IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 22, 2019, accepted January 14, 2020, date of publication January 22, 2020, date of current version February 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2968362

Hyper-Parameter Optimization of Classifiers, Using an Artificial Immune Network and Its Application to Software Bug Prediction

FAIZA KHAN[1], SUMMRINA KANWAL[2], SULTAN ALAMRI[2], AND BUSHRA MUMTAZ[1]
[1]Faculty of Computing, Riphah International University, Islamabad 45211, Pakistan
[2]Department of Computing and Informatics, Saudi Electronic University, Riyadh 11673, Saudi Arabia

Corresponding author: Summrina Kanwal (summrina@gmail.com)

ABSTRACT Software testing is an important task in software development activities, and it requires most of the resources, namely, time, cost and effort. To minimize this fatigue, software bug prediction (SBP) models are applied to improve the software quality assurance (SQA) processes by predicting buggy components. The bug prediction models use machine learning classifiers so that bugs can be predicted in software components in some software metrics. These classifiers are characterized by some configurable parameters, called hyper-parameters that need to be optimized to ensure better performance. Many methods have been proposed by researchers to predict the defective components but these classifiers sometimes not perform well when default settings are used for machine learning classifiers. In this paper, software bug prediction model is proposed which uses machine learning classifiers in conjunction with the Artificial Immune Network (AIN) to improve bug prediction accuracy through its hyper-parameter optimization. For this purpose, seven machine learning classifiers, such as support vector machine Radial base function (SVM-RBF), K-nearest neighbor (KNN) (Minkowski metric), KNN (Euclidean metric), Naive Bayes (NB), Decision Tree (DT), Linear discriminate analysis (LDA), Random forest (RF) and adaptive boosting (AdaBoost), were used. The experiment was carried out on bug prediction dataset. The results showed that hyper-parameter optimization of machine learning classifiers, using AIN and its applications for software bug prediction, performed better than when classifiers with their default hyper-parameters were used.

INDEX TERMS Artificial immune network (AIN), artificial immune system (AIS), hyper-parameter optimization, optimized artificial immune network (opt-aiNet), software bug prediction (SBP).

I. INTRODUCTION

A software bug is a defect, an error or fault due to which incorrect or unpredictable results are produced. The presence of these software bugs directly affects the quality and maintenance cost of software systems. Due to limited resources, software modules that contain bugs must be focused on. To overcome resource allocation problems SBP models have been designed to identify bug-prone software modules using machine learning classification techniques. SBP is an important area in software development activities because predicting bugs in software systems improve their quality and thus decreases maintenance costs and the efforts required. SBP makes it possible for SQA teams to detect the bugs in a buggy code during the software development process in order to prevent it from causing more bugs in other parts of the software modules or components. Moreover, it also helps in optimizing testing by focusing on the components that are directly affected by bugs and thereby improving the overall quality of the software systems. Hyper-parameters are parameters that are tuned for machine learning classifiers so to improve their performance or prediction accuracy. Hyper-parameter optimization is basically the process of tuning the hyper-parameters of machine learning models or the process of finding the best hyper-parameters value; it is also known as model-selection or hyper-parameter tuning. Different classifiers have different features according to which the hyper-parameters need to be optimized [2], [3]. Over the past years, various techniques have been proposed by researchers to examine the performance of machine learning classifiers in software bug prediction [2], [12]. It has been evaluated by Fu *et al.* [31] that 80% of the most cited software bug

The associate editor coordinating the review of this manuscript and approving it for publication was An-An Liu.

TABLE 1. Bug prediction dataset by D 'Ambros *et al.* [29].

system	Release	#classes	%buggy
Eclipse JDT Core	3.4	997	≈ 20%
Eclipse PDE UI	3.4.1	1,497	≈ 14%
Equinox	3.4	324	≈ 40%
Mylyn	3.41	1,862	≈ 13%
Lucene	2.4.0	691	≈ 9%

prediction studies rely on the default setting. The default parameter settings impact the performance of defect prediction models, and as a result, they under-perform.

The prime motivation behind our proposed algorithm for optimization is from Aydin *et al.* [13], Tantithamthavorn *et al.* [2], [12] and Osman *et al.* [3]. Aydin *et al.* [13] applied clonal selection (AIS based technique) to optimize SVM hyper-parameters and reported improvement in the accuracy. Tantithamthavorn *et al.* [2], [12] investigated automated parameter optimization on 20 classification algorithms using grid search, random search, GA, and DE and Caret automated parameter optimization technique for defect prediction models on 18 datasets and reported improved results. Osman *et al.* [3] use hyper-parameter optimization on KNN and SVM and achieved an increase in Accuracy. Moreover, it is also stated that when a Genetic algorithm (GA), Random search (RS), Grid search (GS) and Differential Evolution (DE) are used for hyper-parameter optimization problems it increases computational costs and caused over-fitting. Since GA are non-deterministic methods, the results may differ every-time you run the algorithm on the same sample set. We aimed to overcome these problems by experimenting with Opt-aiNet and for this we have used the bug prediction dataset, shown in Table 1 below.

The rest of the paper is arranged as follows. Section 2 provides related work, section 3 provides the background information about the techniques used in this paper. The proposed method is presented in Section 4. The experimental setup in section 5 and the results of our experimental study in Section 6 is described. The comparison with previous techniques is described in section 7. Conclusion and suggestions for future work are given in Section 8.

The contributions made in this paper is to investigate the impact of hyper-parameter optimization of machine learning classifiers using opt-aiNet on the performance of SBP on 2 performance measures. And comparison with other techniques used by previous researchers for hyper-parameter optimization in SBP.

II. RELATED WORK

Many studies have proposed using machine learning techniques for SBP. In most studies, the hyper-parameters of these machine learning classifiers are left to their default values. But in very few studies the effect of hyper-parameter optimization on bug prediction accuracy and AUC has been examined. Tantithamthavorn *et al.* [2] investigated automated parameter optimization on 26 classification techniques using a grid search, a random search, a genetic algorithm, and a differential evolution for defect prediction models on 18 datasets from National Aeronautics and Space Administration (NASA), Proprietary, Apache and Eclipse. Their results showed that automated parameter optimization increases the accuracy by up to 40 percentage points. Osman *et al.* [3] investigated the optimization of KNN and SVM using a grid search. The results showed that prediction accuracy was improved by up to 20% in KNN and by up to 10% in SVM. Tantithamthavorn *et al.* [12] explored the hyperparameter optimization technique, Caret (an automated parameter optimization technique), on 26 classification techniques as well as using the out-of-sample bootstrap validation technique on 18 datasets from NASA, Proprietary, Apache and Eclipse. The results showed that Caret increases the accuracy of defect prediction models by up to 40 percentage points and with that Caret-optimized classifiers 9 out of 26 classification techniques (35%) are more stable. Aydin *et al.* [13] used a multi-objective artificial immune algorithm to optimize the SVM-RBF kernel for fault diagnosis of induction motors and anomaly detection problems to adjust the parameters of SVM. They reported an increase in performance accuracy by up to 97%. Sarro *et al.* [14] applied the GA to search for suitable settings of SVM parameters to be used for inter-release fault prediction. For this SVM was used with RS and GS configuration strategies. They further compared their proposed algorithm with other machine learning techniques such as Logistic Regression (LR), DT (C4.5), NB, Multi-Layer Perceptron's (MLP), KNN, and RF. They reported that a genetic algorithm with SVM is effective for inter-release fault prediction and the accuracy of SVM with an RS has improved by up to 35%, with Recall up to 80%, with F-measure up to 48% and with Precision up to 4%. The accuracy of SVM with a GS also improved by up to 13%, with Recall up to 19%, with F-measure up to 14% and with Precision up to 4%. Fazel [16] used a GA in conjunction with KNN for software fault prediction on 13 datasets from NASA MDP. To get better performance, the following parameters were changed: the dataset, the number of GA generations, the population and changes in the configuration operator; KNN was also used for calculating the value of indices. The results showed that the proposed method has more than a 95% higher detection rate. Pushpavathi *et al.* [17] using the integrated approach of a GA based on fuzzy C-means clustering and RF classifiers for software defect prediction on 5 NASA defect datasets. The results showed that a fuzzy-c-means-clustering-based approach provided a better result with more than 90% accuracy. Ibrahim *et al.* [19] proposed a bat-search algorithm (BA) for feature selection (FS) and an RF algorithm for software defect prediction on a dataset from a promise repository. The proposed approach was compared with a GA and an Ant search algorithm, such as FS, and Fuzzy Unordered Rule Induction Algorithm (FURIA), Multi-layer Perceptron (MLP), NB, or KStar for prediction. They reported that the RF algorithm and Bat Search Algorithm have higher accuracy than other algorithms. Fu and Menzies [26] used

differential Evolution (DE) to tune SVM-RBF on the dataset of data outcomes. The proposed approach was also compared with a convolutional neural network (CNN). It indicated that tuning SVM improved the performance of the method and the method also ran much faster than it did with CNN. Rathore and Kumar [27] used genetic programming (GP) for predicting the number of faults in a software system on 10 software projects from a promise data repository. They reported that GP was able to predict most of the faults in the software system and the average recall value was 35%. D'Ambros *et al.* [29] used bug prediction approaches such as Change metrics, previous defects, Source code metrics, Entropy of changes, Churn, and Entropy, on bug prediction datasets and also compared bug prediction approaches. They performed class level bug prediction and for FS use principal component analysis; a regression model was also built. They reported that the approach based on churn and entropy had better predictive and explanative power than the other approaches applied. Chakraborty *et al.* [33] used hyperparameter optimization to make a model a fair without losing its predictive power. For hyper-parameter optimization, a Fast Sequential Model-Based Method (FLASH) is used as an optimization technique and the parameters of LR and CART are optimized using 3 datasets Adult Census Income, Compas and German Credit Data. They reported that for the German dataset the three objectives were improved and indicating that if fairness and performance of multi-objective optimization are understood, then it is possible to achieve one without affecting the other one. öztürk *et al.* [34] investigated the effect of hyperparameter optimization in cross-project defect prediction (CPDP) and within-project defect prediction (WPDP). For this they have used RF and SVM using a grid search for searching and Gradient Boosting machine for Boosting and 20 datasets from Softlab, NASA MDP, and open-source are used. They reported that CPDP achieved more successful results than WPDP and in WPDP polynomial kernel of SVM got better results while in CPDP linear kernel have achieved higher AUC.The AUC values were increased in all SVM functions when hyper-parameter optimization was applied. Son *et al.* [35] conducted a systematic mapping in which he selected 98 studies out of 156 regarding defect prediction. They have reported that the most studied search-based techniques used in defect prediction are the Artificial Immune Recognition System (AIRS), Ant Colony Optimization (ACO), Genetic Programming (GP), Evolutionary Programming (EP), Evolutionary Subgroup Discovery (ESD), GA, and Gene Expression Programming (GeP) and Particle Swarm Optimization (PSO). Pandey *et al.* [36] investigated the effect of some Bayesian network (BN) and classifier for bug prediction on NASA and Eclipse datasets. Receiver operating characteristics (ROC) and AUC performance measures are used to measure various parameters performance of the classifiers. They have reported that BN out-performed. Hammouri *et al.* [37] used NB, DT and Artificial Neural Networks (ANN) for SBP on 3 datasets namely DS1, DS2 and DS3. They have reported that DT has out-perform and had better results than NB and ANN. Jayanthi and Florence [38] used an integrated approach of principal component analysis (PCA) and neural networks (NN) for SBP on NASA datasets. In this PCA is used for feature reduction and NN as a classification technique.The proposed method is compared with k-NN, SVM, NB and LDA. They have reported that the proposed method obtains 97.20% AUC and provides better improvement than other methods. Manjula and Florence [39] used an integrated approach for SBP as GA for feature selection and deep neural network (DNN) for classification on a dataset from Promise repository.The proposed method is compared with k-NN, SVM and DT. They have reported that the proposed method performs better than the other techniques and has obtained more than 97% accuracy. Immaculate *et al.* [40] used supervised machine learning algorithms such as LR, NB, DT and RF for SBP on data set made of fifteen Java and Python projects. They have reported that 97% accuracy is achieved in RF and RF out-performed than other machine learning algorithms. Pandey *et al.* [41] used a combined approach of ensemble learning (EL) and deep representation (DR) namely bug prediction using deep representation and ensemble learning technique (BPDET) for SBP on 12 NASA datasets. The class imbalance issue in the dataset is addressed by the SMOTE sampling technique. Staked denoising auto-encoder (SDA) is a method of feature learning that is used for DR.They have reported that BPDET has out-performed on most of the datasets and BPDET is tested using Wilcoxon rank-sum test and as a result, null hypothesis is rejected at $\alpha = 0.025$. Gomes *et al.* [43] performed a comprehensive mapping study in which he had reviewed recent research efforts on automatically bug report severity prediction and for this he had selected 54 papers. They reported that 27 studies are addressing bug report severity prediction on Free or Libre Open Source Software.

Based on the findings from various literature reviews of SBP problem's it can be stated that in most of the bug prediction studies default hyper-parameter values are used in classifiers as a result of which these classifiers did not perform well as they had been expected to do. It was also discovered that 80% of the defect prediction studies rely on default hyperparameter values. In contrast, our study aims to introduce a model for software bug prediction using the AIN and machine learning classifiers. The machine learning classifiers are used in conjunction with the AIN to improve prediction accuracy and AUC through hyper-parameter optimization. We also investigated the effect of hyperparameter optimization on classifiers by comparing the prediction accuracy and AUC of these classifiers before and after hyper-parameter optimization.

III. BACKGROUND

Here we have discussed the techniques, machine learning classifiers, and hyper-parameters in detail which are used in this paper.

A. MACHINE LEARNING CLASSIFIERS

For this study, we have used seven classifiers such as KNN, SVM, NB, DT, LDA, RF and AdaBoost.

KNN is the nearest neighbor classifier which is classified as being an object to the nearest class based on its majority votes to its neighbors or because it stores all available observations and classifies each observation based on its similarity. In this, we have used KNN with two distance metrics namely Euclidean and Minkowski metric. A distance metric or function provides the distance between two elements of a class. And the distance function varies from different distance metrics. In the Euclidean metric, the distance between two data points is calculated in a plane. The function used for the Euclidean metric is:

$$d(x, x') = \sqrt{\sum_{i=1}^{k}(x_i - x'_j)^2}$$

where d is the distance function, n is the number of variables, x_i and x'_i are the variables of vectors x and x' respectively, in the two-dimensional vector space. i.e. x = (x1,x2,x3,...) and x' = (x'1,x'2,x'3,...).

The Minkowski distance is a norm vector space and is a generalized form of both Euclidean and Manhattan distance which means that the function given below for Minkowski metric to calculated the distance between two points can be manipulated.

$$d(x, x') = (\sum_{i=1}^{k}(|x_i - x'_i|)^e)^{1/e}$$

In the above function, the value of e can be manipulated and x_i and x'_i are the variables of vectors x and x' respectively, in the two-dimensional vector space.

SVM is a geometrically-inspired classifier that uses a hyper-plane to separate two classes by choosing the best separating hyper-plane to classify data linearly. For this the hyperplane is:

$$f(x) = w^T x + b.$$

where w is a 2-d vector to the hyper-plane. SVM is trained to generate a model and then testing is assessed. If the sample data is not divided linearly then nearly other methods are used. SVM-RBF is a popular Kernel method used in SVM. SVM-RBF Kernel is a function whose value is dependent on the distance from its original point to some other point and the function for SVM-RBF is given below:

$$k(x, x') = exp(-\gamma||x - x'||^2)$$

In this γ is kernel parameters for each kernel function and these parameters affect the performance of SVM and distance used in the original space is calculated by the similarity of x and x'.

NB is based on the Bayesian theorem and on the fact that it is a probabilistic model that assumes that predictors are independent of each other and that is also used for classification tasks. The function used for NB is:

$$p(x'_i|x) = \frac{1}{\sqrt{2\pi\alpha_x^2}} exp(-\frac{(x'_i - \mu_x)^2}{2\alpha_x^2}$$

The parameters α_x and μ_x are used to estimate the like-hood and x is class variable, x'_i are the dependent features. DT is a classifier that repetitively divides the plot into sub-parts by identifying lines and it uses feature values to classify objects. In DT we have used classification and regression tree (CART). CART is used for both classification and regression decision trees. The decision tree built by using the CART algorithm is a binary tree where each node will have only two child nodes. The function of CART is as follows:

$$1 - \sum_{y=0}^{y=1} P_x^2$$

where P is the proportion of observations, y is the target variable and for binary y takes only 0 and 1 value.

LDA is a dimensionality reduction technique that is used as a processing step in machine learning applications. This dimensionality reduction determines to reduce the dimensions by removing unnecessary features by changing the features from higher dimension space to lower dimension and in doing so it takes labels into considerations. In LDA prediction is made by estimating the probability that a new set of inputs belongs to the class. The class which gets the highest probability is the output class and predictions are made. The discriminate function is given below:

$$DA(i) = i * \frac{muA}{sigma^2} - \frac{muA^2}{2 * sigma^2} + ln(PlA)$$

DA(i) is the discriminate function for class A given input i, the muA, $sigma^2$, and PIA. Where PIA is referred to as the base probability of each class (A) observed in training data, muA is the mean value of each input (i) for each class (A) and $sigma^2$ is the variance across all inputs (i). RF generates a forest with several trees: the more the number of trees in the forest, the greater the accuracy of the results will be. RF uses bagging and features randomness when it is building each tree while creating a forest of trees. classification, RF will use the Gini index or the formula given below will decide how the nodes on DT will branch.

$$= 1 - \sum_{i}^{c} = 1(p_i)^2$$

where p_i represents the relative frequency of the class observed in the dataset and c is the number of classes. The formula is using the class and probability so as to determine which DT branches will occur.

Adaboost tries to generate a robust classifier from several frail classifiers. It is mainly castoff to increase the performance of decision trees on dual classification problems. It is usually referred to as AdaBoost M1. In AdaBoostM1 training

is viewed as stage-wise minimization of the exponential loss. The function of AdaBoostM1 is given below:

$$\sum_{n=1}^{N} w_n exp(-r_n f(y_n))$$

where r_n is the rule class labels which are equal to -1,+1, w_n are the original observation weights passed to Adaboost which are normalized to add up to 1 and $f(y_n)$ is predicted classification score.

We have chosen these classifiers for software bug prediction because they operate differently and have many hyper-parameters. While SVM and KNN have been widely used in the literature, NB, DT(CART), LDA, RF, and AdaBoost have not been widely used in the software bug prediction literature.

B. ARTIFICIAL IMMUNE NETWORK (AIN)

AIS is a sub-field of computational intelligence and it is inspired by the biological immune system. The task of the natural immune system is to detect or recognize and then destroy the antigens. The immune system can stop pathogens and antigens. Pathogens are disease-causing agents such as bacteria, viruses, parasites, and pollen while antigens are toxic substances. Artificial Immune systems have three sub-fields: clonal selection, negative selection, and the immune network.

The artificial immune network algorithm (aiNet) is an immune network algorithm first proposed by de Castro and Timmis [9], [10], [32] from the area of AIS and is inspired by the immune network theory of immune systems. Immune networks are also known as idiotypic networks and are also defined as complex networks of paratopes that identify antigens (Ag). In effect whenever the body is attacked by invaders (pathogens), it responds to antibodies (Ab) which helps them in identifying antigens. The interaction between Ab and Ag is measured by the affinity. This means that the Ab which is more easily recognized by Ag have a higher Ag affinity [8]. The aiNet contains an optimization version for optimization problems called the optimization of the artificial immune network (opt-aiNet) which is a discrete immune network algorithm [8]. Opt-aiNet can perform as a uni-modal or as a multi-modal optimization and has its stopping criteria defined. Opt-aiNet grows a population that consists of a network of antibodies and the size of the population is dynamically adjustable. Opt-aiNet creates a memory for a set of Ab, which represents the best solution for the objective function. Opt-aiNet has to select all cells for the cloning process and for each iteration it gives all the cells the same number of clones [9], [10]. We have tried to use opt-aiNet used by the Brownlee [8] for cost function minimization. The flow chart of opt-aiNet is shown in Fig. 1.

A summary of opt-aiNet is given below [4]:

1) Initialization: Generates a set of N (populations).
2) Antigen representation: While the stopping criterion is not met do

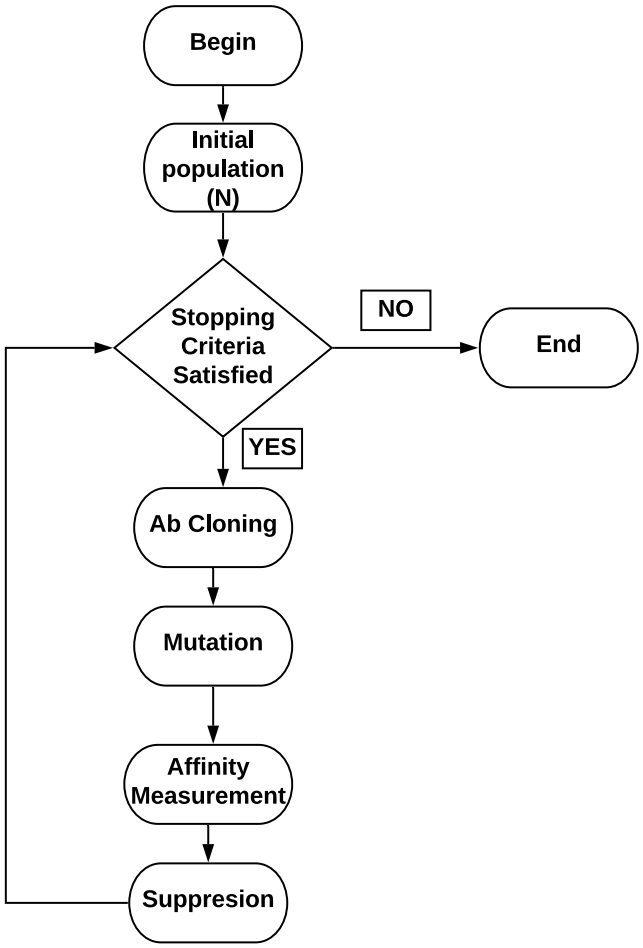

FIGURE 1. Flow diagram of opt-aiNet.

   a) (a) Cloning: NC (number of clones) for each cell are created and are proportional to the cell affinity
   b) (b) Mutation: Each clone, which is inversely proportional to its fitness, is mutated
   c) (c) Affinity Measurement: Determines the distance from the solution.
   d) (d) Suppression: Concludes the highest affinity network and performs suppression.
3) End of process

C. HYPER-PARAMETER OPTIMIZATION

Hyper-parameters are parameters that have tuned for machine learning classifiers to boost their performance or accuracy. These parameters usually affect the learning, construction, and evaluation of machine learning classifiers. Hyper-parameter optimization is the process of finding the best hyper-parameters of machine learning classifiers and is also known as model selection and hyperparameter tuning. The hyper-parameters of the studied classification techniques which are tuned and the parameters of opt-aiNet are given in table 2 and 3 below. To optimize the hyper-parameters of these classifiers, opt-aiNet was used. Details of the

TABLE 2. Hyper-parameters of the classifiers that are tuned.

Machine Learning Classifiers	Parameters Name	Default Parameters value	Optimized Parameters value range
SVM-RBF	Sigma (σ)	0.5	(0.1,0.9)
	Cost (C)	1	(0.25,4)
KNN	Number of neighbors (K)	1	(1,17)
	Exponent (E)	3	(0.5,5)
NB	DistributionNames	0	Normal (0,1)
DT (CART)	MaxNumSplits	5	(1,50)
	MinLeafSize	3	(1,50)
LDA	Discrimtype	linear	linear
	Delta (D)	0	(0,1)
RF	NumLearningCycles	100	(10,100)
AdaBoost	NumLearningCycles	100	(10,100)

TABLE 3. Parameters of opt-aiNet.

Opt-aiNet		
Parameters	Default values	Optimized Values
Suppression threshold	0.1	0.1
Number of clones generated(Nc)	20	50
Number of clones multiplier (N)	10	50
Maximum number of generations	200	50

parameters tuned for these classifiers using opt-aiNet are shown in Table 2 and those of the parameters tuned for opt-aiNet are given in Table 3.

We have chosen hyper-parameters for seven machine learning classifiers mentioned above in section 3(a). The hyper-parameters for SVM are (σ) and C. The parameter C is the cost of misclassification on training data and it controls the distance between the errors and margin. The parameter σ is to handle non-linear classification. As the value of C and σ increases the model gets over-fitted and high value for C gives poor accuracy and high value for σ gives more accuracy. The hyper-parameters for KNN are K and E. K is the number of nearest neighbors and on the basis of K, it is decided that how many nearest neighbors will be there of that particular class should be included in that class and E is Minkowski distance exponent is a comma-separated pair and it should be positive scalar value. The hyper-parameter for NB is distribution type or name which is how data should be distributed normal or kernel density estimation. The hyper-parameter for DT (CART) is the maximum number of decision splits (MaxNumSplits) which is basically the branch nodes in which the tree should be split and the minimum number of leaf node observations (MinLeafSize) which are that how much nodes a leaf have the minimum. The hyper-parameter for LDA is Delta (D) which is a linear coefficient threshold and setting a higher value for delta so that more predictors are eliminated. The hyper-parameter for RF is the number of the ensemble learning cycle (NumLearningCycles) is all predictors combination. If a positive integer is specified then at every cycle the software trains one weak learner for every tree in RF. The hyper-parameter for AdaBoost is the number of the ensemble learning cycle (NumLearningCycles) and if a positive integer is specified then at every cycle the software trains one weak learner for every learner. The software in RF and AdaBoost means all trained learners.

IV. PROPOSED OPT-AINET FOR PARAMETER OPTIMIZATION OF CLASSIFIERS

We have proposed an opt-aiNet algorithm to optimize the parameters of classifiers, such as SVM, KNN, NB, DT, LDA, RF, and AdaBoost. This optimization aims to improve the performance and prediction accuracy of these classifiers. This method takes the accuracy and AUC of the classifiers as the fitness values for optimization. The process of opt-aiNet for parameter optimization of classifiers is shown in Figure 2.

Our proposed approach explains that the data preprocessing was first carried out utilizing which the data was normalized in the form of 0 and 1 because it would also help to increase the classifier accuracy. The formula for data preprocessing normalization is shown as below:

$$x' = \frac{x - xmin}{xmax - xmin}$$

where x' is normalized value for variable x, x is current value for variable x, xmin is a minimum data point in the dataset and xmax is a maximum data point in the dataset. After normalization, training data was used, to train the classifier and testing data was used to calculate the classification accuracy. Once the classifier was trained with training data using 10-fold cross-validation, testing was carried out on the testing data and the results were evaluated using classification accuracy and AUC. When the classification accuracy and the AUC were obtained, then the fitness function was evaluated by the classifier's classification accuracy, namely, AUC and hyper-parameters. The function for evaluating the fitness is given below:

$$f = a\ AUC(D) = \frac{\sum_{i}^{N=1} access(D_i)}{N}$$

where $access(D_i) = 1\ classifiedL(D_i)$ then correct classified & $access(D_i) = 0\ classifiedL(D_i)$ then incorrect classified, f is the fitness function to be evaluated, a is classification accuracy, D ($d_1, d_2, d_3, \ldots d_n$) is dataset which consists of N features with class labels L and classified $L(D_i)$ is correct if the instance is classified correctly like if labels from the dataset is same as the output. While evaluating this fitness function antibodies (Ab) that are between the range of hyper-parameters were produced. Each antibody consisted of accuracy, AUC and hyper-parameters. For each antibody,

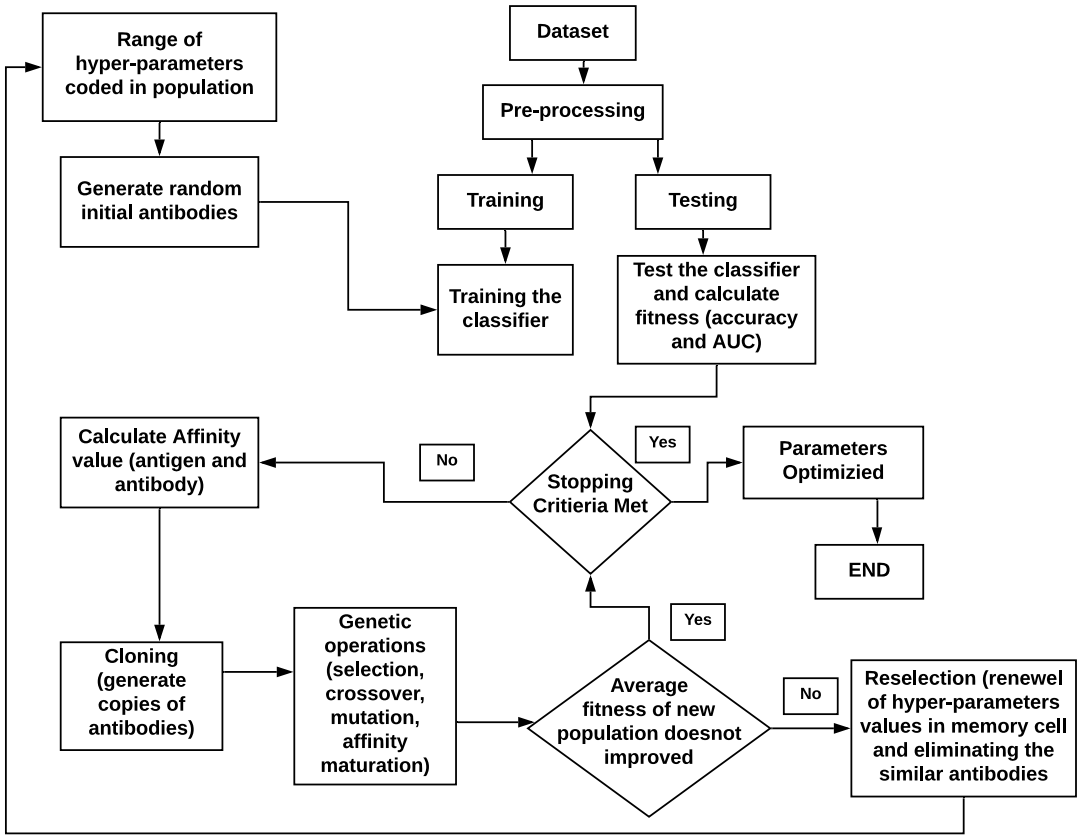

FIGURE 2. Our proposed framework.

the initial population was generated randomly and the size of the population was 50 selected. After the initial population was created, the value of the antigen and the antibody was calculated and also the affinity between the antibody and the antigen (Ag) was noted. The function used for affinity and antigen calculation is given below:

$$Affinity(Ab_x, Ab_y) = \frac{1}{d(Ab_x, Ab_y)'}$$

where $Affinity(Ab_x, Ab_y)$ is the similarity between two antibodies and d is the Euclidean distance between two antibodies Ab_x and Ab_y and function for d is as follows:

$$d = (Ab_x, Ab_y) = \sqrt{\sum_{k=1}^{n} (Ab_x k, Ab_{yk})^2}$$

The performance attained by the classifier was calculated for each one in the population and the algorithm was iterated until the given number of generations was satisfactory. After this cloning, selection, mutation, and crossover had been performed the system searched for one with a higher affinity or a better solution was searched by the system. The affinity mutation was performed according to the following function.

$$C' = c + \alpha.N(0, 1)$$
$$\alpha = (1/\beta)e. - f^*)$$

TABLE 4. Bug prediction dataset source code metrics [3].

Metric Name	Description
CBO	Coupling Between Objects
DIT	Depth of Inheritance Tree
FanIn	Number of classes that reference the class
FanOut	Number of classes referenced by the class
LCOM	Lack of Cohesion in Methods
NOC	Number of Childern
NOA	Number of Attributes in the class
NOIA	Number of Inherited Attributes in the class
LOC	Number of lines of code
NOM	Number of Methods
NOIM	Number of Inherited Methods
NOPRA	Number of PRivate Attributes
NOPRM	Number of PRivate Methods
NOPA	Number of Public Attributes
NOPM	Number of Public Methods
RFC	Response For Class
WMC	Weighted Method Count

where C' is a mutated cell, C is a cell, N(0,1) is a Gaussian random number of zero mean and standard deviation $\sigma = 1$, β is a control parameter which adjusts the mutation range and controls the decay of the inverse exponential function and value for $\beta = 100$ but it is also an user-specified value in the algorithm of opt-aiNet and α is the affinity proportional and f is the fitness of parent cell and f^* is the fitness of an individual Ag or Ab in the population normalized in the interval [0,1]. A mutation is only accepted if the c' lies in the range of its

TABLE 5. Results of classifiers in accuracy before hyper-parameter optimization.

Dataset	SVM-RBF	KNN (Euclidean Metric)	KNN (Minkowski Metric)	NB	Decision Tree (CART)	Linear Discriminate Analysis (LDA)	Random Forest (RF)	Adaptive Boosting (AdaBoost)
Eclipse JDT Core	80.85%	83.24%	83.75%	81.74%	80.35%	84.55%	86.55%	84.65%
Equinox	72.83%	68.51 %	68.51 %	70.98 %	66.66 %	60.18 %	67.28 %	67.90%
Lucene	89.86 %	88.42 %	88.71 %	85.23 %	87.98 %	85.23 %	89.86 %	88.14 %
Mylyn	91.78 %	92.10 %	90.70 %	83.99 %	92.26 %	90.01 %	94.19 %	93.73 %
Eclipse Pde UI	83.95%	83.95%	83.55%	81.74%	81.24 %	84.65%	86.45%	84.86%

TABLE 6. Results of classifiers in accuracy after hyper-parameter optimization using opt-aiNet.

Dataset	SVM-RBF	KNN (Euclidean Metric)	KNN (Minkowski Metric)	NB	Decision Tree (CART)	Linear Discriminate Analysis (LDA)	Random Forest (RF)	Adaptive Boosting (AdaBoost)
Eclipse JDT Core	85.16%	85.84%	85.91%	82.17%	83.46%	84.95%	86.89%	85.75%
Equinox	73.43%	69.96 %	70.78 %	72.02 %	69.70 %	60.18 %	69.79 %	70.15 %
Lucene	90.81 %	90.88 %	90.88 %	86.23 %	90.34 %	90.88 %	90.72 %	89.57 %
Mylyn	94.53 %	93.98 %	93.56 %	84.33 %	94.22 %	94.22 %	95.18 %	94.63 %
Eclipse Pde UI	89.91%	85.92%	85.95%	82.14%	83.37 %	84.90%	86.77%	85.74%

TABLE 7. Results of classifiers in AUC before hyper-parameter optimization.

Dataset	SVM-RBF	KNN (Euclidean Metric)	KNN (Minkowski Metric)	NB	Decision Tree (CART)	Linear Discriminate Analysis (LDA)	Random Forest (RF)	Adaptive Boosting (AdaBoost)
Eclipse JDT Core	0.45	0.56	0.59	0.57	0.54	0.57	0.69	0.62
Equinox	0.57	0.51	0.56	0.55	0.58	0.50	0.55	0.58
Lucene	0.65	0.61	0.63	0.65	0.69	0.67	0.65	0.63
Mylyn	0.70	0.72	0.70	0.63	0.70	0.67	0.74	0.72
Eclipse Pde UI	0.47	0.57	0.59	0.59	0.58	0.61	0.64	0.63

domain. In the end, the new best antibodies that had been found were replaced with the old ones, which were then shown to be the worst [42]. When the stopping criteria had been satisfied the fitness function process was ended and the hyper-parameters were optimized and the average accuracy achieved was obtained.

V. EXPERIMENTAL SETUP

The experiment was conducted using the Matlab (R2018a) software tool on the Intel(R) Core (TM) processor with 8GB RAM installed on it.

A. BUG PREDICTION DATASET

D'Ambros *et al.* [29] provided a bug prediction dataset for bug predictors to identify bugs. A bug prediction dataset is a collection of software system models, metrics and their histories. This dataset is widely used by many researchers in their bug/defect prediction studies and open source available for researchers to investigate more SBP models. This bug prediction dataset has both source code metrics, and change metrics. But we have used a single version approach and this approach does not need a history of the system but instead investigates its current state using a wide range of source code metrics, such as Chidamber and Kemerer (CK) and object-oriented (OO) metrics and line of code (LOC). The single version approach is selected because to investigate the current state of the system in details using a wide range of metrics. The bug prediction dataset has source code metrics, given in Table 4 for the classes in five open-source Java systems, given in Table 1. The column namely ''#classes'' in Table 1 indicates the range of instances in the projects. The number of the instances ranges from 0 to 1900 and as hyper-parameter optimization is a time-consuming tasks so that's why this dataset is taken. The reliability of the experimental results mainly depends on the size of the dataset. Thus, the data sets having large sizes may highly affect the accuracy of the optimization.

VI. RESULTS

This section provides the results of the hyper-parameter optimization of classifiers using opt-aiNet for software bug prediction to boost its prediction accuracy and AUC.

TABLE 8. Results of classifiers in AUC after hyper-parameter optimization using opt-aiNet.

Dataset	SVM-RBF	KNN (Euclidean Metric)	KNN (Minkowski Metric)	NB	Decision Tree (CART)	Linear Discriminate Analysis (LDA)	Random Forest (RF)	Adaptive Boosting (AdaBoost)
Eclipse JDT Core	0.86	0.80	0.78	0.68	0.70	0.65	0.72	0.73
Equinox	0.68	0.67	0.65	0.64	0.63	0.51	0.62	0.65
Lucene	0.78	0.71	0.72	0.68	0.77	0.80	0.79	0.77
Mylyn	0.77	0.78	0.75	0.65	0.86	0.85	0.87	0.83
Eclipse Pde UI	0.78	0.68	0.71	0.62	0.69	0.74	0.73	0.71

TABLE 9. A comparison of our proposed work results with those using the previous techniques.

Papers	Optimization methods	Classifiers	AUC (Improved)	Accuracy (Improved)
The Impact of Automated Parameter Optimization on Defect Prediction Models (2018) (extended work of 2016 paper)	GA Grid Search Random Search DE	SVM-RBF	0.04	
		KNN	0.07	
		NB	0.06	
		Adaboost	0.39	
		RF	0.03	
		CART	0.38	
Automated Parameter Optimization Classification Techniques for Defect Prediction Models (2016)	Caret	SVM-RBF	0.04	
		KNN	0.07	
		NB	0.06	
		Adaboost	0.39	
		RF	0.03	
		CART	0.38	
A multi-objective artificial immune algorithm for parameter optimization in support vector machine	AIS (clonal selection)	SVM-RBF		3%
Hyper Parameter Optimization to improve Bug Prediction Accuracy	Grid Search	SVM-RBF		10%
		KNN		20%
Comparing Hyperparameter Optimization in Cross and Within-Project Defect Prediction: A Case Study	Grid Search	RF	0.7	5%
		SVM	0.8	7%
Hyper-parameter Optimization of Classifiers using artificial immune network and its application to software bug prediction	Opt-aiNet	SVM-RBF	0.41	6%
		KNN	0.24	3%
		NB	0.11	2%
		Adaboost	0.11	3%
		RF	0.03	2%
		CART	0.16	3%
		LDA	0.09	5%

We conducted our experiment with a software bug prediction dataset to classify the data as buggy or not-buggy. The result was evaluated using classification accuracy. Table 5 and Table 6 describe the accuracy while Table 7 and Table 8 describe the AUC performance of the software bug prediction, created using machine learning classifiers such as SVM-RBF, KNN (Minkowski metric), KNN (Euclidean metric), NB, DT (Cart), Discriminate Analysis (linear Discriminate analysis), Random forest (RF) and adaptive boosting (AdaBoost) in conjunction with the optimized Artificial Immune network (opt-aiNet). The methods continued to assess according to their relationship to accuracy. As anticipated, the approach proposed by us produced satisfactory results for all five datasets. The opt-aiNet converged at generation 50 according to the proposed stopping criteria.

In Tables 5 and 6, it can be seen that for hyper-parameters the optimization prediction accuracy of machine learning classifiers is significantly improved for almost all the projects. To compute the difference in the accuracy of each classifier, we compared the accuracy of the classifier using default values for the hyper-parameters and optimized hyper-parameters shown in Table 5 and Table 6. From the results mentioned in this table it can be stated that in SVM-RBF and LDT the 4-6% accuracy is improved on the EclipseJDTCore dataset (SVM), the EclipsePDEUI (SVM), the Lucene (LDA) and the Mylyn (LDA) which had the highest achieved accuracy, while on the rest of the classifiers KNN (Minkowski metric), KNN (Euclidean metric), DT, NB, Random forest (RF) and adaptive boosting (AdaBoost) tuning the hyper-parameters did not improve prediction accuracy so much but in fact only 0.34 - 3% for all projects. In LDA the accuracy is consistent on the Equinox dataset.

From Tables 7 and 8, it can be seen that for hyper-parameters optimization of machine learning classifiers using opt-aiNet the AUC performance is significantly improved for all projects. To estimate the impact of each classifier hyper-parameter, we compared the AUC performance of each classifier using default hyper-parameter values and the optimized hyper-parameter values given in Table 7 and Table 8. From the results given in the above tables, it can be seen that

with SVM-RBF the AUC performance is improved by up to 41% on EclipseJDTCore while in other classifiers the AUC performance is also improved from 1-31% but it remains less than 41%.

Overall, accuracy and AUC performance is improved by hyper-parameter optimization of machine learning classifiers using opt-aiNet rather than using default parameter values.

VII. COMPARISON WITH PREVIOUS TECHNIQUES

In this section, we compare our results with the previous studies according to the optimization methods, classifiers and hyper-parameters used for optimization and what the results were and are thus able to discover if our study is an improvement on the existing ones or not. As in papers [2] and [12], the 30 classification algorithm parameters were tuned, in paper [3] and [34] only 2 classification algorithms parameters were tuned and in paper [13] SVM-RBF parameters were optimized using AIS (clonal selection) and so its results are also comparable with our results due to classifiers and optimization methods used. Hence the results gratifyingly showed that our technique was indeed an improvement on the existing ones. The parameters tuned in papers [2, 12] are a little bit similar to ours; otherwise, the majority of our hyper-parameters were different from those previously used. The details of the comparison are listed in Table 9.

VIII. CONCLUSION AND SUGGESTION FOR FUTURE WORK

Machine learning classifiers that can identify buggy software modules in software Bug Prediction have configurable parameters that control their features but most of the time these classifiers sometimes not perform well when default settings are used. Very few researchers have studied the effect of hyper-parameter optimization on software bug prediction. The aim of this paper was therefore to examine the consequences of the hyper-parameter optimization of machine learning classifiers using opt-aiNet for software bug prediction to boost the prediction accuracy. The accuracy achieved by Machine learning classifiers will be compared before and after hyper-parameter optimization on five open-source datasets. For this purpose, some machine learning classifiers such as SVM- RBF, KNN (Minkowski metric), KNN (Euclidean metric), NB, DT (Cart), Discriminate Analysis (linear Discriminate analysis), Bagging (random forest) and adaptive boosting (AdaBoost) were used in conjunction with opt-aiNet. In this case, the results showed that for SVM-RBF and LDT optimizing the hyper-parameters increases the prediction accuracy by up to 5% for all projects. However, when KNN (Minkowski metric), KNN (Euclidean metric), DT, NB, Random forest (RF) and adaptive boosting (AdaBoost) were used, tuning hyper-parameters did not improve the prediction accuracy that much, in fact only 0.34 - 3% for all projects. In LDA the accuracy is consistent on the Equinox dataset. It must also be mentioned that in SVM-RBF the AUC performance was improved by up to 41 percentage points on Eclipse JDT Core while with other classifiers the AUC performance was also improved from 1-31%. The comparison results show that the hyper-parameter optimization of machine learning classifiers using opt-aiNet for software bug prediction has improved not only software bug prediction accuracy but also AUC performance. From the results, it can be concluded that hyper-parameter optimization has dissimilar effects on other machine learning models having the same datasets and that hyper-parameter optimization should be conducted so that prediction accuracy can be improved.

In a forthcoming study, we plan to widen this study with further machine learning classifiers, such as partial least square, neural networks and the rule-based classifier and also with more hyper-parameter optimization techniques, such as genetic algorithm (GA), Grid search, Differential evolution (DE) and Evolution strategy so that we can then have an even clearer picture of how much machine learning classifiers are sensitive to hyper-parameter optimization in software bug prediction. Hyper-parameter optimization of machine learning classifiers for software bug prediction should be addressed through deep learning. Class imbalance is nowadays the most common classification problem in machine learning. It is important to solve or at-least adjust this class imbalance problem and in future resampling techniques will be used.

REFERENCES

[1] R. S. Wahono, "A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks," *J. Softw. Eng.*, vol. 1, no. 1, pp. 1–16, 2018.

[2] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 45, no. 7, pp. 683–771, Jul. 2019.

[3] H. Osman, M. Ghafari, and O. Nierstrasz, "Hyperparameter optimization to improve bug prediction accuracy," in *Proc. IEEE Workshop Mach. Learn. Techn. Softw. Qual. Eval. (MaLTeSQuE)*, Feb. 2017, pp. 33–38.

[4] H. N. Agiza, A. E. Hassan, and A. M. Salah, "An improved version of opt-AiNet algorithm (I-opt-AiNet) for function optimization," *Int. J. Comput. Sci. Netw. Secur.*, vol. 11, no. 3, pp. 80–85, 2011.

[5] J. Nam, "Survey on software defect prediction," Dept. Comput. Sci. Eng., Hong Kong Univ. Sci. Technol., Hong Kong, Tech. Rep., 2014.

[6] M. Praveena and V. Jaiganesh, "A literature review on supervised machine learning algorithms and boosting process," *Int. J. Comput. Appl.*, vol. 169, no. 8, pp. 32–35, 2017.

[7] I. K. M. Aydin and E. Akin, "A multi-objective artificial immune algorithm for parameter optimization in support vector machine," *Appl. Soft Comput.*, vol. 11, no. 1, pp. 120–129, 2011.

[8] J. Brownlee. (2011). *Clever Algorithms: Nature-Inspired Programming Recipes*. [Online]. Available: https://Lulu.com

[9] L. N. De Castro and F. J. Von Zuben, "Artificial immune systems: Part I—Basic theory and applications," Univ. Estadual Campinas, Campinas, Brazil, Tech. Rep., Dec. 1999, vol. 210, no. 1.

[10] L. N. De Casto and F. J. Von Zuben, "An evolutionary immune network for data clustering," in *Proc. 6th Brazilian Symp. Neural Netw.*, vol. 1, Nov. 2000, pp. 84–89.

[11] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, vol. 1, May 2010, pp. 31–41.

[12] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, May 2016, pp. 321–332.

[13] I. K. M. Aydin and E. Akin, "A multi-objective artificial immune algorithm for parameter optimization in support vector machine," *Appl. Soft Comput.*, vol. 11, no. 1, pp. 120–129, 2011.

[14] F. Sarro, M. S. Di, F. Ferrucci, and C. Gravino, "A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction," in *Proc. 27th Annu. ACM Symp. Appl. Comput.*, Mar. 2012, pp. 1215–1220.
[15] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker, "Efficient hyperparameter optimization for deep learning algorithms using deterministic RBF surrogates," in *Proc. 31st AAAI Conf. Artif. Intell.*, Feb. 2017, pp. 822–829.
[16] F. S. Fazel, "A new method to predict the software fault using improved genetic algorithm," *Bull. la Société Royale des Sci. Liège*, vol. 85, pp. 187–202, 2016.
[17] T. P. Pushpavathi, V. Suma, and V. Ramaswamy, "Defect prediction in software projects-using genetic algorithm based fuzzy C-means clustering and random forest classifier," *Int. J. Sci. Eng. Res.*, vol. 5, no. 9, pp. 888–898 2014.
[18] R. S. Wahono, "A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks," *J. Softw. Eng.*, vol. 1, no. 1, pp. 1–16, 2015.
[19] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *Proc. Int. Conf. New Trends Comput. Sci. (ICTCS)*, Oct. 2017, pp. 252–257.
[20] C. Tantithamthavorn, "Towards a better understanding of the impact of experimental components on defect prediction modelling," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2016, pp. 867–870.
[21] S. R. Young, D. C. Rose, T. P. Karnowski, S. H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proc. Workshop Mach. Learn. High-Perform. Comput. Environ.*, Nov. 2015, p. 4.
[22] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur.*, Aug. 2015, pp. 17–26.
[23] J. Murillo-Morera, C. Castro-Herrera, J. Arroyo, and R. Fuentes-Fernández, "An automated defect prediction framework using genetic algorithms: A validation of empirical studies," *Inteligencia Artif.*, vol. 19, no. 57, pp. 114–137, 2016.
[24] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, May 2016, pp. 297–308.
[25] Y. Tang, "Deep learning using linear support vector machines," 2013, *arXiv:1306.0239*. [Online]. Available: https://arxiv.org/abs/1306.0239
[26] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 49–60.
[27] S. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming," *Procedia Comput. Sci.*, vol. 62, pp. 303–311, Jan. 2015.
[28] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, Jun. 2015, pp. 3460–3468.
[29] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 31–41.
[30] M. Shepperd, D. Bowes, and T. Hall, "Researcher bias: The use of machine learning in software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 603–616, Jun. 2014.
[31] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Inf. Softw. Technol.*, vol. 76, pp. 135–146, Aug. 2016.
[32] L. N. De Castro and J. Timmis, "An artificial immune network for multimodal function optimization," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 1, May 2002, pp. 699–704.
[33] J. Chakraborty, T. Xia, F. M. Fahid, and T. Menzies, "Software engineering for fairness: A case study with hyperparameter optimization," 2019, *arXiv:1905.05786*. [Online]. Available: https://arxiv.org/abs/1905.05786
[34] M. M. Öztürk, "Comparing hyperparameter optimization in cross-and within-project defect prediction: A case study," *Arabian J. Sci. Eng.*, vol. 44, no. 4, pp. 3515–3530, Apr. 2019.
[35] L. H. Son, N. Pritam, M. Khari, R. Kumar, P. T. M. Phuong, and P. H. Thong, "Empirical study of software defect prediction: A systematic mapping," *Symmetry*, vol. 11, no. 2, p. 212, 2019.
[36] S. K. Pandey, R. B. Mishra, and A. K. Triphathi, "Software bug prediction prototype using Bayesian network classifier: A comprehensive model," *Procedia Comput. Sci.*, vol. 132, pp. 1412–1421, 2018.
[37] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 2, pp. 78–83, 2018.
[38] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Comput.*, vol. 22, no. 1, pp. 77–88, 2019.
[39] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Comput.*, vol. 22, no. 4, pp. 9847–9863, 2019.
[40] S. D. Immaculate, M. F. Begam, and M. Floramary, "Software bug prediction using supervised machine learning algorithms," in *Proc. Int. Conf. Data Sci. Commun. (IconDSC)*, Mar. 2019, pp. 1–7.
[41] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques," *Expert Syst. Appl.*, vol. 144, Apr. 2020, Art. no. 113085.
[42] R. Zhang, T. Li, X. Xiao, and Y. Shi, "A danger-theory-based immune network optimization algorithm," *Sci. World J.*, vol. 2013, pp. 1–13, Dec. 2012, Art. no. 810320.
[43] L. A. F. Gomes, S. T. R. da Silva Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Inf. Softw. Technol.*, vol. 115, pp. 58–78, Nov. 2019.

FAIZA KHAN received the master's degree in software engineering from Riphah International University, Islamabad, in September 2019. Her research interests include machine learning, evolutionary computation, and deep learning.

SUMMRINA KANWAL received the Ph.D. degree from the University of Stirling. She is currently working as a Visiting Postdoctoral Fellow with the Cognitive Big Data and Cybersecurity Lab (CogBID) Lab, Napier University. She is also working as an Assistant Professor with the School of Computing and Informatics, Saudi Electronic University, Saudi Arabia.

SULTAN ALAMRI received the master's degree in information technology from the School of Engineering and Mathematical Science, La Trobe University, Australia, in 2010, and the Ph.D. degree from the Clayton School of Information Technology, Monash University, Australia, in 2014. He is currently an Associate Professor with the School of Computing and Informatics, Saudi Electronic University, Saudi Arabia.

BUSHRA MUMTAZ received the master's degree in software engineering from Riphah International University, Islamabad, in September 2019. Her research interest includes artificial intelligence.