, and j represents the j^{th} column of the matrix. $Tolerance$ is a global parameter that contains a value between 0 and 1. P_i in m dimensional space represents each training sample.

3.2 Working of approach

This article proposes a hybrid approach based on distance-based pre-pruned classification and regression tree and weighted k-nearest neighbors. The proposed approach in this article considers all training samples as m dimensional points, where m is number of independent attributes in the dataset. A KNN-matrix ($K - Matrix$) is generated in which element $e_{ij} = 0$ if j^{th} training sample is not the nearest neighbour of i^{th} training sample and element $e_{ij} = 1$ if j^{th} training sample is considered as the nearest neighbor of i^{th} training sample. After calculation of KNN-matrix, maximum distance $Max_{distance}$ among all points is calculated based on Euclidean distance formula and a constant parameter Tolerance is introduced to control the decision tree generation. At each decision tree node, the

Algorithm 1 Pseudo code of proposed approach (WK-Tree)

```
Input: Training Samples $X$, Training Classes $Y$
Output: Binary Classification Confusion Matrix
  // $X$ training samples
  // $Y$ labels attached to training samples
                    *Training phase of proposed approach*
Step 1: All training samples are considered as points in m dimensional space and the largest distance among all points is computed using Euclidean distance shown in equation (4).
Step 2: KNN's of all training samples are calculated using equation (3) and the matrix is created as shown in section 3.
Step 3: The CART decision tree is created with distance-based pre-pruning.
  // pruning strategy is explained in detail in section 4.2.
Step 4: KNN's of all samples are stored in leaf nodes instead of storing class labels.
  // KNN's are stored without repetition
  // all duplicate KNN's are removed
                    *Testing phase of proposed approach*
Step 1: The first step is to reach the leaf node of the decision tree; the testing sample under consideration belongs.
Step 2: Label of the point under consideration is assigned based on weighted labels of nearest neighbors of the leaf node.
//smaller distance has more weight than the larger distance
Step 3: Confusion matrix is created based on predicted values by WK-Tree and actual values
Step 4: Accuracy, Recall (sensitivity), Precision, and F1-Score is calculated based on Confusion Matrix
```

$Max_{distance} * Tolerance$ condition is checked, if the node's training samples satisfy this condition then that node is considered as a leaf node and instead of storing class labels, nearest neighbors based on

KNN-matrix are stored at that leaf nodes. Detailed steps and pseudo code of the proposed approach are given in algorithm 1.

Figure 1 represents the flow chart of our proposed approach.

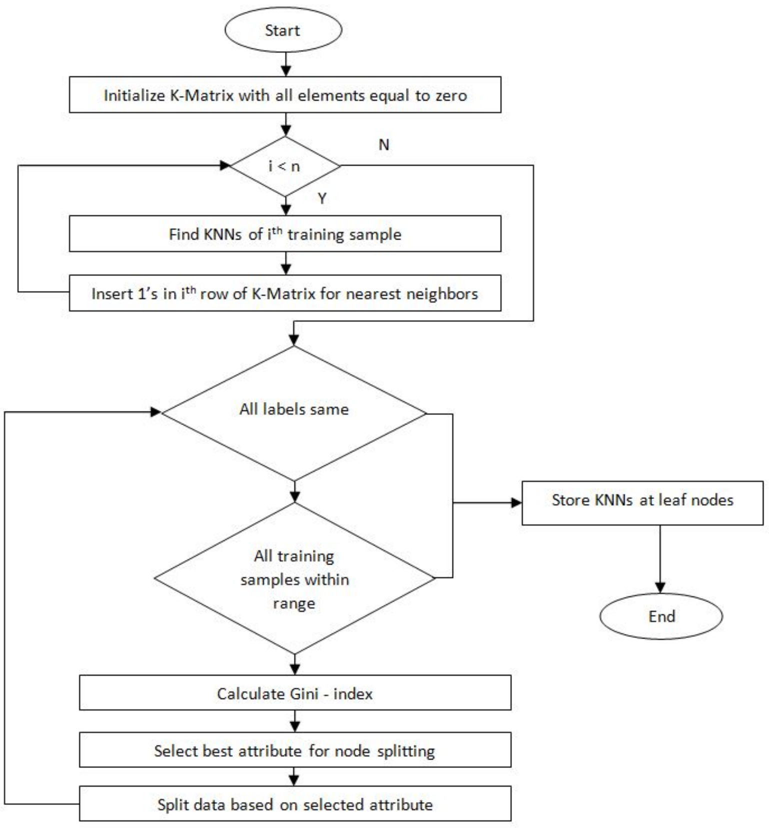
Start
Initialize K-Matrix with all elements equal to zero
i < n
N
Y
Find KNNs of ith training sample
Insert 1's in ith row of K-Matrix for nearest neighbors
All labels same
All training samples within range
Store KNNs at leaf nodes
End
Calculate Gini - index
Select best attribute for node splitting
Split data based on selected attribute

Fig. 1. Flow chart of the proposed approach

3.3 WK-Tree generation

This section of the research article explains the WK-Tree generation process in detail. Steps to develop WK-Tree are explained as below:

1. **Calculation of largest distance:** In the first step of proposed the approach, all training samples are considered as points in m dimensional space and the largest distance among all points is computed as shown in Figure 2.

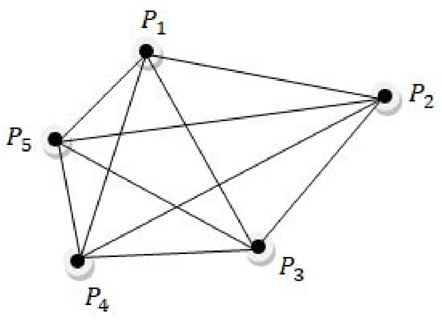

Fig. 2. Grid of training samples in m dimensional space

In figure 2 distance between point P_2 and point, P_5 is the largest among all points. It can be considered for decision tree pruning. Distance computed among all training samples in m dimensional space is Euclidean distance and can be calculated based on equation (1)(Danielsson, 1980).

$$d(P_i, P_j) = \max_{1 \le i,j \le n} \sqrt{\sum_{k=1}^{m} \left(P_i^{(k)} - P_j^{(k)} \right)^2} \tag{1}$$

Where m is the total number of dimensions/features and n is the total number of training samples. k is variable to iterate over the number of dimensions and i & j are variable to iterate over the number of training samples.

2. **KNN matrix generation:** A $n * n$ matrix of k-nearest neighbors is generated for all training samples. Here n is the total number of training samples. $sqrt(n) + c$ function is selected to find the nearest neighbors and an example of $5 * 5$ KNN matrix is shown in equation (2) with all diagonal elements equal to 1. If element e_{ij} of K- matrix is 1 then point j is considered as the nearest neighbor of point i on the other hand if element e_{ij} of KNN matrix is 0 then point j is not considered as the nearest neighbor of point i. The final matrix will be a binary square matrix with all diagonal elements as 1. All diagonal elements are 1 because the point under consideration is always considered as the nearest neighbor of it and stored in the leaf node of the decision tree.

$$K - matrix = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2}$$

   After performing pruning of decision tree node instead of storing class label, nearest neighbors of training samples of the pruned node are stored based on the KNN matrix.

3. **Decision tree creation:** A decision tree is created based on the Gini index node splitting method with distance-based pre pruning. There are two types of decision tree node pruning methods pre-pruning and post-pruning. Pre-pruning based on the maximum depth of each leaf node from the root of the tree is not an effective approach so we have done distance-based pre-pruning which is explained in section 3.4 in detail. In distance-based pruning, a constant parameter $Tolerance$ is initialized between 0 and 1, and $Max_{distance}$ is multiplied with Tolerance to prune decision tree nodes.

4. **Labelling of testing samples:** In the labeling phase first classification and regression tree is traversed to reach the leaf node and then the weighted k-nearest neighbor approach is applied to assign the final label of the testing sample. Weights are assigned to each nearest neighbor based on equation (3) from the testing sample.

$$w_i = \left(1.0 - \frac{1}{1 + e^{-d_{ij}}} \right) * L_i \tag{3}$$

   Where w_i is the weight assigned to nearest neighbor i and value of $L_i = -1$ for non-fault prone classes and $L_i = +1$ for fault-prone classes.

3.4 Distance-based pruning

In distance-based pre-pruning of decision tree first, we find out the maximum distance among all points and then take the fraction of maximum distance to prune decision tree. Maximum distance is calculated to cover all points. Detailed steps of pre-pruning based on the fraction of maximum distance among all training samples are explained as under:

1. All training samples are considered as points in m dimensional space in this strategy. Here m is the number of independent attributes.

2. Parameter $Max_{distance}$ is calculated based on equation (4).

$$Max_{distance} = \max_{0 \leq i,j \leq k} (distance(x_i, y_j)) \tag{4}$$

   Where k is the total number of training samples in a particular node of the decision tree and $(distance(x)_i, y_j)$ is the distance between point x_i and point $(y)_j$.

3. Global parameter $Tolerance$ is adjusted between 1 and 0 manually based on the density of points. If the parameter value is adjusted to 1, it means the whole dataset is considered as nearest neighbors and the decision tree will be able to build only root node on the other hand if the parameter value is adjusted to 0 then the full decision tree will be built without any pruned node.

4. While building the decision tree, at each decision tree node $Max_{distance} * Tolerance$ is tested for all training samples at that node. If the condition is satisfied for all training samples then the decision tree node is pruned and marked as a leaf node.

3.5 Time complexity

Training time is a one-time investment, so we will discuss only the testing time complexity of our proposed approach. The Decision tree traverses from the root node to the leaf node in the prediction phase in $\mathcal{O}(\log n)$ time complexity. Instead of storing labels, k-nearest neighbors are stored at leaf nodes in our proposed approach so a little constant c is added to the actual testing time complexity of the decision tree. The total running time complexity of our proposed approach in this research article is $\mathcal{O}(\log n + c)$.

4. Results and analysis

This section of the research article explains about the model validation approach, performance measurement metrics, datasets used , and comparison of results of the proposed approach with other machine learning models.

4.1 Model validation

In this research article K-fold, cross-validation is used with the value of K set to ten. Dataset is divided into ten equal parts and then the training phase of the approach is applied on nine parts and tested on the remaining part. The process is repeated for each part of the K-fold dataset and the final results are the average of all ten-part results.

4.2 Performance measurement

Four performance metrics are used to evaluate the performance of proposed approach that are calculated based on equations (5), (6), (7), and (8) (Ferri *et al.*, 2009).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

4.3 Datasets used

In this research article, we have used 11 NASA MDP datasets. Datasets are freely available and downloaded from the PROMISE and OPENML repositories (Karim *et al.*, 2017),(Bischl *et al.*, 2017). Dataset names, number of attributes, number of instances, and fault percentage per dataset are given in Table 1.

Table 1. Datasets used for experimentation

Dataset Name	Language	Total attributes	Total instances	Fault percentage
CM1	C	22	498	9.83
KC1	C++	22	2109	15.45
KC2	C++	22	522	20.49
KC3	JAVA	40	458	9.38
MC1	C and C++	39	9466	0.71
MC2	C	40	161	32.29
MW1	C	38	403	7.69
PC1	C	22	1109	6.94
PC2	C	37	5589	0.41
PC3	C	38	1563	10.23
PC4	C	38	1458	12.20

PC2 is the largest dataset with 5589 instances and MC2 is the smallest dataset with only 161 instances. But MC2 has the highest fault rate 32.29% on the other hand PC2 has the lowest fault rate with only 0.41% fault-prone modules.

4.4 Results

The proposed approach in this article is developed and tested on a machine with corei5 processor and 8GB RAM. Anaconda3 is used to develop this approach and compare it with other machine learning models.

4.4.1 Accuracy

In this research article proposed approach is applied to eleven open-source NASA MDP datasets taken from the OPENML repository (Bischl *et al.*, 2017). Table 2 compares our proposed approach with decision tree variations (C4.5 and CART), random forest, k-nearest neighbors, and three variations of support vector machine in terms of accuracy performance metric. The experiment is done by setting a tolerance parameter equal to 0.05. The best results for each dataset are shown in bold letters in Table 1. Accuracy is calculated up to five decimal places.

Out of all eleven datasets, our proposed approach gives better results in the case of nine datasets, including large-scale datasets like KC1, MC1, PC2, PC3, and P4, which contain more than 1500 modules. In the case of MC1, WA-SVM and GA-SVM give similar results as our proposed approach, which is more than 99%. In the case of MC2, the k-nearest neighbor gives slightly better performance, and in the case of PC1, GAWA-SVM gives marginally better results. In both cases, our proposed approach provides the second-best performance. The last row of Table 2 shows the average accuracy comparison of all datasets. Our proposed approach gives better results than all other approaches used for comparison purposes in the case of average accuracy.

Table 2. Accuracy Comparison of the proposed approach with other machine learning approaches, the value of constant $Tolrance = 0.05$

Dataset	C4.5	GA-SVM	CART	RF	KNN	WA-SVM	GAWA-SVM	WK-Tree
CM1/22	0.78417	0.90351	0.78768	0.87311	0.9	0.90175	0.90505	**0.93333**
KC1/22	0.76352	0.85143	0.75134	0.68693	0.75941	0.84527	0.84621	**0.86729**
KC2/22	0.76403	0.79894	0.74316	0.82322	0.75271	0.81611	0.80061	**0.83439**
KC3/40	0.87568	0.90352	0.87501	0.89979	0.90646	0.90391	0.90357	**0.91304**
MC2/40	0.64007	0.67095	0.66471	0.70735	**0.71434**	0.65808	0.67022	0.71428
MW1/38	0.88087	0.91829	0.87535	0.91285	0.92285	0.92317	0.91829	**0.95041**
PC1/22	0.87139	0.93669	0.86678	0.92566	0.93	0.93403	**0.93676**	0.93093
PC2/37	0.99177	0.99588	0.99213	0.99606	0.99588	0.99588	0.99588	**0.99761**
PC3/38	0.8714	0.90144	0.86309	0.89447	0.89767	0.90018	0.90272	**0.91257**
PC4/38	0.88103	0.88206	0.87517	0.87648	0.87789	0.88337	0.87790	**0.88127**
MC1/39	0.99461	**0.99503**	0.9945	0.99408	0.71434	**0.99503**	0.99492	**0.99503**
Average	0.84714	0.88706	0.84444	0.87181	0.85195	0.88697	0.88655	**0.90274**

Figure 3 shows the comparison of the accuracy of our proposed approach with other machine learning approaches. The accuracy distribution of our proposed approach is shown by pink colored box plot, which clearly shows better performance of the proposed approach in this article than other machine learning approaches.

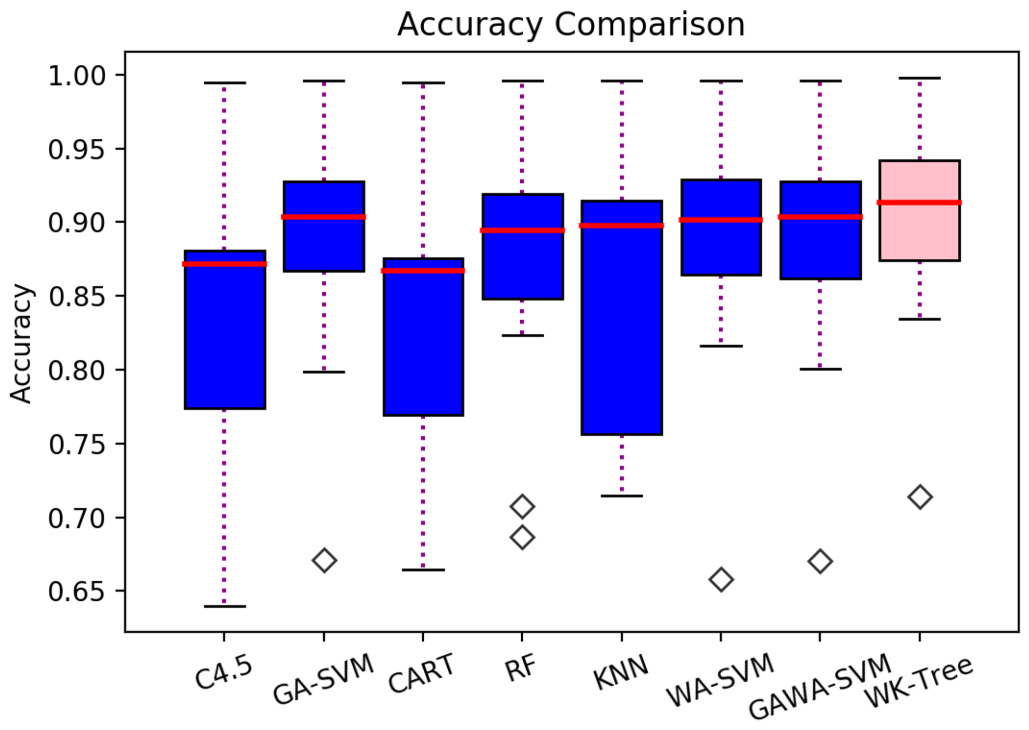

Fig. 3. Average accuracy comparison of all datasets in terms of box plots

4.4.2 Other performance metrics

Table 3 shows the average precision, recall, and f1-score of all techniques used for comparison on eleven datasets. The best performance value is shown in bold letters. For precision and f1-score performance metrics, proposed approach in this article gives better performance than all other approaches used for the comparison but in the case of recall GA-SVM and GAWA-SVM perform slightly better than our proposed approach. GAWA-SVM gives the best performance as shown in bold letters in Table 3.

Table 3. Average precision, recall, and f1-score of all techniques used for comparison

Technique	Precision	Recall	F1-Score
C4.5	0.90790	0.90950	0.89874
GA-SVM	0.89014	0.99470	0.93736
CART	0.90778	0.91206	0.89920
RF	0.88811	0.95419	0.91559
KNN	0.86958	0.96455	0.90392
WA-SVM	0.89398	0.98684	0.93670
GAWA-SVM	0.89022	**0.99223**	0.93665
WK-Tree	**0.91053**	0.98521	**0.94569**

Figure 4 shows the comparison of other metrics like precision, recall, and f1-score for all eight techniques used for comparison purposes. In this parallel bar graph, red-colored bar graphs show the precision value of different techniques, blue-colored bar graphs show recall value and green-colored bar graphs shows the f1-score comparison of all techniques used for comparison purpose.

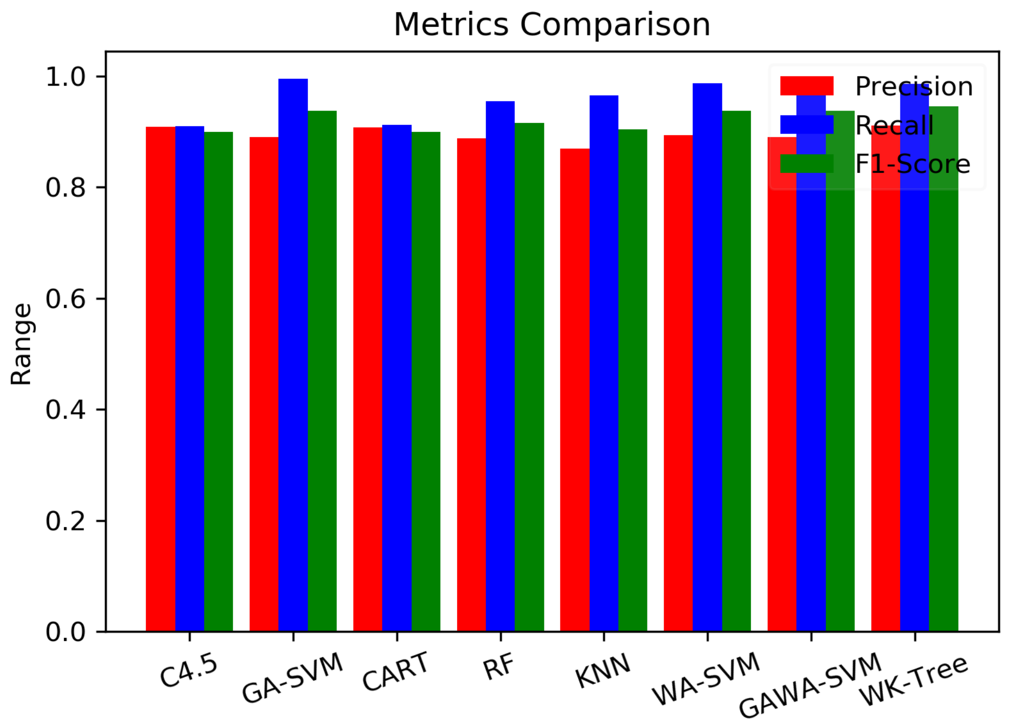

Fig. 4. Average precision, recall, and f1-score comparison of all datasets

4.4.3 Running time

The running time complexity of our proposed approach $\mathcal{O}(\log n)+c$. Figure 5 shows the comparison of running time in seconds of our proposed approach with k-nearest neighbors and weighted k-nearest neighbors.

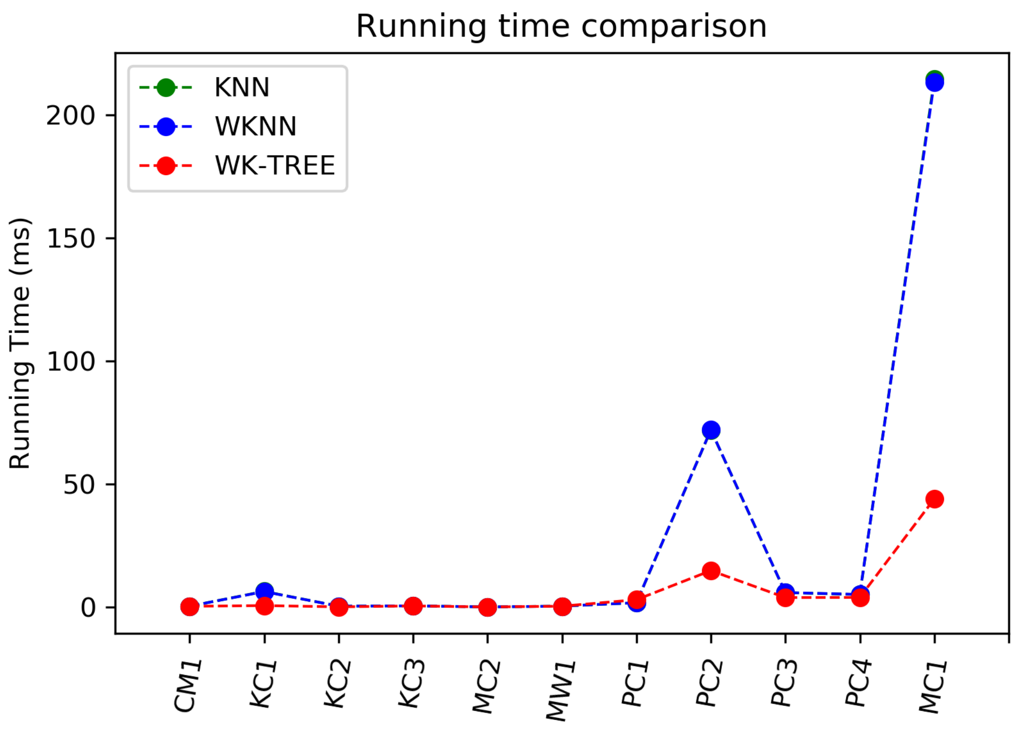

Fig. 5. Running time comparison of the proposed approach with KNN and WKNN

5. Conclusion

This article proposes a hybrid approach of pre-pruned classification and regression tree (CART) and k-nearest neighbors. The decision tree is pruned based on the distance among points in m dimensional space and leaf nodes of the decision tree store nearest neighbors of training samples on leaf nodes instead of storing class labels. The proposed approach is applied on eleven software fault prediction datasets and results are compared with eight machine learning models. Results show significant improvement in performance.

In future work, more hybrid approaches based on standard machine learning approaches can be developed to improve performance and make them work for real-life projects. Work can be done on running time complexity reduction to make the approach practical.

References

Abuassba, A. O., Dezheng, Z., Ali, H., Zhang, F., & Ali, K. (2022). Classification with ensembles and case study on functional magnetic resonance imaging. Digital Communications and Networks, 8(1), 80-86.

Alsghaier, H., & Akour, M. (2020). Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier. Software: Practice and Experience, 50(4), 407-427.

Aziz, S. R., Khan, T., & Nadeem, A. (2019). Experimental validation of inheritance metrics' impact on software fault prediction. IEEE Access, 7, 85262-85275.

Beygelzimer, A., Langford, J., & Zadrozny, B. (2008). Machine learning techniques—reductions between prediction quality metrics. In Performance Modeling and Engineering (pp. 3-28). Springer, Boston, MA.

Biau, G., & Scornet, E. (2016). A random forest guided tour. Test, 25(2), 197-227.

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., ... & Vanschoren, J. (2017). Openml benchmarking suites. arXiv preprint arXiv:1708.03731.

Cheng, D., Zhang, S., Deng, Z., Zhu, Y., & Zong, M. (2014, December). kNN algorithm with data-driven k value. In International Conference on Advanced Data Mining and Applications (pp. 499-512). Springer, Cham.

Danielsson, P. E. (1980). Euclidean distance mapping. Computer Graphics and image processing, 14(3), 227-248.

Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). An experimental comparison of performance measures for classification. Pattern recognition letters, 30(1), 27-38.

Fix, E., & Hodges, J. L. (1989). Discriminatory analysis. Nonparametric discrimination: Consistency properties. International Statistical Review/Revue Internationale de Statistique, 57(3), 238-247.

Gardner, M. W., & Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 32(14-15), 2627-2636.

Haouari, A. T., Souici-Meslati, L., Atil, F., & Meslati, D. (2020). Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction. Applied Soft Computing, 96, 106686.

Karim, S., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, B. (2017, November). Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset. In 2017 IEEE international conference on cybernetics and computational intelligence (CyberneticsCom) (pp. 19-23). IEEE.

Kassaymeh, S., Abdullah, S., Al-Betar, M. A., & Alweshah, M. (2021). Salp swarm optimizer for modeling the software fault prediction problem. Journal of King Saud University-Computer and Information Sciences.

Khan, W., Daud, A., Nasir, J.A. and Amjad, T., 2016. A survey on the state-of-the-art machine learning models in the context of NLP. Kuwait journal of Science, 43(4).

Kozma, L. (2008). k Nearest Neighbors algorithm (kNN). Helsinki University of Technology, 32.

Noble, W. S. (2006). What is a support vector machine?. Nature biotechnology, 24(12), 1565-1567.

Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

Rajkumar, N. and Palanichamy, J., 2015. Optimized construction of various classification models for the diagnosis of thyroid problems in human beings. Kuwait Journal of Science, 42(2).

Ruggieri, S. (2002). Efficient C4. 5 [classification algorithm]. IEEE transactions on knowledge and data engineering, 14(2), 438-444.

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. IEEE transactions on systems, man, and cybernetics, 21(3), 660-674.

Saravanan, P., & Sangeetha, V. (2021). African buffalo optimized multinomial softmax regression based convolutional deep neural network for software fault prediction. Materials Today: Proceedings.

Singh, M., & Chhabra, J. K. (2021). EGIA: A new node splitting method for decision tree generation: Special application in software fault prediction. Materials Today: Proceedings.

Singh, P., Pal, N. R., Verma, S., & Vyas, O. P. (2016). Fuzzy rule-based approach for software fault prediction. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47(5), 826-837.

Yucalar, F., Ozcift, A., Borandag, E., & Kilinc, D. (2020). Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability. Engineering Science and Technology, an International Journal, 23(4), 938-950.

Zhang, M. L., & Zhou, Z. H. (2007). ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition, 40(7), 2038-2048.

Zhang, S., Zong, M., Sun, K., Liu, Y., & Cheng, D. (2014, December). Efficient kNN algorithm based on graph sparse reconstruction. In International Conference on Advanced Data Mining and Applications (pp. 356-369). Springer, Cham.

Submitted: 19/01/2022
Revised: 17/03/2022
Accepted: 25/04/2022
DOI: 10.48129/kjs.18331