Journal of Electronic Testing (2024) 40:229–243
https://doi.org/10.1007/s10836-024-06116-8

Sahand: A Software Fault-Prediction Method Using Autoencoder Neural Network and K-Means Algorithm

Bahman Arasteh[1,2] · **Sahar Golshan**[3] · **Shiva Shami**[4] · **Farzad Kiani**[5]

Received: 3 October 2023 / Accepted: 26 March 2024 / Published online: 12 April 2024

Abstract
Software is playing a growing role in many safety-critical applications, and software systems dependability is a major concern. Predicting faulty modules of software before the testing phase is one method for enhancing software reliability. The ability to predict and identify the faulty modules of software can lower software testing costs. Machine learning algorithms can be used to solve software fault prediction problem. Identifying the faulty modules of software with the maximum accuracy, precision, and performance are the main objectives of this study. A hybrid method combining the autoencoder and the K-means algorithm is utilized in this paper to develop a software fault predictor. The autoencoder algorithm, as a preprocessor, is used to select the effective attributes of the training dataset and consequently to reduce its size. Using an autoencoder with the K-means clustering method results in lower clustering error and time. Tests conducted on the standard NASA PROMIS data sets demonstrate that by removing the inefficient elements from the training data set, the proposed fault predictor has increased accuracy (96%) and precision (93%). The recall criteria provided by the proposed method is about 87%. Also, reducing the time necessary to create the software fault predictor is the other merit of this study.

Keywords Software Fault Prediction · Clustering · Autoencoder · K-means · Accuracy

1 Introduction

Software systems are growing increasingly sophisticated, and despite quality control activities (such as testing), many software systems have been supplied to clients with a non-negligible percentage of faults [1–3]. On the other hand, software is increasingly vital in many safety-critical systems, where faults can threaten users or result in financial losses. As a result, dependability is a critical concern in software systems. Approximately 80% of the faults are caused by 20% of the modules [4]. One method for increasing software dependability is to predict software faults before the testing phase. The ability to detect fault-proneness in software modules can minimize project expense and effort. As a result, one of the major issues in the world of software engineering is the prediction of fault-proneness modules [5].

Software system faults cannot be directly measured, but they can be predicted using software metrics. Previous research has found a link between software metrics and fault-proneness modules [6, 7]. Since the 1990s, software fault prediction models have been investigated, and faulty modules can be found before software testing. The occurrence of a fault in a module depends on n parameters (n independent variables) that should be considered in the fault prediction process. The dependent variable shows if the software module is faulty or not; the independent variables include software product or process metrics such as cyclomatic complexity and lines of code [6–8]. Effective

Responsible Editor: Y. Malaiya.

✉ Bahman Arasteh
b_arasteh2001@yahoo.com

[1] Department of Software Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Turkey

[2] Applied Science Research Center, Applied Science Private University, Amman, Jordan

[3] Department of Software Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran

[4] Department of Software Engineering, Seraj Institute, Tabriz, Azerbaijan Province, Iran

[5] Data Science Application and Research Center (VEBIM), Fatih Sultan Mehmet Vakif University, Istanbul, Turkey

independent variables (metrics) are used to create the prediction model. Hence, a set of effective metrics values gathered from real-world software products (as a dataset) is required for software fault prediction.

Using prior fault datasets and data-mining techniques, certain software fault-prediction approaches identify software modules as faulty or fault-free [6–8]. Corresponding researchers use a variety of public and open resources, including PROMISE and NASA MDP datasets [12]. Machine learning methods are frequently utilized for software fault prediction; Nave Bayes, Random Forest, J48, Neural Network, and SVM are some effective machine-learning algorithms used for fault prediction [9, 10]. Some studies employ statistical methods to predict problems, such as Multivariate Binary Logistic Regression [12]. When the data is unlabeled, supervised learning methods cannot be used to anticipate software faults. New methods, such as unsupervised learning methods, are appropriate in this scenario; to that aim, some researchers used numerous clustering algorithms to cluster software modules [13].

Using prior fault datasets and data-mining techniques, certain software fault-prediction approaches identify software modules as faulty or fault-free [6–8]. Corresponding researchers use a variety of public and open resources, including PROMISE and NASA MDP datasets [11]. Machine learning methods are frequently utilized for software fault prediction; Nave Bayes, Random Forest, J48, Neural Network, and SVM are some effective machine-learning algorithms used for fault prediction [9,]. Some studies employ statistical methods to predict problems, such as Multivariate Binary Logistic Regression [12]. When the data is unlabeled, supervised learning methods cannot be used to anticipate software faults. New methods, such as unsupervised learning methods, are appropriate in this scenario; to that aim, some researchers used numerous clustering algorithms to cluster software modules [13].

The proposed method in this research employs an autoencoder and K-Means to cluster software modules as faulty or faulty-free. The purpose of this study is to improve prediction accuracy while minimizing the time required to develop predictive models (gaps in earlier techniques). The use of an autoencoder in conjunction with K-Means enhances module clustering accuracy while decreasing clustering time. For categorizing data as faulty or fault-free, four traditional fault datasets are employed; datasets are available in the PROMISE [11] data repository. The following are the study's contributions:

- Development of the autoencoder algorithm, as a preprocessing and data-size decreasing step in the software fault prediction process, is the first contribution of this study.
- The development of an effective hybrid method for software fault prediction using the combination of autoencoder and K-means algorithm is the other contribution of this study.
- Increasing the accuracy of software fault prediction models by the proposed hybrid method is the final contribution of this study. The proposed method lowered the time necessary to cluster the software modules as faulty or non-faulty.

In Sect. 2, related works about software fault prediction are reviewed. In Sect. 3, the proposed hybrid method for software fault prediction is described and in Sect. 4, experiments conducted on the NASA datasets and the obtained results are discussed. Then, in Sect. 5 conclusion and future works are given.

2 Related Work

Since the 1990s, software fault-prediction approaches have been used. Various statistical and machine learning methods have been proposed to divide software modules into two classes: fault-prone and non-fault-prone. Since 2005, the use of public datasets and machine learning algorithms has been increased. Many academics have employed genetic programming, neural networks, fuzzy logic, and decision trees to anticipate software faults before the testing process [14]. In the relevant research, the evaluation criteria were false positive rate (FPR), false negative rate (FNR), and total error rate (TER) [13, 14]. The percentage of mislabeled modules is denoted by TER, while the percentage of non-fault-prone modules that were mislabeled as fault-prone is denoted by FPR. Faulty modules that have been designated as non-faulty are denoted as FNR. Finally, utilizing the aforementioned criteria, the accuracy and precision of a prediction model can be calculated. This section examines the software fault-prediction literature.

Previous software metrics were used to predict fault-prone modules in software fault prediction algorithms. Software measurements are employed as independent variables in this study, whereas fault data is used as dependent variables. Software metrics are classified into process, file, class, method, component, and quantitative-level categories. The most widely used class-level metrics are CK metrics [12, 13]. Object-oriented metrics, rather than standard source-code metrics, have been employed in certain related research to forecast fault-prone software [8]. Because the class idea is only utilized in object-oriented paradigms, class-level metrics (object-oriented metrics) are only used for object-oriented systems. Object-Oriented metrics outperform other metrics in software fault prediction [15].

The most conventional product-level metrics employed by the preceding methodologies include method-level, class-level, component-level, and file-level metrics [10, 16]. In [17], a combination of software requirements level and code-level metrics have been employed to increase the accuracy of prediction algorithms. Aside from requirement metrics, researchers may use process-level metrics such as programmer experience level, review time, number of test cases, and unique test cases [18]. Other metrics utilized as "quantitative-level measures" in software fault-prediction algorithms are CPU and disk space usage during the program execution [19].

The other resource required in software fault-prediction systems is datasets. The datasets utilized in the relevant publications were classified as public, private, or unknown [10, 12]. The main sources of public datasets used in the previously proposed methodologies are NASA MDP (metrics data program) sites. Private datasets are owned by private companies and are not available for free. Some studies have employed datasets with no information about them. Since 2005, the PROMISE repository has been one of the primary sources of public datasets used by researchers. The most utilized public datasets in this field of inquiry are NASA fault prediction datasets in PROMISE with a format (ARFF) [11].

Feature selection is a strategy in the ML applications such as classification and regression. While the fault prediction datasets have numerous attributes, some may be ineffective to learning accuracy. Removing these less relevant attributes can both improve accuracy and performance. The aim of attribute selection is to identify the smallest subset of attributes required for the fault prediction models. In the fault prediction models, feature selection includes optimization methods [3]. The optimization method methods evaluate the effectiveness of the features based on specific criteria without incorporating a classification function. Features are sorted and the least important features are eliminated based on a predetermined threshold. The resulting subset is then used in the classification or clustering algorithms.

The set of methods uses an evaluation function based on the error rate of the learning models. These methods select new feature subset using a fitness function, which is then trained using a machine learning method. The effectiveness of the selected features is determined by the number of errors in the test of the learning models. Two main methods used in this context are 'forward selection' and 'backward selection', which are used to identify the most effective set of features. Forward selection evaluates the effectiveness of the selected features at every step, while backward selection starts with all features and iteratively removes them until a predetermined stopping criterion is reached. Both forward and backward selection methods have been implemented in [5].

Blended learning is an effective method in ML. This method increases the accuracy of learning models by integrating the multiple classifiers. Blended learning includes two scenarios. Firstly, different classification algorithms are used to create software defect predictors; this method focuses on finding the most effective classification algorithm. Despite this, this approach fails to identify the best classifier and does not benefit from other algorithms. In the second case, it is not feasible to fit all the characteristics into a single classifier due to the extensive and different feature sets. Choosing a base classification algorithm is an important step in creating a mixed classifier. The variation among classifiers does not increase the efficacy of the blended classifier. According to prior research [13, 15], unsupervised fault-prediction approaches such as clustering algorithms have been employed for unlabeled fault datasets.

Managing the numerous attributes of the datasets can be an important obstacle in the creation of the software fault predictors. In order to reduce the number of features, feature selection techniques are often used. Choosing the best attributes of training datasets for creating fault predictors is an NP-complete optimization problem. The main drawbacks of the previous methods are as follows that have been addressed in this study.

- Ineffective attributes of the training datasets reduce the accuracy of the created fault predictors. Eliminating the ineffective features is an objective of this study.
- Insufficient value of recall criterion and precision are other drawbacks that can be solved by finding the most useful attributes in the fault prediction datasets. A lower value of recall criterion made more faulty modules to be identified incorrectly as fault-free modules.
- The long time required for creating a software fault predictor by the classification and clustering algorithms in the large real-world datasets is another challenge. Alleviating this time overhead is another objective of this study.

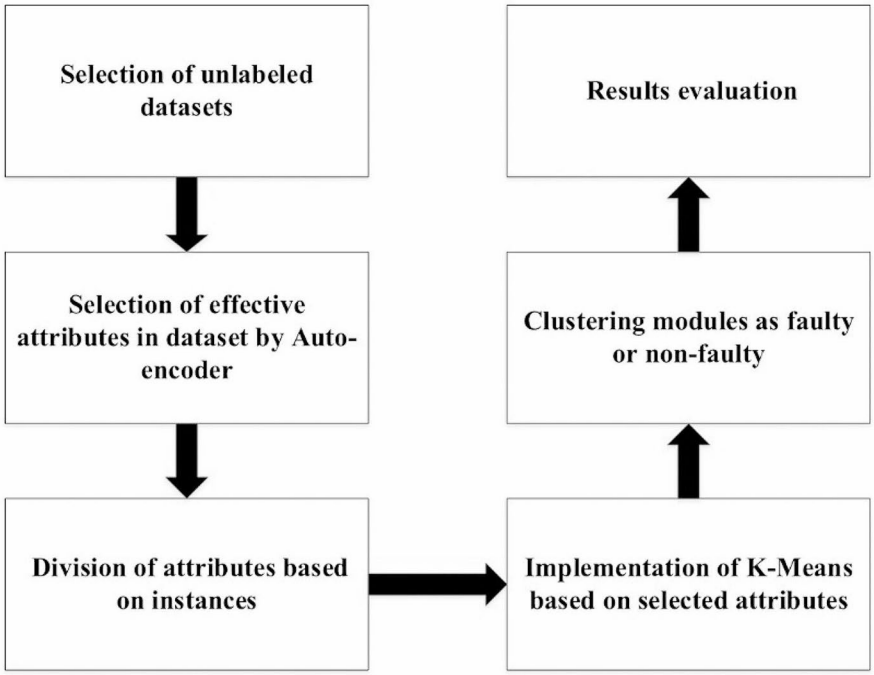
To achieve higher prediction accuracy, precision and recall, an unsupervised method for building software fault predictor is proposed in this research; this method is discussed in Sect. 3.

3 Proposed Method

3.1 The Process of the Proposed Method

The autoencoder and K-means algorithms are utilized in this research to build a machine-learning-based software

Fig. 1 Proposed method's steps



fault predictor. Figure 1 depicts the steps of the suggested technique.

The necessary datasets for developing the fault-prediction model were obtained from the PROMISE data repository [11]; it is the traditional public software data repository that the researchers utilized to construct the fault-prediction model. Each data collection contains historical information about the modules of a program. Each record in the data sets consists of 21 software module attributes. In addition, each data set contains the attributes of both faulty and non-faulty modules. The Preprocessing of the dataset attributes is the first step of the proposed method. The main purpose of this stage is to choose an important and effective subset of characteristics in the datasets using the autoencoder. The elimination of inefficient dataset attributes increases the accuracy of the fault-prediction model generated using the K-means method (second step). Building the prediction model is the third step of the proposed method. The reduced dataset by autoencoder is sent to the K-means clustering method. Clustering is a technique used in this procedure that splits data into two groups (fault-free or faulty). Finally, the created fault predictor by the K-means algorithm was evaluated using the testing data set.

3.2 Obtaining the Training Datasets

The PROMISE software engineering repository (as a public software data repository) is increasingly being used in software engineering research. PROMISE covers a subset of NASA datasets classified as defect prediction, cost estimates, and text mining. The NASA dataset was compiled over time from various initiatives (satellite instruments, ground control systems, attitude control systems, and so on) in the United States. The fault-prediction datasets include McCabe, Halstead, and line of code metrics. The datasets used in this study are shown in Table 1. The goal of this research is to create a fault prediction model at the source-code level utilizing object-oriented metrics. The object-oriented metrics (given in Table 2) are thought of as independent variables, while fault is thought of as the dependent variable. The final attributes (fault) were removed and are not shown in Table 2.

Table 1 Description of used datasets

Dataset	Project source	Instances	Attributes	Faulty modules (%)
ant 1.7	NASA	745	21	22.3
arc	NASA	234	21	11.54
camel 1.4	NASA	965	21	16.6
zuzel	NASA	29	21	45

3.3 Selecting the Most Effective Attributes of the Training Data Set

The autoencoder detects and selects the most significant and effective subset of attributes in each dataset in this step (filtering step). Attribute selection minimizes data dimension (number of attributes) by removing ineffective attributes. Because each instance is unique, the autoencoder learns which attribute plays a significant role in system failure by

Table 2 Object-oriented metrics (attributes) of datasets used in the related studies and our study [7, 8, 12]

WMC	This metric is the total of its approaches' complexity. Cyclomatic complexity can be used as a measure of complexity.
DIT	This metric calculates the inheritance levels from the top of the object hierarchy for each class.
NOC	This metric counts the number of the class's direct descendants.
CBO	This measure reflects the number of classes that are linked to a specific class. This can happen via field accesses, method calls, inheritance, arguments, and exceptions.
RFC	This metric counts the number of methods that can be called when an object of that class receives a message.
LCOM	This measure counts the sets of methods in a class that aren't related by the fact that they share some of the class's fields.
Ca	This is a count of how many other classes the specified class uses (coupling).
Ce	This is a count of how many other classes the specified class uses (efferent coupling).
NPM	The number of publicly available methods measures counts the number of methods in a class.
LCOM3	Lack of cohesion in methods. LCOM3 varies between 0 and 2
LOC	Lines of Code.
DAM	This measure represents the proportion of private attributes to total attributes (Metric for Data Access).
MOA	Aggregation measurement.
MFA	Functional abstraction represents the ratio of a class's inherited methods to the overall number of methods.
CAM	Cohesion of class computes the relatedness of a class's methods.
IC	Coupling of inheritance indicates how many parent classes a specific class is linked to.
CBM	The combination of methods counts the number of new methods to which all inherited methods are linked.
AMC	The average method size for each class is measured by this metric.

examining all of them. All of the datasets utilized in this study share the same characteristics. Figure 2 depicts the autoencoder structure during the attribute selection step. The autoencoder is a simple three-layer neural network with output units that are directly coupled to input units [20].

The first and last levels are referred to as the input and output layers, respectively. Between the first and last layers, there is at least one concealed layer. The autoencoder employed in this work prioritizes all 21 attributes of each dataset in the input layer based on their influence on the risk of software fault. Indeed, the dataset properties have been mapped to the autoencoder's input neurons. During the middle layers, the autoencoder analyzes the attributes of an input dataset and learns the valuable and effective attributes of the dataset. The autoencoder compresses (encodes) the input data in each middle layer to fit in a smaller representation and then attempts to reconstruct (decode) it. The purpose of the Autoencoding encoder and decoding layers (middle layers) is to discover erroneous and ineffective qualities in datasets and prioritize the most effective attributes.

In terms of the last attribute of the datasets (faults of a software module), the data instances were split into two groups: software modules with no bugs (fault-free) and software modules with one or more bugs (faulty). During the decoding phase, the autoencoder analyzes the problematic instances of datasets and learns which attributes have the greatest impact on software failures. The autoencoder examines the amount of each attribute's effect on the software failure. Finally, the qualities are categorized into three types based on the extent of their effect on the number of software faults:

- High Priority Class: The autoencoder classifies the characteristics with the greatest effect size into this class after determining the effect size of each attribute.
- Medium Priority Class: The autoencoder assigns this class to attributes that have a medium impact on the software fault.
- Low Priority Class: This class contains attributes with the smallest effect size on the software problem.

Table 3 shows the prioritized attributes as the autoencoder output. The qualities have been categorized into three classes, as shown in this table. Class H qualities have the greatest effect size on fault probability. Attributes in class M have a medium effect on fault likelihood, whereas attributes in class L have the least effect. After the autoencoder has classified the attributes (attribute selection), the clustering step is used to build the predicting model.

3.4 Creating the Software Fault Predictor by K-means Algorithm

The goal of this stage is to build a fault prediction model using K-means clustering methods, which predicts faulty and non-faulty modules. The K-Means technique divides data into k clusters. In this study, software modules will be divided into two groups: faulty and faulty-free (k = 2). The K-means algorithm sets a center for each cluster and correlates data objects with the clusters that are closest to them; when no data objects remain, new k centroids are established. Centroids that are determined change their location step by step until no further modifications occur. The steps of the K-Means method are:

1. K points (initial centroids) are chosen at random from the clustered datasets.

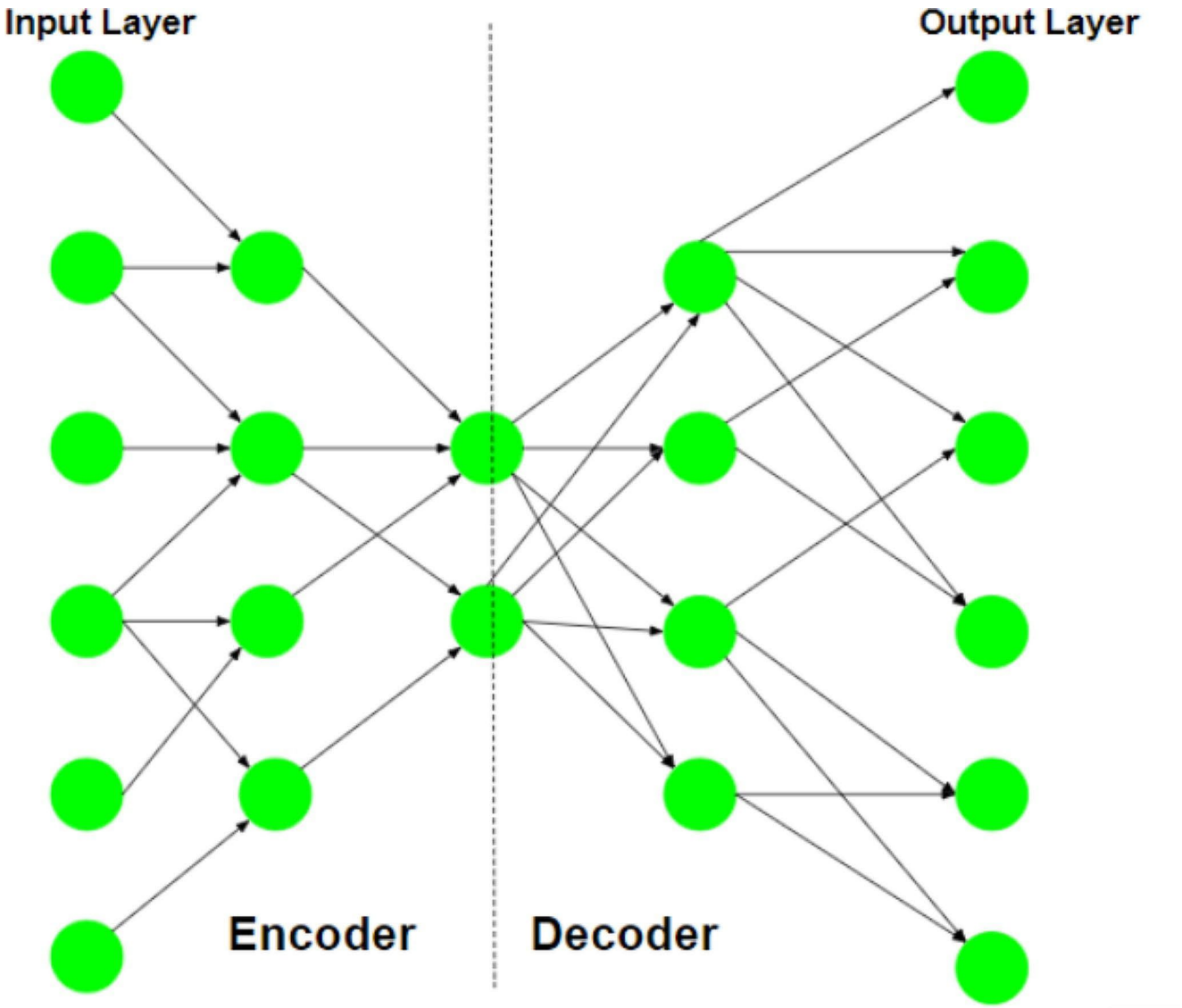


Fig. 2 The structure of autoencoder in the attribute (feature) selection

Table 3 Prioritized attributes by the autoencoder as the output of the preprocessing step

#	Attribute	Priority	#	Attribute	Priority	#	Attribute	Priority
1	WMC	H[1]	8	CE	H	15	CAM	H
2	DIT	H	9	NPM	H	16	IC	L
3	NOC	L[2]	10	LCOM	H	17	CBM	L
4	CBO	H	11	LOC	H	18	AMC	H
5	RFC	H	12	DAM	M	19	CC (Max and AVG)	H
6	LCOM	M[3]	13	MOA	L			
7	CA	M	14	MFA	M			

[1] High Priority.

[2] Low Priority.

[3] Medium Priority.

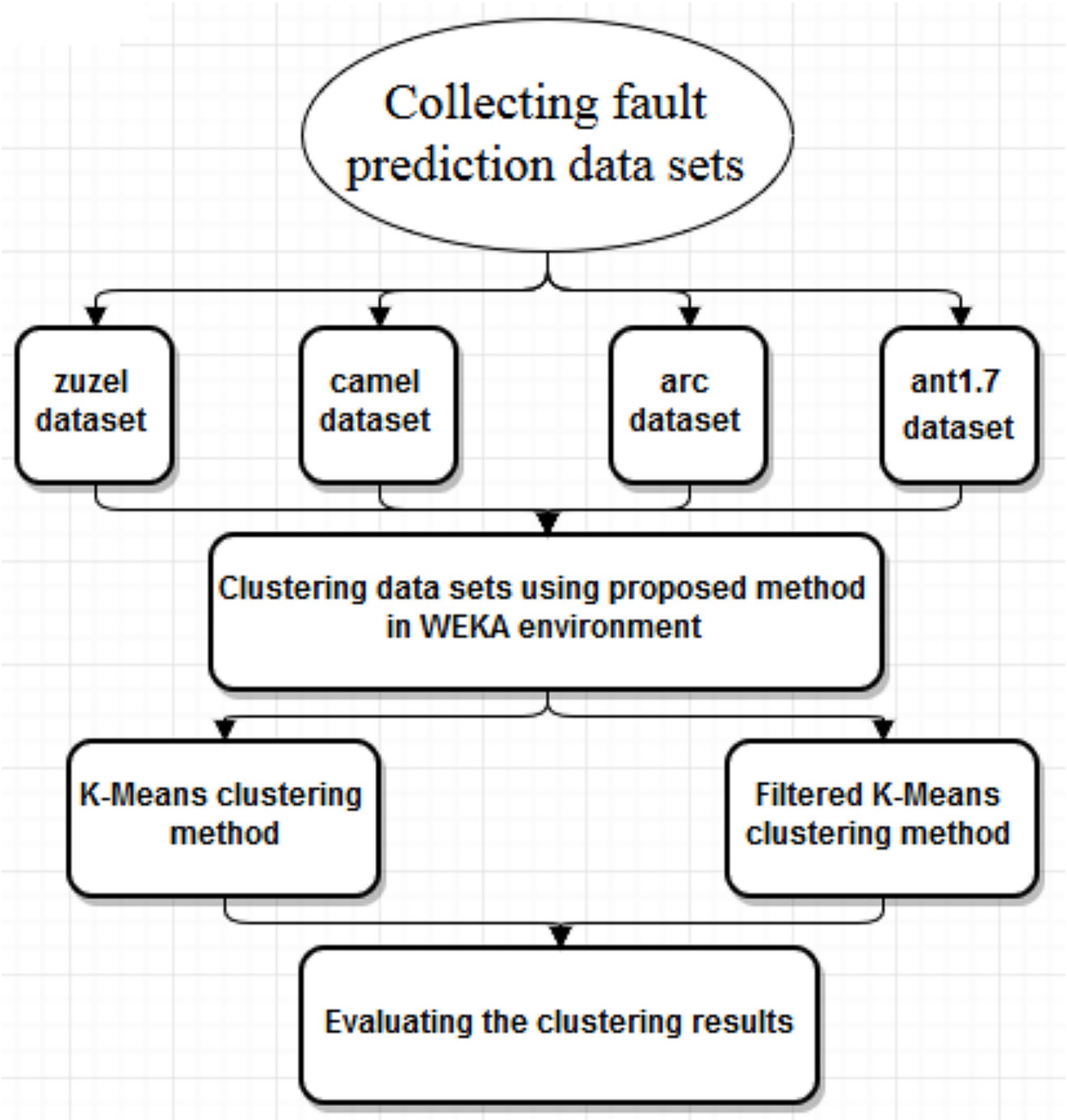


Fig. 3 Process of evaluation system

2. Each dataset instance is assigned to a cluster group based on the distance to the nearest centroid.
3. After all the objects have been dedicated, recalculate the positions of the K centroids.

Steps 2 and 3 should be repeated until the centroids are stable. The K-Means technique is used to minimize the objective function. Equation (1) depicts the objective function.

$$\text{Distance} = \sum_{j=1}^{k} \sum_{i=1}^{n} \left(||X_i^{(j)} - C_j|| \right)^2 \quad (1)$$

In Eq. (1), $||X_i^{(j)} - C_j||$ is the distance measure between data point $X_i^{(j)}$ and the cluster center (c_j). Variable *n* is the number of data points in *jth* cluster and *k* is the number of cluster centers. The autoencoder calculates the priority of attributes on software faults, and low-priority attributes

(effect-less attributes) are not considered in the clustering step; hence, minimizing the number of training data increases clustering accuracy and decreases clustering time. Indeed, after classifying the attributes according to their importance, K-means groups the instances of training datasets into faulty and fault-free classes using the selected attributes by the autoencoder.

The datasets described in Table 1 were used to build and test a predictive model using k-means clustering. The k-fold cross-validation procedure is employed in this study to evaluate our predictive models. This method divides the original data set into two parts: a training set for training the model and a test set for evaluating it (test). The data set is randomly partitioned into 10 equal-size subsamples in k-fold (10-fold) cross-validation. A subsample is kept as validation data to evaluate the predictive model, while the remaining k-1 subsamples are used as training data. The cross-validation procedure is carried out ten times.

4 Experiments and Results

4.1 Experimental System and Evaluation Criteria

The generated predicting model was assessed (examined) using the 10-fold cross-validation methodology to assess the accuracy and time of the suggested method. As previously stated, the distribution of attributes in train and test data is the same. The performance requirements for the fault prediction method are accuracy, precision, and clustering time. Figure 3 depicts the evaluation system procedure. Clustering time is defined as the time required to develop (build) the prediction model using the provided method. The datasets ant1.7, arc, camel, and zuzel were employed in this study to evaluate the suggested fault-prediction model. These datasets are described in Table 1. Autoencoder (as a preprocessor) and K-Means (as a clustering technique) are used to build a fault-predicting model. The proposed method was developed in the WEKA environment, which is an open-source and Java-based data mining tool. In the testing process, the outputs of the predicting model are binary (faulty or fault-free) and are classified as follows:

- True Positive (TP): The predicting model predicts a positive outcome correctly.
- True Negative (TN): The predicting model predicts a negative outcome correctly.
- False Positive (FP): The predictive model forecasts a positive outcome inaccurately.
- False Negative (FN): The predictive model forecasts a negative result inaccurately.

A true positive is correctly predicting a software module's code as defective. A genuine negative is correctly predicting a code with no errors. A false positive occurs when a fault-free code is expected to be faulty, and a false negative occurs when the predicting model expects a code to be clean when it is defective. The accuracy of the forecasting model is calculated using Eq. (2). Another criterion that should be calculated for our forecasting model is the error rate. The error rate is calculated using Eq. (3), and the value of this criterion should be as small as feasible. Another criterion is FPR (false positive rate), which is the proportion of modules that are genuinely non-faulty but anticipated to be defective, and FNR (false negative rate), which is the percentage of modules that are faulty but predicted to be non-faulty.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$Overall\ Error\ Rate = \frac{FP + FN}{TP + TN + FP + FN} \tag{3}$$

Using the autoencoder as a preprocessor for the K-means algorithm enhances model accuracy while decreasing the time necessary to build the model (clustering time). K-means and filtered K-means were built with and without the autoencoder (as a preprocessor) to test the effectiveness of the suggested method on clustering time and accuracy.

4.2 Results

There are 745 instances and 21 attributes in the ant1.7 dataset when the autoencoder algorithm is not used as a preprocessor. For this dataset, the number of iterations is three. Centroids were stabled after three repetitions. In the second implementation, the K-Means clustering method (simple and filtered) was used with the autoencoder preprocessor on the ant1.7 dataset. In this implementation, there are 745 instances and 3 attributes. Autoencoder reduced the data dimension (number of attributes). It has three layers: the input layer, the hidden layer, and the output layer. The number of iterations for this dataset was 4. After 4 iterations, centroids were stabled, and the dataset was clustered as faulty or not. Table 4 shows the performance of the k-means algorithm in the fault prediction problem in four implementations.

The second benchmark dataset is arc. In the second experiment, the K-Means clustering method was implemented in two forms (simple and filtered) using an arc dataset. In this implementation, there are 234 instances and 21 attributes. The number of iterations for this dataset was 2 without using an autoencoder. After 2 iterations, centroids were stabled, and data was clustered as faulty

Table 4 The effect of the autoencoder algorithm on the performance of k-means in ant1.7 dataset

	K-means without autoencoder	K-means with autoencoder	Filtered K-means without autoencoder	Filtered K-means with auto-encoder
Time Taken to Build Model	0.4 s	0.1 s	0.55 s	0.14 s
Clustered instances as fault-free	223 (30%)	384 (52%)	349 (47%)	383 (51%)
Clustered instances as faulty	522 (70%)	361 (48%)	396 (53%)	362 (49%)
Number of Iterations	3	4	5	4

Table 5 The effect of the autoencoder algorithm on the performance of k-means in arc dataset

	K-means without auto-encoder	K-means with auto-encoder	Filtered K-means without auto-encoder	Filtered K-means with auto-encoder
Time Taken to Build Model	0.04 s	0.01 s	0.19 s	0.07 s
Clustered instances as fault-free	106 (45%)	147 (63%)	146 (62%)	145 (62%)
Clustered instances as faulty	128 (55%)	87 (37%)	88 (38%)	89 (38%)
Number of Iterations	2	5	4	4

Table 6 The effect of the autoencoder algorithm on the performance of k-means in camel dataset

	K-means without autoencoder	K-means with autoencoder	Filtered K-means without autoencoder	Filtered K-means with auto-encoder
Time Taken to Build Model	0.06 s	0.01 s	0.46 s	0.13 s
Clustered instances as fault-free	600 (62%)	429 (44%)	431 (45%)	452 (37%)
Clustered instances as faulty	365 (38%)	536 (56%)	534 (55%)	513 (63%)
Number of Iterations	5	5	6	6

Table 7 The effect of autoencoder algorithm on the performance of k-means in zuzel dataset

	K-means without autoencoder	K-means with autoencoder	Filtered K-means without autoencoder	Filtered K-means with auto-encoder
Time Taken to Build Model	0.002 s	0.001 s	0.05 s	0.03 s
Clustered instances as fault-free	18 (62%)	19 (66%)	18 (62%)	19 (66%)
Clustered instances as faulty	11 (38%)	10 (34%)	11 (38%)	10 (34%)
Number of Iterations	6	3	4	4

or not. In the second round of the second experiment, the K-Means clustering method was implemented based on the autoencoder preprocessor using arc dataset. In this implementation, there are 234 instances and 3 Attributes. Autoencoder's target is reducing data dimension. The number of iterations for this dataset was 5. After 5 iterations, centroids were stabled, and data was clustered as faulty or not. Similar experiments were implemented with the filtered k-means algorithm. Table 5 shows the performance of the k-means algorithm in the fault prediction problem in two implementations.

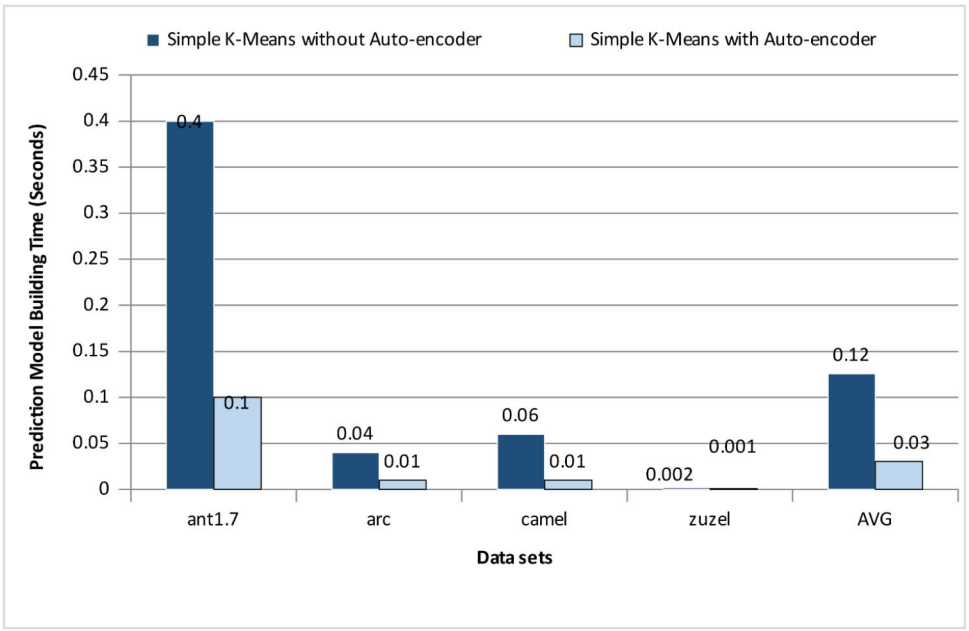
In the third experiment, the K-Means clustering method was implemented using camel dataset in simple and filtered forms. In this implementation, there are 965 instances and 21 attributes. The number of iterations for this dataset is 5 when simple k-means is used. After 5 iterations, centroids are stable, and data is clustered as faulty or not. In the second round of the third experiment, the K-Means clustering method was implemented based on the autoencoder preprocessor using the camel dataset. In this implementation, there are 965 instances and 3 attributes. The autoencoder's target is reducing data dimension. The number of iterations for this dataset is 5. The effects of the autoencoder were evaluated on the filtered K-means algorithm. Results of experiments confirm that the autoencoder has considerably positive effects on the performance of the K-means algorithm. Table 6 shows the performance of the k-means algorithm in the camel dataset.

In the fourth experiment, the K-Means clustering method was implemented to cluster the zuzel dataset. In this dataset, there are 29 instances and 21 attributes. The

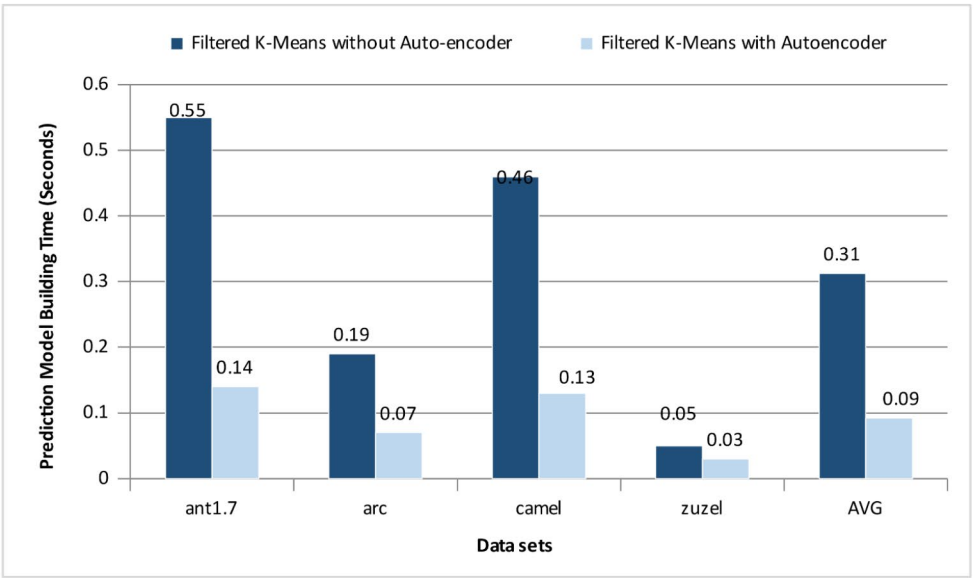
number of iterations for this dataset is 5. After 6 iterations, centroids are stable, and data is clustered as faulty or not. On the other hand, the filtered K-Means clustering method is implemented with and without an autoencoder preprocessor using zuzel dataset. In this implementation, the number of attributes was reduced to 3 by the autoencoder. The number of iterations for this dataset is 4. After 4 iterations, centroids are stable, and data is clustered as faulty or not. Tale 7 indicates the performance of the K-means clustering algorithm in the zuzel.

4.3 Explanation

Figure 4 depicts the impact of the suggested approach on clustering time (model building time). In our trials, the average clustering time (time required to develop a prediction model) by simple K-means with and without autoencoder is 0.03 and 0.12, respectively. Furthermore, the average clustering time for filtered K-means with and without autoencoder is 0.09 and 0.31, respectively. As a result, the time required to create a predicting model without the autoencoder preprocessor is longer than the time required to develop the



(a)



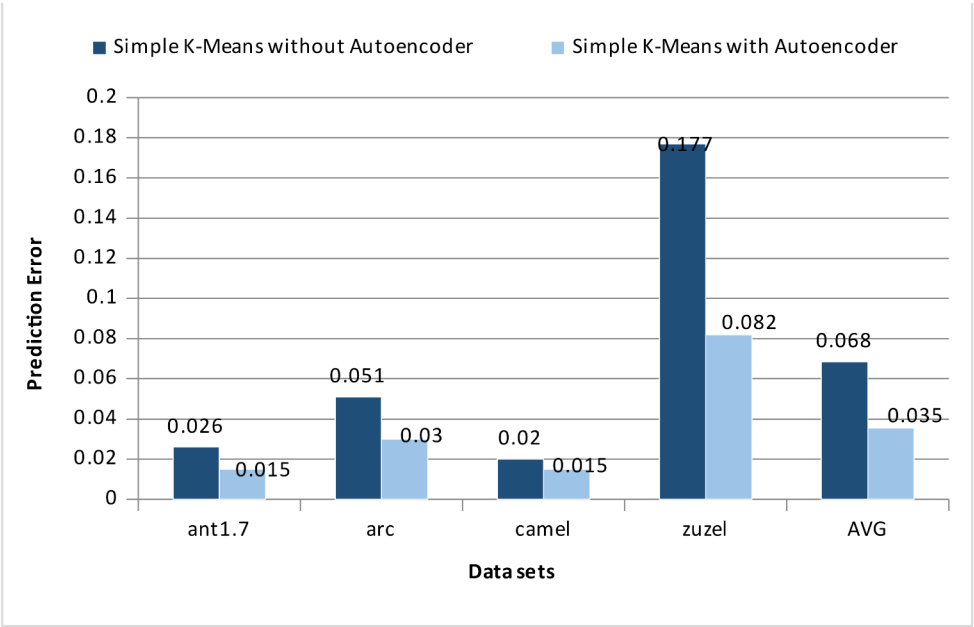
(b)

Fig. 4 (**a**) Clustering time reduction by autoencoder using *ant1.7*, *arc*, *camel* and *zuzel* datasets for simple K-Means, (**b**) Clustering time reduction by autoencoder using *ant1.7*, *arc*, *camel* and *zuzel* datasets for Filtered K-Means

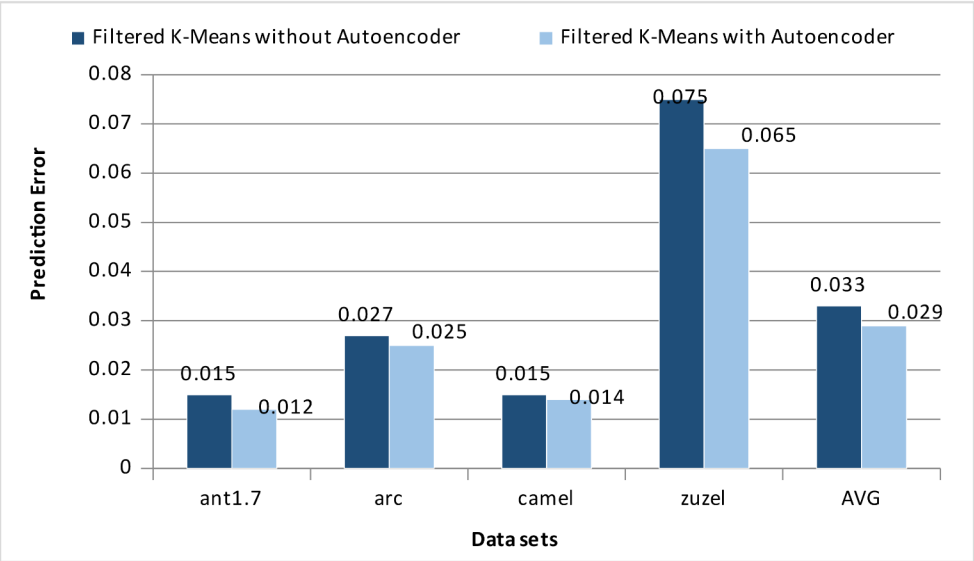
model with the autoencoder preprocessor. The accuracy of the constructed model using the provided method was evaluated in the following step of the evaluation procedure. To that purpose, the developed predicting model was put to the test using the testing data (k-fold cross-validation).

Figure 5 depicts the proposed method's error in forecasting software faults. The size of training data is reduced when the autoencoder determines the priority of the characteristics and eliminates the effect-less attributes, resulting in improved prediction accuracy. Indeed, utilizing an autoencoder as a preprocessor of the K-means method allows us to improve the accuracy of the forecasting model. Figure 6 depicts the proposed predictive model's accuracy. As illustrated in Fig. 6, the suggested method outperforms the pure k-means clustering algorithm in software defect prediction. In terms of defect prediction accuracy and time, the combination of autoencoder with k-means produces better results.

Precision is one of the important criteria for the performance of the predicting models. The correctness of the proportion of positive alarms (faulty prediction) is determined by precision criterion. Figure 7 shows the effects of the autoencoder on the precision of the predicting model created by the k-means algorithm. The



(a)



(b)

Fig. 5 (**a**) clustering error in simple K-Means and autoencoder using *ant1.7*, *arc*, *camel* and *zuzel* datasets, (**b**) clustering error in filtered K-Means and autoencoder using *ant1.7*, *arc*, *camel* and *zuzel* datasets

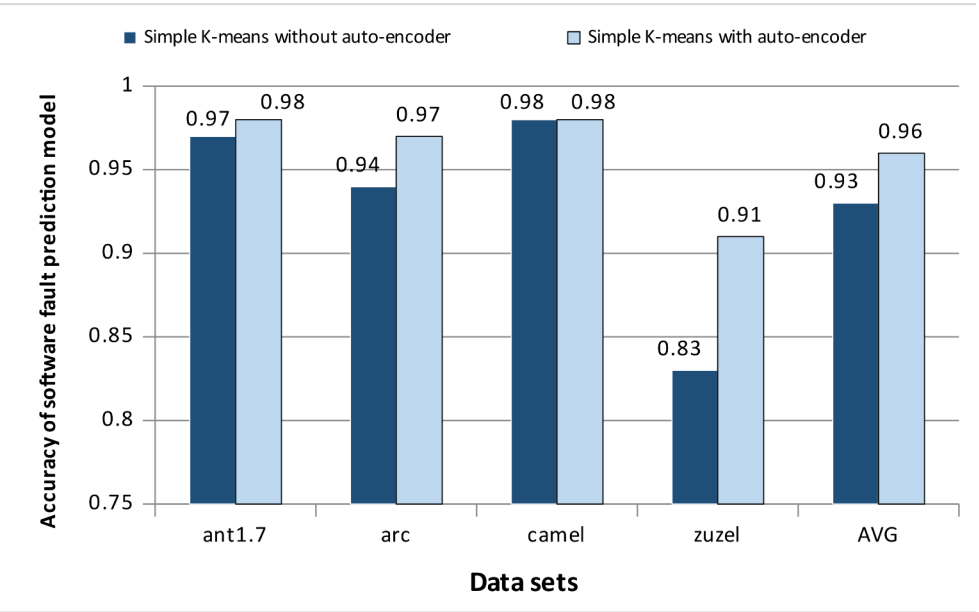


Fig. 6 The accuracy of the proposed method on the different data sets

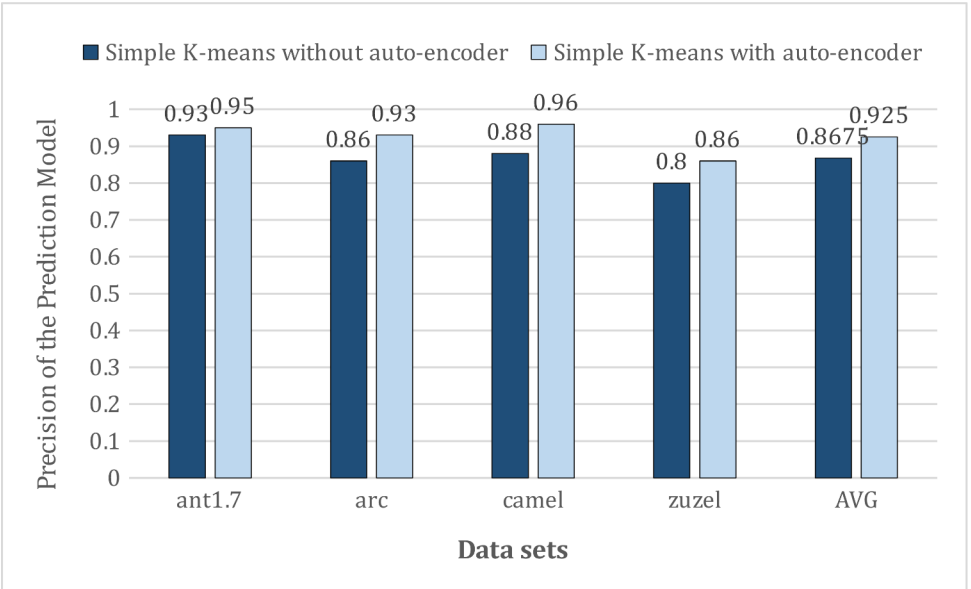


Fig. 7 The precision of the proposed method on the different data sets

autoencoder improves the precision of the created fault prediction model in all datasets. In the *ant1/7* dataset, the precision of the created fault prediction model by the k-means algorithm without autoencoder is about 0.93, whereas this figure was improved to 0.95 using the autoencoder preprocessing algorithm. The precision criterion in the arc data set is approximately 0.86; this figure was approximately 0.93 when the autoencoder algorithm preprocessed the dataset before clustering stepwise by the k-means algorithm. Similar results were obtained on the *camel* and *zuzel* datasets. On average, the precision of the fault prediction model without and with the autoencoder algorithm as the preprocessing step is 0.86 and 0.92, respectively. Overall, the results of the conducted experiments confirm that the autoencoder has positive effects on the precision of the fault prediction model created by the k-means algorithm.

Recall is the other evaluation criteria that was considered in this study. The recall criterion tries to evaluate the proportion of actual positives (actual faulty modules) that were identified correctly by the predicting model. Equation 4 was used to calculate the recall criterion for the created fault prediction model. The results of experiments depict that the autoencoder enhances the recall of the created predicting model by the k-means algorithm. The most improvement has been made in the *zuzel* dataset. On average, the recall of the predicting model without autoencoder is 0.845; this figure is 0.865 when using

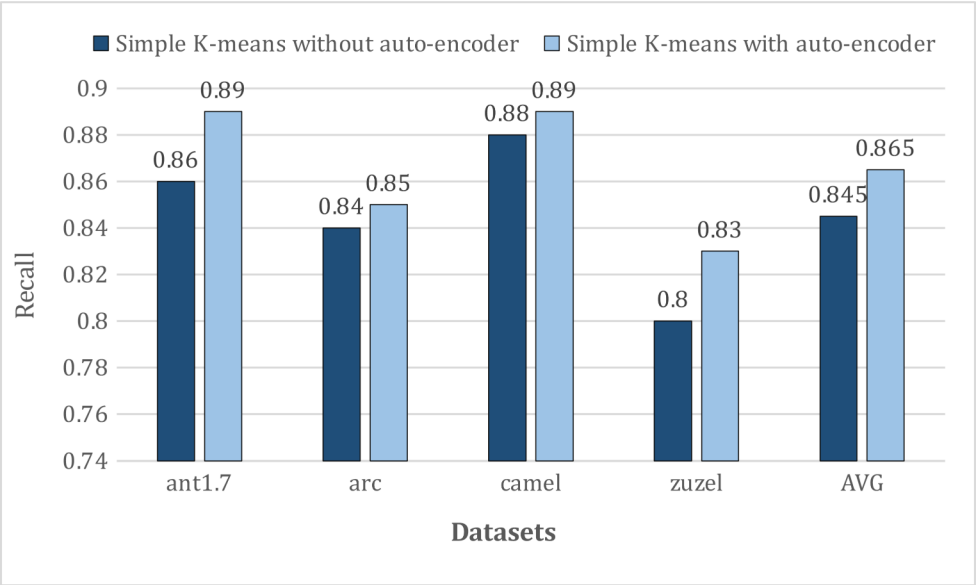


Fig. 8 The recall of the proposed method on the different data sets

Table 8 Comparing the accuracy of different methods on different data

	ant1.7	arc	camel	zuzel	AVG
K-means without auto-encoder	0.98	0.94	0.98	0.83	0.93
K-means with auto-encoder	0.98	0.97	0.98	0.91	0.96
K Nearest Neighbor	0.83	0.82	0.84	0.81	0.82
Decision Tree	0.84	0.86	0.87	0.79	0.84
Multi-Layer Perceptron	0.79	0.80	0.84	0.82	0.81

the autoencoder algorithm as the preprocessing algorithm. Indeed, close to 87% of the faulty modules can be correctly predicted by the proposed method (See Fig. 8).

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

Table 7 shows the accuracy of the created fault predictor by different ML algorithms using different datasets. The fault predictor created by the proposed method in *ant1.7* dataset provides 98% accuracy which is higher than the other fault predictors created by the other ML algorithms. In the *arc* dataset, the fault predictor by the proposed method provides 97% accuracy; this figure in the *camel* and *zuzel* datasets is respectively 98% and 91%. Overall, the proposed method is superior to the other methods in terms of accuracy and performance. This improvement is the result of the developed autoencoder. The developed autoencoder identifies the most effective features of the training dataset. The elimination of the ineffective attributes increases the performance and the accuracy of the created software fault predictor (See Table 8).

Table 9 The results of the ANOVA test on the performance values (time) of the k-means algorithm and proposed method

Source	Sum of Square	df	MS	
Between Groups	0.121	1	0.121	F = 5.6891, $P = 0.044187$
Within Groups	0.1702	8	0.0213	
Total	0.2912	9		

4.4 Statistical Analysis

To statistically investigate the results, the ANOVA test (as statistical analysis) was used to evaluate the significance of the obtained results. Analysis of variance (ANOVA) is a statistical analytical method that divides observed results into two parts: systematic and random factors. Random factors have no statistical impact on the supplied data set, but systematic factors do. ANOVA employs the F-statistic and its accompanying p-value to assess if data is from the same population. A p-value indicates that there are significant differences among the results provided by the methods. Table 9 shows the results of the ANOVA test on the performance values (model creation time) of the k-means algorithm and proposed method; indeed, the effect of the proposed method on the performance of the k-means algorithm has been investigated using the ANOVA test. The main hypothesis of this study is that the autoencoder technique makes a systematic reduction in the time of the fault prediction process. In contrast, the null hypothesis indicates the lack of the systematic influence of the autoencoder technique on the efficiency of the k-means algorithm. The results of the ANOVA test indicate that the f-ratio value is 5.6891 and the p-value is 0.044187. Hence, the null hypothesis is rejected, and the result is significant at $p < 0.05$.

Table 10 The most effective attributes selected by the autoencoder for the ant1.7 dataset as a case study in the best execution

#	Attribute	Priority	#	Attribute	Priority	#	Attribute	Priority
1	WMC	1	8	CE	1	15	CAM	1
2	DIT	1	9	NPM	1	16	IC	0
3	NOC	0	10	LCOM	1	17	CBM	0
4	CBO	1	11	LOC	1	18	AMC	1
5	RFC	1	12	DAM	0	19	CC (Max and AVG)	1
6	LCOM	1	13	MOA	0			
7	CA	1	14	MFA	0			

Machine learning techniques dealing with high-dimensional data, such as datasets with a large number of features, encounter several challenges. The large number of attributes increases computational complexity, and difficulty in extracting meaningful prediction model. In order to avoid overfitting during the training stage of the machine learning algorithm, the number of attributes should be decreased. Attribute reduction methods are employed to address these issues, aiming to represent data in a lower-dimensional space while preserving essential characteristics of the original dataset. These methods reduce the dimensionality of the data by either combining existing feature values to form a smaller, more manageable set of attributes or by selecting a subset of the most relevant features. The proposed method identifies a subset of attributes in the training dataset. In this study, an autoencoder algorithm is used for attribute selection. This process involves identifying the most effective attributes within the training dataset before creating the fault predictor. Ineffective features can hinder the capability of the created fault predictor and reduce the overall accuracy. This method (Eliminating the ineffective attributes) not only shortened the execution time of the machine learning algorithm but also increased its efficiency. Table 10 shows the most prior attributes identified by autoencoder for the *ant1.7* case study in its best-performing runs.

5 Conclusion and Future Studies

In this study, a method for predicting the faulty modules of software using a combination of autoencoder and K-means was proposed. The autoencoder algorithm was used to find out the most effective features of the training dataset; in the next stage the created fault predictor by the K-means algorithm provides lower error-rate, higher precision, and higher performance thanks to the developed autoencoder algorithm. On average, the proposed fault predictor provides 96% accuracy and 92% precision, respectively. The results of ANOVA test, statistical analysis, confirms the significance of the provided improvement by the proposed method.

One limitation of our work is the use of public NASA datasets; the extension of proposed method using datasets from other new software industry is considered as one of future studies. The features of new programming languages, programming structures, and frameworks can be considered in the training datasets. The success of the proposed method is also dependent on the size of the training data sets, which is another drawback of this method. Another future study would be to build a predictive model utilizing deep learning algorithms. A variety of heuristic methods which have been developed in [21–24] can be used as attributes selectors.

Authors' Contributions The proposed method was developed and discretized by B. Arasteh and S. Golshani. The designed algorithm was implemented and coded by B. Arasteh and S. Shami. The implemented method code was adapted and benchmarked by B. Arasteh. The data and results analysis were performed by B. Arasteh and S. Golshani. The manuscript of the paper was written by B. Arasteh and F. Kiani.

Data Availability Access.
The data relating to the current study is available via the following link:
https://drive.google.com/drive/folders/1-aX_QueAUV1PhL9rBOAF-n0ZzS5RcnNXF? usp=drive_link.

Declarations

Ethical and Informed Consent for data used The data used in this research does not belong to any other person or third party and was prepared and generated by the researchers themselves during the research. The data of this research will be accessible to other researchers.

Competing Interests The authors declare that no funds, grants, or other support were received during the preparation of this manuscript. The authors have no relevant financial or non-financial conflict of interest.

References

1. Iqra Batool B, Tamim Ahmed Khan AK (2022) Software fault prediction using data mining, machine learning and deep learning techniques: a systematic literature review. Comput Electr Eng 100:0045–7906. https://doi.org/10.1016/j.compeleceng.2022.107886
2. Al-Laham M, Kassaymeh S, Al-Betar MA, Makhadmeh SN, Albashish D, Alweshah M, Part A (2023) 0045–7906, https://doi.org/10.1016/j.compeleceng.2023.108923
3. Mafarja M, Thaher T, Al-Betar MA et al (2023) Classification framework for faulty-software using enhanced exploratory whale optimiser-based feature selection scheme and random forest ensemble learning. Appl Intell 53:18715–18757. https://doi.org/10.1007/s10489-022-04427-x

4. Yousef HA (2015) Extracting Software Static defect models using Data Mining. Ain Shams Eng J 6(1):133–144
5. Jayanthi R, Florence L (2019) Software defect prediction techniques using metrics based on neural network classifier. Cluster Comput 22(1):77–88. https://doi.org/10.1007/s10586-018-1730-1
6. Arasteh B (2018) Software Fault-Prediction using combination of neural network and Naive Bayes Algorithm. J Netw Technol 9(3):94–101. https://doi.org/10.6025/jnt/2018/9/3/94-101
7. Catal C, Diri B (2009) Investigating the Effect of Dataset Size, Metrics Sets and Feature Selection Techniques on Software Fault Prediction Problem, *Information Sciences*, Vol. 179, No. 8, pp. 1040–1058, Mar
8. Radjenović D, Heričko M, Torkar R, Živkovič A (Aug 2013) Software Fault Prediction Metrics: a systematic literature review. Inf Softw Technol 55(8):1397–1418
9. Anbu M, Anandha GS (2019) Feature selection using firefly algorithm in software defect prediction. Cluster Comput 22:10925–10934. https://doi.org/10.1007/s10586-017-1235-3
10. Rathi SC, Misra S, Colomo-Palacios R, Adarsh R et al (2023) Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction. Expert Syst Appl 223:0957–4174. https://doi.org/10.1016/j.eswa.2023.119806
11. Promise software engineering repository [Online Available:http://promise.site.uottawa.ca/SERepository/datasets-page.html
12. He P, Li B, Liu X, Chen J, Ma Y (2015) An Empirical Study on Software Defect Prediction with a Simplified Metric Set, *Information and Software Technology*, Vol. 59, pp. 170–190, Mar
13. Sujitha KC, Leninisha S (2014) Software Fault Prediction Using Single Linkage Clustering Method, *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, Vol. 3, No. 2, Apr
14. Rathore SS, Kumar S (March 2017) Linear and non-linear heterogeneous ensemble methods to predict the number of faults in Software systems. Knowl Based Syst 119:232–256
15. Kaur S, Kumar D (2011) Quality Prediction of Object-Oriented Software Using Density Based Clustering Approach, *International Journal of Engineering and Technology*, Vol. 3, No. 4, pp. 440–445, Aug
16. Catal C (April 2011) Software Fault Prediction: A literature review and current trends. Expert Syst Appl 38(4):4626–4636
17. Jiang Y, Cukicc B, Menzies T (2007) Fault Prediction using Early Lifecycle Data, in *Proceedings of 17th IEEE international symposium on software reliability*, Sweden, pp. 237–246
18. Kaszycki G (1999) Using Process Metrics to Enhance Software Fault Prediction Models, *Proceedings of 10th international symposium on software reliability engineering*, Boca Raton, Florida
19. Moeyersoms J, Junqu E, Dejaeger K, Baesens B, Martens D (February 2015) Comprehensible Software Fault and Effort Prediction: A Data Mining Approach. J Syst Softw 100:80–90
20. İrsoy O, Alpaydın E (2017) Unsupervised feature extraction with autoencoder trees, Neurocomputing, Volume 258, Pages 63–73, ISSN 0925–2312, https://doi.org/10.1016/j.neucom.2017.02.075
21. Gharehchopogh F, Abdollahzadeh B, Arasteh B (2023) An Improved Farmland Fertility Algorithm with Hyper-Heuristic Approach for solving travelling salesman problem. CMES-Computer Model Eng Sci 135(3):1981–2006. https://doi.org/10.32604/cmes.2023.024172
22. Arasteh B, Miremadi SG, Rahmani AM (2014) Developing inherently resilient Software against soft-errors based on Algorithm Level inherent features. J Electron Test 30:193–212. https://doi.org/10.1007/s10836-014-5438-8
23. Soleimanian F, Abdollahzadeh B, Barshandeh S, Arasteh B (2023) A multi-objective mutation-based dynamic Harris Hawks optimization for botnet detection in IoT, Internet of things. 24:2542–6605. https://doi.org/10.1016/j.iot.2023.100952
24. Arasteh B, Sadegi R, Arasteh K (2021) Bölen: software module clustering method using the combination of shuffled frog leaping and genetic algorithm. Data Technol Appl 55(2):251–279. https://doi.org/10.1108/DTA-08-2019-0138

Bahman Arasteh was born in Tabriz. He received the master's degree in software engineering from IAU of Arak, and the Ph.D. degree in software engineering from IAU, Tehran Science and Research Branch. Currently, he is an associate professor at Istinye University, Istanbul, Turkey. He has published more than 60 papers in refereed international journals and conferences. He is a coordinating editor for Springer's Journal of Electronic Testing and Journal of Assurance Engineering and Management. His research interests include search-based software engineering, Software testing, optimization algorithms, software fault tolerance, and software security.

Sahar Golshan received a master's degree in software engineering from Azad University of Tabriz. His research interests include Software development, maintenance and testing.

Shiva Shami received a master's degree in software engineering from Seraj Institute. His research interests include Software engineering, dependability and testing.

Farzad Kiani is an associate professor at Fatih Sultan Mehmet Vakif University in Turkey. His research interests include software testing, evolutionary algorithms and network security.