e-Informatica Software Engineering Journal, Volume 18, Issue 1, 2024, pages: 240103, DOI: 10.37190/e-Inf240103

Software Defect Prediction Using Non-Dominated Sorting Genetic Algorithm and *k*-Nearest Neighbour Classifier

Mohammad Azzeh*, Ali Bou Nassif**, Manar Abu Talib***, Hajra Iqbal****

*Department of Data Science, Princess Sumaya University for Technology, Jordan
**Department of Computer Engineering, University of Sharjah, UAE
***Department of Computer Science, University of Sharjah, UAE
****Department of Computer Engineering, University of Sharjah, UAE

m.azzeh@psut.edu.jo, anassif@sharjah.ac.ae, mtalib@sharjah.ac.ae, u16107036@sharjah.ac.ae

Abstract

Background: Software Defect Prediction (SDP) is a vital step in software development. SDP aims to identify the most likely defect-prone modules before starting the testing phase, and it helps assign resources and reduces the cost of testing.
Aim: Although many machine learning algorithms have been used to classify software modules based on static code metrics, the *k*-Nearest Neighbors (*k*NN) method does not greatly improve defect prediction because it requires careful set-up of multiple configuration parameters before it can be used. To address this issue, we used the Non-dominated Sorting Genetic Algorithm (NSGA-II) to optimize the parameters in the *k*NN classifier with favor to improve SDP accuracy. We used NSGA-II because the existing accuracy metrics often behave differently, making an opposite judgment in evaluating SDP models. This means that changing one parameter might improve one accuracy measure while it decreases the others.
Method: The proposed NSGAII-*k*NN model was evaluated against the classical *k*NN model and state-of-the-art machine learning algorithms such as Support Vector Machine (SVM), Naïve Bayes (NB), and Random Forest (RF) classifiers.
Results: Results indicate that the GA-optimized *k*NN model yields a higher Matthews Coefficient Correlation (*MCC*) and higher balanced accuracy based on ten datasets.
Conclusion: The paper concludes that integrating GA with *k*NN improved defect prediction when applied to large or small or large datasets.

Keywords: software defect prediction, genetic algorithm, multi-objective optimization, *k*-nearest neighbor

1. Introduction

Software systems are integral to daily life [1–4]. As such, the quality of our software systems must be monitored for defects in order to produce perfectly running, defect-free systems and limit software failures [3, 5–7]. A software defect is simply defined as an error, fault, bug, or problem in a computer program [8]. Software defects are caused by errors made during the implementation phase and can lead to software failures when the software

Submitted: 30 Apr. 2023; Revised: 28 Sep. 2023; Accepted: 18 Oct. 2023; Available online: 25 Oct. 2023

program is executed [9, 10]. Studies have shown that such defects lead to decreased customer satisfaction and increased maintenance costs [11]. Therefore, it is essential to locate software defects and identify which modules require repair or retesting before running a program application. Software Detection Prediction (SDP) is the process of predicting defect-prone modules before starting the testing phase [12]. As software complexity increases, the use of SDP in the software development process has become even more crucial. The necessity of having proper SDP models results in improved efficiency, reduced development time, and reducing time spent on testing and error management [4, 5, 13]. Many SDP models were built in the last decades using several different datasets. The most commonly used publicly accessible dataset is the NASA repository, produced in 2005 [14]. Researchers have used the NASA repository with many machine learning models with promising results [15–19]. The most widely used machine learning algorithms include Naïve Bayes (NB), Neural Networks (NN), k-Nearest Neighbor (kNN), Support Vector Machine (SVM), decision trees, etc. [20, 21].

Our focus in this paper is on kNN algorithm because it has less attention form researchers, and it did not offer significant improvements on the prediction accuracy. One reason for that is the large space of configuration possibilities that govern the execution of kNN which includes 1) choosing distance measure, 2) optimal features sets, 3) optimal feature weights, and most notably, 4) the number of nearest neighbors. This resulted in a massive number of configuration possibilities that extend 100,000 possible solutions. Therefore, searching for the best subset of kNN configuration parameters is relatively impossible unless a more robust searching algorithm is used.

This paper applied Non-dominated Sorting Genetic Algorithm (NSGAII) optimization technique to search for the best kNN solution that fits training data [22]. This algorithm finds the best solution amongst a vast space of solution sets or a primary population where an individual is referred to as a chromosome [23], inspired by Darwin's theory of "survival of the fittest." The best individuals have the best probability of being reproduced in the next generation [24]. GA has been widely used to solve complex problems in computing and engineering fields [25]. The GA process consists of four main phases: 1) initialization, 2) selection, 3) crossover, and 4) mutation. In each cycle of GA construction, the chromosomes with high fitness are most likely reproduced to generate new generations. With the help of crossover and mutation, the GA can make a minor modification to the most fitted individuals to generate hopefully better solutions [26, 27].

However, the existing accuracy metrics often present opposite judgments in evaluating SDP models [19, 28]. This means that changing one parameter could improve one accuracy metric while reducing the other accuracy metrics. Therefore, we used a Pareto front multiobjective NSGAII to come up with an optimal solution that improves the overall accuracy of SDP-based kNN without reducing the other accuracy metrics. The effect of these conflicting parameters together has not been examined before. Thus, we propose to see the construction of kNN as a multiobjective optimization problem. This research is driven by the following research questions:

- **RQ1.** Do the use of NSGAII and the proposed solution vector improve accuracy of kNN model for software defect prediction problem?
- **RQ2.** Is there sufficient evidence that the NSGAII algorithm can find the best k value for each module?

To answer these questions, we integrated the kNN algorithm with NSGAII. We proposed a new solution vector that combines two main kNN variables that must be optimized. The first variable is an integer that represents best k value and the second one is a binary

vector of length m, where m is the number of features. Furthermore, since the evaluation measure cannot give indication to the superiority of the generated configuration, we used mutliobjective optimization based on two evaluation measures that are less vulnerable to imbalanced data, namely, Matthew's Correlation Coefficient (MCC) and Balanced Accuracy (BA). In general, the proposed model produced good performance when compared to baseline kNN and other machine learning algorithms.

The rest of the paper is structured as follows. The first section introduced the research and the subject matter of the paper, an overview of the literature available on software defect prediction, the algorithms we tested, and the results of significance obtained from these research papers. Section 2 presents related work. Section 3 presents a problem presentation. Section 4 presents multiobjective optimization algorithms. The research methodology is presented in Section 5. The obtained results are covered in Section 6. Section 7 introduced threats to validity. Section 8 provides a summarized conclusion to the efforts of this study and the future works, along with limitations.

2. Related work

Many machine-learning models have been used to predict the defects in software systems. Shuai et al. [29, 30] Investigated the use of GA with an SVM classifier and particle swarm algorithm for software fault prediction. Another study investigated a novel dynamic SVM method based on improved Cost-Sensitive SVM (CSSVM) optimized by GA [29]. Results demonstrated a high Area Under the Curve (AUC) value for the GA-CSSVM model compared with only SVM or CSSVM. Recorded results were 0.721 with the KC1 dataset, 0.832 with PC1, and 0.897 with the MC1 dataset. The proposed method resulted in an F1 score of 94.88% with the CM1 dataset, 91.89% with KC1, 94.90 with KC3, 99.7% with MC1, and 95.78% with the MW1 dataset. Elish et al. [31] compared the effectiveness of SVM prediction to that of eight machine learning models in predicting defect-prone software modules. Using the PC1, KC1, and KC3 datasets, the findings showed that SVM outperformed other machine learning models. Hammad et al. [32] used the kNN machine learning algorithm to predict faulty software projects. They used public datasets and four different similarity measures, achieving an accuracy of 87.2% using the Euclidean distance measure. The researchers [7, 11, 33] studied the detailed performance analysis of various machine learning classification techniques using publicly available NASA datasets. They found that evaluation metrics like ROC and accuracy are ineffective performance measures because they do not react to class imbalance. In contrast, precision, recall, f-measure, and MCC react to class imbalance. Moreover, Khoshgoftaar et al. [34] worked on a pre-existing model to enhance the performance of Random Forest by applying feature selection to detect software defects. They achieved high accuracies with PC2, PC2, PC3, and PC4 datasets.

Mabayoje et al. [35] focused on understanding the effect of kNN tuning on the abilities of the SDP based on 6 datasets. The result of significance was that the k value should be greater than 1 and the distance weighting improved the predictive performance of the SDP by 8.82%. Iqbal et al. [36] used Use of a feature selection-based ensemble classification framework, divided over four stages, with the framework implemented with and without feature selection. They Found that the framework proposed in the paper could perform better in comparison to other popular classification techniques including kNN, NB, MLP and OneR among others. One issue that the study faced was with regards to imbalance; a solution was not presented in the paper. Jindal et al. [37] proposed a Solution for defect severity

assessment using text mining and machine learning. The results showed that the kNN technique could predict defects across all severity levels, with the performance improving every time the number of corresponding words was increased. The study concluded that the kNN method is best used for defects of medium severity. Ulumi et al. [38] focused on subject of SDP for cross-project domain. The study made use of kNN to fill in missing values from a dataset, followed by classification using the NB or RF methods. Goyal et al. [39] used the kNN regression as opposed to Ordinary Least Square (OLS) parametric regression for defect prediction. They found the kNN regression remains unaffected with increasing number of predictors and provide better performance when using linear regression.

Based on existing research, it appears that using the kNN method in conjunction with other techniques, such as machine learning, can help develop models capable of conducting SDP with higher levels of accuracy and reliability. In addition to SDP for single projects, kNN presents an exciting front for cross-project defect prediction. Without using kNN, this can be a challenging task since the datasets across domains contain many features. An issue that is also faced is data imbalance, which is a significant issue in SDP; one possible solution is the use of resampling techniques, which enhance the accuracy of machine learning classifiers, including kNN.

Above all, there is little research on the kNN efficacy in predicting defect-prone software systems, and it does not offer significant improvements in prediction accuracy. This is due to the multiple configuration possibilities that govern kNN, including 1) choosing distance measures, 2) optimal feature sets, 3) optimal feature weights, and most importantly, 4) the number of nearest neighbors [35, 40]. Because these configuration possibilities result in 100 000 possible solutions, a more robust searching algorithm is needed to identify the best subset of kNN configuration parameters. One possible solution to this problem is to apply the Multi-Objectives Genetic Algorithm (MOGA) optimization technique to search for the best kNN solution that fits training data. This study hypothesized that optimizing kNN with GA would create a more robust algorithm for SDP.

3. Objective functions

Objective function is important termination factor that tells MOGA to stop iteration before reaching maximum number of iterations. Usually, the classical GA uses one objective function that tell us we found the best solution for a problem. In case of SDP studies, the objective function is frequently one of the accuracy measures. However, there are multiple accuracy measures that are used evaluate SDP models where some of them are less informative for the model performance. Since these accuracy measures (a.k.a. objective functions) behave differently we prefer to use multiple objective function functions. The second important question is which reliable accuracy metrics should be used as objective functions. After careful investigation we found that almost all SDP datasets suffer from imbalanced data distribution where number of non-defected modules outnumbers the defected modules which lead to prediction bias towards majority class which is non-defected modules. Therefore, we searched in the literature for the best evaluation metrics that are considered reliable and work well with the existent of imbalanced data. We found that Matthews Correlation Coefficient (MCC) is the most credible metrics for imbalanced datasets [19, 41]. The MCC as shown in Eq. (1) uses binary confusion matrix values (i.e., True Positive (tp), True Negative (tn), False Positive (fp) and False Negative (fn)) and returns a value between -1 and $+1$ corresponding to the relationship between predicted

output labels and actual output labels. Any value closes to +1 means strong positive correlation, any value closes to −1 means strong negative correlation and value around zero means no correlation.

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp+fn)(tp+fp)(tn+fp)(tn+fn)}} \tag{1}$$

The second objective function is the Balanced Accuracy ($mathitBA$) as shown in Eq. (2). This measure is considered less important than MCC when data is imbalanced, but it is still useful because it computes the average of compromises between Sensitivity and Specificity. This metric is less sensitive to bias than other accuracy metrics.

$$BA = \frac{\frac{tp}{tp+fn} + \frac{tn}{tn+fp}}{2} \tag{2}$$

The Area Under Precision-Recall Curve is the third objective function ($AUPRC$). This measure is thought to be more stable and less susceptible to data imbalances than Area Under ROC [19]. $AUPRC$ is a binary response evaluation metrics statistic that is acceptable for unbalanced data and is independent of model specificity. In other fields, precision-recall curves have been recognized as useful for assessing classification performance for unbalanced binary responses; its tolerance of skewed data (e.g., many absences and few presences) makes it well suited for quantifying distribution model performance for minority cases. All objective functions are supposed to be maximized. These objective functions have been selected because, even though all of them were initially designed to show the performance of a model, they can behave in a different way as mentioned in [6, 42, 43]. This enables us to choose as many as possible good solutions that can produce tradeoff between these objective functions.

4. Multiobjective genetic algorithms

4.1. Basic concepts

Evolutionary algorithm is widely used to solve complex problems with interrelated parameters as encountered in computing. The problem of searching can be defined as follows: Given a function $f : S \to \Re$ from some set of decision vectors (S) to the set of real numbers ($\Re$), the aim is to find a solution $\vec{s_o}$ in S such that the objective function is either minimized ($f(\vec{s_o}) \leq f(\vec{s}), \forall s \in S$) or maximized ($f(\vec{s_o}) \geq f(\vec{s}), \forall s \in S$), where each solution $\vec{s}$ is defined as vector of variables in the d-dimensional space as shown in Eq. (3).

$$\vec{s} = [s_1, s_2, \ldots, s_d]^T \tag{3}$$

For single objective problem, finding the optimal solution is straight forward. But, when the problem should be optimized by multiple objective functions. We arrive at the conclusion that there is no one optimum solution, but rather a good trade-off solution that represents the best compromises between the objectives. Since there are several linked decisions that need to be optimized based on discovering trade-offs between different accuracy metrics, the challenge of tuning kNN process can be considered as a multiobjective optimization problem. The GA has been chosen for three reasons: 1) widely used among research community,

2) simple to implement and 3) it showed astonished accuracy against other evolutionary algorithms such as Particle Swarm Optimization and Simulated Annealing.

4.2. Genetic algorithm

GA is an evolutionary algorithm based on natural selection. The GA algorithm simulates natural selection, in which the fittest chromosomes are picked for reproduction in order to generate the next generation. A particular number of solutions are chosen initially to run the problem. Some of these solutions (fittest ones) are combined, resulting in new solutions including pieces (genes) from the generating pair (crossover). A new generation is defined when the components of other solutions are randomly transformed (mutation). The goal of all of these procedures is to identify the individuals who perform the best in terms of the objective function. The current best solutions are always moved to the next generation to ensure that the algorithm progresses toward improving the objective functions. A careful selection of all of these parameters, with the conventional technique being to test various combinations of their values. The size and makeup of the original population, as well as the number of generations to compute, are also essential assumptions. The last parameter of the algorithm can be securely examined by observing the evolution of the objective function over generations and terminating the algorithm when no improvement is obtained after a specific number of iterations. Finally, because the majority of the above mechanisms are influenced by random factors, it is common practice to repeat the entire operation for each fixed set of parameter values. The crossover operator randomly recombines specific parts from two selected solutions and creates a new solution (chromosome) for the new population. The mutation operator picks out a point in parent solutions and generates a new random solution to replace the previously selected solution. In contrast, the reproduction operator propagates the selected solution to the new population [44]. The process repeats until the maximum iteration set has been reached or the objective function has been met [44].

4.3. Chromosome construction

This section describes the proposed chromosome that is used with NSGAII to improve kNN model for defect prediction. Usually, the kNN model can accept two decision variables: 1) feature weights (w), and 2) number of nearest analogies (k). The first decision variable determines the importance of each feature in the training data set during distance calculation as shown in Eq. (4). The kNN classifier is defined as retrieving by similarity [42, 45]. The Euclidean distance is usually used to determine the closest observations. The value of k determines the number of nearest neighbors that will be used to make final prediction. Figure 1 illustrates the effect of k variable on the final prediction. Usually choosing small k value will result in ignoring other useful instances, whereas choosing large k value will results in potential misclassification. Therefore, the good k value is the one that can identify the actual nearest instances without degrading the accuracy.

The use of kNN is a practical application for SDP applications, given its ability to predict defects across all severity levels. The kNN method is used in addition to the other techniques, including practices such as the Euclidean distance measure to improve the accuracy of fault prediction.

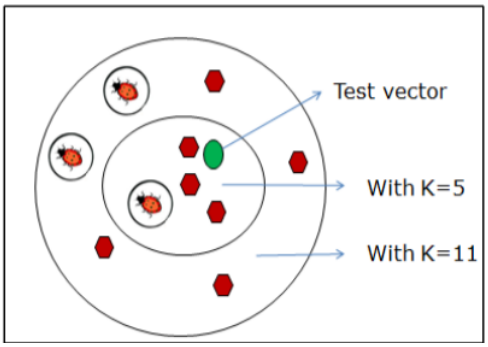

Figure 1. kNN classification example [46]

$$\text{distance} = \sqrt{\sum_{u=1}^{m} w_u \left(x_{iu} - x_{ju}\right)^2} \tag{4}$$

where m is the number of features, x_i and x_j are instances under investigation, u is the index of the feature and w_u is the weight of feature u that is identified by MOGA algorithm.

The second decision variable determines how many nearest neighbors should be retrieved from training data sets for which the final prediction will be made from them using voting technique. Each feasible solution ($\overrightarrow{s}$) in the search space is represented as a vector of two choice variables, as indicated in Eq. (5).

$$\overrightarrow{s} = \langle k, w \rangle \tag{5}$$

where k is the number of nearest neighbors which must be bounded by 1 and $n/2$ (i.e., $k \in [1, n/2]$) where n is the number of training instances, w is a numeric vector whose coordinate represent the weight in addition to the presence or absence of feature. If the any value in the w is zero, then that feature is not important. Each possible weight can take value between zero and one (i.e., $w_{ij} \in [0, 1]$) and the summation of weight values should equal to 1 as shown in Eq. (6). Finally, the dimension of w should be equivalent to number of input features in the training dataset.

$$\sum_{i=1}^{m} w_i = 1 \tag{6}$$

To illustrate that, assume the following solution vector $\overrightarrow{s} = \langle 5, 0.2, 0.4, 0.1, 0.15, 0.05, 0.1 \rangle$ are the best identified solution and assume the number of features is 6 ($m = 6$). This solution vector shows that only 5 nearest analogies should be retrieved, and the remaining values after 5 (i.e., from 0.2 to 0.1) represent the weight of each feature. It is important to note that all features should be included in distance computation because all of weights are above zero. As mentioned earlier, this solution vector is generated from NSGAII as explained in the methodology section.

4.4. Multiobjective genetic algorithm

The multiobjective optimization is defined as searching for a solution vector ($\vec{s}$) that meet d inequality constraints ($g_i(x) \geq 0, i = 1, 2, \ldots, d.$) and p equality constraints ($h_i(x) = 0, i = 1, 2, \ldots, p.$) while simultaneously optimizing a vector of M conflict objective

functions as indicated in Eq. (7). The constraints determine the feasible zone, which includes all viable options. As depicted in Figure 2, the ideal solution is reached as a trade-off between two or more competing objectives, which is known as a Pareto optimal solution. Here are some key terms to remember when it comes to multiobjective optimization:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_M(\vec{x})] \tag{7}$$

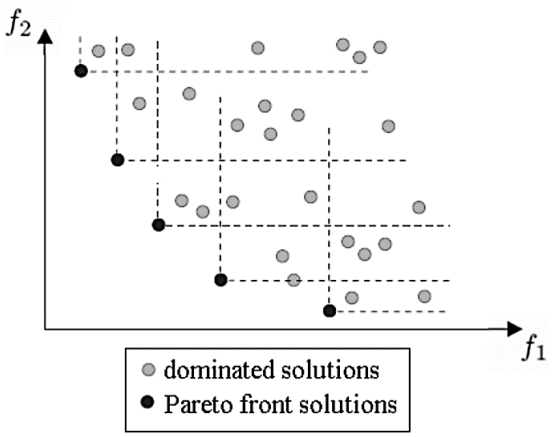

Figure 2. The Pareto front of a set of solutions in a two-objectives space (f_1 and f_2 are supposed to be minimized)

- Definition 1 (Dominance): A solutionx $\vec{x_i} \in \Re^m$ is strictly dominated by a solution $\vec{x_j} \in \Re^m$ ($\vec{x_i} \prec \vec{x_j}$ and $\vec{x_i} \neq \vec{x_j}$) iff $f_l(\vec{x_i}) \leq f_l(\vec{x_j}), \forall l \in 1, 2, \ldots, M$ and $f_l(\vec{x_i}) < f_l(\vec{x_j}), \exists l \in 1, 2, \ldots, M$.
- Definition 2 (Non-dominance): The solution $\vec{x_i} \in \Re^m$ is non-dominated solution, if there does not exist another solution $\vec{x_j} \in \Re^m$ such that $f_l(\vec{x_i}) \leq f_l(\vec{x_j}), \forall l \in 1, 2, \ldots, M$ and $i \neq j$.
- Definition 3 (Pareto optimal): We say that a solution $\vec{x}^* \in \eta \subset \Re^m$ is a Pareto optimal if it is non-dominated with respect to the feasible region (η).
- Definition 4 (Pareto optimal set): a set $\rho \subset X$ non-dominated solutions is called Pareto Optimal set which is formally defined as: $\rho = \{\vec{x} \in \eta \mid \vec{x}\}$ is a Pareto optimal set.
- Definition 5 (Pareto front): is defined as $\rho f = \{f(\vec{x}) \in \Re^m \mid \vec{x} \in \eta\}$.

In this paper, we used a modified version of the GA method called NSGAII [22], which supports Pareto-front optimization. This algorithm employs a strong elitism technique to provide a number of Pareto fronts solutions, taking non-dominance into account as illustrated in Figure 3. All non-dominated solutions make up the initial front solution. The second one has the solutions dominated by only one solution, and the fronts are generated until all solutions are classified. To maintain the diversity of solutions, another sort is performed using the crowding distance for solutions of the same front. The crowding distance defines how distant a solution's neighbors are from it, where the solutions are ranked in decreasing order. The selection operator employs both front and crowding distance sorting processes. Individuals with lower fronts are picked in the binary tournament; if the fronts are the same, the solution with the greatest crowding distance is chosen. Recombination and mutation are used to create new populations.

```
Procedure NSGA-II
  Input: N', g, f_k(X) ▷ N' members evolved g generations to solve f_k(X)
Initialize Population ℙ';
Generate random population - size N';
Evaluate Objectives Values;
Assign Rank (level) based on Pareto - sort;
Generate Child Population;
  Binary Tournament Selection;
  Recombination and Mutation;
for i = 1 to g do
  for each Parent and Child in Population do
      Assign Rank (level) based on Pareto - sort;
      Generate sets of nondominated solutions;
      Determine Crowding distance;
      Loop (inside) by adding solutions to next generation starting from
      the first front until N' individuals;
  end
  Select points on the lower front with high crowding distance;
  Create next generation;
    Binary Tournament Selection;
    Recombination and Mutation;
end
```

Figure 3. NSGAII algorithm pseudo code [22]

5. Methodology

5.1. Using the solutions produced by NSGAII in *k*NN

Figure 4 illustrates the research methodology of our study. We used repeated 10 folds cross validation to validate our model. In order to show how the NSGAII algorithm works with kNN we first start with describing the process of initialization as shown in Figure 5. Each vector represents a potential solution which is composed of two variables – k and w. Each solution is randomly initialized such that the value of k can take random integer number between 1 and $n/2$ and w can randomly take value between 0 and 1.

```
M ← 10      no. of repetition
N ← 10      no. of folds
dataset ← {CM1, MC1, AR3, PC4, KC1, KC3, Safe, Poi − 1.5, Ant − 1.3, Redktor}
foreach data ∈ dataset do
  foreach times ∈ [1, M] do
    data' ← randomize instances order
    binData ← generate N bins from data'
    foreach fold ∈ [1, N] do
      testData ← binData[fold]
      trainingData ← binData − testData
      bestSolution ← NSGAII(trainingData)
      model ← KNN(trainingData, bestSolution)
      accuracy ← evaluate(model, testData)
    end
  end
end
```

Figure 4. Research methodology

```
S ⟵ []
for i: 1 To t do
  s[t].k ⟵ Rand(1, n/2)
  foreach feature j do
    s[t].w[j] ⟵ Rand(0, 1)
  end
  S.add(s)
end
```

Figure 5. The algorithm of population generation

When NSGAII starts execution, the k and w values are updated based on crossover and mutation. Figure 3 depicts how each solution in this study is updated using Pseudo code. Recent research publications show that the solution's values frequently surpass the search space's bounds. This is most likely to take place when a solution is far away from best solution. The conventional method is to truncate the location at the exceeded boundary in this iteration and reflect the values in the boundary in the following generation such that the solution moves away. It does, however, limit the size of the solution step, limiting additional solution divergence and allowing the solution to stay near to the bounds during the search process.

In this study, we used the following configuration parameters for GA: 1) The mutation probability was 0.05, and 2) cross over operation has been performed using single point cross over.

To test the accuracy of our proposed, we also implemented a model without GA. To do this, we ran the classical kNN model and recorded the evaluation metrics to verify that the model with optimization gave us good results. We used the kNN classifier in our experiment to check the accuracy of our proposed model with and without optimization. One additional benefit of using kNN is its ability to enhance its performance as the size of the dataset increases. A larger training dataset typically provides a more comprehensive representation of the underlying data distribution. This increased coverage helps the kNN algorithm make more accurate predictions by capturing a wider range of data patterns and reducing the influence of outliers. The kNN model has also a better chance of learning the underlying patterns and relationships within the data. This improved generalization ability enables the model to make more accurate predictions on unseen or test data. The kNN model is less likely to overfit the training data. Overfitting occurs when the model becomes too specialized to the training set and fails to generalize well to new data. Increasing the training dataset size helps alleviate this issue. The kNN determines the class of a new data point based on the majority vote of its kNN. When the training dataset is larger, the decision boundaries between different classes can become more refined, leading to better classification performance.

These features ensure that software of higher complexities can use the kNN method for SDP. It is also assuring that the methodology will consider all the different parameters and produce an accurate representation of the defect probability in the system. In addition to the kNN classifier, we also tested the code using SVM, NB, and RF classifiers to compare our results and prove that the optimized model gave the best results. We chose these classifiers because they have been used widely in SDP in combination with other algorithms to evaluate results.

5.2. Datasets

The employed datasets have been obtained from the PROMISE repository [5, 12], which consisted of six imbalanced datasets from the NASA Metrics Data Program (NASA MDP), one dataset from ReLink [47], and three from the Modular Open Robotics Platform for Hackers (MORPH) [48]. These datasets are used in many studies related to SDP due to similar data points and the respective fault data, which helped us assess NSGAII-kNN performance during the testing, training, and evaluating process. Table 1 lists and describes the datasets we used in this experiment. Before implementation, we checked the datasets for missing attributes, and we applied data normalization. Before running NSGAII-kNN,

we also normalized the input features using the Min-Max scaler to avoid errors in our results.

Table 1. Datasets description

Dataset	Source	No. of attributes	No. of instances	Defective instances
CM1	NASA	37	344	12
KC1	NASA	21	2107	15
KC3	NASA	40	458	9
MC1	NASA	38	9277	1
AR3	NASA	29	63	12
PC4	NASA	37	1399	12
Safe	ReLink	26	56	39
Poi- 1.5	MORPH	20	237	59
Ant- 1.3	MORPH	20	125	16
Redktor	MORPH	20	176	15

6. Results

This section focuses on the design and implementation of NSGAII-kNN.

To answer the research questions, we first installed required libraries and packages, followed by the dataset import. We set up GA parameters manually before running the experiment. The GA was used to find the best features weight and k number that produced a minimized objective function value. Therefore, we tuned the GA parameters to a set number of maximum iterations, creating a population size. The objective function class defined how we made the population and assigned weights. Then we split our data into training and testing sets using the 10 $\times$ 10-Folds cross-validation method. For the genetic algorithm, the objective function focused on minimizing the false-negative and maximizing the true positive. Before generating the optimized function, we normalized the input features using the Min-Max scaler and then applied the kNN classifiers to output results. Next, with the aid of this optimized function class, we passed on the genetic algorithm and found out the best solution that consisted of optimal weights and k numbers. We used the 10 $\times$ 10-Folds cross-validation technique, and the iterations were repeated many times for each dataset. Finally, we recorded the output results. We used *MCC* and balanced accuracy to conclude our results. In addition, we ran the overall model on datasets CM1, KC1, KC3, MC1, PC4, AR3, Safe, Ant- 1.3, Poi- 1.5, and Redktor. Along with the optimized model setup, we ran a classical kNN model, splitting the dataset into training and testing, then normalizing the features using the Min-Max scalar before fitting the model and predicting results. We tested the accuracy of this model using the same evaluation measures as we did for the optimized model. We also set up classical SVM, NB, and RF models to compare the results with GA-kNN. The configuration parameters of these machine learning alogrithms are presented in Table 2. The layout of the code was the same as the classical kNN model. We used the same evaluation metrics to record the results.

Tables 3 and 4 summarize the results. Table 3 summarizes the *MCC* values obtained with all the models, including the SVM, NB, and RF classifier models, while table 4 shows the balanced accuracy for all models. The tables show that the NSGAII-kNN classifier performed better than the other classifier models in both *MCC* and balanced accuracy. The

Table 2. Configuration parameters of the employed machine learning algorithms

Model	Best configuration parameters
kNN	$n\_neighbors$ = 5, weights='uniform', algorithm='ball_tree'
RF	n_estimator= 100, learning_rate=0.01, tree_method='hist'
NB	Kernel='Gaussian function'
SVM	Gamma='scale', kernel='rbf', epsilon=0.1

Table 3. *MCC* recorded for all models

Datasets	NSGAII kNN	Classica kNN	SVM	NB	RF
CM1	0.893	0.796	0.511	0.256	0.0963
KC1	0.974	0.950	0.967	0.433	0.379
KC3	0.326	0.198	0.056	0.278	0.0987
MC1	0.241	0.189	0.0	0.117	0.241
AR3	0.330	0.293	0.130	0.401	0.330
PC4	0.342	0.339	0.156	0.319	0.389
Safe	0.426	0.367	0.325	0.288	0.304
Poi- 1.5	0.405	0.367	0.386	0.184	0.532
Ant- 1.3	0.341	0.326	−0.009	0.234	0.0240
Redktor	0.466	0.447	0.383	0.099	0.370

Table 4. Balanced accuracy recorded for all models

Datasets	NSGAII kNN	Classical kNN	SVM	NB	RF
CM1	0.941	0.897	0.686	0.622	0.527
KC1	0.987	0.975	0.987	0.678	0.676
KC3	0.716	0.582	0.517	0.674	0.536
MC1	0.821	0.796	0.700	0.804	0.800
AR3	0.882	0.782	0.715	0.860	0.815
PC4	0.683	0.669	0.547	0.650	0.633
Safe	0.734	0.682	0.659	0.649	0.666
Poi- 1.5	0.718	0.687	0.683	0.574	0.767
Ant- 1.3	0.690	0.668	0.495	0.630	0.489
Redktor	0.771	0.723	0.719	0.510	0.722

highest *MCC* was scored with the KC1 dataset, while the highest balanced accuracy was also with the KC1 dataset. If we compare the classical kNN model with other classifiers, the kNN classifier proved to be better than the others in most of the datasets, proving to be a good classifier for SDP.

Tables 3 and 4 provide different combinations of parameters and conditions, which we used to obtain different results. NSGAII-kNN yielded the highest overall recorded *MCC* of 0.974 on the KC1 dataset, while NSGAII-kNN's lowest *MCC* value was .241 in MC1 dataset. By comparison, the classical kNN model's highest *MCC* was 0.796 in the CM1 dataset and an *MCC* of 0.189 in the MC1 dataset. Overall, the NSGAII-kNN model yielded *MCC* values ranging from 0.241 to 0.974 while the classical kNN model yielded *MCC* values ranging from 0.189 to 0.950. For balanced accuracy, again, NSGAII-kNN outperformed, yielding the highest value of 0.987. By comparison, the classical kNN model's highest value was 0.975. GA-kNN yielded balanced accuracy values between 0.650 and 0.987, while the classical kNN model yielded balanced accuracy values between 0.582 and 0.975. These results demonstrate that GA performed well on all datasets and that the NSGAII-kNN model

works better than the classical kNN model across the majority of datasets. When comparing NSGAII-kNN with the SVM, NB, and RF models, NSGAII-kNN performed better again. *MCC* and balanced accuracy remained high for GA-kNN in all the datasets and low in the SVM model, proving that NSGAII-kNN performs SDP better. The NB model also yielded poor results with *MCC* ranging from 0.099–0.433, a short-range compared to 0.241–0.974 for NSGAII-kNN. Balanced accuracy recorded with the SVM model also produced poor results with the minimum *MCC* value of 0.517 with the KC3 dataset compared to 0.650 with the optimized model. Lastly, the RF model's *MCC* range was relatively low compared to NSGAII-kNN, similar to the other classifier models. The balanced accuracy was also low, 0.489–0.815, compared to NSGAII-kNN's range of 0.650–0.975. These results demonstrate that overall, NSGAII-kNN performed better than SVM, NB, and RF.

The results obtained from these experiments used the same high-performance methods to define the accuracy of the results. Our results support previous findings that both kNN and NSGA-II are effective methodologies in SDP. Our experiments resulted in balanced accuracy and *MCC* values comparable to those found in the literature, supporting our hypothesis that using kNN with NSGAII is an effective method of SDP. Using NSGAII with kNN is a unique approach in terms of the algorithm that governs this experiment's classification methodology. Most of the literature review focused on the sole use of kNN, which has produced high-reliability results. However, our study demonstrates that the addition of GA enhances the accuracy and reliability of results and is a methodology superior to one solely reliant on kNN. Table 5 provides an analytical comparison of some of these literature reviews to our study. Many different methodologies and frameworks have been considered for application for SDP. The kNN classification method has been assessed and applied under various conditions and to other datasets, yielding results that display a high potential for application in SDP modules. The classical kNN model gave us good results when testing it as a classifier for a software defect prediction model with high accuracy values.

Finally, we revisited the proposed research questions to facilitate drawing the conclusions:

– **RQ1.** Do the use of NSGA-II and the proposed solution vector improve accuracy of kNN model for software defect prediction problem?
Answer. Yes, as we have seen from Tables 3 and 4, the proposed NSGA-II model can produce significant results over all results.

Table 5. Analytical comparison with the existing literature

Ref.	Used algorithms	Key findings in comparison to our study
[32]	Used the kNN algorithm, like our study, in addition to constructing using Euclidean distance, weighted ED, Manhattan distance, and Hausdorff distance measures.	This study achieved an accuracy of 87.2% with the same datasets as our research used. However, we used an optimized approach with GA and kNN and gained higher balanced accuracy, 94.1%, and 98.7%, with CM1 and KC1 datasets, respectively.
[49]	This study used variant-based ensemble learning and feature selection techniques.	KC1 dataset achieved an *MCC* of 0.482, very low in comparison to our study. We achieved an *MCC* of 0.974 with the KC1 dataset, proving to be a better algorithm for SDP.
[50]	The authors in this study used the kNN based probability density estimation approach using fuzzy memberships to eliminate classification errors.	The results were a balanced accuracy of 72.76% with CM1, 71.78% with KC1, and 64.27% with KC3. In comparison to these values, we obtained high-performance results that are 94.1%, 98.7%, and 65% with the CM1, KC1, and KC3 datasets, respectively.

– **RQ2.** Is there sufficient evidence that the NSGA-II algorithm can find the best k value for each module?
Answers. Yes, if we compare performance between our proposed model and classical kNN, we can reach to a conclusion that our model that uses dynamic selection of k produces better performance than classical kNN that uses static k value for all samples.

7. Threats to validity

This section aims to discuss the threats to validity encountered in the present study. The identified threats are categorized as either internal or external. External validity is further subdivided into three aspects: datasets, SDP scenarios, and evaluation measures. To ensure accurate conclusions, the utilization of an adequate number of diverse datasets that encompass a wide spectrum of features is vital. In this investigation, a comprehensive analysis was performed on 10 public datasets obtained from three software repository, which exhibit variations in module numbers and defect percentages. Moreover, the findings of this research are specific to "within-project" defect prediction scenarios, and the practical guidelines derived may not be readily applicable to other contexts. Additionally, while evaluation measures such as *MCC* and *BA* are generally robust against changes in the confusion matrix, the utilization of *MCC* and *BA* is recommended for enhanced sensitivity and informative results when dealing with imbalanced data.

The internal validity of this study is susceptible to three potential threats: the validation approach and feature selection. Although the conducted experiments employed a 10-fold cross-validation technique, it is acknowledged that leave-one-out cross-validation may yield better results by minimizing potential bias stemming from data sampling. Furthermore, feature selection algorithms were not employed in the current investigation to identify the optimal features for each dataset. Previous empirical studies have indicated that utilizing all available features often produces accuracy similar to that achieved by using the best feature subset. Consequently, the impact of the feature selection process on the final results is deemed insignificant and thus not recommended for utilization.

8. Conclusion

Predicting defects in the software modules helps developers detect faulty modules and identify the classes that might need refactoring. In this research paper, we trained the kNN classifier with a multi-optimization model called Genetic Algorithm (GA), resulting in our model, GA-kNN. Model performance was measured using *MCC* and balanced accuracy. The dataset KC1 yielded the highest *MCC* of 0.974 with $k = 3$ for GA-kNN, while the classical kNN model recorded an *MCC* value of 0.950 with the same k number. All the other datasets yielded a higher *MCC* and AUC when trained with the NSGAII-kNN model. Our experiments proved that the NSGAII-kNN model is better than the classical kNN model and can be used with various datasets. Our results demonstrated that the kNN classifier gave us good results for all the datasets proving that it is a good classifier for SDP datasets when compared to other classifiers such as SVM, NB, and RF, and thus, a good base for our model. Furthermore, NSGAII-kNN also showed superior performance when compared to other classifier models, including classical kNN because it yielded a high range of *MCC* and balanced accuracy values. In this study, we analyzed the kNN method for the purpose

of SDP, and we evaluated its performance based on three different classifiers. Our study was limited to understanding the optimization of kNN parameters using the GA. This is one possible method, although alternatives such as multilayer perceptron, neural networks and binary trees need to be analyzed for similar SDP applications on a dataset. As well, our study applied NSGAII-kNN to a limited number of datasets. Ideally, NSGAII-kNN should be applied to other varying datasets consisting of different types of data to analyze whether the results and accuracy are maintained. As well, we used three classifiers as evaluation metrics to determine the accuracy of our NSGAII-kNN model; this proves to be a limitation as numerous classifiers are available and the accuracy for SDP should be evaluated against each one of these classifiers. One final limitation of the study is that it was limited to the study of single platform algorithms and not on cross-platform systems, so it is unknown whether the proposed algorithm will provide the same results in a cross-platform environment. Future studies could investigate more developed optimization algorithms and other classification techniques for SDP. Furthermore, the NSGAII-kNN could be applied to in cross-platform applications, applications of software development, with different types and weights of datasets. Finally, future studies could evaluate the accuracy and efficiency of the proposed algorithm for use as a mainstream algorithm for SDP.

References

[1] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang et al., "Software defect prediction based on kernel PCA and weighted extreme learning machine," *Information and Software Technology*, Vol. 106, 2019, pp. 182–200.

[2] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman et al., "Local versus global lessons for defect prediction and effort estimation," *IEEE Transactions on Software Engineering*, Vol. 39, No. 6, 2013, pp. 822–834.

[3] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, Vol. 59, 2015, pp. 170–190.

[4] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, Vol. 44, No. 9, 2018, pp. 874–896.

[5] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering, PROMISE '07*, 2007.

[6] C. Tantithamthavorn, A.E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, 2018.

[7] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 63, No. 2, 2014, pp. 676–686.

[8] M. Singh Rawat and S. Kumar Dubey, "Software defect prediction models for quality improvement: A literature study," *IJCSI International Journal of Computer Science Issues*, Vol. 9, No. 5, 2012. [Online]. www.IJCSI.org

[9] E. Erturk and E.A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Systems with Applications*, Vol. 42, No. 4, 2015, pp. 1872–1879.

[10] B. Turhan, T. Menzies, A.B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, Vol. 14, No. 5, 2009, pp. 540–578. [Online]. https://link.springer.com/article/10.1007/s10664-008-9103-7

[11] I. Arora and A. Saha, "Software defect prediction: A comparison between artificial neural network and support vector machine," in *Advances in Intelligent Systems and Computing*, Vol. 562. Springer Verlag, 2018, pp. 51–61. [Online]. https://link.springer.com/chapter/10.1007/978-981-10-4603-2_6

[12] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang et al., "Defect prediction from static code features: Current results, limitations, new approaches," *Automated Software Engineering*,

Vol. 17, No. 4, 2010, pp. 375–407. [Online]. https://link.springer.com/article/10.1007/s10515-010-0069-5

[13] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: Do different classifiers find the same defects?" *Software Quality Journal*, Vol. 26, No. 2, 2018, pp. 525–552.

[14] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, 2013, pp. 434–443.

[15] X. Yang and W. Wen, "Ridge and lasso regression models for cross-version defect prediction," *IEEE Transactions on Reliability*, Vol. 67, No. 3, 2018, pp. 885–896.

[16] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Transactions on Reliability*, Vol. 64, No. 1, 2015, pp. 234–246.

[17] F. Wu, X.Y. Jing, Y. Sun, J. Sun, L. Huang et al., "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Transactions on Reliability*, Vol. 67, No. 2, 2018, pp. 581–597.

[18] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, Vol. 46, No. 12, 2020, pp. 1267–1293.

[19] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, Vol. 45, No. 12, 2019, pp. 1253–1269.

[20] M.A. Khan, N.S. Elmitwally, S. Abbas, S. Aftab, M. Ahmad et al., "Software defect prediction using artificial neural networks: A systematic literature review," *Scientific Programming*, Vol. 2022, No. 1, 2022.

[21] M.S. Alkhasawneh, "Software defect prediction through neural network and feature selections," *Applied Computational Intelligence and Soft Computing*, Vol. 2022, No. 1, 2022, pp. 1–16.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, 2002, pp. 182–197.

[23] I. Maleki, A. Ghaffari, and M. Masdari, "A new approach for software cost estimation with hybrid genetic algorithm and ant colony optimization," *International Journal of Innovation and Applied Studies*, Vol. 5, No. 1, 2014, pp. 72–81. [Online]. https://www.academia.edu/download/52287242/IJIAS-13-292-35.pdf

[24] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," *Software: Practice and Experience*, Vol. 50, No. 4, 2020, pp. 407–427.

[25] M.M. Rosli, Noor Hasimah Ibrahim Teo, Nor Shahida M. Yusop, and N. Shahriman Mohamad, "Fault prediction model for web application using genetic algorithm," in *International conference on computer and software Modeling (IPCSIT)*, 2011, pp. 71–77. [Online]. https://www.researchgate.net/profile/Marshima-Rosli/publication/264890205_Fault_Prediction_Model_for_Web_Application_Using_Genetic_Algorithm/links/55d4fcc308ae43dd17de4df4/Fault-Prediction-Model-for-Web-Application-Using-Genetic-Algorithm.pdf

[26] W. Afzal and R. Torkar, "On the application of genetic programming for software engineering predictive modeling: A systematic review," *Expert Systems with Applications*, Vol. 38, No. 9, 2011, pp. 11 984–11 997.

[27] S. Chatterjee, S. Nigam, and A. Roy, "Software fault prediction using neuro-fuzzy network and evolutionary learning approach," *Neural Computing and Applications*, Vol. 28, No. 1, 2016, pp. 1221–1231. [Online]. https://link.springer.com/article/10.1007/s00521-016-2437-y

[28] S. Goyal, "Handling class-imbalance with knn (neighbourhood) under-sampling for software defect prediction," *Artificial Intelligence Review*, Vol. 55, No. 3, 2022, pp. 2023–2064.

[29] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, "Software defect prediction using dynamic support vector machine," in *9th International Conference on Computational Intelligence and Security*, 2013, pp. 260–263.

[30] M.P. Sasankar and G. Sakarkar, "Cross project defect prediction using deep learning techniques," in *International Conference on Artificial Intelligence and Big Data Analytics*, 2022.

[31] K.O. Elish and M.O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, Vol. 81, No. 5, 2008, pp. 649–660.

[32] M. Hammad, A. Alqaddoumi, H. Al-Obaidy, and K. Almseidein, "Predicting software faults based on k-nearest neighbors classification," *International Journal of Computing and Digital Systems*, Vol. 8, No. 5, 2019, pp. 461–467.

[33] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," *Information and Software Technology*, Vol. 93, 2018, pp. 1–13.

[34] T.M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *International Conference on Tools with Artificial Intelligence, ICTAI*, Vol. 1, 2010, pp. 137–144.

[35] M.A. Mabayoje, A.O. Balogun, H.A. Jibril, J.O. Atoyebi, H.A. Mojeed et al., "Parameter tuning in kNN for software defect prediction: an empirical analysis," *Jurnal Teknologi dan Sistem Komputer*, Vol. 7, No. 4, 2019, pp. 121–126.

[36] A. Iqbal, S. Aftab, I. Ullah, M. Salman Bashir, and M. Anwaar Saeed, "Modern education and computer science," *Modern Education and Computer Science*, Vol. 9, 2019, pp. 54–64. [Online]. http://www.mecs-press.org/

[37] R. Jindal, Ruchika Malhotra, and Abha Jain, "Analysis of software project reports for defect prediction using kNN," in *Proceedings of the World Congress on Engineering*, Vol. 1, 2014. [Online]. http://www.iaeng.org/publication/WCE2014/WCE2014_pp180-185.pdf

[38] D. Ulumi and D.S. Series, "Weighted knn using grey relational analysis for cross-project defect prediction," *Journal of Physics: Conference Series*, Vol. 1230, No. 1, 2019, p. 12062. [Online]. https://iopscience.iop.org/article/10.1088/1742-6596/1230/1/012062/meta

[39] R. Goyal, P. Chandra, and Y. Singh, "Suitability of kNN regression in the development of interaction based software fault prediction models," *IERI Procedia*, Vol. 6, 2014, pp. 15–21.

[40] M.A. Mabayoje, A.O. Balogun, S.M. Bello, J.O. Atoyebi, H.A. Mojeed et al., "Wrapper feature selection based heterogeneous classifiers for software defect prediction," 2019.

[41] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, Vol. 21, No. 1, 2020, pp. 1–13. [Online]. https://link.springer.com/articles/10.1186/s12864-019-6413-7https://link.springer.com/article/10.1186/s12864-019-6413-7

[42] D. Tomar and S. Agarwal, "Prediction of defective software modules using class imbalance learning," *Applied Computational Intelligence and Soft Computing*, Vol. 2016, 2016, pp. 1–12.

[43] A.D. Chakravarthy, S. Bonthu, Z. Chen, and Q. Zhu, "Predictive models with resampling: A comparative study of machine learning algorithms and their performances on handling imbalanced datasets," *18th IEEE International Conference on Machine Learning and Applications, ICMLA 2019*, 2019, pp. 1492–1495.

[44] P.K. Kudjo, E. Ocquaye, and W. Ametepe, "Review of genetic algorithm and application in software testing," *International Journal of Computer Applications*, Vol. 160, No. 2, 2017, pp. 1–6.

[45] S. Agarwal, D. Tomar, and Siddhant, "Prediction of software defects using twin support vector machine," in *Proceedings of the International Conference on Information Systems and Computer Networks, ISCON*. IEEE, 2014, pp. 128–132.

[46] G.D. Boetticher, "Nearest neighbor sampling for better defect prediction," *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, 2005, pp. 1–6.

[47] L. Gong, S. Jiang, Q. Yu, and L. Jiang, "Unsupervised deep domain adaptation for heterogeneous defect prediction," *IEICE Transactions on Information and Systems*, Vol. E102D, No. 3, 2019, pp. 537–549.

[48] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, Vol. 95, 2018, pp. 296–312.

[49] U. Ali, S. Aftab, A. Iqbal, Z. Nawaz, M. Salman Bashir et al., "Software defect prediction using variant based ensemble learning and feature selection techniques," *Modern Education and Computer Science*, Vol. 5, 2020, pp. 29–40. [Online]. http://www.mecs-press.org/

[50] S. Zheng, J. Gai, H. Yu, H. Zou, and S. Gao, "Software defect prediction based on fuzzy weighted extreme learning machine with relative density information," *Scientific Programming*, Vol. 2020, No. 1, 2020.