International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V73I7P122

Volume 73 Issue 7, 280-292, July 2025

Orginal Article

Feature Weight Based Fuzzy C-Means Clustering with Optimal Initialization for Software Fault Prediction

Yuvaraj K[1], Balaji N V[2]

[1,2]*Department of Computer Science, Karpagam Academy of Higher Education, Coimbatore, Tamil Nadu, India.*

[1]*Corresponding Author : kyuvarajj@gmail.com*

Received: 12 March 2025 Revised: 11 June 2025 Accepted: 30 June 2025 Published: 30 July 2025

Abstract - *In this digital era, software is ruling the world by making the life of humans easier and more convenient in many ways. Not only in business, but software is also required for each specific field. Software development has become a predominant and common field that provides services to every other field of science and engineering. However, the primary challenge in developing software is to identify and fix the faults that occur in various circumstances as early as possible to minimize th e time, effort, and associated inconvenience. This paper proposes an effective software fault prediction framework to identify the fault modules in software projects. The model applies accelerated k-means clustering for identifying the count of clusters by evaluating gap statistics. Then, fuzzy clustering is applied over the training set, which makes use of a probability distribution for initializing cluster centroids and feature weights to compute the similarity between the samples and the cluster centroids. As a result, samples inside the cluster are strengthened and samples outside the cluster are weakened. Moreover, it also helps to increase the quality of the clusters and accelerates the convergence of the clustering process by reducing the iterations. Using the classification model, the modules are categorized as non-defective or defective based on their high population in the relevant cluster. The effectiveness of the proposed model has been tested experimentally, and the findings show that the framework can suc cessfully identify defective software modules in less time and with higher accuracy.*

Keywords - *Accelerated k-means, Feature weight, Fuzzy c-means clustering, Probability distribution, Software fault prediction.*

1. Introduction

A software fault or defect generally refers to the failure or error that occurred in the software program or situation in which the actual results of the software are different from the expected results. In this digital era controlled by software, identifying and rectifying the defects or faults in the software before dispatching it to the customers is a critical event for software developers [1]. Various issues lead to software defects, including unclear or incorrect software requirements, incorrect design, improper coding, insufficient testing, and other environmental issues such as uncertain documentation, hardware configuration and deployment [2]. The automation of various operations in companies and industries often leads to the deployment of software systems to enhance time and quality. However, to maintain such systems, each industry has invested huge amounts of money in the information technology sector [3]. However, apart from the significance of IT projects, not all projects achieve the essence of success. According to the CHAOS Report published by the Standish Group in the year 2020, it was revealed that only 31% of the software projects were successful on time and cost with good sponsors, good teams and a good place. On the other hand, 50% of the software projects were highly challenged, and 19% were cancelled due to various failures [4].

Since software development involves a significant time and cost investment, errors or failures result in increased expenses and time and lower customer satisfaction. Thus, identifying the faulty modules in the software at an early stage helps the testers and developers to focus on specific problems to rectify the defects, thereby reducing the time, effort and resources [5]. Thus, software fault prediction has become an attractive field of research that can increase software quality and reduce the effort in the software development phase. The fault prediction helps to identify the faults promptly and deliver the software on time and at minimum cost. It identifies software faults based on the history of data and previous knowledge or information about them [6]. Numerous studies that use data mining, machine learning, soft computing, and deep learning techniques to anticipate defective software modules have been included in the literature. In general, these techniques are used in building a model that classifies the faulty modules from the non-faulty modules [7].

With simple statistical and standard Machine Learning (ML) models, prediction accuracy is compromised due to the complex and difficult relationship between the quality factors and the software metrics [5]. Numerous models to forecast faulty modules have been suggested over recent years.

Identifying the most appropriate method for predicting faults is very difficult as the performance of the models varies due to different influential factors such as software metrics used in feature extraction, data quality, classification algorithms used and so on [8]. Though all the method aims at providing an accurate prediction of faulty modules, most of the methods do not seem to be more applicable in the software industry [5]. Recently, soft computing has been most widely used in many applications, including software fault prediction. The main fact of employing soft computing is that it effectively deals with the imprecise, uncertain, and incomplete data to achieve improved performance at a low cost [9]. The most common methods used in soft computing models are Neural Networks (NN), Support Vector Machines (SVM), Fuzzy Logic (FL), evolutionary models, metaheuristics, and swarm intelligence [10].

Thus, owing to the significance of early detection of software faults, an effective software fault prediction framework has been suggested to identify the faulty modules of software projects. The model aims to perform the prediction using fuzzy clustering, in which the feature weights are also employed to enhance the results. To reduce the computational and time complexities, the initialization of cluster centroids is carried out using probability distribution and the count of clusters is initialized by performing k-means clustering with gap statistics. The model aids in proliferating the cluster quality and accelerates the convergence of the clustering process, thereby reducing the computational time.

Faulty modules on highly populated clusters have been predicted using the SVM algorithm, whereas the simple similarity measure is used in sparsely populated clusters. The efficiency of the proposed framework is approved through various results from the experimental analysis. The paper is structured as follows. Section 2 presents the associated studies on fault prediction from the literature. Section 3 discusses the background of the study. Section 4 describes the suggested model along with the overall design of the framework and algorithm pseudocodes. The experimental study, including the datasets employed, performance metrics in section 5. The result analysis, is discussed in section 6, and finally, the study is concluded in section 7.

2. Related Works

A recent review on software fault prediction intends to gain knowledge on existing models suggested to improve the prediction performance [11]. Though it is claimed by the developers that software metrics determine prediction performance, few researchers insist on the impact of the prediction model on performance [12, 13]. The ML algorithms employed in forecasting software faults include Bayesian learners [14], decision trees [15], evolutionary algorithms [16], ensemble learners [17], NN [18], SVM [19], and rule-based learning [20]. A high-performance fault predictor was suggested that makes use of ML algorithms and a computational intelligence approach. This study found that while the Naïve Bayes (NB) algorithm functioned superior with smaller datasets, the Random Forest (RF) approach showed superior results with larger datasets [21]. Since the quality of the data collected from the existing software versions has a greater impact on the performance of the prediction model, an outlier detection approach was proposed by utilizing the metrics threshold and class label for identifying the outliers. The model was proved to be more effective in performance with the NASA dataset than improved random forest (ACF) and improved naïve Bayes (ACN) algorithms [22].

For classifying the software faults, an adaptive neuro-fuzzy c-means clustering was suggested that applies a reasoning fuzzy system on the fuzzy c-means clustering. The approach lacks an in-depth examination of other datasets, while it has demonstrated efficacy on the NASA-PC1 dataset [23]. A new framework clustering was used to group similar modules, thereby reducing the irrelevant and inconsistent modules. Then, a Probabilistic Neural Network (PNN) classifier was applied for the training and test phase. With the NASA dataset, the model was proven to have an improved false negative rate than RF and NB [24]. The author also extended their work to use fuzzy clustering and majority ranking for improving prediction accuracy. They also evaluated the model with a fuzzy majority ranking approach with outliers (FMRT) and a fuzzy majority ranking approach without outliers (FMR). Additionally, it was noted that it lessens the influence of unreliable and insignificant information on prediction performance [24].

A simple model using the k-means algorithm was proposed to improve the performance. The initialization of cluster centroids was carried out by applying a novel point centre algorithm, which is then used to forecast the faulty modules. The model was more effective than the standard k-means algorithm [25]. To prioritize test cases in software projects, two fuzzy-based clustering techniques were suggested that use a new similarity coefficient and dominance measure to cluster the project. After that, prioritization would be done using the mathematical weighted sum product calculation for ranking [26]. A comprehensive evaluation of the 10 ML classifiers' performance in detecting software faults was conducted using 12 NASA datasets. From the detailed study, the Radial Basis Function (RBF), SVM, RF, and Multilayer Perceptron (MLP) offered better prediction accuracy than other simple classifiers [27]. Other than clustering, the software change metrics were also employed in predicting software faults [28]. Additionally, it was demonstrated that using code metrics and change metrics improves prediction accuracy [29].

3. Background

This section presents the basic background study on various state-of-the-art clustering algorithms.

3.1. k-means Clustering

A commonly and widely used paradigm for unsupervised learning, k-means clustering, groups related data items to identify hidden patterns. The simple and fastest algorithm iteratively groups the unlabeled dataset into k clusters by satisfying the criteria that the samples inside the clusters have high similarities and the samples outside the cluster have high dissimilarities [30]. It works with k centroids representing the centre of the k clusters, which is an imaginary location, and the samples nearest to the centroids are assigned to the respective clusters. The centroid is computed by computing the average of the samples of each cluster.

Generally, Euclidean distance is utilized for computing the separation between the centroids and the points. Though the clustering model is simple and fast, choosing arbitrary cluster centroids at the initialization phase leads to a bad approximation when compared with optimal clustering. Algorithm 1 demonstrates the algorithm's pseudocode.

Algorithm 1: means clustering

Input: D={x_1, x_2, …., x_n}
Output: k clusters
Procedure kmeans ()
  //Initialization of k clusters
  Arbitrarily choose the initial k cluster centroids
  C = {c_1, c_2, …. , c_k}.
  Repeat
    Assign each datapoint d_i to the cluster C_j that is closest
    to its centroid, c_j
    Update the cluster centroids, c_j, by computing the means of the samples $c_j = \frac{1}{|c_j|}\sum_{x \in c_j} x$
    Until the convergence criteria (no change in the assignment) are met
    Return k clusters
    End Procedure

3.2. k-means++ Clustering

A specific and effective way of choosing the first centroids of the k-means method was proposed, which offers a substantial improvement in the final error of clustering [31]. Here, the first cluster centroid is selected at random, and the other cluster centroids are selected based on a probability based on their squared distance from the samples that are nearest to the existing cluster centre.

With this, k-means as the base, several variations on the algorithm were suggested to decrease the computational complexity and improve the clustering accuracy. The triangle inequality was included while computing the distance between the samples and centroids, reducing the computations [32]. To make it even faster, the bound is applied to reduce the distance computations [33]. Algorithm 2 displays the k-means++ clustering algorithm.

Algorithm 2: k-means++ clustering

Input: D={x_1, x_2, …., x_n}
Output: k initial cluster centroids
Procedure kmeans++()
  Arbitrarily choose the cluster centroid c1 uniformly at random among datapoints x_n.
  Repeat until k centroids are chosen:
    For each sample x other than x_n,
      Compute d(x), the distance between x and the nearest centroid.
    Choose a point as the new centroid x using weighted probability distribution as $\frac{d(x)^2}{\sum_{x \in D} d(x)^2}$
  //Proceed with the standard k-means algorithm for performing clustering
End Procedure

3.3. Fuzzy c-means Clustering

The subject of fuzzy sets was familiarized with cluster analysis in 1973 [34]. With this as a base, various other developments have been suggested, and it has become the most frequently employed algorithm in various applications. The main thing is that the method performs soft clustering, in which, unlike assigning each sample into a specific cluster, each sample ends with a fuzzy membership that specifies the membership degree concerning each cluster [35]. With a dataset $X = \{x_1, x_2, \dots, x_n\}$ into c clusters where $x_i = \{x_{i1}, x_{i2}, \dots x_{il}\}$ It is a dimensional object. $\mathcal{V} = \{\vartheta_1, \vartheta_2, \dots, \vartheta_c\}$ Is the c cluster centroids where $\vartheta_k = \{\vartheta_{k1}, \vartheta_{k2}, \dots \vartheta_{kl}\}$ l dimensions of the k^{th} cluster and $\mathcal{U} = (\mu_{ik})_{n \times c}$ is a fuzzy partition matrix, μ_{ik} ($1 \le i \le n$), ($1 \le k \le c$) represents the membership degree of the i^{th} sample in the k^{th} cluster. Thus, as stated in Equation 1, the clustering algorithm seeks to minimize the objective function.

Minimize

$$J_m(\mathcal{U}, \mathcal{V}) = \sum_{i=1}^{n}\sum_{k=1}^{c} \mu_{ik}^m d_{ik}^2 \quad (1)$$

Subject to $\mu_{ik} \in [0,1]$; $\sum_{k=1}^{c}\mu_{ik} = 1 \forall i = 1,2, \dots n; 0 < \sum_{i=1}^{n}\mu_{ik} < n \forall n$

Here, d_{ik} represents the closeness between the samples. x_i and ϑ_k Which is calculated employing the Euclidean distance computed using $d_{ik} = \|x_i - \vartheta_k\|$ And m represents the fuzzier parameter that determines the cluster fuzziness [36].

The membership becomes increasingly ambiguous as m increases, and though it was suggested that m can range between (1.5, 2.5), the optimum value recommended is $m=2$ [37, 38]. The objective function Jm is minimized by computing the values of $\mathcal{U}$ and $\mathcal{V}$ as in Equations 2 and 3.

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c}(\frac{d_{ik}}{d_{ij}})^{\frac{2}{m-1}}} \quad (2)$$

$$\vartheta_k = \frac{\sum_{i=1}^{n} \mu_{ik}^{m} x_i}{\sum_{i=1}^{n} \mu_{ik}^{m}} \quad (3)$$

Apart from these state-of-the-art clustering algorithms, various other algorithms were also proposed. Fuzzy c-means++ is a seeding algorithm that initializes the cluster centroids by dispersing them across the data space [39]. Instead of Euclidean distance, Canberra weighted distance was proposed to enhance the FCM algorithm's effectiveness [40]. The idea behind the membership scaling FCM was to make the samples close to the centroid converge while blocking the convergence of the other samples [41]. The algorithm pseudocode is presented in Algorithm 3.

Algorithm 3: Fuzzy c-means clustering (FCM)

Input: dataset $X = \{x_1, x_2, \ldots, x_n\}$, cluster count c, fuzzy exponent m, termination limit ε and maximum iteration I_{max}
Output: Membership degree matrix $\mathcal{U}$ and $\mathcal{V}$
Procedure fuzzy_c_means()
Initialize membership degree matrix $\mathcal{U}^{(0)}$ With random values meeting the constraints.
Set t=1
  Repeat
    Calculate the cluster centroids $\mathcal{V}^{(t)}$
    For each i from 1 to n
      Compute $d_{ik} = \|x_i - \vartheta_k\|$ for $1 \le i \le n$, $1 \le k \le c$;
    Calculate the objective function $J_m^{(t)}$
    Update the membership degree matrix. $\mathcal{U}^{(t+1)}$
  Until $|J_m^{(t)} - J_m^{(t-1)}| < \varepsilon$ or ($t > I_{max}$)
End Procedure

4. Proposed Clustering Framework

The proposed model aims at utilizing clustering techniques for predicting the software fault modules. The process is broken down into two basic stages: the first is training, and the second is testing. During the training phase, the preprocessing is carried out on the training samples. Here, the recommended model uses accelerated k-means clustering based on harmonic mean with a seeding strategy and gap statistics to find an optimal count of clusters. It employs fuzzy c-means clustering to group related modules after the ideal k value for the number of clusters has been determined. The clusters are separated into two categories: high-population and low-population, in which only high-population clusters are trained with the SVM classifier to classify the samples as faulty or non-faulty. During the testing phase, the prediction is simply carried out by comparing the test sample with the samples of the low-population clusters to determine if the test sample is similar. Conversely, the SVM classification forecasts the high population clusters' non-faulty samples. The general layout of the suggested prediction model is depicted in Figure 1.

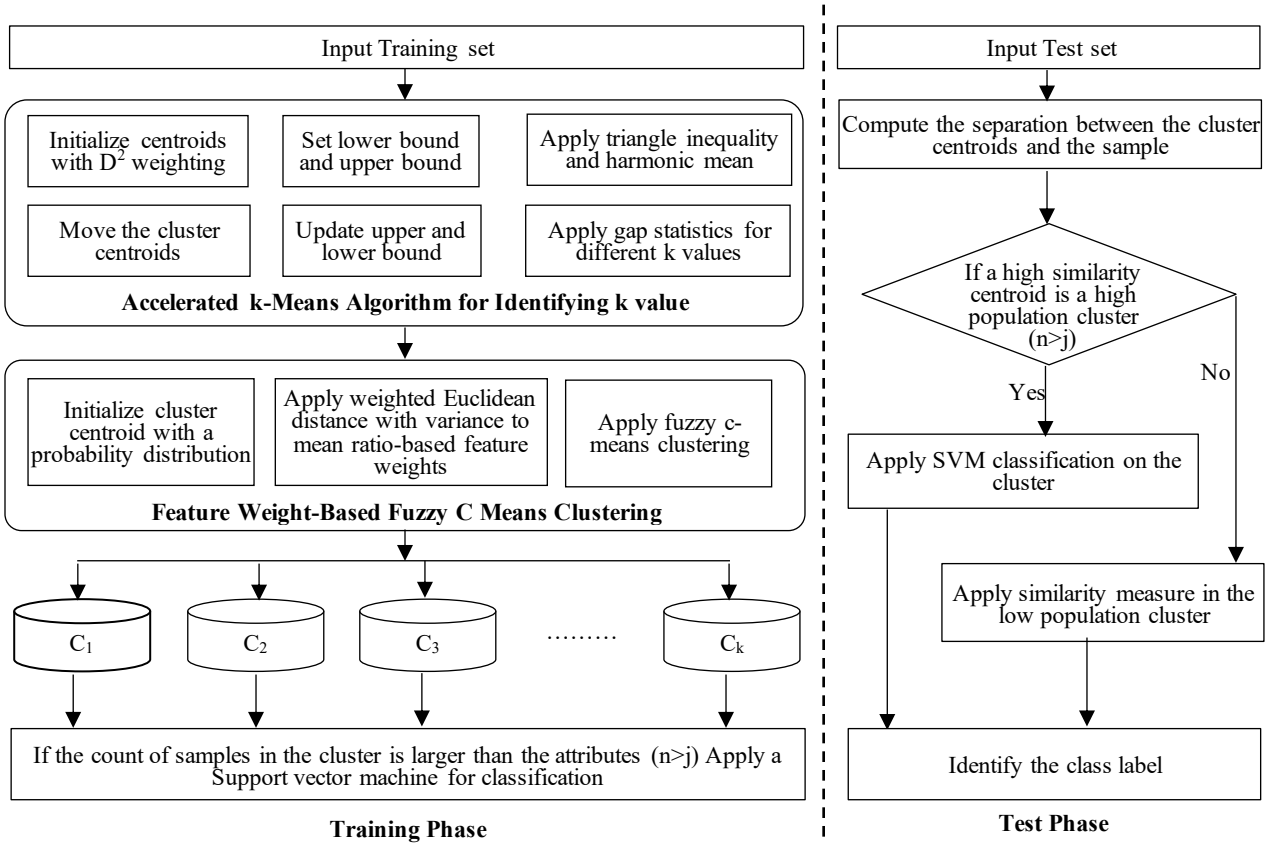

Fig. 1 Overall framework of the proposed prediction model

4.1. Accelerated k-means Algorithm

In the proposed model, the accelerated k-means approach that employs distance bounds and triangle inequality proposed by Elkan [32] has been used with a simple modification. This algorithm highly reduces the computations by avoiding point-centred distance calculations and applying the triangle inequality.

It achieves the speed by storing the upper limit representing the distance to the centroid to which it is assigned, and the lower limit representing the distance to all the cluster centroids. It verifies whether the point is closer to any other centroids before computing the distance. This is carried out by applying the triangle inequality as $d(x,z) \leq d(x,y)\ d(y,z)$. It uses the two conditions that $d(c,c') \geq 2d(x,c)$ then $d(x,c') \geq d(x,c)$ and $d(x,c') \geq max\{0,\ d(x,c)-d(c,c')\}$ where the centroids are c and c', and x is a point.

Thus, if the higher limit of x and c is less than the lower limit of x and c' or if the higher limit is less than half the distance between c and c', then c' cannot be the closest point, and thus the calculation can be avoided. The algorithm is utilized in the proposed model since it outperforms other models in high-dimensional space.

Instead of random initialization, the seeding approach has been utilized in the proposed model. Also, to perform cluster centroids, the harmonic mean, an advanced algebraic method, is applied as it is more reliable and not affected by the sample variations. The formula for the harmonic mean of n points is computed as in Equation 4.

$$h_\mu = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \left(\frac{\sum_{i=1}^{n} x_i^{-1}}{n}\right)^{-1} \quad (4)$$

Also, the model utilizes a seeding approach instead of choosing random centroids initially. Here, only the first centroid is chosen randomly, and the consecutive centroids are initialized based on the probability computed with the squared distance to their closest centroid [31]. Algorithm 4 displays the pseudocode for the algorithm.

Algorithm 4: Accelerated k-means Algorithm

Input: dataset χ, the k value
Output: k clusters

```
Procedure acc_kmeans()
    //Initialize k cluster centroids
    Choose the first cluster centroid c1 uniformly at random
     from χ
    For j from 2 to k
        cj ← x ϵ χ with probability D(x)^2 / Σ_{x∈χ} D(x)^2
    //Set lower and upper bounds
    Lower limit l(x,c)=0 ∀ x∈χ and c∈k
    For all x ∈χ do
        Assign x to the closest initial centroid c(x)=argmin_c
        ||x − c||
        Lower limit l(x,c) = ||x − c||
        Upper limit u(x)=min_c ||x − c||
    While not converged, do
        Compute ||c − c'||, ∀c, c' ∈ k
        Compute s(c)= (1/2) min_{c≠c'} ||c − c'|| ∀c, c' ∈ k
        For all x ∈χ do
            If u(x)≤s(c(x)) then continue with next x
            r=Ture
            for all c ∈k do
                z = max(l(x, c), ||c(x) − c||/2)
                if c=c(x) or u(x)≤z then continue with next c
                If r then
                    u(x)= ||x − c(x)||
                      r=False
                      If u(x) ≤z, then continue with the next c
                l(x, c) = ||x − c||
                if l(x,c)<u(x) then c(x)=c
        //Update the centroid value
        For all c in K do
            m(c)= h_μ (c) as in Equation 1
        //Update upper and lower limits
        For all x in χ do
            u(x)=u(x)+ ||m(c(x)) − c(x)||
              For all c in k do
                l(x,c)=max(l(x,c)- ||c − m(c)||, 0)
```

4.2. Gap Statistics

The accelerated k-means algorithm is executed by varying the k values from 2 to $\sqrt{|\chi|}$. With k clusters $c_1, c_2, c_3, \ldots, c_k$, in which n_r represents $|c_r|$, the number of samples in cluster c_r.

Next, the gap statistics are computed to determine the count for the clusters [42]. The formula to compute the gap statistics is given in Equation 5.

$$Gap_n(k) = E_n^*\{\log(W_k)\} - \log(W_k) \quad (5)$$

Where, E_n^* represents the expectation of sample size n and the formula to compute the W_k is given in Equation 6.

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r} \sum_{i,i' \in c_r} \sum_j (x_{ij} - x_{i'j})^2 \quad (6)$$

With this idea, the gap statistics are implemented in the following steps.

1. Cluster the observed data into k clusters for which W_k is evaluated.
2. Create the B reference datasets by consistently generating each reference feature across the feature's observed value range, then calculate W_{kb}^*.

3. Estimate the gap statistics $Gap(k) = \left(\frac{1}{B}\right)\sum_b \log(W_{kb}^*) - \log(W_k)$
4. Compute standard deviation $sd_k = [\left(\frac{1}{B}\right)\sum_b \{\log(W_{kb}^*) - \bar{l}\}^2]^{1/2}$ where $\bar{l} = \left(\frac{1}{B}\right)\sum_b \log(W_{kb}^*)$ and define $S_k = sd_k\sqrt{(1+\frac{1}{B})}$
5. Select the smallest k such that Gap(k)≥Gap(k+1)- S_{k+1}, for the cluster count, k.
6. Select the least of the clusters. $\hat{k}$ As the smallest k.

4.3. Feature Weighting-Based Fuzzy C-Means Clustering

This section presents the fuzzy c-means clustering for feature weighting, inspired by the model suggested by Setyawan and Ilham [40]. It uses weighted Euclidean distance, incorporating each feature's feature weights and variance-to-mean ratio in identifying the similarity between the samples and the cluster centroids.

The main issues in fuzzy c-means clustering are the random initialization and the distance measures used in the clustering process, which often lead to an ineffective result.

Thus, the fuzzy c-means++ approach was employed in the proposed model to initialize the spread-out representative samples as cluster centroids with which the membership matrix can be assigned a reasonable value [39]. When the clustering process starts with a better position close to the real values, the number of iterations to attain the convergence will be very low.

In this process, the first cluster centroid is chosen randomly and added to the set $\mathcal{V}$. With this point, the probability distribution for all the points x_i In the dataset is evaluated, and the point having a larger distance has a higher chance of being the next centroid. The probability distribution is computed as in Equation 7.

$$pb = \frac{d^p(x,\mathcal{V})}{sum(d^p)} \tag{7}$$

Here $d^p(x,\mathcal{V})$ represents the distance between the sample x to the nearest centroid in $\mathcal{V}$. p signifies the spreading factor, in which lower values pick the nearest point as the cluster centroid, whereas higher values pick the outlier as the cluster centroid.

Thus, p = 1.8 could be chosen as a factor for achieving better performance. This process continues until the initial cluster centroids are identified, with which the membership matrix is computed as in Equation 2.

However, the distance between the points is computed using Euclidean distance and feature weighting. This is carried out in three steps by computing the variance-to-mean ratio, weights for the features and distance with feature weighting [40]. As each feature possesses a varying degree of relevance to the clustering process, the weight for each feature in the dataset is computed using variance-to-mean ratio, a statistical measure to compute the feature weights has been employed in the model [43].

The formula to compute the variance-to-mean ratio is evaluated in Equation 8.

$$V_j = \frac{s_j^2}{\bar{x_j}} \Rightarrow \frac{\frac{1}{n-1}\sum_{i=1}^n (x_{ij}-\bar{x_j})}{\bar{x_j}} \tag{8}$$

Here s_j^2 represents the variance of the features in the dataset, n represents the dataset's sample count, and $\bar{x_j}$ represents the mean of each feature and is computed as $1/n\sum_{i=1}^n x_{ij}$

Following the computation of the variance-to-mean ratio, the feature weight from the fuzzy set is determined using Equation 9.

Here $\mathcal{s}$ and ℓ represent the smallest and largest variance-to-mean ratio of the features, and w_j signifies the feature weights. This filter method selects the significant features, thereby eliminating features with weights of 0.

$$w_j = \begin{cases} 0 & V_j = \mathcal{s} \\ \frac{V_j - \mathcal{s}}{\mathcal{s}-\ell} & \mathcal{s} < V_j < \ell \\ 1 & V_j = \ell \end{cases} \tag{9}$$

With the computed weights, Equation 10 is used to calculate the distance between the two points, where l represents the number of features and x_{ij} and ϑ_{kj} Signify the ith sample and the kth centroid.

$$d_{ik}^w = \sqrt{\sum_{j=1}^l w_j \left(x_{ij} - \vartheta_{kj}\right)^2} \tag{10}$$

Thus, the weighted distance can redefine the membership matrix and objective function, as shown in Equations 11 and 12.

Minimize

$$J_m(\mathcal{U},\mathcal{V}) = \sum_{i=1}^n \sum_{k=1}^c \mu_{ik}^m d_{ik}^{w2} \tag{11}$$

$$\mu_{ik} = \frac{1}{\sum_{j=1}^c (\frac{d_{ik}^w}{d_{ij}^w})^{\frac{2}{m-1}}} \tag{12}$$

The overall working procedure for the proposed feature weight-based fuzzy c-means clustering is shown in Figure 2.

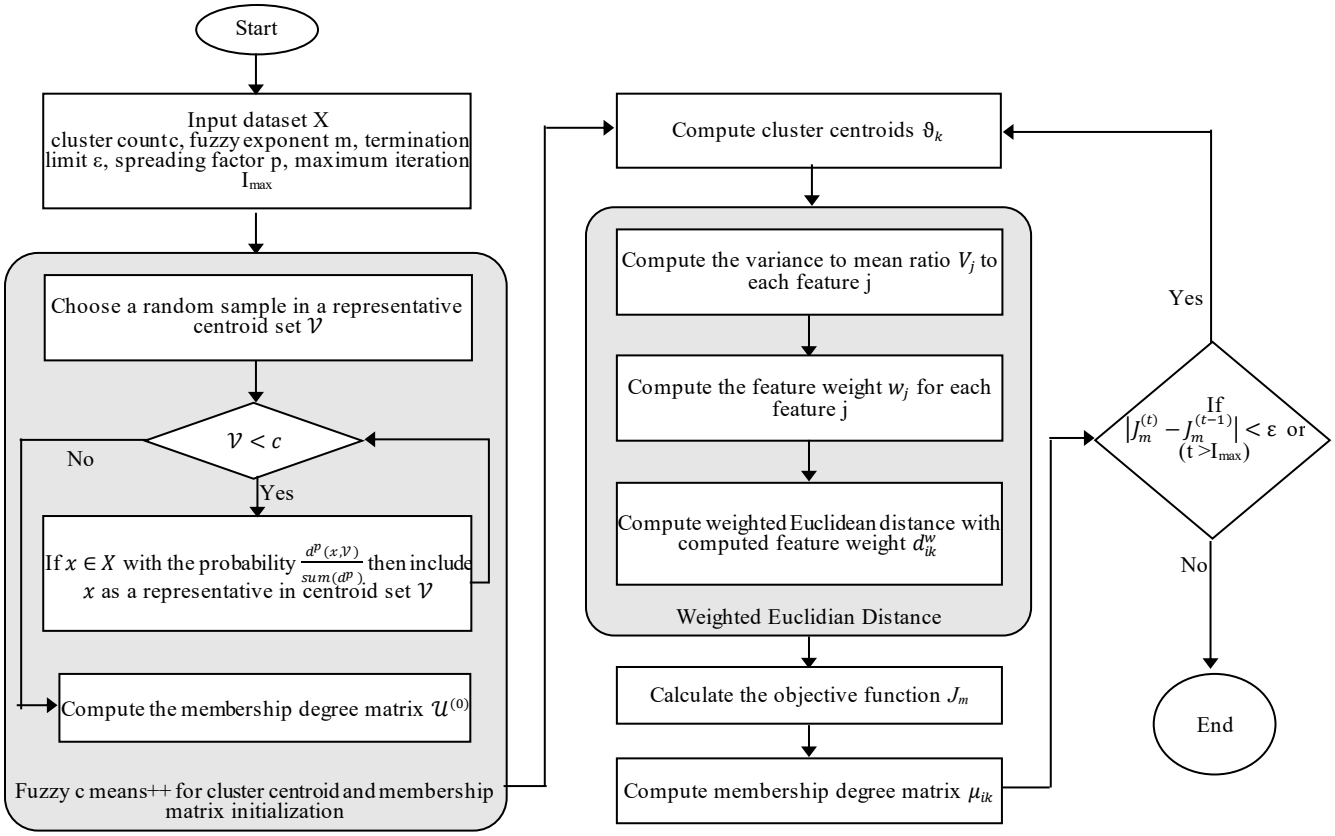

Fig. 2 Overall working procedure of the proposed clustering

Thus, the pseudocode for the proposed feature weight-based fuzzy c-means clustering is presented in Algorithm 5.

Algorithm 5: Feature weight-based Fuzzy c-means clustering

Input: dataset X = $\{x_1, x_2, \ldots, x_n\}$, cluster count c, fuzzy exponent m, termination limit ε, spreading factor p, and maximum iteration I_{max}
Output: Membership degree matrix $\mathcal{U}$ and $\mathcal{V}$
Procedure fuzzy_c_means()
Initialize_cluster_centroids (dataset X, cluster c)
Compute the membership degree matrix $\mathcal{U}^{(0)}$
Set t=1
  Repeat
    Calculate the cluster centroids $\mathcal{V}^{(t)}$
    For each i from 1 to n
      Compute d_{ik}^w for $1 \le i \le n$, $1 \le k \le c$
    Compute the objective function $J_m^{(t)}$
    Update the membership degree matrix. $\mathcal{U}^{(t+1)}$
  Until $|J_m^{(t)} - J_m^{(t-1)}| < \varepsilon$ or (t >I_{max})
End Procedure

Procedure Initialize_cluster_centroids(X, c)
$\mathcal{V} = \mathcal{V} \cup random(x)$
While $|\mathcal{V}| < c$ do
    For each $x \in$ X
      If x with probability $\frac{d^p(x,\mathcal{V})}{sum(d^p)}$
        $\mathcal{V} = \mathcal{V} \cup x$
      End For
End While
End Procedure

Following the clustering of the training samples using the proposed fuzzy c-means clustering with feature weight, which divides the clusters into high-population and low-population categories. The process involves comparing the clusters' sample count and the attribute count. The high-population clusters are then trained using the most popular and effective algorithm, the SVM classifier. It is a supervised ML classification that utilizes the kernel trick to transform the given data to find an optimal boundary between the target classes. SVM is an appropriate choice for various classification problems, especially with high-dimensional spaces and memory efficiency. It also helps to identify the complex associations with the data samples without performing difficult transformations.

In the test phase, the input test module is compared with the cluster centroids by evaluating the Euclidean distance between them. The test sample that has a minimum distance from the cluster centroid is then included in the respective cluster. Then, the cluster population is evaluated, and if the selected cluster has a dense population, the SVM classifier is

applied to the specific cluster to identify the class label as defect or non-defect samples. On the other hand, if the specific cluster has had a low population in which the sample count is less than the attribute count, then the Euclidean distance is utilised to evaluate the similarity among the test sample and the other samples in the cluster.

Here, the top three similar modules with the minimum distance are taken for the analysis. The majority of the top three sample labels will be chosen as the class label for the test sample.

5. Experimental Study

This section presents the experimental analysis carried out for the suggested research. Initially, the datasets utilized for the analysis are described. The various performance analyses used to evaluate the study are also discussed. The experiments are performed in Rstudio installed on an Intel (R) Pentium CPU 6405U with a 64-bit Windows operating system and 8 GB RAM.

5.1. Dataset Description

Five NASA datasets from the PROMISE repository were employed for evaluation. The five datasets are CM1, PC1, KC1, KC2, and JM1, in which all the datasets contain 22 module-based measures as features that include 5 different Lines of Code (LOC), 3 McCabe metrics, 4 bases and 8 derived Halstead measures, a branch-count, and 1 goal field. Thus, software metrics are used to evaluate the modules and the outcomes of these measures are used as features. Commonly used software metrics include Halstead software metrics, which measure base and derived measures in addition to lines of code, and McCabe software metrics, which measure essential complexity, cyclomatic complexity, design complexity, and lines of code. Details regarding the datasets are provided in Table 1.

Table 1. Dataset from PROMISE repository

Datasets	Details	Programming Language	#Samples	#Non-Defects	#Defects	Defect%
CM1	NASA spacecraft instrument	C	498	449	49	10
PC1	Software for satellites in Earth-orbiting flights	C	1109	1032	77	7
KC1	Ground data storage and processing	C++	2109	1783	326	15
KC2	Science data processing	C++	522	105	415	20
JM1	Predictive ground system in real time	C	10,885	8779	2106	19

NASA Metrics Data Program (MDP) datasets are widely available in different versions. The cleaned version of the datasets was created by Shepperd et al. [44]. It is available as a dataset with duplicate and inconsistent samples and a dataset with duplicate and inconsistent samples removed. The nine datasets from dataset versions such as PC1, PC2, PC3, PC4, MW1, CM1, KC1, KC3, and MC2 have been employed and are available on GitHub. The particulars of the cleaned NASA MDP datasets are presented in Table 2.

Table 2. NASA MDP datasets

Datasets	# Features	# Samples	# Non-Defects	# Defects	Defect %
PC1	37	759	698	61	8
PC2	36	1585	1569	16	1
PC3	37	1125	985	140	12
PC4	37	1399	1221	178	13
MW1	37	264	237	27	10
CM1	37	344	302	42	12
KC1	21	2096	1771	325	16
KC3	39	200	163	36	19
MC2	39	127	83	44	35

5.2. Performance Metrics

In general, the prediction carried out by the models is evaluated by generating the confusion matrix for the N samples grouped as True Positives (TP), False Negatives (FN), False Positives (FP) and True Negatives (TN). TP and TN represent correct classification as the count of defective test instances classified as faulty and non-faulty test instances classified as non-faulty.

FN and FP represent incorrect classification as the count of defect test samples classified as non-defective and the count of non-faulty test samples predicted as defective. With these four elements of the confusion matrix, various performance measures can be evaluated.

However, according to Equations 13 to 16, the study makes use of measures such as Overall Error Rate (OER), False Positive Rate (FPR), False Negative Rate (FNR) and the Rand index.

$$OER = \frac{FN+FP}{N} \tag{13}$$

$$FPR = \frac{FP}{FP+TN} \tag{14}$$

$$FNR = \frac{FN}{TP+FN} \tag{15}$$

$$Rand\ Index = \frac{TP+TN}{N} \tag{16}$$

6. Results Analysis

This section discusses the various results obtained for the anticipated model and compares the results with those of the other existing models. The first stage is to evaluate and assess the outcomes of the proposed Accelerated k-means algorithm (AkM) with those of other existing methods, including the Point Centre k-means algorithm (PCKM) and the k-means algorithm (kM) [25]. The prediction of defects has been analyzed with defect prediction rate and defect miss rate for these models, and the results are presented in Table 3, in which the high prediction rate and the low miss rate are highlighted in boldface. Here, the defect prediction represents the percentage of defects predicted, and the detection miss rate represents the percentage of defects missed by the models.

Table 3. Comparative analysis of defect prediction

Datasets	Defect Prediction %			Defect Miss %		
	kM	PKCM	AkM	kM	PKCM	AkM
PC1	98.36	98.36	**100.00**	1.64	1.64	**0**
PC2	93.75	93.75	**100.00**	6.25	6.25	**0**
PC3	**100.00**	**100.00**	**100.00**	**0**	**0**	**0**
PC4	96.07	96.07	**97.19**	3.93	3.93	**2.81**
MW1	77.78	92.59	**96.30**	22.2	7.41	**3.7**
CM1	**100.00**	**100.00**	**100.00**	**0**	**0**	**0**
KC1	87.08	87.08	**88.31**	12.9	12.9	**11.7**
KC3	80.56	**86.11**	**86.11**	19.4	16.7	**13.9**
MC2	88.37	86.36	**88.64**	14	13.6	**9.09**

From the analysis, the defect prediction rate for PC1, PC2, PC3 and CM1 with the proposed model is 100%. For the other datasets, the proposed model offers an improved defect prediction rate compared to the other existing models. The average defect prediction rate for the proposed model is 95.2% whereas that of the k-means and PCKM models is 91.3% and 93.4% respectively. Similarly, the defect miss rate is highly appreciable for the proposed model, with an average of 4.6% less than k-means (8.9%) and PCKM (6.9%) models. The obtained defect miss rate is visualized using a line graph in Figure 3 for better understanding.

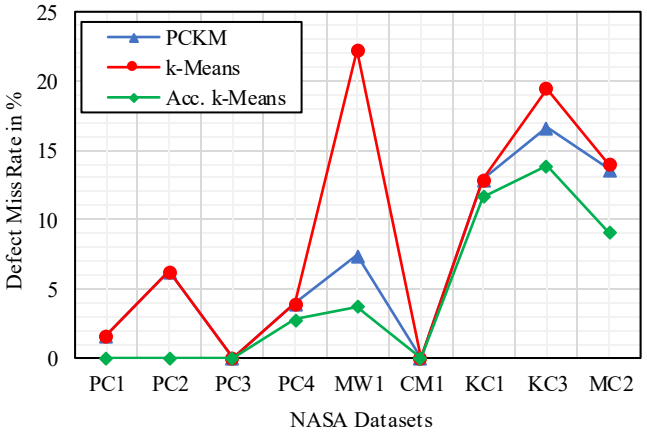

Fig. 3 Comparison of defect miss rate

Using the nine NASA MDP datasets, the OER, FPR, FNR, and the Rand Index are assessed for the k-means (kM), PKCM, and proposed AkM in addition to the defect prediction analysis. Table 4 displays the observed values from the analysis. It is evident from the elaborated outcome that the suggested accelerated k-means model outperformed the conventional kM and PCKM models in terms of error rate, false prediction, and Rand index.

Table 4. Comparative analysis using NASA MDP datasets

Datasets	OER			FPR			FNR			RAND		
	kM	PKCM	AkM	kM	PKCM	AkM	kM	PKCM	AkM	kM	PKCM	AkM
PC1	0.003	0.003	0	0.001	0.001	0	0.016	0.016	0	0.997	0.997	1.000
PC2	0.005	0.008	0.003	0.004	0.008	0.003	0.063	0.063	0	0.995	0.992	0.997
PC3	0.001	0.001	0	0.001	0.001	0	0	0	0	0.999	0.999	1.000
PC4	0.015	0.019	0.009	0.011	0.016	0.006	0.039	0.039	0.028	0.985	0.981	0.991
MW1	0.030	0.125	0.008	0.025	0.114	0.004	0.074	0.222	0.037	0.970	0.875	0.992
CM1	0.006	0.006	0.003	0.007	0.007	0.003	0	0	0	0.994	0.994	0.997
KC1	0.035	0.035	0.031	0.018	0.018	0.015	0.129	0.129	0.117	0.965	0.965	0.969
KC3	0.055	0.075	0.040	0.031	0.049	0.018	0.162	0.194	0.139	0.945	0.925	0.960
MC2	0.055	0.055	0.031	0.012	0.012	0	0.136	0.136	0.093	0.945	0.945	0.969

Thus, the average error rate, FPR, and FNR for proposed accelerated k-means are 0.014, 0.005 and 0.046, which is less than standard k-means that have OER as 0.036, FPR as 0.025, FNR as 0.089 and PCKM that has OER as 0.023, FPR as 0.012 and FNR as 0.069. On the other hand, the average rand index for the standard k-means, PCKM, and proposed accelerated k-means are 0.964, 0.977 and 0.986, respectively. Thus, accelerated k-means clustering is a better choice for predicting software faults.

The PROMISE Repository's five datasets—KC1, KC2, CM1, PC1, and JM1—are used to conduct the overall analysis of the suggested model. A comparison is made between the suggested model's (FW_FCM+SVM) performance with the other models with conventional classifiers such as NB, RF, and other existing models such as ACN [22], ACR [22], fuzzy c-means clustering with probabilistic neural network (FCM_PNN) [5], FMRT [24], and FMR [24]. The obtained outcomes are presented in Table 5.

Table 5. Comparative analysis with different prediction models

Datasets	Metrics	NB	RF	ACN	ACF	FCM_PNN	FMRT	FMR	FW_FCM+SVM
KC1	FPR	0.09	0.06	0.07	**0.01**	0.19	0.06	0.03	0.02
	FNR	0.63	0.73	0.16	0.31	0.21	0.74	0.13	**0.11**
	OER	0.17	0.16	0.06	**0.03**	0.17	0.16	**0.03**	0.04
KC2	FPR	0.05	0.09	0.06	**0.03**	0.25	0.06	0.04	**0.03**
	FNR	0.53	0.53	**0.1**	0.27	0.12	0.52	0.18	0.15
	OER	0.15	0.18	0.07	0.06	0.23	0.17	0.06	**0.05**
CM1	FPR	0.08	0.02	0.05	0.02	0.04	0.04	**0.01**	0.02
	FNR	0.71	0.81	0.29	0.42	0.45	0.61	0.37	**0.21**
	OER	0.14	0.09	0.07	0.05	0.28	0.16	0.04	**0.03**
PC1	FPR	0.08	0.02	0.05	0.02	0.17	0.04	**0.01**	0.02
	FNR	0.71	0.81	0.29	0.42	0.38	0.61	0.37	**0.24**
	OER	0.14	0.09	0.07	0.05	0.19	0.16	**0.04**	**0.04**
JM1	FPR	0.08	0.02	0.05	0.02	0.08	0.04	**0.01**	0.02
	FNR	0.71	0.81	0.29	0.42	0.18	0.61	0.37	**0.23**
	OER	0.14	0.09	0.07	0.05	0.14	0.16	0.04	**0.03**

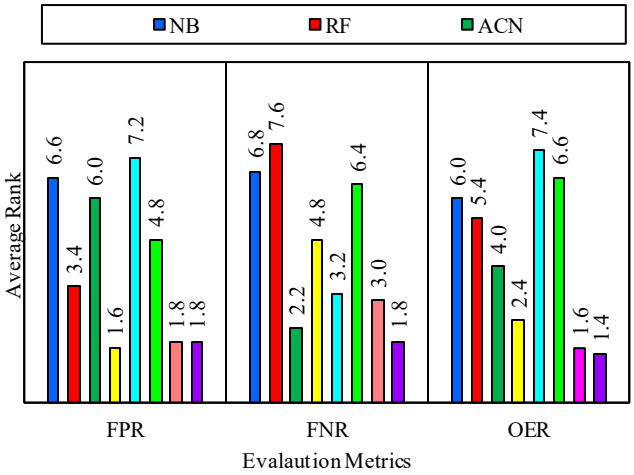

Fig. 4 Comparison of average ranks of various models

While analyzing the FPR, the ACF provides better results for KC1 and KC2 datasets and the FMR model for CM1, PC1 and JM1 datasets. On the other hand, the proposed FW_FCM+SVM provides improved FNR for KC1, CM1, PC1 and JM1 datasets and ACN for the KC2 dataset. The FMR model offers less OER for KC1 and the proposed FW_FCM+SVM for KC2, CM1, PC1 and JM1 datasets. Thus, the average ranks are computed for the 8 models compared, and the obtained value is depicted in Figure 4. The result indicates that the ACF model offers better FPR with an average rank of 1.6. The FMR model and proposed models acquired a second top position concerning the FPR. Meanwhile, the FNR and OER for the proposed model are better, with an average rank of 1.8 and 1.4, respectively.

Table 6. Analysis with different test and training sets

Test set	Models	FPR	FNR	EOR
CM1	ACN	0.03	0.41	0.06
	ACF	**0.02**	0.44	0.05
	FMR	**0.02**	0.48	0.05
	FW_FCM+SVM	**0.02**	**0.32**	**0.03**
KC1	ACN	**0**	0.71	0.08
	ACF	0.01	0.5	0.07
	FMR	**0**	0.63	0.07
	FW_FCM+SVM	0.01	**0.36**	**0.04**
KC2	ACN	**0**	0.54	0.12
	ACF	0.02	0.44	0.11
	FMR	**0**	0.48	0.1
	FW_FCM+SVM	0.01	**0.32**	**0.1**
PC1	ACN	**0.02**	0.39	0.04
	ACF	**0.02**	0.33	0.04
	FMR	**0.02**	0.33	0.04
	FW_FCM+SVM	**0.02**	**0.27**	**0.03**

Finally, as discussed by Abaei and Selamat [5], the various models are analyzed with different training and test datasets. The models, such as ACN, ACF, FMR and proposed FW_FCM+SVM, are evaluated with JM1 as the training set and CM1, KC1, KC2 and PC1 as test datasets. The results obtained are presented in Table 6. Using CM1 as the test sample and JM1 as the learning sample, the suggested model produces better results with lower FPR, FNR, and OER. The average FPR for ACN, ACF, FMR and FW_FCM+SVM are 0.013, 0.18, 0.10 and 0.15, respectively. Thus, the FMR has enhanced FPR results. The FNR values for ACN, ACF, FMR and FW_FCM+SVM are 0.513, 0.428, 0.48 and 0.318, respectively, whereas the OER values for ACN, ACF, FMR and FW_FCM+SVM are 0.075, 0.068, 0.065 and 0.05,

respectively. Thus, the FNR and OER for the proposed model with different datasets have better results, indicating that they effectively detect faults from non-faulty modules.

7. Conclusion

Software fault prediction has gained more attention among researchers who apply predictive analysis in software development. Yet, the models still need further enhancement with high precision and low time complexity. This paper suggests the feature weight-based fuzzy c-means clustering for predicting faulty modules from non-faulty ones. The method clusters the modules based on their characteristics using fuzzy feature weights and then applies a classification algorithm to the high-populated clusters to predict software faults and simple similarity measures to classify them in low-populated clusters. This method improves fault detection and reduces computational complexity. The various experimental analysis have been carried out using 5 datasets from the PROMISE repository and 9 datasets from NASA MDP. The outcome shows that compared to other conventional and existing fault prediction approaches, the suggested model offers a better FNR and OER. The result analysis shows that the proposed model is appropriate for predicting software faults.

References

[1] Norah Abdullah Al-Johany et al., "Static Analysis Techniques for Fixing Software Defects in MPI-Based Parallel Programs," *Computers, Materials & Continua*, vol. 79, no. 2, pp. 3139-3173, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[2] Shahzad Ashiq et al., "Challenges and Barriers to Software Testing," *Bulletin of Business and Economics*, vol. 13, no. 1, pp. 628-640, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[3] Marek Molęda et al., "From Corrective to Predictive Maintenance - A Review of Maintenance Approaches for the Power Industry," *Sensors*, vol. 23, no. 13, pp. 1-47, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[4] Jim Johnson, *CHAOS 2020: Beyond Infinity*, The Standish Group, Boston, MA, 2021. [Google Scholar] [Publisher Link]

[5] Golnoosh Abaei, and Ali Selamat, "Software Fault Prediction Based on Improved Fuzzy Clustering," *International Conference on Distributed Computing and Artificial Intelligence*, pp. 165-172, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[6] Lov Kumar, Sanjay Misra, and Santanu Ku. Rath, "An Empirical Analysis of the Effectiveness of Software Metrics and Fault Prediction Model for Identifying Faulty Classes," *Computer Standards & Interfaces*, vol. 53, pp. 1-32, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[7] Aswathy Rajendra Kurup et al., "Ensemble Models for Circuit Topology Estimation, Fault Detection and Classification in Distribution Systems," *Sustainable Energy, Grids and Networks*, vol. 34, pp. 1-32, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Santosh S. Rathore, and Sandeep Kumar, "A Study on Software Fault Prediction Techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255-327, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[9] Prachi Pramod Shedge et al., "Enhancing Maternal Health: A Soft Computing Approach to Pregnancy Risk Management," *Modernizing Maternal Care with Digital Technologies*, pp. 65-96, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[10] Sarita Negi, Devesh Pratap Singh, and Man Mohan Singh Rauthan, "A Systematic Literature Review on Soft Computing Techniques in Cloud Load Balancing Network," *International Journal of System Assurance Engineering and Management*, vol. 15, no. 3, pp. 800-838, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi, "Machine Learning Based Methods for Software Fault Prediction: A Survey," *Expert Systems with Applications*, vol. 172, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[12] Erik Arisholm, Lionel C. Briand, and Eivind B. Johannessen, "A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2-17, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[13] Haonan Tong, Bin Liu, and Shihai Wang, "Software Defect Prediction using Stacked Denoising Autoencoders and Two-Stage Ensemble Learning," *Information and Software Technology*, vol. 96, pp. 94-111, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[14] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Triphathi, "Software Bug Prediction Prototype using Bayesian Network Classifier: A Comprehensive Model," *Procedia Computer Science*, vol. 132, pp. 1412-1421, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Aleksey Borodulin et al., "Using Machine Learning Algorithms to Solve Data Classification Problems using Multi-Attribute Dataset," *BIO Web of Conferences*, vol. 84, pp. 1-11, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[16] Asad Ali, and Carmine Gravino, "Bio-inspired Algorithms in Software Fault Prediction: A Systematic Literature Review," *2020 14th International Conference on Open Source Systems and Technologies (ICOSST)*, Lahore, Pakistan, pp. 1-8, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[17] Santosh S. Rathore, and Sandeep Kumar, "An Empirical Study of Ensemble Techniques for Software Fault Prediction," *Applied Intelligence*, vol. 51, no. 6, pp. 3615-3644, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Jian Li et al., "Software Defect Prediction via Convolutional Neural Network," *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Prague, Czech Republic, pp. 318-328, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[19] Chetan Shelke et al., *Optimized Machine Learning Techniques for Software Fault Prediction*, Natural Language Processing for Software Engineering, pp. 207-219, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[20] Nikhil Saji Thomas, and S. Kaliraj, "An Improved and Optimized Random Forest Based Approach to Predict the Software Faults," *SN Computer Science*, vol. 5, no. 5, pp. 1-18, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[21] Misbah Ali et al., "Enhancing Software Defect Prediction: A Framework with Improved Feature Selection and Ensemble Machine Learning," *PeerJ Computer Science*, vol. 10, pp. 1-37, 2024. [Google Scholar] [Publisher Link]

[22] Oral Alan, and Cagatay Catal, "Thresholds Based Outlier Detection Approach for Mining Class Outliers: An Empirical Case Study on Software Measurement Datasets," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3440-3445, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[23] T. Pushpavathi, V. Suma, and V. Ramaswamy, "Analysis of Software Fault and Defect Prediction by Fuzzy C-Means Clustering and Adaptive Neuro Fuzzy C-Means Clustering," *International Journal of Scientific & Engineering Research*, vol. 5, no. 9, 2014. [Google Scholar] [Publisher Link]

[24] Golnoush Abaei, and Ali Selamat, *Increasing the Accuracy of Software Fault Prediction using Majority Ranking Fuzzy Clustering*, Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 179-193, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[25] Riski Annisa, Didi Rosiyadi, and Dwiza Riana, "Improved Point Center Algorithm for K-Means Clustering to Increase Software Defect Prediction," *International Journal of Advances in Intelligent Informatics*, vol. 6, no. 3, pp. 328-339, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[26] D. Shrivathsan et al., "Novel Fuzzy Clustering Methods for Test Case Prioritization in Software Projects," *Symmetry*, vol. 11, no. 11, pp. 1-22, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[27] Ahmed Iqbal et al., "Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 5, pp. 1-19, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[28] Wasiur Rhmann et al., "Software Fault Prediction Based on Change Metrics Using Hybrid Algorithms: An Empirical Study," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 4, pp. 419-424, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[29] Garvit Rajesh Choudhary et al., "Empirical Analysis of Change Metrics for Software Fault Prediction," *Computers & Electrical Engineering*, vol. 67, pp. 15-24, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[30] José M. Pena, Jose Antonio Lozano, and Pedro Larranaga, "An Empirical Comparison of Four Initialization Methods for the K-Means Algorithm," *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1027-1040, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[31] David Arthur, and Sergei Vassilvitskii, "*k-Means++: the Advantages of Careful Seeding*," Technical Report, Stanford, pp. 1027-1035, 2006. [Google Scholar] [Publisher Link]

[32] C. Elkan, "Using the Triangle Inequality to Accelerate k-Means," *Proceedings of the 20th International Conference on Machine Learning,* Washington, DC, pp. 147-153, 2003. [Google Scholar] [Publisher Link]

[33] Greg Hamerly, "Making k-means Even Faster," *Proceedings of the 2010 SIAM International Conference on Data Mining*, pp. 130-140, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[34] Enrique H. Ruspini, "New Experimental Results in Fuzzy Clustering," *Information Sciences*, vol. 6, pp. 273-284, 1973. [CrossRef] [Google Scholar] [Publisher Link]

[35] James C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, 1st ed., Advanced Applications in Pattern Recognition, Springer New York, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[36] R.E. Hammah, and J.H. Curran, "Fuzzy Cluster Algorithm for the Automatic Identification of Joint Sets," *International Journal of Rock Mechanics and Mining Sciences*, vol. 35, no. 7, pp. 889-905, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[37] Nikhil R. Pal, and C. James, "On Cluster Validity for the Fuzzy C-Means Model," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, pp. 370-379, 1995. [CrossRef] [Google Scholar] [Publisher Link]

[38] Min Ren et al., "A Self-Adaptive Fuzzy C-Means Algorithm for Determining the Optimal Number of Clusters," *Computational Intelligence and Neuroscience*, vol. 2016, no. 1, pp. 1-12, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[39] Adrian Stetco, Xiao-Jun Zeng, and John Keane, "Fuzzy C-Means++: Fuzzy C-Means with Effective Seeding Initialization," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7541-7548, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[40] Andy Arief Setyawan, and Ahmad Ilham, "A Novel Framework of the Fuzzy C-Means Distances Problem Based Weighted Distance," *Journal of Applied Computing and Informatics*, pp. 1-25, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[41] Shuisheng Zhou et al., "A New Membership Scaling Fuzzy C-Means Clustering Algorithm," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 9, pp. 2810-2818, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[42] Robert Tibshirani, Guenther Walther, and Trevor Hastie, "Estimating the Number of Clusters in a Data Set via the Gap Statistic," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411-423, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[43] Xizhao Wang, Yadong Wang, and Lijuan Wang, "Improving Fuzzy C-Means Clustering Based on Feature-Weight Learning," *Pattern Recognition Letters*, vol. 25, no. 10, pp. 1123-1132, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[44] Martin Shepperd et al., "Data Quality: Some Comments on the NASA Software Defect Datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208-1215, 2013. [CrossRef] [Google Scholar] [Publisher Link]