Journal of Theoretical and Applied Information Technology
15th June 2022. Vol.100. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

CROSS-VERSION SOFTWARE FAULT DETECTION MODEL WITH AUTOMATIC DATA SELECTION

*[1]KIRAN JAMMALAMADAKA, [2]NIKHAT PARVEEN

*[1]Research Scholar, Department of Computer Science and Engineering,
KoneruLakshmaiah Education Foundation, Vaddeswaram, Guntur, A.P.
[2]Associate Professor, Department of Computer Science and Engineering,
KoneruLakshmaiah Education Foundation, Vaddeswaram, Guntur, A.P.
*Email:[1]jvskkiran@yahoo.com, [2]nikhat0891@gmail.com

ABSTRACT

Fault detection in software engineering is gaining attention due to the severe impacts of the faults in the software and malfunctioning of software used in diverse domains. Among the various faults, the class overlapping and the data distribution difference are one of the major issues that are required to be properly addressed. To deal with these issues, we propose a concept on cross-version software fault detection with data selection (CV-FDDS). This scenario is much reasonable and significant to detect the above-mentioned issues by predicting the faults in the new versions based on the labelled data of previous versions. The proposed framework works on identifying the faults in the newer versions of the software projects on the basis ofolder version labels through the use of learning strategies. Initially, the dataset is pre-processed using an appropriate data filter. The older and newer files are treated in two different ways to accomplish this task. To deal with the existing or the older files, an adaptive spiking atom search recurrent neural network classifier (ASARNN) has been proposed through which the training and testing data are automatically selected. This is done by assigning higher weight values to the much relevant and noise-free older versions. Based on the training and testing, the newer versions comprising faults are detected using the proposed kernel optimized extreme learning machine (KOELM). The implementation of the developed approach is implemented on PYTHON. The implemented approach performance is compared against the conventional methods with respect to the major performance indicators. The implemented nodel achievesa geometric-mean (G-mean) of about 0.7815, f-measure of about 0.775 and balance is about 0.8055 respectively.

Keywords: *Fault Detection, Software Engineering, Data Distribution Difference, Class Overlapping, Data Selection, Neural Network.*

1. INTRODUCTION

The software processing and performing several operation on it is known as software engineering. There are many detection techniques in the software engineering fields like cost prediction and security. Software systems are highly essential in the modern society and for human activities as they occupy most of the safety-critical domains. Due to the growth and importance of the software systems, it is important to ensure efficiency and reliability for the software [1]. There are a vast count of testing and debugging strategies available to identify the faults in the software systems. In this domain, software defect prediction (SDP) is an area of research intending the researchersto localize the vulnerable or the fault-prone modules efficiently [2]. The concept inspired by the SDP strategies is that it uses the faults identified in the older versions for predicting the faults in the newer versions of the projects. The major component used in SDP is the project data collected from the older versions. The defects in the software

projects are normally relevant for the same projects and hence the versions of the same projects are much useful to detect the faults of newer versions of the same projects [3, 4].

Cross-Version Defect-Prediction (CVDP) works on the principle of detecting the software faults based on the prior released project data. The pre-released versions of any projects are used to label the current versions belonging to the same project [5]. The historic data of the software system provide much useful and relevant data required to trace the faults in the system. Through this, the vulnerability of the new software can be avoided and the developed systems will be error-free with high reliability [6]. Constructing the defect prediction models based on mining of software history database, the constructed models will be highly efficient without any prior faults [7]. This scenario involves three major metrics such as the evolution metrics, code metricsand process metrics. The code metrics illustrate the static characteristicslike cohesion, code complexity, inheritance,coupling and code scale. The other two metrics describe the dynamic properties such as the differences and evolution patterns [8-10].

Accordingthe training and testing datasets used, the fault prediction approaches can be categorized into within-project and cross-project defect prediction (CPDP) models. The CPDP models are highly significant as the training data comprises information gathered from diverse software projects to predict the faults from a particular project contrary to the within-project model for which both the training and testing datasets are gathered from the similar project [11, 12]. Though the CPDP models are efficient in overcoming the data insufficiency problem, the performance in prediction for most of these models are degraded due to the data distribution difference issue [13]. Since the projects consist of multiple prior versions, the distribution difference arises among multiple versions and hence the labelling of versions become critical. Moreover, labelling the modules of the newer projects manually are literally time-consuming and error-prone [14]. Therefore, the automatic selection of the training data from the older versions are helpful in achieving the desired objectives. The CPDP models work on using these training data for labelling the newer versions of the projects automatically [15].

In most of the prediction approaches, the learning strategies are used as those strategies are capable of retaining a large amount of information and those models are able to be trained with the prior versions more efficiently than the other techniques [16]. Almost all the existing techniques relevant to CPDP, as these strategies are able to automate the prediction process more reliably and are able to label the versions of the software projects more accurately [17].The current research work has been formulated with the aid of overcoming the issues while classifying the defects of the versions of the software projects.

Motivation

Software faults are intolerable as it leads to various critical problems including financial loss or even accidents due to failures in the safety equipment. One of the most reasonable approach is the cross-version fault detection of the software based on the labelling details of the previous versions. Due to the existence of numerous previous versions, the scenario faces two major issues such as the data distribution difference and class overlapping resulting from the existence of redundant files in more than one version. These issues lead to wrong detections of the software programs. Though there are methods available for the cross-version fault detection, almost all the methods considered only the prior version of the model. But any software project can have multiple versions and hence data distribution difference and class overlapping issues will probably occur leading to critical problems. These issues are rarely addressed in any of the available techniques. Due to the lag in this research field and to improve the scope while providing a keen knowledge about these issues, this article has been proposed.

Objectives

- Fault detection in software engineering is regarded as a supervised binary classicationissue. The defect detection techniques can be classified on the basis of trainingand test data. Here cross-version software fault detection is used.
- Validating the existence of two critical issues in the CV-FDDS scenario such as the data distribution difference and class overlapping arising due to redundant files in more than one version, to identify and understand the impacts in fault detection techniques.
- Proposing a novel adaptive spiking atom search recurrent neural network (ASARNN) for the older versions of files to generate the testing and training data for the classifier to detect the faults in the newer versions of files.
- Proposing a novel Kernel Optimized Extreme Learning Machine (KOELM) for identifying the faults in the newer versions of the files on the basis oflabelling data generated by the pre-mentioned classifier.
- Evaluating the performance of the implemented fault detection scheme against the existing approaches of the major metrics like geometric-mean (G-mean), balance and f-measure.

2. RELATED WORK

Some of the recently introduced techniques in software fault detection using different learning strategies are listed below:

Jie Zhang *et al.* [18] introduced a fault prediction strategy to predict the faults in the present version of a software project based on the faults identified in the previous versions of the project. The authors tried to address the software problems that are seldom reported in the previous studies related to faults predictions and the model called cross-version model with data selection (CDS). To deal with the old files, the authors introduced the clustering-based multi-version classifier (CMVC) through which the training data were automatically selected from the most relevant and noise-free versions and higher weight values were assigned to them. For the new files, a Weighted Sampling Model (WSM) was introduced through the incorporation of the outputs of CMVC. Upon simulations, the model proved to outperform the other existing baseline and state-of-the-art approaches.

Zhidan Yuan *et al.* [19] introduced a technique called ALTRA for CPDP. To overcome the problem of data distribution difference due to a large number of existing versions of the projects, the authors introduced ALTRA integrating active learning and TrAdaBoost. Initially, the Burak filter was used to filter out the similar labeled modules based on the unlabeled modules. Then active learning was used to label the unlabeled representative modules of the target project where the modules were labeled into defective and non-defective by the experts. Then the weights are determined for the labeled modules with the help of the TrAdaBoost in both the source and target projects. Finally, the model was constructed by weighted support vector machine (SVM). Under experimentations, the approach proved to outperform the other models on the basis ofthe certain measures.

Mengmeng Zhu and Hoang Pham [20] presented a model for software fault detection and to improve the system reliability based on multiple environmental factors. The authors presented a generalized multiple-environmental-factors reliability growth model under the martingale model. The model randomness was analyzed while detecting the faults in the software projects. For illustration, two environmental factors were considered for developing the model. Because of the randomness, a stochastic fault detection model was developed to identify the faults occurring in a software system. The simulation results demonstrated the performance of the scheme in detecting the faults occurring in a system in terms of software failure and reliability.

Hiba Alsghaier and Mohammed Akour [21] suggested a software fault detection model hybridizing genetics algorithm (GA) with SVM and particle swarm optimization (PSO) algorithm. The model was constructed on different phases each with the integration of an algorithm with the SVM classifier for predictions and to identify the best model for prediction. The introduced integration approaches proved to provide better results and were sufficient enough for all kind of datasets. The search models worked on finding the faults in the software projects through traversal and helped the classifier to predict the faults in the project. The simulation results demonstrated the efficiency and effectiveness of the suggested approach over the other existing approaches.

Swapnil Shukla *et al.* [22] formulated the CVDP model for maximizing the recall with the following objectives: (a) minimizing the misclassification cost and (b) minimizing the Quality Assurance (QA) cost over the vulnerable files. Initially, the pre-processing was carried out using data standardization and then the training and testing processes were sequentially carried out to identify the faults in the projects. The objectives for optimization were solved using the GA and the faults were predicted using the logistic regression technique. Finally the simulation results suggested that the technique was capable of locating the faults more accurately than the other available techniques in fault identification in software projects.

Zhou Xu *et al.* [23] introduced a model for the cross-version fault prediction to overcome the issue of data distribution differences. The authors designed a Two-Stage Training Subset Selection (TSTSS) to identify the defects of the newer versions of the projects. In the initial stage, the model used a selection method called as the sparse modelling representative to select the suitable prior version for reconstructing the data. Then the model introduced a dissimilarity-based sparse subset selection modelfor refining the best modules from the selected modules that were capable of labelling the current version. Finally, the cross-version defect prediction model was constructed using the weighted extreme learning machine classifier. The experimental results demonstrated that the technique improved the performance compared to 11 existing baseline models. Table 1 delineates the Features and Challenges of the existing approaches

.Table 1: Features and Challenges of the existing approaches

Authors and citation	Methods	Features	Challenges
Jie Zhang *et al.* [18]	CMVC and WSM	Achieves better F-score and G-mean value	When the test set size is small, active learning property may cause some degree of unfair results
Zhidan Yuan *et al.* [19]	ALTRA	Achieves better precision, F-score and Recall	This method is suitable only for small number of module, does not support when the module is large
Mengmeng Zhu and Hoang Pham [20]	GA with SVM	Achieves high accuracy, F-score and less error rate	This method suffer from slow convergence and trapped by local optima
Swapnil Shukla *et al.*[22]	GA	Faults are successfully predicted Minimize the cost of QA and misclassification	Sometimes it achieves less recall and misclassification cost
Zhou Xu *et al.* [23]	TSTSS	This model successfully reconstructed prior and current version data	It is a lengthy and time consumingprocess It leads to over fitting problem

Journal of Theoretical and Applied Information Technology
15th June 2022. Vol.100. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. PROPOSED METHODOLOGY

Initially, the dataset is pre-processed using a missing value imputation which is used to secure the loss of data and normalization process is carried out.Let the prior version labeled dataset L is $Z^1, Z^2, ...Z^n$ and the present version dataset Z. The version j^{th} has instances about m_i, $Z^i = \{z^i_1 z^i_2, ...z^i_{mi}\}$ has a label of $g^i = \{g^i_1 g^i_2, ...g^i_{mi}\}$ where g^i is 1 when the respective document is defective and -1 when the file is not defective. The aim of ASARNN is for detecting the label of the present version $g = \{g_1, g_2, ...g_m\}$.

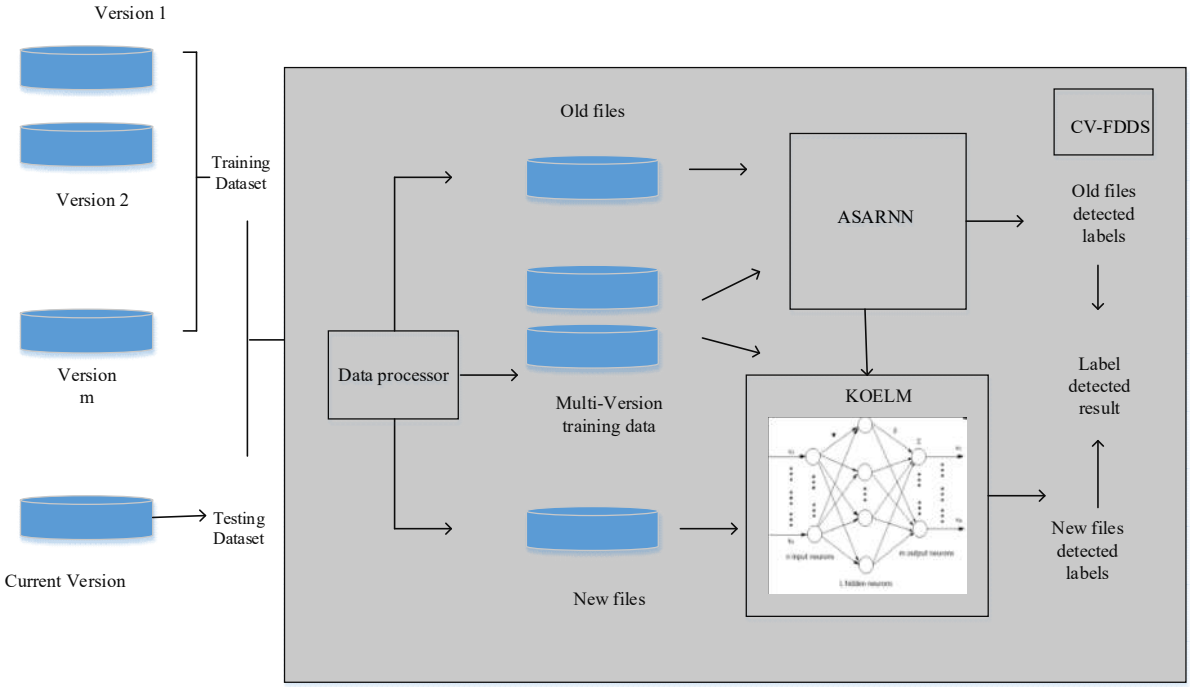


Figure 1: Framework of the developed CV-FDDS

Figure 1 delineats the framework of the developed CV-FDDS. The architecture comprises of three major parts like data processor, ASARNN and KOELM. The data processor divides the Z into new and existing file as Z^{New} and Z^{exist}, where Z^{New} is the files that are not exist in the prior version and Z^{exist} is the files exist in the prior version. In addition the data processor expands the vectors of label $g^1, g^2, ...g^n$ to the similar size by adding zero and computes the matrix of the indicator of the matrix P^j for each prior version that is a matrix of diagonal with $P^j_{kk} = 1$ when k exist in both the j^{th} and the present version, otherwise $I^j_{kk} = 0$.

When detecting the label Z^{New} and Z^{exist} are treated differently. This is because the statistical or machine learning techniques are on the basis of the hypothesis that the dataset instance must be sampled independently from same distribution. This hypothesis holds true for new files. Z^{exist} file is based on the similar name document in the training set. Here ASANN is implemented to find

the solution for version weighting and to detect the Z^{exist} label by solving the optimization issue. In addition by solving the version weight,KOELM is developed to detect the Z^{New} label by classification method trained on the sample of data from prior versions based on theresult of ASARNN weights.

The main objectives in detecting the existing files are given below:

(a) Files with the same features must belabelled similarly, which is a first step of thedeep learning models.

(b)The documents in the presentpatternwillreceive defect patterns from priorpattern. When detecting the label of document in the present version, the labels of similar namedocumentin the prior versions must be taken.

(c)Prior versions with the relevance of higher data must assign more weights.

(d) Prior versions with low data noise are assigned to havemore weights. Sometimes the defect detection datasets consist of mislabelling noises. Hencehigher weights must be given to the versions with low noise.

3.1 Adaptive Spiking Atom Search Recurrent Neural Network Classifier (ASARNN)

In this work, a hybrid model of deep learning with optimization technique is employed to obtain old version file detected label.Spiking neurons emerge from methods that express the characteristics of real biological neurons. A SRNN [24] has 2 recurrent layers; neurons inside the layer are linked fully recurrently among the layers neurons are fully connected with forwarding connections.

Let us consider the RNN with continuous variable and spiking RNN. The continuous variable with N rate is given as

$$\tau_i^d \frac{dy_i}{dt} = -xi + \sum_{j=1}^{N} w_{ij}^{rate} r_j^{rate} + J_{ext} \quad (1)$$

$$r_j^{rate} = \phi(y_i) \quad (2)$$

where τ_i^d is the synaptic decay time, w_{ij}^{rate} is the synaptic strength from the unit j to i. J_{ext} is the input of the external current input to the unit i. r_j^{rate} is the filtering rate and $\phi(y_i)$ is a transfer function. The firing rate in spiking network should not be negative;therefore the activation fuction is a standard sigmoid function.

The 2nd RNN with N spiking unit is given as

$$\tau_n \frac{dv_i}{dt} = -v_i + \sum_{j=1}^{N} w_{ij}^{spk} r_j^{spk} + J_{ext} \quad (3)$$

Where τ_n is the constant time of membrane, v_i is the i^{th} unit voltage membrane, w_{ij}^{spk} is the synaptic strength from the unit j to i and r_j^{spk} is the synaptic filtering rate of unit j. The back propoagation through time (BPTT) is used for training the RNN for creating the target signal. The units in RNN are linked sparely by W^{rate} and input signal is received by the weights W_{in} derive from a normal distribution with zero mean and unit variance. The output of the network output is

$$output^{rate}(t) = W_o^{rate} . r^{rate}(t) \quad (4)$$

Where W_o^{rate} is the weights of the readout, $r^{rate}(t)$ is the rate of firing at the time t in the network. The weight matrix of recurrent W^{rate}, W_o^{rate} and τ_i^d are optimized on the process of the training and the matrix of the input weight W_{in} is constant.

A new metaheuristic approachAtom Search Optimization (ASO) [25] is influenced the atom. In ASO, everyposition of atom within the search space indicates a solution computeded by its mass.The atoms will attract or repel each other on the basis ofdistance among them, motivating the lesser atoms to move to heavy atoms. Lighter atoms have higher acceleration which identify a newencouraging region in the whole search space and heavy atom have lessacceleration which ensures better solutions in local spaces. Atom search start by N atoms with dimension D. Let the mass (M) for every atom is calculated by

$$M(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_i(t)} \quad (5)$$

$$m_i(t) = e^{-\frac{fit_i(t) - fit_b(t)}{fit_w(t) - fit_b(t)}} \quad (6)$$

Where $fit_w(t)$ and $fit_b(t)$ are the best and the worst fitness factor at the t iteration. $fit_w(t)$ and $fit_b(t)$ are given below.

$$fit_w(t) = \max_{i \in (1,2....,N} fit_i(t) \quad (7)$$

$$fit_b(t) = \min_{i \in (1,2....,N} fit_i(t) \quad (8)$$

Where $T_{\max}$ is the overall iterations. Then the step is to calculate the constraint force CF and interaction force IF.

$$IF^d(t) = \sum_{j \in zbest} \eta_j \times f_{ij}^{\ d}(t) \quad (9)$$

Where $\eta_j \in [0,1]$ is a random number, $zbest$ is a atoms subset which has the best k atoms. $f_{ij}^{\ d}(t)$ is the force of interaction given on atom i from atom j at t iteration. Constraint force can be written as

$$CF^d(t) = \alpha(t)(Y_b^d(t) - Y_i^d(t)) \quad (10)$$

$$\alpha(t) = \lambda e^{-20\frac{T}{T_{\max}}} \quad (11)$$

Where $Y_b^d(t)$ is best atom position, $\alpha(t)$ is a lagrangian multiplier, λ is the multiplier weight. Then for each atom, the acceleration is calculated by the below expression

$$A_i^d(t) = \frac{IF^d(t)}{m_i^d(t)} + \frac{CF^d(t)}{m_i^d(t)} \quad (12)$$

$$A_i^d(t) = -\gamma\left(1 - \frac{T-1}{T_{\max}}\right)^3 e^{-20\frac{T}{T_{\max}}} \times \sum_{j \in zbest} \frac{r_i[2 \times (h_{ij}(t))^{13} - (h_{ij}}{m_i(t)}$$

$$\frac{\lambda e^{-20\frac{T}{T_{\max}}} (Y_b^d(t) - Y_i^d(t)}{m_i(t)} \quad (13)$$

Where r_i is the random number. Then the velocity and position updationat $(t+1)$ is given by the following expression

$$Y_i^d(t+1) = Y_i^d(t) + V_i^d(t+1) \quad (14)$$

$$V_i^d(t+1) = r_i V_i^d(t) + A_i^d(t) \quad (15)$$

In atom search, for improving the exploration in the 1st phase of iteration, every atom requires to interact with more atoms with best fitness value at the neighbour k. To improve exploitation in the last phase of iteration, every atom requiresfor interacting with less atoms with best value of fitness. The neighbour $K(T)$ is computed by

$$K(T) = N - (N-2)\sqrt{\frac{t}{t_{\max}}} \quad (16)$$

Finally, the existing files are detected by ASARNN. This deep learning model ensures an optimal solution for detecting the file in the prior version. To detect the file Y^{New} is the files that are not exist in the prior version.

3.2 Kernel Optimized Extreme Learning Machine (KOELM)

KOELM [26]test the data from every prior version on the basis of the respective weight. The versions with less noise and relevance of higher data are considered with more weights, higher instances sampled from these versions. Likewise, the training sets are assigned. KOELM has fast learning speed and has a better generalization. Because there is no need to tune the initial parameters of the hidden layer.The hidden layer Feed Forward Network (FFN) is converted into the linear equation by minimum norm least squares. The aim of the KOELM is to reduce the output norm weight and training error at the same time. For the samples $\{(Z_i, T_i) \mid X_i \in S^m, T_i \in S^n, i = 1,2,...N\}$, the P neurons hidden layer with the output function is:

$$f_P(Z) = \sum_{i=1}^{P} \beta_i r(Z) = r(Z)\beta \quad (17)$$

Where $\beta = [\beta_1, \beta_2 .....\beta_P]$ is the output weight vector between output neuron and P. The hidden layer output vector to the input X is given by

$$r(Z) = [r_1(Z), r_2(Z) .....r_P(Z)] \quad (18)$$

For enhancing the generalization and to reduce the training error of neural networks, at the same time both output weight and the training error must be minimized.

$$\min : \| h\beta - T \|, \| \beta \| \quad (19)$$

According to (Karush–Kuhn–Tucker) the equation (19) can be written as

$$\beta = h^T \left(\frac{1}{R} + hh^T \right)^{-1} T \quad (20)$$

Where h is the output matrix of the hidden layer, R is the coefficient of the reflection and T is the expected samples and the ELM algorithm output function is

$$f(Z) = g(Z)h^T \left(\frac{1}{R} + hh^T \right)^{-1} T \quad (21)$$

When the feature mapping function $r(Z)$ is unknown, ELM kernel matrix on the basis of Mercer's condition is given by

$$M = hh^{T,} : m_{ij} = r(Z_i)r(Z_j) = L(Z_i, Z_j) \quad (22)$$

The output function $g(Z)$ on the basis ofKOELM is given by

$$f(Z) = L(Z, Z_1) .....L(Z, Z_n)] \left(\frac{1}{R} + M \right)^{-1} T \quad (23)$$

Where $L(Z, Z_1)$ and $M = hh^T$ are the hidden neurons kernel function of single hidden layer FFN networks. The functions like polynomial kernel, exponential kernel, linear kernel and Gaussian kernel will satisfy the Mercer condition. Here three typical kernel functions are selected for the kernel functions are given below.

(i) Gaussian kernel:

$$L(Z, Y) = e(-b \| Z - Y \|) \quad (24)$$

(ii) Hyperbolic tangent kernel:

$$L(Z, Y) = \tanh(aZ^T Y \mid + C) \quad (25)$$

(iii) Wavelet kernel:

$$L(Z,Y)=\cos\left(f\frac{\|Z-Y\|}{d}\right)\exp\left(-\frac{\|Z-Y\|}{e}\right) \quad (26)$$

Where a,b,c,e and f are the parameters that are the majorpart in the performance of neural network and must be tuned on the basis of the solved problem. KOELM depends on the prior version with higher weights, which has low noise and applicable for the current version based on ASRNN.After detection by the KOELM, the detected model will be trained on the classification approaches. In this work for classificationNaive Bayes, Linear RegressionandRandom forestare used for training.

4. RESULT AND DISCUSSION

This section gives the performance analysis and discussion aboutdeveloped scheme. The entire implementations have been processed on a system with 8 GB RAM and Intel Core i5 CPU with 3.0 GHz speed. To implement the proposed scheme, PYTHON 3.8 is utilized. The developed approach performance is implemented with three metrics Accuracy, F-score, Balance, and G-mean. The performance eof the developed model is compared with Linear Regression [18], Naive Bayes [18] and Random forest [18], Hybrid Active Learning and Kernel PCA (HALKP) [30] and CDS [18].

4.1 Dataset Detail

The dataset used is MORPH and it is collected from the PROMISE repository. The reason for choosing this dataset is it was used in many existing approaches [23],[27], [28], and [29]and this dataset comprises of multiple previous version for each software project. The proposed model is implemented on 28 versions of eight software projects.

4.2 Performance Measures

The formulas for calculating thethree metrics F-score, Balance, and G-mean are given below.

The number of accurately classified data to the total number of data is called as accuracy and it is given in Eq. (27), where TP, TN, FP and FN are True Positive, True Negative, False Positive, False Negative.

$$Accuracy=\frac{TP+TN}{TP+FP+TN+FN} \quad (27)$$

F_1-score is the harmonic mean of the precision and recall and it is given in Eq. (28)

$$F-score=\frac{2\times precision\times \mathrm{Re}call}{precision+\mathrm{Re}call} \quad (28)$$

G-mean is the geometric mean of sensitivity and specificity and it is given in Eq. (29)

$$g-mean=\sqrt{\left(\frac{TN}{TN+FP}\right)\times\left(\frac{TP}{TP+FN}\right)} \quad (29)$$

$$Balance=1-\sqrt{\frac{(0-FPR)^2+(1-TPR)^2}{2}} \quad (30)$$

where FPR and TPR are the False Positive Rate, True Positive Rate

Table 2: projects and defective ratio of MORPH dataset

Projects	Version	File Instance	Defective (%)
camel	1.0	339	3.8
	1.2	608	35.5
	1.4	872	16.6
	1.6	965	19.5
	3.2	273	33.1
	4.0	306	24.5

jedit	4.1	312	25.3
	4.2	367	13.1
Log4j	1.0	135	25.2
	1.1	109	33.9
	1.2	205	92.2
poi	1.5	237	59.2
	2.0	314	11.8
	2.5	385	64.4
	3.0	442	63.6
synapse	1.0	157	10.2
	1.1	222	27.0
	1.2	256	33.6
velocity	1.4	196	75
	1.5	214	66.4
	1.6	229	34.1
xalan	2.4	723	15.2
	2.5	803	48.2
	2.6	885	46.4
xerces	1.1	162	47.5
	1.2	440	16.1
	1.3	453	15.2
	1.4	588	63.7

Table 1shows the projects and defective ratio of MORPH dataset. There are 8 open source project and the defective ratio differ for each version. For example in the software project, in the seventh project xalan in the version 2.4, only 15.2% files are defective then in version 2.5 the defective percentage is increases to 48.2% and again reduces to 46.4%. therefore it is proved that the variation pf data distribution among a version of similar projects does not exist.

Table 3: Detection performance of the proposed method

projects	Version pair		Accuracy	F-score	Balance	Gmean
	prior	current				
camel	1.0	1.6	0.961	0.945	0.896	0.890
	1.2	1.6	0.792	0.424	0.543	0.375
	1.4	1.6	0.934	0.859	0.879	0.8749
	1.0+1.2+1.4	1.6	0.889	0.749	0.837	0.833
jedit	3.2	4.2	0.672	0.410	0.710	0.708
	4.0	4.2	0.818	0.573	0.796	0.795
	4.1	4.2	0.818	0.573	0.796	0.795
	3.2+4.0+4.1	4.2	0.763	0.522	0.812	0.809
Log4j	1.0	1.2	0.426	0.822	0.698	0.629
	1.1	1.2	0.3934	0.810	0.6810	0.601
	1.0+1.1	1.2	0.442	0.828	0.706	0.643
poi	1.5	3.0	0.734	0.777	0.727	0.726
	2.0	3.0	0.386	0.701	0.539	0.2820
	2.5	3.0	0.6969	0.694	0.642	0.620
	1.5+2.0+2.5	3.0	0.7348	0.806	0.75	0.748
synapse	1.0	1.2	0.802	0.702	0.793	0.793
	1.1	1.2	0.868	0.798	0.853	0.852
	1.0+1.1	1.2	0.802	0.701	0.807	0.807
velocity	1.4	1.6	0.794	0.703	0.8115	0.810

	1.5	1.6	0.867	0.8102	0.851	0.850
	1.4+1.5	1.6	0.823	0.738	0.845	0.843
xalan	2.4	2.6	0.962	0.9535	0.963	0.963
	2.5	2.6	0.9547	0.959	0.953	0.9528
	2.4+2.5	2.6	0.950	0.953	0.9495	0.949
xerces	1.1	1.4	0.920	0.9495	0.925	0.925
	1.2	1.4	0.909	0.923	0.896	0.895
	1.3	1.4	0.931	0.939	0.918	0.918
	1.1+1.2+1.3	1.4	0.9090	0.923	0.896	0.8959

Table 2 illustrates the detection performance of the implemented model. The specific defect detection approach can differ a lot because of the variation of training set from different versions. Let us consider the project synapse, here the G-mean, balance and F-measure values are less in the case of version 1.0. but when using 1.1 version as a training set, detection performance is increased significantly for three metrics. The data distribution variation affect the performance. Hence, selecting the training data with low noise is suitable for building CV-FDDS model.

Table 4: The overall performance of CV-FDDS and existing models with respect to F-score

project	RF [18]	LR [18]	NB [18]	HALKP [30]	CDS [18]	proposed
camel	0.371	0.240	0.305	0.428	0.516	0.744
jedit	0.482	0.503	0.475	0.564	0.525	0.520
Log4j	0.505	0.409	0.452	0.612	0.718	0.820
poi	0.654	0.634	0.294	0.674	0.767	0.744
synapse	0.361	0.438	0.510	0.457	0.538	0.734
velocity	0.562	0.530	0.525	0.527	0.609	0.750
Xalan	0.427	0.284	0.427	0.610	0.710	0.955
xerces	0.233	0.069	0.224	0.401	0.538	0.933
Average	0.44	0.391	0.401	0.531	0.612	0.775

For every project, the training data is the data in the prior version, and test data is the present version. Table 3 shows the overall performance of CV-FDDS and existing models with respect to F-score. Here in all cases the developed method CV-FDDS achieves better results. The average of the developed method is 0.775 while the existing approaches like HALKP and CDS achieve less average of about 0.531 and 0.612.

Table 5: The overall performance of CV-FDDS and existing models with respect to G-mean

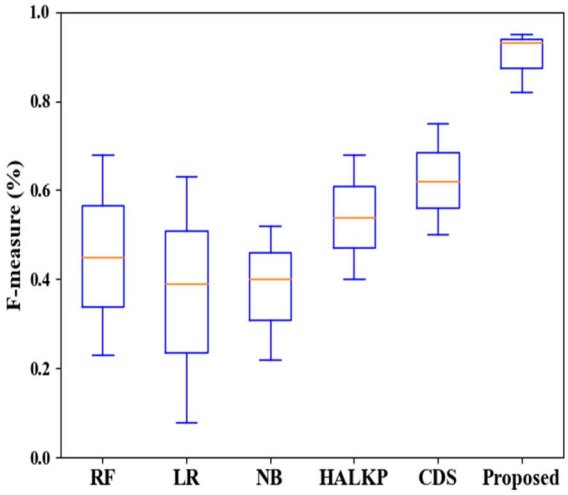
project	RF [18]	LR [18]	NB [18]	HALKP [30]	CDS [18]	proposed
camel	0.541	0.395	0.481	0.582	0.710	0.743
jedit	0.782	0.745	0.712	0.709	0.783	0.777
Log4j	0.526	0.492	0.508	0.621	0.636	0.624
poi	0.648	0.621	0.410	0.680	0.705	0.594
synapse	0.487	0.543	0.615	0.568	0.641	0.817
velocity	0.563	0.350	0.549	0.494	0.676	0.834
Xalan	0.523	0.407	0.523	0.635	0.691	0.955
xerces	0.352	0.190	0.351	0.467	0.508	0.908
Average	0.55275	0.467875	0.518625	0.579875	0.6687	0.7815

Table 4 shows the overall performance of CV-FDDS and existing models with respect to G-mean. Here in all cases the developed method CV-FDDS achieves better results. The average of the developed method is 0.7815 while the existing approaches like HALKP and CDS achieves less average of about 0.579 and 0.6687 respectively.

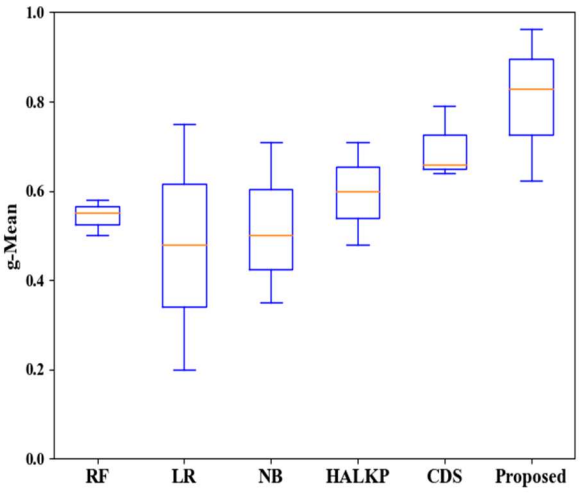
Table 6: The overall performance of CV-FDDS and existing models with respect to balance

project	RF [18]	LR [18]	NB [18]	HALKP [30]	CDS [18]	proposed
camel	0.519	0.407	0.471	0.551	0.702	0.789
jedit	0.782	0.730	0.690	0.698	0.699	0.778
Log4j	0.518	0.473	0.498	0.597	0.63	0.695
poi	0.641	0.617	0.417	0.666	0.703	0.664
synapse	0.476	0.512	0.591	0.543	0.638	0.818
velocity	0.540	0.381	0.537	0.484	0.682	0.836
Xalan	0.492	0.413	0.497	0.635	0.509	0.955
xerces	0.385	0.318	0.383	0.465	0.667	0.909
Average	0.542	0.318	0.381	0.462	0.507	0.8055

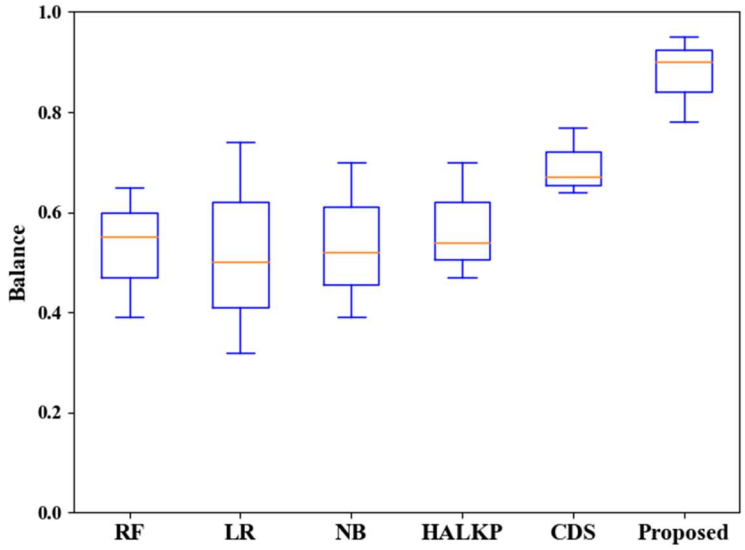
Table 5 shows the overall performance of CV-FDDS and existing models with respect to balance. Here in all cases, the developed method CV-FDDS achieves better results. The average of the developed method is 0.805. in the first project camel, the proposed method achieves the balance value of 0.789 and the existing model like HALKP and CDS achieves 0.771 and 0.702 respectively.



(a)



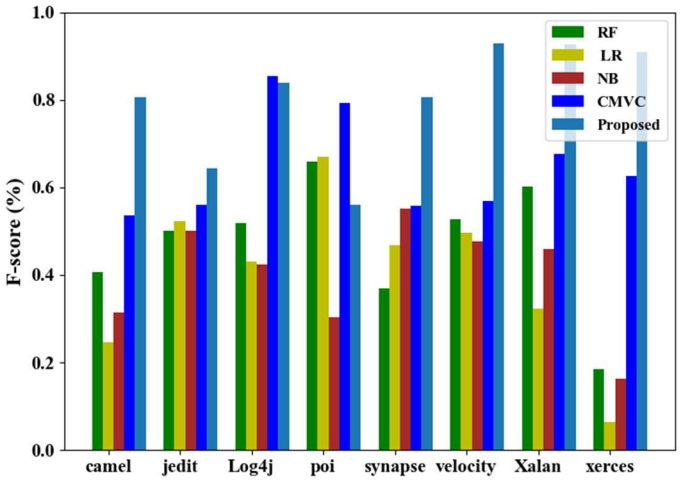
(b)



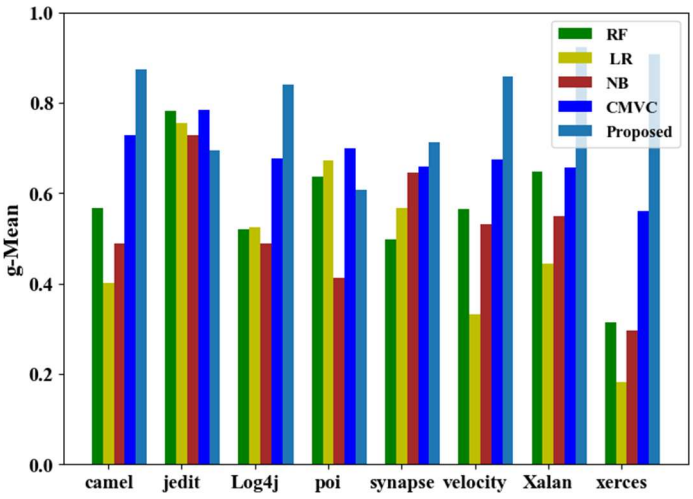
(c)

Figure 2: Box Plot Representation Of Various Models

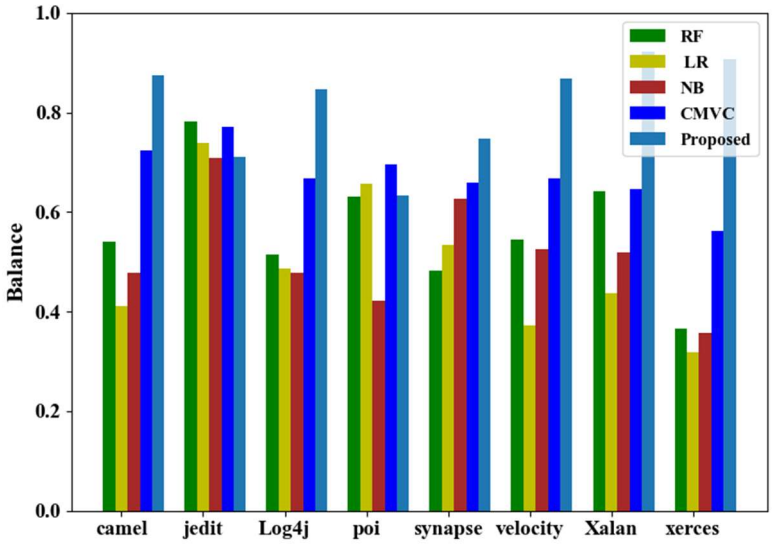
Figure 2represents the box plot representation of various models likeRF, LR, NB, HALKP and CDS. This graph depicts the higher performance of each model on the basis of balance , G-mean and F-score. In the case of F-score, in the project xerces the developed model achieves higher value of about 0.955. In the case of G-mean, in the project xerces the developed model achieves higher value of about 0.955. In the case of balance also, in the project xerces the developed model achieves higher value of about 0.955. Thus the proposed model proved its betterment.



(a)



(b)



(c)

Figure 3: Detection Performance Of Existing File

Figure 3 shows the detection performance of the methods like CMVC, RF, LR, NB and the proposed model on the basis of F-score, G-mean and balance. Here in figure 3(a), in the project camel, the proposed method achieves F-score of about 0.806, while the CMVC method achieves 0.536 only. Here in figure 3(b), in the project camel, the proposed method achieves G-mean of about 0.873, while the CMVC method achieves 0.729 only. Here in figure 3(c), in the project camel, the proposed method achieves balance of about 0.874, while the CMVC method achieves 0.724 only. Finally it is proved that the developed model achieves higher performance in existing files in all cases respectively.

Table 7: Detection performance of new file KOELM-RF

	F-score		G-mean		Balance	
project	WSM-RF [18]	KOELM-RF	WSM-RF [18]	KOELM-RF	WSM-RF [18]	KOELM-RF
camel	0.257	0.550	0.457	0.272	0.454	0.531
jedit	0.414	0.586	0.811	0.477	0.807	0.601
Log4j	0.515	0.640	0.553	0.0	0.533	0.482
poi	0.432	0.749	0.557	0.690	0.547	0.710
synapse	0.363	0.630	0.494	0.728	0.488	0.729
velocity	0.827	0.746	0.628	0.723	0.590	0.743
Xalan	0.941	0.656	0.941	0.647	0.938	0.655
xerces	0.422	0.820	0.495	0.759	0.481	0.772
Average	0.521375	0.672125	0.617125	0.537	0.605	0.652875

Table 6 shows the detection performance of new file KOELM-RF. After detecting the new file, this model is trained on the classifier RF. When comparing the performance of KOLEM-RF and the existing model WSM-RF, the proposed model achieves high F-score value of about 0.820, high G-mean of about 0.759 and high balance value of about 0.772 in the xerces software. In addition, the average value of the three metrics also high for the developed approach

Table 8: Detection of new file KOELM- LR

	F-score		G-mean		Balance	
project	WSM-LR [18]	KOELM-LR	WSM-LR [18]	KOELM-LR	WSM-LR [18]	KOELM-LR
camel	0.240	0.504	0.408	0.357	0.41	0.549
jedit	0.52	0.405	0.85	0.239	0.85	0.518
Log4j	0.503	0.655	0.546	0.453	0.52	0.5
poi	0.44	0.721	0.537	0.652	0.53	0.676
synapse	0.35	0.630	0.47	0.728	0.45	0.729
velocity	0.82	0.456	0.62	0.407	0.59	0.531
Xalan	0.22	0.659	0.35	0.607	0.38	0.636
xerces	0.13	0.793	0.261	0.714	0.34	0.740
Average	0.402	0.602875	0.505	0.519625	0.511	0.609875

Table 7shows the detection performance of new file KOELM-LR. After detecting the new file, this model is trained on the classifier LR. When comparing the performance of KOELM-LR and the existing model WSM-LR, the proposed model achieves a high F-score value of about 0.793 in xerces software, a high G-mean of about 0.728 in the synapse software and a high balance value of about 0.740 in xerces software. In addition, the average value of the three metrics also high for the developed approach when compared to the existing model WSM-LR.

Table 9: Detection of new file KOELM- NB

	F-score		G-mean		Balance	
project	WSM-NB [18]	KOELM-NB	WSM-NB [18]	KOELM-NB	WSM-NB [18]	KOELM-NB
camel	0.275	0.199	0.46	0.135	0.454	0.494
jedit	0.4	0.586	0.70	0.477	0.682	0.6015
Log4j	0527	0.655	0.56	0.5453	0.540	0.5
poi	0.17	0.761	0.312	0.704	0.362	0.727
synapse	0.54	0.598	0.63	0.7106	0.608	0.711
velocity	0.88	0.597	0.85	0.576	0.851	0.626
Xalan	0.65	0.703	0.69	0.663	0.653	0.68
xerces	0.29	0.862	0.4	0.816	0.416	0.818
Average	0.45	0.620125	0.58	0.578363	0.57	0.644688

Table 8 shows the detection performance of new file KOELM-NB and the WSM-NB. After detecting the new file, this model is trained on the classifier NB. When comparing the performance of KOLEM-NB and the existing model WSM-NB, the proposed model achieves high F-score value of about 0.862, high G-mean of about 0.816 and high balance value of about 0.818 in the xerces project. In addition, the average value of the three metrics also high for the developed approach.Thus the proposed model proved its superiority.

5. CONCLUSION

Fault detection in a software engineering is gaining attention due to the severe impacts of the faults in the software and malfunctioning of software used in diverse domains. Among the various faults, two faults are required to be properly addressed. To deal with those issues, we propose a concept on cross-version software fault detection with data selection (CV-FDDS). This scenario is much reasonable and significant to detect the above-mentioned issues by predicting

the faults in the new versions based on the labelled data of previous versions. The proposed framework works on identifying the faults in the newer versions of the software projects on the basis of older version labels through the use of learning strategies. Initially, the dataset is preprocessed using an appropriate data filter. The older and newer files are treated in two different ways to accomplish this task. To deal with the existing or the older files, an adaptive spiking atom search recurrent neural network classifier (ASARNN) has been proposed through which the training and testing data are automatically selected. This is done by assigning higher weight values to the much relevant and noise-free older versions. Based on the training and testing, the newer versions comprising faults are detected using the proposed kernel optimized extreme learning machine (KOELM). The implementation of the developed approach is implemented on PYTHON. The implementation of the developed approach is implemented on PYTHON. The implemented approach performance is compared against the conventional methods with respect to the major performance indicators. The implemented nodel achieves a geometric-mean (G-mean) of about 0.7815 , f-measure of about 0.775 and balance is about 0.8055 respectively. In the future, the developed model can be applied to other software engineering problems like selection of test case.

REFERENCES:

[1] Xiao, Hui, Minhao Cao, and Rui Peng. "Artificial neural network based software fault detection and correction prediction models considering testing effort." *Applied Soft Computing* 94 (2020): 106491.

[2] Amasaki, Sousuke. "Cross-version defect prediction: use historical data, cross-project data, or both?." *Empirical Software Engineering* 25, no. 2 (2020): 1573-1595.

[3] Yao, Zhexi, Jiawei Song, Yushuai Liu, Tao Zhang, and Jinbo Wang. "Research on Cross-version Software Defect Prediction Based on Evolutionary Information." In *IOP Conference Series: Materials Science and Engineering*, vol. 563, no. 5, p. 052092. IOP Publishing, 2019.

[4] Huo, Xuan, and Ming Li. "On cost-effective software defect prediction: Classification or ranking?." *Neurocomputing* 363 (2019): 339-350.

[5] Haouari, Ahmed Taha, LabibaSouici-Meslati, FadilaAtil, and DjamelMeslati. "Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction." *Applied Soft Computing* 96 (2020): 106686.

[6] Chen, Xiang, Dun Zhang, Yingquan Zhao, Zhanqi Cui, and Chao Ni. "Software defect number prediction: Unsupervised vs supervised methods." *Information and Software Technology* 106 (2019): 161-181.

[7] Gao, Houleng, Minyan Lu, Cong Pan, and Biao Xu. "Empirical Study: Are Complex Network Features Suitable for Cross-Version Software Defect Prediction?." In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 1-5. IEEE, 2019.

[8] Bai, Xue, Hua Zhou, Hongji Yang, and Dong Wang. "Connecting historical changes for cross-version software defect prediction." *International Journal of Computer Applications in Technology* 63, no. 4 (2020): 371-383.

[9] Sultana, KaziZakia, VaibhavAnu, and Tai-Yin Chong. "Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach." *Journal of Software: Evolution and Process* 33, no. 3 (2021): e2303.

[10] Manjula, C., and Lilly Florence. "Deep neural network based hybrid approach for software defect prediction using software metrics." *Cluster Computing* 22, no. 4 (2019): 9847-9863.

[11] Ni, Chao, Xin Xia, David Lo, Xiang Chen, and Qing Gu. "Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction." *IEEE Transactions on Software Engineering* (2020).

[12] Xu, Zhou, Tao Zhang, Yifeng Zhang, Yutian Tang, Jin Liu, Xiapu Luo, Jacky Keung, and Xiaohui Cui. "Identifying crashing fault residence based on cross project model." In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 183-194. IEEE, 2019.

[13] Pandey, Sushant Kumar, and Anil Kumar Tripathi. "BCV-Predictor: A bug count vector predictor of a successive version of the software system." *Knowledge-Based Systems* 197 (2020): 105924.

[14] Sun, Ying, Xiao-Yuan Jing, Fei Wu, and Yanfei Sun. "Manifold embedded distribution adaptation for cross-project defect prediction." *IET Software* 14, no. 7 (2021): 825-838.

[15] Sohan, MdFahimuzzman, Md Ismail Jabiullah, Sheikh Shah Mohammad Motiur Rahman, and SM Hasan Mahmud. "Assessing the effect of imbalanced learning on cross-project software defect prediction." In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1-6. IEEE, 2019.

[16] Sohan, MdFahimuzzman, MdAlamgirKabir, Mostafijur Rahman, TouhidBhuiyan, Md Ismail Jabiullah, and EbubeoguAmarachukwu Felix. "Prevalence of Machine Learning Techniques in Software Defect Prediction." In *International Conference on Cyber Security and Computer Science*, pp. 257-269. Springer, Cham, 2020.

[17] Tanaka, Kazuya, AkitoMonden, and ZeynepYücel. "Prediction of software defects using automated machine learning." In *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 490-494. IEEE, 2019.

[18] Zhang, Jie, Jiajing Wu, Chuan Chen, Zibin Zheng, and Michael R. Lyu. "CDS: A Cross–Version Software Defect Prediction Model With Data Selection." *IEEE Access* 8 (2020): 110059-110072.

[19] Yuan, Zhidan, Xiang Chen, Zhanqi Cui, and Yanzhou Mu. "ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost." *IEEE Access* 8 (2020): 30037-30049.

[20] Zhu, Mengmeng, and Hoang Pham. "A generalized multiple environmental factors software reliability model with stochastic fault detection process." *Annals of Operations Research* (2020): 1-22.

[21] Alsghaier, Hiba, and Mohammed Akour. "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier." *Software: Practice and Experience* 50, no. 4 (2020): 407-427.

[22] Shukla, Swapnil, T. Radhakrishnan, K. Muthukumaran, and LalitaBhanu Murthy Neti. "Multi-objective cross-version defect prediction." *Soft Computing* 22, no. 6 (2018): 1959-1980.

[23] Xu, Zhou, Shuai Li, Xiapu Luo, Jin Liu, Tao Zhang, Yutian Tang, Jun Xu, Peipei Yuan, and Jacky Keung. "TSTSS: A two-stage training subset selection framework for cross versiondefect prediction." *Journal of Systems and Software* 154 (2019): 59-78.

[24] Kim, Robert, Yinghao Li, and Terrence J. Sejnowski. "Simple framework for constructing functional spiking recurrent neural networks." *Proceedings of the national academy of sciences* 116, no. 45 (2019): 22811-22820.

[25] Zhao, W., Wang, L. and Zhang, Z., 2019. Atom search optimization and its application to solve a hydrogeologic parameter estimation problem. *Knowledge-Based Systems*, 163, pp.283-304.

[26] Li, B., Rong, X. and Li, Y., 2014. An improved kernel based extreme learning machine for robot execution failures. *The Scientific World Journal*, 2014.

[27] Yang X, Wen W. Ridge and lasso regression models for cross-version defect prediction. *IEEE Transactions on Reliability*. 2018 Jun 29;67(3):885-96.

[28] Xu, Z., Li, S., Tang, Y., Luo, X., Zhang, T., Liu, J. and Xu, J., 2018, May. Cross version

defect prediction with representative data via sparse subset selection. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)* (pp. 132-13211). IEEE.

[29] Yuan, Zhidan, Xiang Chen, Zhanqi Cui, and Yanzhou Mu. "ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost." IEEE Access 8 (2020): 30037-30049.

[30] Xu, Z., Liu, J., Luo, X. and Zhang, T., 2018, March. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. *In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 209-220). IEEE.*