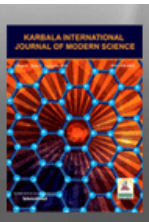

Karbala International Journal of Modern Science

Volume 9 | Issue 2 Article 9

Machine learning based Software Fault Prediction models

Gurmeet Kaur
Research Scholar CST, MRU, India, grmtkaur02@gmail.com

Jyoti Pruthi
Professor CST, MRU, India

Parul Gandhi
Professor FCA, MRIIRS, India

Machine learning based Software Fault Prediction models

Abstract

The study aims to identify soft-computing-based software fault prediction models that assist in resolving issues related to the quality, reliability, and cost of the software projects. It proposes models for implementation of software fault prediction using decision-tree regression and the K-nearest neighbor technique of machine learning. The proposed models have been designed and implemented in Python using designed metric suites as input, and the predicted-faults as output, for the real-time, wider dataset from the Promise repository. By comparing the prediction and validation results of the proposed models for the same dataset, it has been concluded that the decision-tree regression-based fault prediction model has the best performance with values of MMRE, RMSE, and accuracy of 0.0000204, 3.54, and 99.37, respectively.

Keywords

Process Metrics; Agile Metrics; predicted-fault; Software Fault Prediction; Soft-computing;

Cover Page Footnote

To annotate the PDF file, the Foxit reader software has been used.

RESEARCH PAPER

Machine Learning Based Software Fault Prediction Models

Gurmeet Kaur [a,*], Jyoti Pruthi [a], Parul Gandhi [b]

[a] MRU, India
[b] MRIIRS, India

Abstract

The study aims to identify soft-computing-based software fault prediction models that assist in resolving issues related to the quality, reliability, and cost of the software projects. It proposes models for implementation of software fault prediction using decision-tree regression and the K-nearest neighbor technique of machine learning. The proposed models have been designed and implemented in Python using designed metric suites as input, and the predicted-faults as output, for the real-time, wider dataset from the Promise repository. By comparing the prediction and validation results of the proposed models for the same dataset, it has been concluded that the decision-tree regression-based fault prediction model has the best performance with values of MMRE, RMSE, and accuracy of 0.0000204, 3.54, and 99.37, respectively.

Keywords: Process metrics, Agile metrics, Predicted-fault, Software fault prediction, Soft-computing

1. Introduction

A software bug is a flaw, error, failure, or defect that occurs when a software program is being executed and stops it from carrying out the intended function. The software flaw or defect results in a functional failure of the software. The software malfunction suggests unclear results caused by different state and environmental circumstances that result in defaults when a utility application is being executed. Both time and value can be regularized if these problems are regularly identified at the beginning of the software development phase [1]. The testing process is made simple and rapid by the early detection of faulty modules. If Software Fault Prediction (SFP) is applied additionally to advancing development for all software activities, it will raise programming standards. Defective software modules increase development and maintenance costs, contribute to software failures, and decrease user satisfaction. The goal of constructing software fault expectation models is to use methods that will typically be acquired in the life cycle of project development at the appropriate times to provide suitable early assessments of the quality of developing software projects. Additionally, measurements are commonly used for product development, internal control, programming evaluations, and process creation and verification.

Software development measures generally use the cyclomatic complexity metric (CCM), lines of codes (LOC), and Halstead complexity metric (HCM) to analyze the product's complexity. Defect Density is a product reliability metric that assesses the defect per function point or defect per KLOC. One of the important calculations in a programming standard is defect removal efficiency [2].

Chidamber Kemerer proposed class-level metrics in 1994, and they are still adopted by a community of researchers and software companies nowadays.

Received 23 November 2022; revised 8 March 2023; accepted 15 March 2023.
Available online 22 April 2023

* Corresponding author.
E-mail addresses: grmtkaur02@gmail.com (G. Kaur), jyoti@mru.edu.in (J. Pruthi), parul.fca@mriu.edu (P. Gandhi).

https://doi.org/10.33640/2405-609X.3297

Class-level metrics can only be applied with object-oriented programming. WMC, DIT, CBO, RFC, NOC, and LCOM are examples of these metrics. Process metrics also include an arrangement of measurements that is based on the data gathered during the software development life cycle. Requirement metrics, change metrics, code delta, and churn measures are examples of process metrics [3].

A software fault prediction intends to foresee the defect-prone programming modules by making use of some stressful attributes of the software project. It's regularly accomplished for a known project via preparing an expectation model by making use of project properties or attributes supported by defect data, and progressively applying the forecast model to foresee shortcomings for an uncertain task. Fig. 1 gives a summary of the software fault prediction model [4].

It is regularly shown in Fig. 1 that there are three significant parts of software fault prediction model which are as follows.

- Software defect/fault forecast methods,
- Software defect/fault dataset, and
- Functional assessment measures

In the first place, estimations of different programming metrics (like LOC, cyclomatic complexity, and so forth) are extricated, which fills in as an autonomous variable, and thus, the necessary fault information regarding the fault forecast (e.g., the measure of issues, flawed and non-broken) performs like the dependent variable.

From the programming project files containing information related to the iteration of the development of software projects, like the source code file, modify logs, and etc., the software defect information is gathered, and subsequently, the fault data is gathered from the comparing flaw archive.

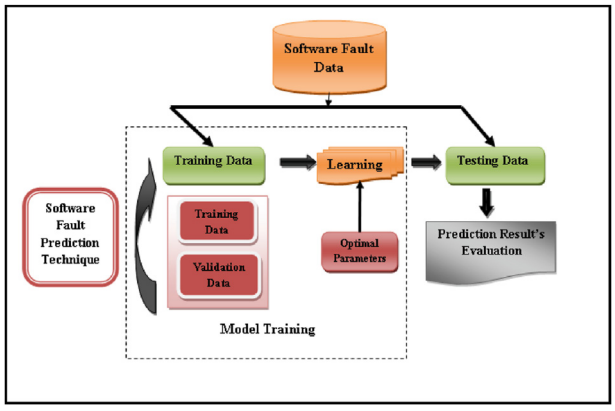

Fig. 1. Summary of Software Fault Prediction model.

At long last, the performance of the constructed defect forecast model is assessed by making use of the different execution assessment measurements such as review, exactness, precision, and accuracy.

Soft computing techniques are an assortment of useful, dynamic, and precise theories that plan to resolve complexities faced in real world problems. The strategies in soft computing include information portrayal, securing and preparing to unravel various issues in the region of computing, telecommunication networks, bio and genetic-informatics, and so forth, where the information and framework could likewise be probabilistic or questionable and would give the ideal solutions. Various techniques are employed for software fault prediction based on soft computing approaches like machine learning, neural networks, and fuzzy logic systems. Many of the existing fault prediction algorithms focus on estimating the number of defects in the modules of software based on the product metrics [4]. However, it is important to consider high-level software reliability over a possible number of faults based on standard data. As several fault prediction or expectation techniques used in the existing literature present these issues, such types of techniques have been taken under review.

The SFP model has an impact on a number of software development processes, which benefits quality, reliability, completion time, and cost. The following steps are among the objectives of this study:

- To identify and design an appropriate fault prediction metric suite for the software projects using a real-time, wider dataset.
- To evaluate the proposed metric suite using a real-time, wider dataset for the prediction of faults.
- To apply the designed metric suite as input to develop a new sophisticated fault prediction model using machine learning computing techniques with high accuracy.

2. Related study

Software fault prediction is the most effective method of identifying faulty items in modules prior to the deployment of the software since the increasing number of defects has an effect on development time, cost, and quality of a software package. This paper is intended to give a

comprehensive review of the software defect prediction field, which has been handled by various researchers utilizing a variety of soft computing methodologies.

Chatterjee [5] planned a model to arrive at an assessment of weaknesses during the initial development of the programming lifecycle and to anticipate the total number of faults. The author applied interval type-2 fuzzy reasoning to get the unexpected probability values inside the node probability tables of the assumption network. The benefits of the suggested model are that it can be used by the product engineers to accomplish a focus on the condition of the whole number of programming defects and anticipate the complete number of faults.

P. Sharma [6] proposed a framework utilizing 21 process metrics that included various software development phases and a fuzzy inference system to forecast faults. They then applied the back propagation algorithm to train the fuzzy set of rules to enhance the accuracy of forecasting and validated the explained model, which used 29 projects from the PROMISE repository. Different performance metrics, including MMRE, BMMRE, RMSE, and NRMSE, were computed by the author.

Pandey [7] designed a model using process metrics, incorporating the basic four phases of software development as inputs and faults as outputs. The results of the framework are obtained at the edge of each phase of SDLC as a predictor of fault density. The author designed a model using the fuzzy inference system and implemented it using Matlab.

Arshad [8] proposed a methodology based on semi-supervised deep fuzzy c-mean clustering (DFCM). The compression strategy was used in this study to revise programming standards. For the dataset's imbalanced classes, a classification model for programming defect forecasting was designed. The author suggested deep fuzzy C-Mean clustering that is semi-supervised, random sampling, and compression techniques. The datasets from a real-world software project by NASA and Eclipse were selected, and their performance was analyzed using F-measures and area under the curve, in addition to being compared with a variety of the most recent standard baseline approaches.

Table 1 describes the comparison of related studies performed by various researchers on soft computing-based software fault prediction. The different studies include representations of different software fault prediction models or frameworks using product or process metrics.

There are some other related studies that have been described as follows:

Yadav [15] to foresee deficiencies at each phase of the SDLC, the author depicted a fuzzy logic based on a fault expectation model that made use of the principal reliability measurements.

He [16] portrayed the metric selection process as being performed by a well-qualified assessment, and the assessment could then be one-sided as a result of this non-automated process.

Raslan [17] proposed a fuzzy-based structure for assessing effort in agile software development. To enhance the exactness of effort assessment, the focal point of this research is the use of symbolic or fuzzy logic. By utilizing the trapezoidal fuzzy membership function, the effort assessment was performed by making use of the user stories with input parameters. The author proposed a fuzzy-based procedure during their research that got Implementation level factor and fuzzy information estimations of story points, dynamic forces, and friction elements to be dealt with in numerous progressive steps to provide the final effort assessment.

Hall [18] investigated and described a quantitative model to analyze the raw presentation of measurements without contemplating the setting in which the investigations were performed. Various investigations were carried out to investigate the product measurements' capacities for programming defect expectations. The author used PROMISE and NASA data sets and began presenting their investigations using open-source software (OSS) projects. The advantage of OSS was that it was simple to check the investigation's findings and for anyone to copy the investigation.

Bishnu [19] to foresee defects inside the programme modules, the author used a quad tree-based K-means algorithm. To foresee the fault-proneness of the product package, the author employed the logistic regression (LR) procedure and the Naive Bayes (NB) algorithm. The findings indicate a need for advancement in the quality of the AI technique.

Jin [20] established fault-prone programming modules by making use of the only programming metric. They make use of ANNs to dispose of poor attributes and then utilize the SVM to predict fault-proneness by making use of the chosen features. The author described that use of the proposed approach requires neither master's information nor additional expense for the event group as the proposed approach is predicated exclusively on programming measurements.

The coupling and cohesion estimation data acquired for the 83 framework classes was evaluated by Briand [21]. The examination strategy included a look at the different statistics, a look at univariate

Table 1. Comparative study of various soft-computing based SFP models.

Ref. no.	Objective	Methodology/Metrics used	Data Set/Features	Findings	Performance measurement tool
[6]	To measure the effectiveness of various metrics in predicting defects in agile software projects, the fuzzy inference system proposed.	Neuro-Fuzzy hybrid approach and Process metric	Data sets from PROMISE repository	As per performance evaluation methods, the proposed framework provides improved accuracy (especially for projects with a size of less than 50 KLOC).	MMRE, BMMRE, RMSE, NRMSE,
[9]	Proposed new variations of WOA as wrapper algorithms to deal with the feature (metric) selection issues in SFP applications.	Roulette wheel, Tournament, Linear position, and random based, stochastic universal sampling and Process metric.	To pass judgment on the presentation of the proposed upgrade from the PROMISE archive, the Author used 17 accessible SFP datasets.	The deep investigation presented that the suggested TBWOA (tournament approach) exceeded the principal WOA.	Average AUC and running time.
[10]	Proposed a system using inter-version and inter-project assessment, to recognize the product defect.	Based on fuzzy logic. During the working of the framework, the past assignments or adventure variations are taken for preparing sets, and thus, the present version or exercises are taken as testing sets. Product & Process metric	The author performed investigation taken from PROMISE store, even as on PDE and JDT adventures and other more nine open source adventures.	The assessment results showed that the suggested system showed outstandingly good outcomes for Eclipse-PDE and Eclipse-JDT based projects.	AUC and GM, Root mean square error RMSE.
[11]	Proposed an exact framework to anticipate programming deficiency and assisted the feature selection algorithm for classification issues.	Proposed upgraded binary moth flame Optimization EBMFO with adaptive synthetic sampling ADASYN. Product & Process metric	It makes use of the wrapper feature selection and improves the data set, and it portrays three unique classifiers decision tree DT, Linear discriminant analysis (LDA), and k-nearest neighbors KNN.	ADASYN attempts to beat the imbalanced data issue while BMFO perform as a feature selection.	Average AUC
[12]	Planned to deal with a layered recurrent neural organization (L-RNN) and to choose significant programming measurements by making use of the diverse feature selection algorithms.	The author applied distinctive Feature Selection (FS) algorithms in iteration as (i.e., Binary Particle Swarm Optimization (BPSO), Binary Genetic Algorithm (BGA), and Binary Ant Colony Optimization (BACO)). Process metric	To recognize the introduction of the suggested approach, the author analyzed 19 real-world programming projects from PROMISE store.	The results provided assurance of the importance of feature selection in developing a good classifier as opposed to employing a predefined arrangement of features.	AUC

(*continued on next page*)

Table 1. (continued)

Ref. no.	Objective	Methodology/Metrics used	Data Set/Features	Findings	Performance measurement tool
[13]	Performed near-investigation of anticipated for different kinds of ANNs	Spotlighted on two kinds of neural network as Elman neural network and ANN-feed forward back-propagation neural network. Product & Process metric	From different software companies, the author applied data sets from 21 agile based projects.	The feed forward back-propagation neural network has high algorithm speed, fixed algorithm time and adaptation to non-critical failure concerning Elman organization.	MSE, MMRE, and assumption (PRED (X)).
[14]	Depicted adequately anticipate the product defect proneness in the soft modules with a tendency to lessen programming maintenance cost.	A DDN (deep neural network) forecast technique for programming flaw proneness module assisted by BAPS (Bound particle swarm optimization) dimensional reduction and 21 programming deficiencies estimation indexes suggested. Process metric	To see the proficiency of the approach, a data set from real-world programming projects (NASA and Eclipse) was made use.	The analysis result showed that the BPSO-based dimensionality reduction innovation can enhance the organization structure and acquire better execution.	F-measure, area under the curve (AUC), and probability of detection (pd).

regression, and a look at principal component analysis against the faulty data and their relationship to measure. The author made several recommendations based on the findings, including: it has been demonstrated to import solid coupling, a consistent pointer of flaw trace, and a solid stress should be placed on method innovation.

For the purpose of predicting software faults, Catal [22] reviewed the literature on the various methodologies, techniques, metrics, and datasets. The conclusion demonstrates that, following the design of the PROMISE data store in 2005, the application of machine learning computation increased steadily.

Chinna Gounder Dhanajayan [23] uses Bayesian classification strategies for effective programming defect forecasting. A Bayesian arrangement is applied to spot defective and non-defective modules by making use of a probability distribution. He applied a vigorous similarity-aware clustering algorithm to assist the similarity measure of the feature inside the selected data sets. The proposed methodology achieves a lower mistake rate. The precision, valuable review, and F-measures of the proposed strategy are predominant. The author described the proposed rank-based methodologies for predicting the precise number of programming modules against product defects.

Ghosh D [24] built up a system for multi-fault localization utilizing deep learning convolution neural networks (CNN) might be a notable design for deep learning CNN. The framework was designed to gather the massive amount of information from the experiments and concentrate the significant features. For multi-flaw restriction, CNN decreases the parameters and complexity of the product, and along these lines, it speeds up the training cycle. Later on, to execute CNN on archived multi-shortcoming benchmark programs, and, furthermore, to carry out recurrent neural networks (RNN) to limit the different flaws.

N Kalaivani [25] derived an analysis focused on processes for the introduction of the number of faults, number of modifications, and number of newly revised lines. They analyzed and showed that OO metrics believe data for changes in artefacts within the source code document. It had been noticed that a number of the changes might not have an immediate reference to the software process. Finally, the researchers reported that system

metrics were a frequent and efficient addition to forecast modules that usually encouraged prediction methods.

In addition, there are some other studies [26–29] that have been included in the present research but not described in detail.

In the present study, the proposed framework described the design of the proposed metric suite based on formulas using process metrics as inputs for a real-time, wider dataset. Next, for the implementation of SFP models, the proposed metric suite is considered as inputs (independent variables) and predicted-faults as outputs (a dependent variable).

3. Metric design framework

This section includes the design of the proposed metric suite requirement phase-based metrics (M_r) and agile-based metrics (M_a) based on a formula using process metrics as inputs for a real-time, wider dataset from the Promise repository. A portion of a real-time, wider dataset described in Appendix 1 from Ref. [30] presents the different columns of variables categorized as requirement phase, design phase, coding phase, testing phase, size, and faults of each software project.

Variables can either be dependent or independent. The RC, RS, RIW, DTE, PM, CTE, DPF, TTE, and SI are considered independent or input variables that are used to predict faults. The dependent variables are response or output variables and also represent the effect of independent variables on the present dataset. Fault is considered a dependent or output variable. Table 2 describes the description of variables at different phases of software development.

Table 2. Description of variables.

Independent Variables	Description
Requirement Phase	
RC	Requirements Complexity
RS	Requirements Stability
RIW	Review, Inspection And Walk-Through
Design Phase	
DTE	Design Team Experience
PM	Process Maturity
Coding Phase	
CTE	Coding Team Experience
DPF	Defined Process Followed
Testing Phase	
TTE	Testing Team Experience
SI	Stake-Holders Involvement
Dependent Variables	
Faults(s)	Fault(s) (Predicted)

Table 3 (a). Python code for M_r.

```
import pandas as pd
"""### Load Data"""
path = r"C:\Users\Dell\Desktop\dataset03.csv"
df = pd.read_csv(path)
df
fori in range(0,len(df)):
Mr = 2*df['RC']/(df['RS']+df['RIW'])
print("Mr values are:")
print(Mr)
```

3.1. Requirement-based metric (M_r)

The metric has been proposed for the dataset described in Appendix 1 from Ref. [30] based on requirement phase process metrics using RC, RS, and RIW. The RC is directly related to the faults, while the RS and RIW are inversely related to the faults. The design of the requirement phase-based metric M_r has been detailed in equation (2) for i = 1 … n projects of dataset.

As M_r shows the relationship between the requirement phase metrics RC, RS, and RIW, it is named as requirement-based metric M_r. The Python code for the implementation of the metric M_r has been described in Table 3(a).

$$m1(i) = \frac{2\,and RC(i)}{RS(i) or RIW(i)} \tag{1}$$

$$M_r(i) = \begin{cases} m1(i) & if m1(i) < 0.99 \\ 0.99 & if m1(i) \geq 0.99 \end{cases} \tag{2}$$

Table 3(b). Implementation results of the M_r.

Sr. No.	RC	RS	RIW	M_r
1	0.63	0.63	0.92	0.812903
2	0.63	0.34	0.75	0.99
3	0.15	0.34	0.92	0.238095
4	0.34	0.63	0.75	0.492754
5	0.34	0.63	0.75	0.492754
6	0.63	0.63	0.75	0.913043
7	0.63	0.15	0.75	0.99
8	0.63	0.15	0.5	0.99
9	0.86	0.63	0.75	0.099
10	0.63	0.05	0.75	0.99
11	0.05	0.34	0.75	0.091743
12	0.05	0.34	0.5	0.119048
13	0.63	0.34	0.75	0.099
14	0.86	0.05	0.5	0.099
15	0.05	0.34	0.75	0.091743
16	0.34	0.15	0.5	0.99
17	0.63	0.34	0.92	0.99
18	0.34	0.34	0.92	0.539683
19	0.34	0.86	0.92	0.382022
20	0.05	0.86	0.92	0.05618

Table 4(a). Python code for M_a.

```
import pandas as pd
"""### Load Data"""
path = r"C:\Users\Dell\Desktop\dataset04.csv"
df = pd.read_csv(path)
df
fori in range(0,len(df)):
m2=(df['DTE']+df['PM']+df['CTE']+df['DPF']+df['TTE']+df['SI'])/6
print("Ma values are:")
print(Ma)
```

The implementation results of M_r for the sample of 20 software projects from a real-time, wider dataset have been presented in Table 3(b).

3.2. Agile-based metric (M_a)

Another metric has been proposed for the dataset described in Appendix 1 from Ref. [30] based on the metric of the design, coding, and testing phase of software development, using DTE, PM, CTE, DPF, TTE, and SI as independent variables, and these variables are inversely related to the "faults", which is the dependent variable. Here, a set of the independent variables relate in the same way with the dependent variables, and all the independent variables are in the same place at the same time. This behaviour of metric describes the nature of agile and is named an agile-based metric, M_a and explained in equation (4) for i = 1 ….n projects in a dataset.

The Python code for the implementation of metric M_a has been described in Table 4(a).

Table 4(b). Implementation results of the M_a.

Sr. No.	DTE	PM	CTE	DPF	TTE	SI	M_a
1	0.63	0.92	0.86	0.75	0.63	0.86	0.775
2	0.15	0.75	0.34	0.75	0.34	0.34	0.445
3	0.34	0.75	0.86	0.75	0.34	0.86	0.65
4	0.63	0.5	0.63	0.5	0.34	0.63	0.538333
5	0.63	0.75	0.63	0.75	0.34	0.63	0.621667
6	0.63	0.75	0.63	0.5	0.63	0.63	0.628333
7	0.86	0.75	0.34	0.5	0.63	0.63	0.618333
8	0.63	0.75	0.63	0.75	0.34	0.63	0.621667
9	0.63	0.75	0.63	0.75	0.63	0.63	0.67
10	0.63	0.75	0.63	0.75	0.63	0.86	0.708333
11	0.63	0.75	0.63	0.75	0.63	0.86	0.621667
12	0.34	0.75	0.34	0.75	0.15	0.34	0.445
13	0.63	0.75	0.63	0.75	0.34	0.63	0.621667
14	0.05	0.75	0.05	0.25	0.05	0.63	0.296667
15	0.63	0.75	0.63	0.75	0.63	0.63	0.67
16	0.63	0.75	0.34	0.25	0.34	0.63	0.49
17	0.15	0.75	0.63	0.75	0.63	0.63	0.59
18	0.34	0.75	0.15	0.5	0.34	0.34	0.403333
19	0.86	0.75	0.86	0.75	0.86	0.86	0.823333
20	0.63	0.75	0.63	0.75	0.63	0.86	0.708333

$$m2(i) = \frac{DTE(i) or PM(i) or DPF(i) or CTE(i) or SI(i) or TTE(i)}{6} \quad (3)$$

$$M_a(i) = \begin{cases} m2(i) if m2(i) < 0.99 \\ 0.99 \ if m2(i) \geq 0.99 \end{cases} \quad (4)$$

The implementation results of the M_a for a sample of 20 software projects from a real-time, wider dataset have been presented in Table 4(b).

Both of the metric suites, M_r and M_a, had been validated theoretically and empirically in previous research work for a real-time, wider dataset of software projects.

4. Proposed models

Soft computing techniques are an assortment of useful, dynamic, and precise theories that plan to resolve the complexities faced in real-world problems. The strategies in soft computing include information portrayal, securing and preparing to unravel various issues in the region of computing, telecommunication networks, bio-genetic-informatics, and so forth, where the information or framework could likewise be probabilistic or questionable and would give the ideal solutions. The soft computing methods are primarily classified as: neural networks, fuzzy computing, evolutionary computing, machine learning, and a few more techniques.

To implement a fault prediction model with the proposed metrics M_r and M_a as inputs and predicted-faults as an output parameter, machine learning-based techniques have been used. Fig. 2 shows the basic workings of machine learning for

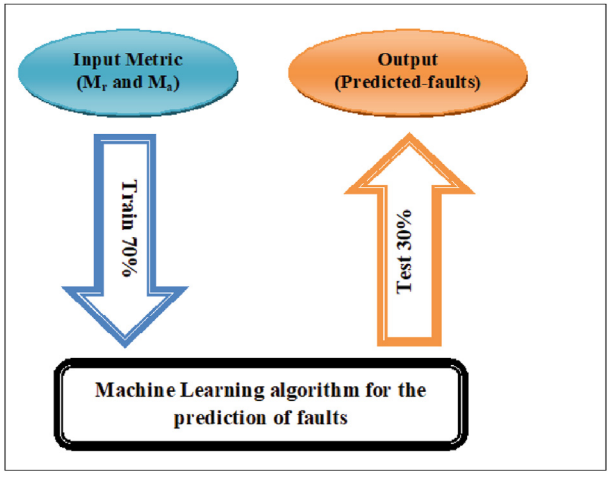

Fig. 2. SFP model using machine learning.

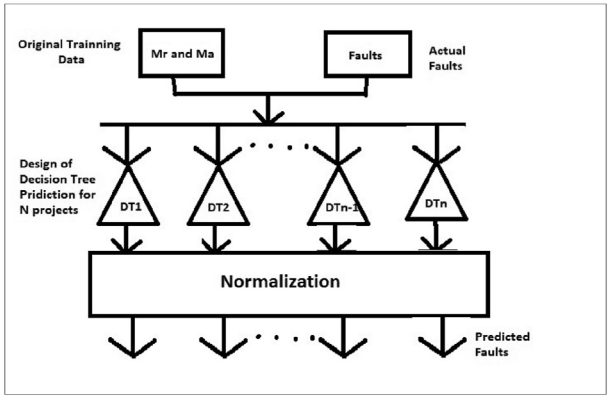

Fig. 3. SFP model using DTR computing.

prediction. The machine learning techniques are either supervised or unsupervised. The supervised learning algorithms can be applied to regression and classification problems. The types of supervised learning algorithms are: linear, logistic, decision tree, k-nearest neighbor, and random forest. The unsupervised learning algorithms are based on clustering and associations.

In the present work of the implementation of fault prediction models, the supervised learning algorithms Decision tree regression (DTR) and K nearest neighbor (KNN) computing have been applied.

4.1. Decision tree regression model

DTR is a supervised learning algorithm that is rule-based and non-parametric and can be applied to predict faults in software projects. The DTR algorithm has been designed using implicit criteria that are applicable for linear and non-linear relationships between dependent variables and independent variables [31]. Fig. 3 shows the workings of the decision tree regression-based SFP model. Hence, it has been applied to manage the critical non-linear relationship between the proposed metric suite, M_r and M_a, and predicted-faults of the software project. The pseudo-code for decision tree prediction is as follows:

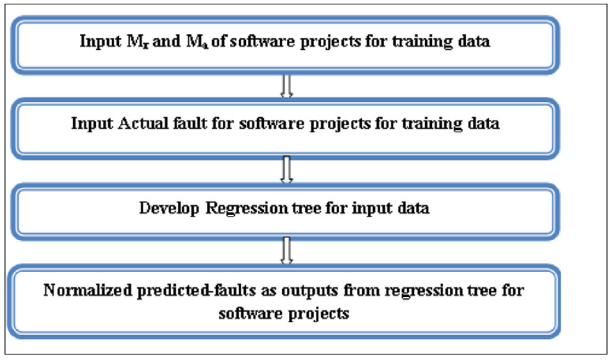

Fig. 4. Flowchart for SFP model using DTR.

1. The complete training data set is treated as the root.
2. The values of the independent variable are selected to be definite.
3. The documentations are assigned recursively by using independent variable values.
4. By referring to analytical technique, the order of appending an independent variable as the inner or root node of the tree is finalized.

To implement a software fault prediction model, a regression-based tree for each project has been designed. The proposed metric suite M_r and M_a treated as independent variables, and predicted-faults of the projects are considered as dependent or target variables of each regression tree. The values of M_r and M_a are fed as input for each regression tree, and predicted-faults have been obtained as output.

4.1.1. FlowChart

The flowchart for the SFP model using DTR has been described in Fig. 4 as follows:

Algorithm:

Input states: The proposed metric suites are M_r, M_a and actual-faults for the training dataset, while M_r, and M_a for the testing dataset.
Output states: The predicted-faults for the training and testing datasets.
The algorithm for SFP model using DTR by making the use of training data set has been described with following steps as follow.
Step 1: Input the M_r and M_a metric suite of projects as an independent or predictor variable.
Step 2: Input the actual-faults of project i as the independent variable.
Step 3: Develop the regression tree, for project i ….. 1 to n, where n is the number of projects
Step 4: The predicted-faults are normalized by each tree using the formula of equations (5) and (6) [32].

Table 5(a). Decision Tree Regression –based SFP Model training code.

```
from sklearn.tree import DecisionTreeRegressor
regressor1 = DecisionTreeRegressor(criterion = 'mse')
regressor1.fit(X_train, y_train)
print("accuracy of the model through Decision Tree Regression - ML Model Training is:")
regressor1.score(X_test, y_test)
```

Table 5(b). Distance computation formulas.

Distance Computations Methods	Formula
Euclidean	$\sqrt{\sum(x_i - y_i)^2}$
Manhattan	$\sum\lvert x_i - y_i \rvert$
Hamming distance	$\sum\lvert x_i - y_i \rvert$ If x = y then D = 0 If x≠y then D = 1

Hence, the predicted faults are normalized by each tree denoted by DT(i), i = 1 ….n as

$$T(i) = \sum_{i=1}^{n} DT(i) \tag{5}$$

$$P(i) = \frac{DT(i)}{T(i)} \tag{6}$$

The implementation of the software fault prediction using DTR for a real-time, wider dataset using Python has been presented in Table 5(a) as follows:

4.1.2. Execution

The outcomes predicted-faults using decision tree regression are proven in Table 6 for a sample of 20 software projects from a real-time, wider dataset [30]. For example, Project No. 11 has values of M_r and M_a of 0.091743 and 0.621667, with computed predicted-faults of 109 that are the same as the actual fault. The accuracy of the model through decision-tree regression is 0.9937344910413299, and the prediction results have been computed and analyzed in Table 6.

4.2. K nearest neighbor model

K-nearest neighbor (KNN) computing is the basic and simplest supervised learning technique, which is applicable when there is little or no advance knowledge of the distribution of the dataset. The KNN algorithm can be applied for both regression and classification-based cases. The KNN algorithm is used to predict any value of a dataset using feature similarity. The value of the dataset is computed on the basis of how nearly it matches the point in the training data set. The KNN algorithm includes the following computations:

Table 5(c). KNN Regression - based SFP Model training code.

```
from sklearn.neighbors import KNeighborsRegressor
regressor1 = KNeighborsRegressor(n_neighbors = 3)
regressor1.fit(X_train, y_train)
y_pred = regressor1.predict(X_test)
print("accuracy of the model through KNN - ML Model Training
  is:")
regressor1.score(X_test, y_test)
```

4.2.1. Distance computation

As discussed, the algorithm KNN predicts the outcome value of a point using K neighbors nearer to that point. To do this, it requires measuring the distance between given points and cases from the given dataset. There are different methods available to compute distance D(x, y), such as Euclidean, Manhattan, and Hamming distances [33]. Table 5(b) describes the different methods for distance computations.

Where x is a given point from training data and y is any case from the given dataset. Both the Euclidean and Manhattan methods are used for continuous variables, while the Hamming distance method is applicable for category variables. The Euclidean method computes as the sum of squares of X and Y values. The Manhattan method computes as the sum of their absolute difference, and Hamming distance computes as follows: if x and y are not the same, then D will be one, otherwise it will be zero.

4.2.2. Prediction using the K factor

The next step is to select the value of k. The key value provides the number of neighbors that can be

Table 6. Comparison of prediction results of DTR and KNN-based SFP models.

Pr.No.	Inputs		Predicted-faults using		Actual Faults
	M_r	M_a	DTR	KNN	
1	0.812903	0.775	343	249	209
2	0.99	0.445	373	365	373
3	0.238095	0.65	204	218	204
4	0.492754	0.538333	53	53	53
5	0.492754	0.621667	29	53	29
6	0.913043	0.628333	71	71	71
7	0.99	0.618333	254	239	90
8	0.99	0.621667	129	139	129
9	0.099	0.67	672	502	672
10	0.99	0.708333	1768	1371	1768
11	0.091743	0.621667	109	127	109
12	0.119048	0.445	688	485	688
13	0.099	0.621667	476	504	476
14	0.099	0.296667	928	765	928
15	0.091743	0.67	196	188	196
16	0.99	0.49	184	323	184
17	0.99	0.59	680	651	680
18	0.539683	0.403333	412	529	412
19	0.382022	0.823333	91	89	91
20	0.05618	0.708333	5	5	5

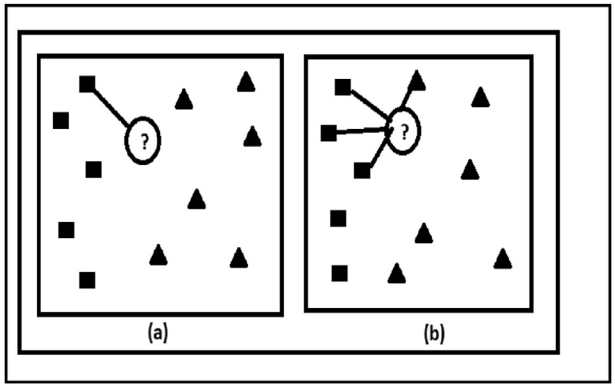

Fig. 5. (a) & (b) selection of K-factor.

chosen to assign a value to any new point in the dataset. For example, for a given point Y, the value of K is 3, which means Y has three neighbors, which gives the best value. Fig. 5(a) & (b) have been described the selection of value of k.

For regression, the mean of the k nearest neighbors provides the KNN prediction as:

$$Y = Y\frac{1}{k}\sum_{i=1}^{k} yi \tag{7}$$

Where Y is the predicted value of a new given point and y_i is the ith project of the training sample of the dataset.

It noticed that final output can be changed on the basis of selection of value of k, e.g. When K = 3, the output value will be calculated by averaging the three best neighboring values, and if K = 4, the final output will be calculated by averaging the four best neighboring values.

If K is equal to 1, a very low value and 10 or 11 is a very high value, the algorithm is likely to give high errors on the training dataset as well as on the validation of the dataset and model over fit. To locate the best value of k, grid search techniques can be used.

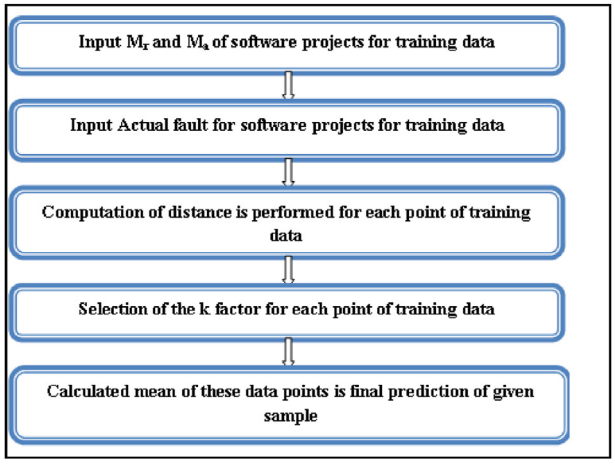

Fig. 6. Flowchart for SFP model using KNN.

Algorithm:

Input states: The proposed metric suites M_r, M_a and actual-faults for training dataset while M_r, and M_a for testing dataset.
Output states: The predicted-faults for the training and testing datasets.
The KNN algorithm for the SFP model is as follows:
Step 1: Input the M_r and M_a metric suite of projects as an independent or predictor variable.
Step 2: Input the actual-faults of Project i as the independent variable.
Step 3: The computation of distance is performed for each point of training data.
Step 4: Selection of the k factor for each point of training data.
Step 5: The final prediction for the given data sample is given by calculating mean of these data points.
The implementation of the KNN algorithm for software fault prediction on a real-time, wider dataset using Python has been presented in Table 5(c) as follows:

4.2.3. Flowchart

The flowchart for the SFP model using KNN has been described in Fig. 6 as follows:

4.2.4. Execution

The outcomes predicted-faults using KNN are proven in Table 6 for a sample of 20 software projects from a real-time, wider dataset [30]. For example, Project No. 11 has values M_r and M_a of 0.091743 and 0.621667, with computed predicted-faults of 127 and actual faults of 109. The accuracy of the model through KNN is 0.9527495684080002, and prediction results have been computed and analyzed in Table 6.

5. Results and discussion

After implementation, the proposed fault prediction model using a soft computing approach has been verified by using a real-time, wider dataset of software projects. A sample of 20 software projects has been presented in Table 6. The metric suites M_r and M_a have been considered as inputs to implement software fault prediction models using DTR and KNN techniques of machine algorithms, which give outcomes as predicted-faults.

5.1. Prediction results

The predictions of proposed models have been computed for a real-time, wider dataset of software projects. Table 6 describes the predicted faults as outcomes computed using DTR and KNN computing for a sample of 20 projects. Also, Table 6 has described the comparison of similar outcomes computed by DTR and KNN computing for the same real-time, wider dataset [30].

It has been deduced from Table 6 that the proposed DTR-based fault prediction model is showing the best results with an accuracy of 0.9937 as compared to KNN-based fault prediction models for the same real-time, wider dataset of software projects.

5.2. Validation results

There are several analytical techniques that may be used to compare the performance of the developed SFP model and quantify the degree of accuracy of the proposed SFP models. These methods can be either continuous or categorical to compare the performance of the developed SFP model and quantify the degree of accuracy of the proposed SFP model. Continuous studies on the difference between actual and predicted outcomes, goodness-of-fit, classified and appropriate outcomes, and accuracy were conducted. These measurements focused on predicting the number of defects. To validate the proposed SFP models, the outcomes classified as "predicted-fault" have been analyzed, and their performance measures based on continuous studies, namely MMRE, RMSE, and R-Square, have been computed. In addition to this precision, recall, F1-measure, and geometric mean have been computed for the proposed SFP models. Table 7 compares the performance measures of the DTR-based SFP model along with the KNN-based SFP model for the same real-time, wider dataset from Ref. [30].

Table 7. Compare the performance measures of DTR-based SFP models to KNN-based SFP models.

	DTR −based	KNN-based
RMSE	3.54	9.029
MMRE	0.0000204	0.000124
R-Square	99.373449	95.274956
Precision	0.99876	0.987654
Recall	1.00	0.97453
F1-measure	0.999786	0.98654
Geometric Mean	0.999876	0.97689

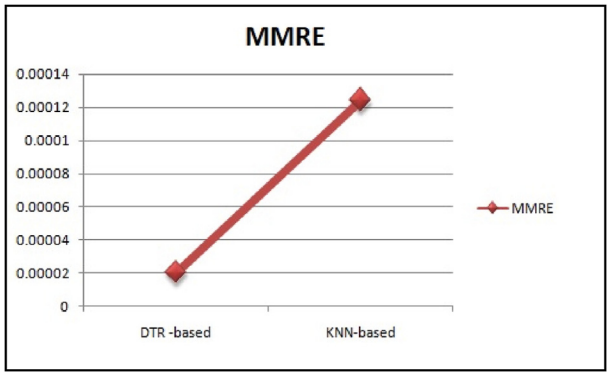

Fig. 7(a). MMRE for SFP models.

Fig. 7a and b shows the graphical representation of MMRE and RMSE, respectively, computed by the DTR-based SFP model and their comparison with the KNN-based SFP model.

6. Threats to validity

The proposed machine learning-based software fault prediction was examined in this study using a case study methodology. Measures of construct, internal, external, and reliability validity can be used to evaluate the effectiveness of the proposed models [34].

Construct validity is a measure of how closely the operational measures examined in the research reflect the researcher's views. By developing models based on machine learning for software failure prediction and employing software metrics suites, these risks have been reduced to a minimum. Additionally, the framework's training and algorithm allow for defects and provide accessibility in the prediction system. A real-time, wider data set taken from the PROMISE repository has been used

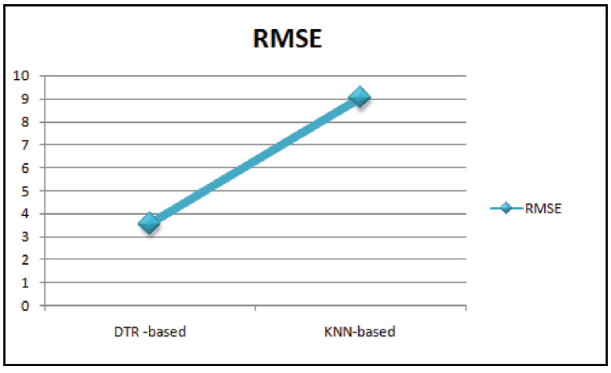

Fig. 7(b). RMSE for SFP models.

to test and validate the models by applying statistical measures.

Generally, the development of study is associated with internal validity [35]. The process measurements from the real-time data set from the study were used to create the metric suites. To guarantee the qualitative properties of the metrics suites used, the metrics suites have been theoretically and empirically validated in previous research studies. Additionally, a well-defined approach that uses a domain expert for the evaluation of metrics has been used to assure internal validity.

External validity refers to a result's ability to represent the findings of a study [36,37]. Since there isn't a large enough population to generate statistically representative samples, the sample size is a potential threat to generalization. This kind of risk is avoided by choosing all the project cases listed in the PROMISE data repository. In addition, the goal of the case study research is to compare the study results and, if applicable, draw an analytical generalization from them. Consequently, new studies with special datasets need to be performed.

Reliability In the study, RMSE and MMRE are appropriate statistical metrics that have been used to measure reliability. Additionally, the size of the training dataset has been made flexible to achieve minimal mean square error, which improved the prediction accuracy of the SFP models.

7. Conclusion

The present study focuses on the implementation of software fault prediction models that predict faults in software projects. This paper describes the implementation of software fault prediction models using soft computing-based DTR and KNN. The metric suite M_r and M_a have designed using the formula given by equations (2) and (4) and implemented it in python for a real-time, wider dataset of software projects. The M_r and M_a have been validated theoretically and empirically by performing correlation and regression analysis in previous research work.

In the present study, the software fault prediction models have been designed and implemented using DTR and KNN algorithms of machine learning by applying the proposed metric suite M_r and M_a as input or independent variables and predicted-faults as output or dependent variables, and their prediction results have been presented in Table 6 for the real-time wider dataset of software projects. Table 6 describes the comparison of the results of DTR and KNN computing for a sample dataset of 20 software projects. To validate the proposed models, the performance measures MMRE, RMSE, R-square, precision, recall, accuracy, and geometric mean have been computed. It has been deduced from Table 7 that the performance of proposed models based on DTR is much superior as compared to the KNN for the same dataset, with values of MMRE, RMSE, and R-square of 3.54, 0.0000204, and 99.373449, respectively.

The proposed model provides strength in concern of reducing time complexity and budget improving reliability and quality of the software projects in development.

In future studies, for the same dataset, the validation of proposed models can be compared with that of already developed soft computing-based SFP models.

The weakness of the proposed DTR-based SFP model needs to be validated for other datasets. It can be completed in a future study.

Funding

No financial support was received for this work.

Conflict of interest

There is no conflict of interest in this work.

Acknowledgement

I want to express my heartfelt gratitude and appreciation to Prof. Jyoti Pruthi, who has guided me during the research work. Without her constant support and inspiration, my efforts would not have taken shape. I am truly grateful for her guidance and motivation to complete the work.

I am deeply grateful to Prof. Parul Gandhi, who guided and inspired me. Her unwavering support played a pivotal role in shaping my efforts, and I want to express my sincere appreciation to her. I would be honoured to have her continued support in all my future endeavours.

I sincerely appreciate the managing editor and the anonymous reviewers for their insightful feedback and comments to enhance the contribution's excellence.

Last but not least, I express my heartfelt gratitude to Dr. Ashwini Kush, who has suggested that I submit my research work to this prestigious journal.

Appendix 1.

Sr. No.	RS	RC	RIW	DTE	PM	CTE	DPF	TTE	SI	Size (KLOC)	Faults
1	0.86	0.34	0.92	0.86	0.75	0.86	0.75	0.86	0.86	11	91
2	0.8428	0.3332	0.9016	0.8428	0.735	0.8428	0.735	0.8428	0.8428	10.78	89.18
3	0.825944	0.326536	0.883568	0.825944	0.7203	0.825944	0.7203	0.825944	0.825944	10.5644	87.3964
4	0.809425	0.320005	0.865897	0.809425	0.705894	0.809425	0.705894	0.80942512	0.80942512	10.353112	85.64847
5	0.793237	0.313605	0.848579	0.793237	0.691776	0.793237	0.6917761	0.793236618	0.793236618	10.1460498	83.9355
6	0.777372	0.307333	0.831607	0.777372	0.677941	0.777372	0.6779406	0.777371885	0.777371885	9.94312877	82.25679
7	0.761824	0.301186	0.814975	0.761824	0.664382	0.761824	0.6643818	0.761824448	0.761824448	9.74426619	80.61166
8	0.746588	0.295163	0.798675	0.746588	0.651094	0.746588	0.6510942	0.746587959	0.746587959	9.54938087	78.99942
9	0.731656	0.289259	0.782702	0.731656	0.638072	0.731656	0.6380723	0.731656199	0.731656199	9.35839325	77.41944
10	0.717023	0.283474	0.767048	0.717023	0.625311	0.717023	0.6253108	0.717023075	0.717023075	9.17122538	75.87105
11	0.702683	0.277805	0.751707	0.702683	0.612805	0.702683	0.6128046	0.702682614	0.702682614	8.98780088	74.35363
12	0.688629	0.272249	0.736673	0.688629	0.600549	0.688629	0.6005485	0.688628962	0.688628962	8.80804486	72.86655
13	0.674856	0.266804	0.721939	0.674856	0.588538	0.674856	0.5885375	0.674856382	0.674856382	8.63188396	71.40922
14	0.661359	0.261468	0.707501	0.661359	0.576767	0.661359	0.5767668	0.661359255	0.661359255	8.45924628	69.98104
15	0.648132	0.256238	0.693351	0.648132	0.565231	0.648132	0.5652315	0.64813207	0.64813207	8.29006136	68.58142
16	0.635169	0.251113	0.679484	0.635169	0.553927	0.635169	0.5539268	0.635169428	0.635169428	8.12426013	67.20979
17	0.622466	0.246091	0.665894	0.622466	0.542848	0.622466	0.5428483	0.62246604	0.62246604	7.96177493	65.86559
18	0.610017	0.241169	0.652576	0.610017	0.531991	0.610017	0.5319913	0.610016719	0.610016719	7.80253943	64.54828
19	0.597816	0.236346	0.639525	0.597816	0.521351	0.597816	0.5213515	0.597816385	0.597816385	7.64648864	63.25732
20	0.58586	0.231619	0.626734	0.58586	0.510924	0.58586	0.5109245	0.585860057	0.585860057	7.49355887	61.99217
21	0.34	0.63	0.75	0.15	0.75	0.34	0.75	0.34	0.34	14	373
22	0.3332	0.6174	0.735	0.147	0.735	0.3332	0.735	0.3332	0.3332	13.72	365.54
23	0.326536	0.605052	0.7203	0.14406	0.7203	0.326536	0.7203	0.326536	0.326536	13.4456	358.2292
24	0.320005	0.592951	0.705894	0.141179	0.705894	0.320005	0.705894	0.32000528	0.32000528	13.176688	351.0646
25	0.313605	0.581092	0.691776	0.138355	0.691776	0.313605	0.6917761	0.313605174	0.313605174	12.9131542	344.0433
26	0.307333	0.56947	0.677941	0.135588	0.677941	0.307333	0.6779406	0.307333071	0.307333071	12.6548912	337.1625
27	0.301186	0.558081	0.664382	0.132876	0.664382	0.301186	0.6643818	0.301186409	0.301186409	12.4017933	330.4192
28	0.295163	0.546919	0.651094	0.130219	0.651094	0.295163	0.6510942	0.295162681	0.295162681	12.1537575	323.8108
29	0.289259	0.535981	0.638072	0.127614	0.638072	0.289259	0.6380723	0.289259428	0.289259428	11.9106823	317.3346
30	0.283474	0.525261	0.625311	0.125062	0.625311	0.283474	0.6253108	0.283474239	0.283474239	11.6724687	310.9879
31	0.277805	0.514756	0.612805	0.122561	0.612805	0.277805	0.6128046	0.277804754	0.277804754	11.4390193	304.7682
32	0.272249	0.504461	0.600549	0.12011	0.600549	0.272249	0.6005485	0.272248659	0.272248659	11.2102389	298.6728
33	0.266804	0.494372	0.588538	0.117708	0.588538	0.266804	0.5885375	0.266803686	0.266803686	10.9860341	292.6993
34	0.261468	0.484484	0.576767	0.115353	0.576767	0.261468	0.5767668	0.261467612	0.261467612	10.7663135	286.8454
35	0.256238	0.474794	0.565231	0.113046	0.565231	0.256238	0.5652315	0.25623826	0.25623826	10.5509872	281.1084
36	0.251113	0.465299	0.553927	0.110785	0.553927	0.251113	0.5539268	0.251113495	0.251113495	10.3399674	275.4863
37	0.246091	0.455993	0.542848	0.10857	0.542848	0.246091	0.5428483	0.246091225	0.246091225	10.1331681	269.9765
38	0.241169	0.446873	0.531991	0.106398	0.531991	0.241169	0.5319913	0.241169401	0.241169401	9.93050473	264.577
39	0.236346	0.437935	0.521351	0.10427	0.521351	0.236346	0.5213515	0.236346012	0.236346012	9.73189463	259.2855
40	0.231619	0.429177	0.510924	0.102185	0.510924	0.231619	0.5109245	0.231619092	0.231619092	9.53725674	254.0998
41	0.15	0.63	0.75	0.86	0.75	0.34	0.5	0.63	0.63	19	90
42	0.34	0.15	0.92	0.34	0.75	0.86	0.75	0.34	0.86	21	204
43	0.3332	0.147	0.9016	0.3332	0.735	0.8428	0.735	0.3332	0.8428	20.58	199.92
44	0.326536	0.14406	0.883568	0.326536	0.7203	0.825944	0.7203	0.326536	0.825944	20.1684	195.9216
45	0.320005	0.141179	0.865897	0.320005	0.705894	0.809425	0.705894	0.32000528	0.80942512	19.765032	192.0032
46	0.313605	0.138355	0.848579	0.313605	0.691776	0.793237	0.6917761	0.313605174	0.793236618	19.3697314	188.1631
47	0.307333	0.135588	0.831607	0.307333	0.677941	0.777372	0.6779406	0.307333071	0.777371885	18.9823367	184.3998
48	0.301186	0.132876	0.814975	0.301186	0.664382	0.761824	0.6643818	0.301186409	0.761824448	18.60269	180.7118
49	0.295163	0.130219	0.798675	0.295163	0.651094	0.746588	0.6510942	0.295162681	0.746587959	18.2306362	177.0976
50	0.34	0.05	0.75	0.63	0.75	0.63	0.75	0.63	0.63	22	196

51	0.3332	0.049	0.735	0.6174	0.735	0.6174	0.735	0.6174	0.6174	21.56	192.08
52	0.326536	0.04802	0.7203	0.605052	0.7203	0.605052	0.7203	0.605052	0.605052	21.1288	188.2384
53	0.320005	0.04706	0.705894	0.592951	0.705894	0.592951	0.705894	0.59295096	0.59295096	20.706224	184.4736
54	0.313605	0.046118	0.691776	0.581092	0.691776	0.581092	0.6917761	0.581091941	0.581091941	20.2920995	180.7842
55	0.307333	0.045196	0.677941	0.56947	0.677941	0.56947	0.6779406	0.569470102	0.569470102	19.8862575	177.1685
56	0.301186	0.044292	0.664382	0.558081	0.664382	0.558081	0.6643818	0.5580807	0.5580807	19.4885324	173.6251
57	0.295163	0.043406	0.651094	0.546919	0.651094	0.546919	0.6510942	0.546919086	0.546919086	19.0987617	170.1526
58	0.289259	0.042538	0.638072	0.535981	0.638072	0.535981	0.6380723	0.535980704	0.535980704	18.7167865	166.7496
59	0.34	0.05	0.75	0.63	0.75	0.63	0.75	0.63	0.86	26.67	109
60	0.3332	0.049	0.735	0.6174	0.735	0.6174	0.735	0.6174	0.8428	26.1366	106.82
61	0.326536	0.04802	0.7203	0.605052	0.7203	0.605052	0.7203	0.605052	0.825944	25.613868	104.6836
62	0.320005	0.04706	0.705894	0.592951	0.705894	0.592951	0.705894	0.59295096	0.80942512	25.1015906	102.5899
63	0.313605	0.046118	0.691776	0.581092	0.691776	0.581092	0.6917761	0.581091941	0.793236618	24.5995588	100.5381
64	0.307333	0.045196	0.677941	0.56947	0.677941	0.56947	0.6779406	0.569470102	0.777371885	24.1075677	98.52737
65	0.301186	0.044292	0.664382	0.558081	0.664382	0.558081	0.6643818	0.5580807	0.761824448	23.6254163	96.55682
66	0.295163	0.043406	0.651094	0.546919	0.651094	0.546919	0.6510942	0.546919086	0.746587959	23.152908	94.62568
67	0.34	0.05	0.5	0.34	0.75	0.34	0.75	0.15	0.34	33	688
68	0.3332	0.049	0.49	0.3332	0.735	0.3332	0.735	0.147	0.3332	32.34	674.24
69	0.326536	0.04802	0.4802	0.326536	0.7203	0.326536	0.7203	0.14406	0.326536	31.6932	660.7552
70	0.320005	0.04706	0.470596	0.320005	0.705894	0.320005	0.705894	0.1411788	0.32000528	31.059336	647.5401
71	0.313605	0.046118	0.461184	0.313605	0.691776	0.313605	0.6917761	0.138355224	0.313605174	30.4381493	634.5893
72	0.307333	0.045196	0.45196	0.307333	0.677941	0.307333	0.6779406	0.13558812	0.307333071	29.8293863	621.8975
73	0.301186	0.044292	0.442921	0.301186	0.664382	0.301186	0.6643818	0.132876357	0.301186409	29.2327986	609.4596
74	0.295163	0.043406	0.434063	0.295163	0.651094	0.295163	0.6510942	0.13021883	0.295162681	28.6481426	597.2704
75	0.15	0.34	0.5	0.63	0.75	0.34	0.25	0.34	0.63	44	184
76	0.147	0.3332	0.49	0.6174	0.735	0.3332	0.245	0.3332	0.6174	43.12	180.32
77	0.14406	0.326536	0.4802	0.605052	0.7203	0.326536	0.2401	0.326536	0.605052	42.2576	176.7136
78	0.141179	0.320005	0.470596	0.592951	0.705894	0.320005	0.235298	0.32000528	0.59295096	41.412448	173.1793
79	0.138355	0.313605	0.461184	0.581092	0.691776	0.313605	0.230592	0.313605174	0.581091941	40.584199	169.7157
80	0.135588	0.307333	0.45196	0.56947	0.677941	0.307333	0.2259802	0.307333071	0.569470102	39.7725151	166.3214
81	0.132876	0.301186	0.442921	0.558081	0.664382	0.301186	0.2214606	0.301186409	0.5580807	38.9770648	162.995
82	0.130219	0.295163	0.434063	0.546919	0.651094	0.295163	0.2170314	0.295162681	0.546919086	38.1975235	159.7351
83	0.15	0.63	0.5	0.63	0.75	0.63	0.75	0.34	0.63	49.1	129
84	0.147	0.6174	0.49	0.6174	0.735	0.6174	0.735	0.3332	0.6174	48.118	126.42
85	0.14406	0.605052	0.4802	0.605052	0.7203	0.605052	0.7203	0.326536	0.605052	47.15564	123.8916
86	0.141179	0.592951	0.470596	0.592951	0.705894	0.592951	0.705894	0.32000528	0.59295096	46.2125272	121.4138
87	0.138355	0.581092	0.461184	0.581092	0.691776	0.581092	0.6917761	0.313605174	0.581091941	45.2882767	118.9855
88	0.135588	0.56947	0.45196	0.56947	0.677941	0.56947	0.6779406	0.307333071	0.569470102	44.3825111	116.6058
89	0.132876	0.558081	0.442921	0.558081	0.664382	0.558081	0.6643818	0.301186409	0.5580807	43.4948609	114.2737
90	0.130219	0.546919	0.434063	0.546919	0.651094	0.546919	0.6510942	0.295162681	0.546919086	42.6249637	111.9882
91	0.127614	0.535981	0.425382	0.535981	0.638072	0.535981	0.6380723	0.289259428	0.535980704	41.7724644	109.7484
92	0.34	0.63	0.75	0.63	0.75	0.63	0.75	0.34	0.63	87	476
93	0.323	0.5985	0.7125	0.5985	0.7125	0.5985	0.7125	0.323	0.5985	82.65	452.2
94	0.05	0.86	0.5	0.05	0.75	0.05	0.25	0.05	0.63	50	928
95	0.049	0.8428	0.49	0.049	0.735	0.049	0.245	0.049	0.6174	49	909.44
96	0.04802	0.825944	0.4802	0.04802	0.7203	0.04802	0.2401	0.04802	0.605052	48.02	891.2512
97	0.04706	0.809425	0.470596	0.04706	0.705894	0.04706	0.235298	0.0470596	0.59295096	47.0596	873.4262
98	0.046118	0.793237	0.461184	0.046118	0.691776	0.046118	0.230592	0.046118408	0.581091941	46.118408	855.9577
99	0.045196	0.777372	0.45196	0.045196	0.677941	0.045196	0.2259802	0.04519604	0.569470102	45.1960398	838.8385
100	0.044292	0.761824	0.442921	0.044292	0.664382	0.044292	0.2214606	0.044292119	0.5580807	44.292119	822.0617

References

[1] R. Kaur, E.S. Sharma, Various techniques to detect and predict faults in software system: survey, Int. J. Fut. Revolut. Comput. Sci. Commun. Eng. (IJFRSCE) 4 (2) (2018) 330–336.

[2] A.A. Porter, R.W. Selby, Empirically guided software development using metric-based classification trees, IEEE Software 7 (2) (1990) 46–54.

[3] S. Chidamber, C. Kemerer, A metrics suite for object-oriented design, IEEE Trans. Software Eng. 20 (6) (1994) 476–493.

[4] S.S. Rathore, S. Kumar, A study on software fault prediction techniques, Artif. Intell. Rev. 51 (2019) 255–327, https://doi.org/10.1007/s10462-017-9563-5.

[5] S. Chatterjee, B. Maji, A bayesian belief network based model for predicting software faults in early phase of software development process, Appl. Intell. 48 (2018) 2214–2228, https://doi.org/10.1007/s10489-017-1078-x.

[6] P. Sharma, A.L. Sangal, Building and testing a fuzzy linguistic assessment framework for defect prediction in ASD environment using process-based software metrics, Arabian J Sci. Eng 45 (2020) 10327–10351, https://doi.org/10.1007/s13369-020-04701-5.

[7] A.K. Pandey, N.K. Goyal, Multistage model for residual fault prediction, in: Early Software Reliability Prediction, Springer. 2013, pp. 59–80.

[8] A. Arshad, S. Riaz, L. Jiao, A. Murthy, Semi-supervised deep fuzzy c-mean clustering for software fault prediction, IEEE Access 6 (2018) 25675–25685, https://doi.org/10.1109/ACCESS.2018.2835304.

[9] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, J. Too, Boosted whale optimization algorithm with natural selection operators for software fault prediction, IEEE Access 9 (2021) 14239–14258, https://doi.org/10.1109/ACCESS.2021.3052149.

[10] K. Juneja, A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation, Appl. Soft Comput. Elsevier 77 (2019) 696–713, https://doi.org/10.1016/j.asoc.2019.02.008.

[11] I. Tumar, Y. Hassouneh, H. Turabieh, T. Thaher, Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction, IEEE Access 8 (2020) 8041–8055, https://doi.org/10.1109/ACCESS.2020.2964321.

[12] H. Turabieh, Iterated feature selection algorithms with layered recurrent neural network for software fault prediction, Exp. Syst. Appl. Elsevier 122 (2019) 27–42, https://doi.org/10.1016/j.eswa.2018.12.033.

[13] S. Bilgaiyan, S. Mishra, M. Das, Effort estimation in agile software development using experimental validation of neural network models, Int. J. Inf. Technol. 11 (2019) 569–573, https://doi.org/10.1007/s41870-018-0131-2.

[14] Geng Wang, Cognitive deep neural network prediction methods for software fault tendency module based on bound particles swarm optimization, Cognit. Syst. Res. 52 (2018) 12–20, https://doi.org/10.1016/j.cogsys.2018.06.001.

[15] H.B. Yadav, D.K. Yadav, A fuzzy logic based approach for phase wise software defects prediction using software metrics, Inf. Software Technol. 63 (2015) 44–57.

[16] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software defect prediction with a simplified metric set, Inf. Software Technol. 59 (2015) 170–190.

[17] A.T. Raslan, N.R. Darwish, H.A. Hefny, Towards a fuzzy based framework for effort estimation in agile software development, Int. J. comput. Knowl. Sec. 13 (1) (2015) 37.

[18] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic review of fault prediction performance in software engineering, IEEE Trans. Software Eng. 38 (6) (2012) 1276–1304.

[19] P.S. Bishnu, V. Bhattacherjee, Software fault prediction using quadtree-based k-means clustering algorithm, IEEE Trans. Knowl. Data Eng. 24 (2012) 1146–1150.

[20] C. Jin, S.W. Jin, J.M. Ye, Artificial neural network-based metric selection for software fault-prone prediction model, IET Softw. 6 (6) (2012) 479–487.

[21] L. Briand, V. Basili, C. Hetmanski, Developing interpretable models with optimized set reduction for identifying high risk software components, IEEE Trans. Software Eng. 19 (11) (1993) 1028–1044.

[22] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert. Syst. Appl. 36 (2009) 7346–7354. http://10.1016/j.eswa.2008.10.027.

[23] R.C.G. Dhanajayan, S.A. Pillai, SLMBC: spiral life cycle model-based bayesian classification technique for efficient software fault prediction and classification, Soft Comput. 21 (2017) 403–415, https://doi.org/10.1007/s00500-016-2316-6.

[24] D. Ghosh, S.C. Satapathy, A.K. Jena, S. Bilgaiyan, J. Singh, A novel approach of software fault prediction Using deep learning technique, in: Automated Software Engineering: A Deep Learning-Based Approach. Learning and Analytics in Intelligent Systems vol. 8, Springer. 2020, pp. 73–91.

[25] N. Kalaivani, R. Beena, Overview of software defect Prediction using machine learning algorithms, Int. J. Pure Appl. Math. 118 (20) (2018) 3863–3873.

[26] G. Abaei, A. Selamat, J. Al Dallal, A fuzzy logic expert system to predict module fault proneness using unlabeled data, J. King Saud Univ. Comp. Info. Sci. (2018), https://doi.org/10.1016/j. jksuci.2018.08.003.

[27] R. Ozakõncõ, A. Tarhan, Early software defect prediction: a systematic map and review, J. Syst. Software 144 (2018) 216–239, https://doi.org/10.1016/j.jss.2018.06.025.

[28] G. Kaur, J. Pruthi, A study of agile-based approaches to improve software quality, Int. J. Comp. Syst. Eng. 16 (5) (2022) 158–163. https://publications.waset.org/computer-and-systems-engineering.

[29] E. Arisholm, L.C. Briand, E.B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, J. Syst. Software 83 (1) (2010) 2–17.

[30] PROMISE Repository, 2004. http://promise.site.uottawa.ca/SERepository/datasets-page.html.

[31] S. Wang, X. Yao, Using class imbalance learning for software defect prediction, IEEE Trans. Reliab. 62 (2) (2013) 434–443.

[32] Min Xu, P. Watanachaturaporn, P.K. Varshney, M.K. Arora, Decision tree regression for soft classification of remote sensing data, Rem. Sens. Environ. (3) (2005) 322–336.

[33] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, Energy Aware Consolidation Algorithm Based on K-Nearest Neighbor Regression for Cloud Data Centers. 6th International Conference on Utility and Cloud Computing, IEEE. 2013, pp. 256–259.

[34] R.K. Yin, Case Study Research: Design and Methods, sage 5, 2009.

[35] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering, Kluwer Academic Publishers. 2000.

[36] P. Runeson, M. Höst, Guidelines for Conducting and Reporting Case Study Research in Software Engineering 14, Empirical software engineering. 2009, pp. 131–164.

[37] Björn Regnell, P. Runeson, T. Thomas, Are the Perspectives Really Different?–Further Experimentation on Scenario-Based Reading of Requirements 5, Empirical Software Engineering. 2000, pp. 331–356.