Feature Transformation for Improved Software Bug Detection Models

Shamse Tasnim Cynthia
University of Saskatchewan
Saskatoon, SK, Canada
shamse.cynthia@usask.ca

Banani Roy
University of Saskatchewan
Saskatoon, SK, Canada
banani.roy@usask.ca

Debajyoti Mondal
University of Saskatchewan
Saskatoon, SK, Canada
d.mondal@usask.ca

Abstract

Testing software is considered to be one of the most crucial phases in software development life cycle. Software bug fixing requires a significant amount of time and effort. A rich body of recent research explored ways to predict bugs in software artifacts using machine learning based techniques. For a reliable and trustworthy prediction, it is crucial to also consider the explainability aspects of such machine learning models. In this paper, we show how the feature transformation techniques can significantly improve the prediction accuracy and build confidence in building bug prediction models. We propose a novel approach for improved bug prediction that first extracts the features, then finds a weighted transformation of these features using a genetic algorithm that best separates bugs from non-bugs when plotted in a low-dimensional space, and finally, trains the machine learning model using the transformed dataset. In our experiment with real-life bug datasets, the random forest and k-nearest neighbor classifier models that leveraged feature transformation showed 4.25% improvement in recall values on an average of over 8 software systems when compared to the models built on original data.

CCS Concepts: • **Software and its engineering → Software testing and debugging**; **Maintaining software**.

Keywords: software bug, machine learning, t-SNE, genetic algorithm

ACM Reference Format:
Shamse Tasnim Cynthia, Banani Roy, and Debajyoti Mondal. 2022. Feature Transformation for Improved Software Bug Detection Models. In *15th Innovations in Software Engineering Conference (ISEC 2022), February 24–26, 2022, Gandhinagar, India.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3511430.3511444

1 Introduction

The increasing complexity of today's software artifacts has made the software development and maintenance more challenging than ever. To deliver a high quality software to the end-users and maintain it, developers invest much of their effort into various tasks such as feature upgrades, code refactoring, bug detection, bug fix, etc. In this paper, we investigate the area of bug detection. In the case of a bug, a software may deviate from its expected behavior in a way that may result into significant cost to the end users. To test and fix a bug, it can take up to 50-60% of the software development effort [37]. This indicates the significance of detecting bugs at the earlier level of software development. For this purpose, machine learning classifiers can come in handy to detect bugs effectively. But with the detection of the bugs, it is also important for the developers to visualize the bug data, learn the reasons for bug occurrence and the features which influenced the most in inducing any bugs in the software system. Therefore, visualization techniques can be an efficient solution for understanding the bug data and gather insights into the explainability of the machine learning models that predict such bugs.

Visualization is known to be an effective tool to understand patterns in large amount of information. In the field of software visualization, visualization algorithms and techniques can provide meaningful graphical representations for the code and data in a software artifact [16]. In particular, for the code clones and bugs, a well-designed visualization can provide more information regarding bug propagation, features associated with the bugs, and confidence into why some machine learning classifiers are working better than the others. A rich body of previously proposed approaches [29, 39, 40] for detecting bugs was based on applying different algorithms and classifiers to detect the bugs with a high accuracy. However, the machine learning classifiers are often treated as black box models, and model explainability has recently been a crucial issue to gain confidence on these models [36]. Thus with proper detection of bugs, we think, it is also necessary to visualize the separation of the buggy and non-buggy data so that we can gain confidence on the working of the prediction models. A meaningful question in this context is whether one can find a weighted transformation of the features that can well separate the bug and non-bug data when they are visualized in a low-dimensional space (e.g., 2D scatter plot).

In software bug detection, machine learning classifiers have gained much popularity for their ability to predict bugs by learning from the code and features extracted from raw data repositories. A number of benchmark datasets are now available that include code blocks or commits, their features, and a label indicating whether a bug is present or not [14]. Researchers have noticed that some features in a dataset may be more distinguishing (than the rest) when predicting bugs. Therefore, it is natural to transform the features by weighting them with proper weights such that the bug data can be separated from the non-buggy ones in low-dimensional space. The hope is that such a proper weighting would also make the detection easier and increase the accuracy of the machine learning models. In this regard, we have used Genetic Algorithm (GA) in selecting the best weights to put on the feature values that will separate the bugs efficiently. Genetic Algorithm is considered to be an optimization algorithm that works by impersonating biological evolution [25]. It finds the best-fitted survivors by selecting the best individuals and performing crossover and mutations among them. Here we defined an individual to have a high fitness when the corresponding feature transformation is able to well-separate the bugs from the non-bugs. In our study, with the help of GA and low-dimensional embedding (t-SNE) based fitness, we designed a set of weights, used them to transform the data features, and applied machine learning classifiers thereafter to obtain better results.

1.1 Research Questions

In the light of the above discussion, in this paper we consider the following research questions.

RQ1: Do there exist data transformations that visually cluster and separate bugs from non-bugs in low-dimensional space? Does such data transformation help Machine Learning models to better detect the bugs in software artifacts?

Our study focuses on applying machine learning models to investigate whether the data transformation obtained from GA search actually helps the models to perform better while predicting bugs, and also serves the purpose of separating bug and non-bugs when the transformed data is visualized in a two-dimensional plot.

RQ2: Can a data transformation that visually cluster and separate bugs from non-bugs in low-dimensional space help build confidence in explaining model performance?

Here, we have investigated whether our visualization technique can give insights into why a machine learning model built one the transformed data would perform well over the ones built using the original data.

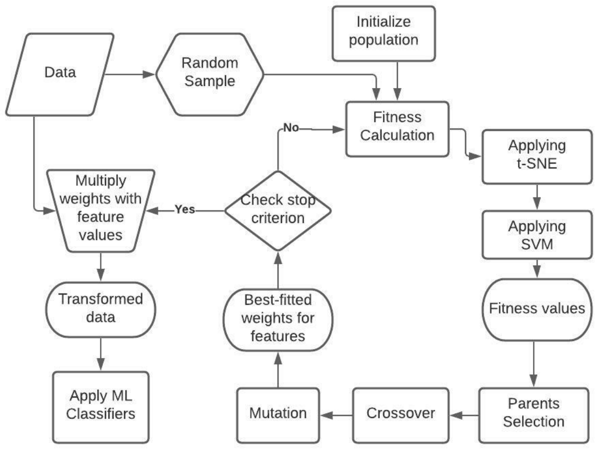

Fig. 1. A flow diagram of the proposed approach.

1.2 Steps to Solution

In this paper, we have used the BugHunter [13] and Jenkins [1] datasets for our study. Figure 1 illustrates the workflow of our approach.

Given the features of the commits in a dataset, we used a genetic algorithm to find a set of weights such that weighting the features and then embedding the commits into a low-dimensional space would separate the buggy commits from the non-buggy ones. At each iteration of GA, we have used a t-distributed stochastic neighbor embedding (t-SNE) [44] to compute such a low-dimensional embedding, i.e., t-SNE helps to visualize high dimensional data by giving each data point a location in a two or three-dimensional map. We computed the fitness of an individual (i.e., the quality of a low-dimensional embedding) using a support vector machine (SVM) classifier [47] that finds a hyper-plane that best separates the buggy commits from the non-buggy ones in the low-dimensional space.

The end result of the GA search is a set of weights that we multiply to the features to transform the data. We use four machine learning classifiers to evaluate whether the data transformation helps achieve better bug detection accuracy.

1.3 Contributions

We observed that weights computed using the genetic algorithm could transform data such that the buggy commits get well separated from the others when visualized in a scatter plot. Furthermore, the random forest and k-neighbors classifiers built on the transformed data achieved better classification accuracy than the one built on the original data. Specifically, the improvement in the recall value was sometimes upto 8% over 8 software systems (4.25% on an average). But the rest of the two machine learning classifiers did not show improvement on recall values when applied on the transformed data. In some cases, the classification accuracy is lower than the original data. For this reason, we have further changed the fitness value calculation for the genetic algorithm. This time, we have used each of the classifiers in the fitness function and evaluated them with the same classifier. We found that here is also some of the classifiers are performing better and others are not showing better prediction results. The code and sample data can be found in the shared github repository [1].

2 Related Research

In this section, we review the role of machine learning models in bug detection, the research related to software bug visualization, and the literature on dimensionality reduction and explainability of learning models.

2.1 Machine Learning Approaches to Software Bugs

Many machine learning based approaches have been designed to detect software bugs over the past decades [19, 34, 35, 42]. A recent study by Awni Hammouri et al. [19] presented a bug prediction model based on three supervised machine learning algorithms (Naive Bayes, Decision Tree and Artificial Neural Network), and observed a high accuracy rate in predicting the bugs. Miltiadis Allamanis et al. [4] addressed a new approach called BugLab, which is a self-supervised model for bug detection and fix. Their proposed approach first learns to detect bugs and repair them in code, and then it generates its own training data by creating buggy code for the detector to use, and the implementation showed an improvement of 30% than the baseline methods. Researchers have also attempted to tune the parameters of different machine learning algorithms. Aashish Gupta et al. [2] showed that tuning and changing the parameters of the existing XGBoost model can actually outperform state-of-art models. They used Logistic Regression, Decision Tree, Random Forest, Adaboost and XGBoost as their state-of-art models and applied them on four datasets. Neysiani et al. [38] conducted a comparative study between information retrieval (IR) based and machine learning based model to detect duplicate bug reports. Their study showed that ML based approaches achieved 40% better result than IR based approaches in validation performance.

2.2 Visual Analytics of Software Bugs

Visual analysis of software bugs is an active area of research and researchers are striving to find more suitable ways to relate the visualization techniques in understanding the bug entities. D'Ambros et al. [11] described the importance of visualization in analyzing the sheer data in a software artifact. Their paper focused on the bug life cycle, i.e., the information of a bug's history and its traversal states. Later, they extracted data from Bugzilla and presented two visualization techniques focused on understanding bugs based on various granularity. Hora et al. [22] proposed a tool that can be used to retrieve information about software bugs from bug-tracking systems, link the extracted information to other systems and also to help explore interactive visualizations about bugs. They suggested that to tackle bug in legacy software complications, more information is needed about the software systems, and visual analysis can be of great help. In another study, Hammad et al. [18] discussed the importance of understanding and monitoring bug report status as it can change over time. Since the relationship between the developers and bug reports is often problematic, they were inspired to propose a visualization approach that can reveal various bug status reports and their alliance with the bug trackers or developers working on them. Yeasmin et al. [46] in their paper, explained the importance of a project manager's struggle to be aware of all possible bug reports for a software's current version. Therefore, they presented a new prototype that can help the developers to give feedback on a project's bug report with the help of interactive visualization of the bug report's important information by using topic analysis.

2.3 Dimensionality Reduction Approaches

Existing benchmarks for bug related datasets are often high-dimensional, i.e., a commit may contain about a hundred features [1]. While analyzing high-dimensional data, a common approach to to reduce the dimensionality and visualize it in a low-dimensional space for better interpretation. The PCA, MDA, t-SNE are some commonly used techniques for dimensionality reduction. Visualization of the data after transforming them using these techniques can significantly impact our understanding of the data, as well as they can often help build better machine learning models. Jonsson [23] examined that t-SNE rendering of a bug database can reveal meaningful structure in the dataset. Mondal et al. [33] used dimensionality reduction technique to create a geospatial map of the code clones that clusters related code fragments in clusters revealing the communities of the code clones. The use of t-SNE and PCA techniques are widespread also beyond software research, e.g., document analysis [12], image processing [31], bioinformatics [27], etc. We have also seen dimensionality reduction techniques to be combined with metaheuristic search approaches such as genertic algorithms in different areas of research. For example, Firoz Mahmud et al. [28] proposed an approach to recognize a face by combining PCA with a Genetic Algorithm in the Computer Vision area. PCA was applied to retrieve the features, and GA was used to find an optimal solution from the ample search space. Rahul Adhao et al. [3] combined PCA and GA in feature engineering to assist machine learning algorithms in providing efficient results. Their study showed that applying PCA

[1]https://github.com/cynthia247/Feature-Transformation-for-Improved-Bug-Detection-Model.git

before GA improved the model to have better accuracy with fewer features.

Although there exist several examples that combine dimensionality reduction and metaheuristic search to solve various research problems, we are not aware of any approach that augments genetic algorithm and t-SNE in the software bug detection research area.

3 Technical Background

In this section we review various algorithmic tools and techniques that we will use in our experiments.

3.1 t-SNE algorithms

One of the core components used in our approach is the t-distributed Stochastic Neighbor Embedding (t-SNE) [44]. This algorithm is based on the stochastic neighbor embedding (SNE) technique that embeds high-dimensional data into a low-dimensional space (two or three dimensions for visualization) by creating clusters putting similar datapoints close together. The t-SNE has two main stages. The first one consists of defining a probability distribution over pairs of high-dimensional objects, which is constructed in such a way that a higher probability is given to similar objects and a lower probability is assigned to dissimilar points. In the second part, a similar probability distribution is defined in the low-dimensional map over the points. Then the Kullback-Leibler divergence between the two distributions have been minimized by computing the gradients.

Formally, given a set of N high-dimensional objects $x_1, \ldots, x_N$, t-SNE first calculates the probabilities $p_{i|j}$ that is proportional to the similarity of objects x_i and x_j, as follows. For $i \neq j$,

$$p_{i|j} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k\neq 1} exp(-||x_i - x_k||^2/2\sigma^2)}$$

, where $p_{i|i} = 0$ and $\sum_j p_{j|i} = 1$ for all i, σ_i denotes the bandwidth of the Gaussian kernels which is set following that the perplexity of the conditional distribution is equal to a preset perplexity which uses the bisection method. So in denser parts of data space, smaller values of σ_i are used.

The t-SNE focuses on learning a d-dimensional map $y_1, \ldots, x_N$ that considers the similarities $q_{i|j}$, which is modeled by the similarities between two points in the low-dimensional map. For $i \neq j$,

$$q_{i|j} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_k \sum_{l\neq k}(1 + ||y_k - y_l||^2)^{-1}}$$

.

The point locations y_i are set by minimizing the Kullback-Leibler divergence of the distribution P from the distribution Q, which is:

$$KL(P||Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

This minimization is done using gradient descent which results in reflecting the similarities between the high-dimensional points in the low-dimensional map.

The t-SNE has been used in various applications over the years, such as computer security research, cancer research, genomics, bioinformatics etc. The plots generated by t-SNE often display clusters, and the clusters are influenced by the parameters of t-SNE that the users choose. Hence understanding the parameters properly is necessary [5], where an interactive exploration can help choosing appropriate parameters.

3.2 Genetic Algorithms

Genetic algorithms [21] are considered as a type of optimization algorithm which is inspired by the theory of natural evolution introduced by Charles Darwin [30]. They are mainly used to produce better solutions to optimizations and search problems by mutation, crossover and selection techniques. Genetic algorithm focuses on the process of selecting the best-fitted individuals through natural selection. This aims to reproduce better offspring of the next generation. Natural selection starts with selecting the fittest individuals collected from a population. The main idea is to generate offspring that take the characteristics of the parents and add them to the next generation. The quality of the offspring gets better if the parents have better fitness. By repeating this process, in the end, a new generation can be found that has the fittest individuals. There are five stages in a genetic algorithm. Those are described below:

Initial population is a set of individuals which is identified by a set of parameters called genes. These genes are united together into a string to make a solution or chromosome. In our approach, each individual of the population is a set of weights. The *fitness* function aims to determine the fitness of an individual which shows the capability of an individual to complete compared to other individuals. Each individual receives a fitness score, and thus, this score is used in selecting the best individual for reproduction. Then, the *selection* phase consists of selecting the best-fitted individual and allows them to pass their genes to the next generation. Best on the fitness score, two pairs of parents are selected. After that, *crossover* plays an important role in genetic algorithms by creating offspring until a random crossover point is reached within the genes. The genes of the parents are exchanged, and the population gets new offspring. Lastly in Mutation, as the new offspring are formed, in some cases, the genes may require a mutation because of the low random probability. This is how diversity within the population can be maintained, and premature convergences can be prevented. At last, when the algorithm reaches a point where the population has converged by getting the best-fitted offspring.

3.3 Support Vector Machine (SVM)

Support vector machine [10] is one of the widely used machine learning tools for classification, regression, detection and feature reduction tasks. It mainly focuses on finding a hyperplane that separates two classes (or a set of hyperplanes for more classes). The plane is chosen such that it separates the two classes of data points by maximizing the distance from the two data clusters. The reason to maximize such a distance measure is that it provides some confidence that the future data can be classified more accurately. The data points which fall close to the hyperplane are called support vectors. Based on the position of the support vectors, the hyperplane position is changed so that it can maintain the highest possible distance between the data points of both classes.

3.4 Random Forest

Random forest [8] is a supervised learning algorithm, which depicts the idea of combining learning models so that it can perform better than a single model's result. In particular, it combines multiple Decision Trees to achieve a more accurate prediction. This algorithm adds randomness in creating multiple decision trees from the dataset and combines them to make a final decision. But the depth of the tree can be controlled using max_depth variable, and the number of decision trees can be controlled by n_estimators. One advantage of using random forest is that this can be used to compute an importance score for each feature, and thus provides us with some insights into the features that are relevant to obtain an accurate prediction.

3.5 K Nearest Neighbors

In k-nearest neighbor [15], the main deciding factor is the number of neighbors that is k. This classifier is used in both regression and classification problem and for both of the cases, k closest training examples are considered as input and the output is dependant on the application of k-NN in classification or regression model. The algorithm takes the assumption of the similarity between the new data and the available data. When a new case or data comes, then the algorithm put the new data into that category which is the most similar one to the available categories.

3.6 Logistic Regression

Logistic Regression [6] is used for categorical dependant variable. The parameters of logistic regression mostly depends on the number of input features and the output is the categorical prediction. Like the linear regression, logistic regression does not fit a straight line to the data. Rather, it fits a S shaped curve which is called Sigmoid. The output is based on the estimated probability and it is used to indicate the confidence of the prediction value being the actual value when an input X is given. A threshold is set to predict the class of the data and this decision boundary can be linear or non-linear.

3.7 Naive Bayes

Naive Bayes [45] is a classification technique and it is based on the Bayes' Theorem which assumes that a particular feature in a class is unrelated to any other features present in the data. Naive Bayes works fast in predicting the class of a dataset and it also performs well in multi class prediction. In our study, we have used the Gaussian Naive Bayes. This is an extension of the naive Bayes and the Gaussian or Normal distribution is used here to estimate the distribution of the data. The mean and the standard deviation are calculated to estimate from the training data. The main problem with naive Bayes is that the algorithm depends on the assumption that the features of the data would be independent but in real life, independent features can be hardly found.

4 Our Approach (GA+t-SNE)

Our approach aims at finding the best-fitted weights for the features that can distinguish between bug and non-bug data when the transformed data (with weighted features) is plotted using t-SNE. The pseudocode of our approach is illustrated in Algorithm 2.

Note that the genetic algorithm finds the best-fitted solution by computing a fitness value to each individual in the population. The individual which gets the highest fitness is considered to be a better offspring for the next generation. In our approach, the fitness of an individual is computed with respect to a sample dataset. We choose a sample of the dataset (ranging between 5%-10% in random). To compute the fitness of an individual we first transformed the sample data using the weight vector of the individual. We then evaluated how well the transformed data separates the bugs from the non-bugs when plotted on a low-dimensional space as follows.

We used t-SNE to plot the transformed dataset into two dimensions. In fact, this gives us a two-dimensional scatterplot. To calculate the fitness value, we used SVM to find a hyper-plane that best separates the bug and non-bug data. After applying the model, we computed the performance metric recall value, which is set to be the fitness value for the individual. Note that the SVM has been applied only to two-dimensional embedding obtained from the t-SNE plot. The population with high recall values were considered to take part in parents selection.

5 Experimental Design

In this section, we discuss the dataset and experimental setup.

5.1 Dataset Description

For our experiments, we have used Jenkins dataset (an open-source project [1]), and 7 other datasets (software systems)

Algorithm 1: Fitness_Calculation

```
Input  : A population with $m$ individuals and a
         random sample $S$
Output : A set of parents with high fitness
highest_fitness ⟵ 0
temporary_fitness ⟵ $\Phi$
for $i = 1, 2, \ldots, m$ do
    $S_i$ ⟵ transform $S$ using the $i$th individual or
      weight vector
    tsne_plot$_i$ ⟵ run TSNE on $S_i$
    run SVM on tsne_plot$_i$
    set the fitness of $i$th individual to be the SVM
      recall value and append this recall value to
      temporary_fitness
end
if highest_fitness ≤ max(temporary_fitness) then
    highest_fitness ⟵ temporary_fitness
end
parents ⟵ random selection from best-fitted
  individuals
```

Algorithm 2: Genetic Algorithm

```
Input  : A $n$-dimensional dataset $D$
Output : A set of $n$ weights
Initialize a population $P$ with $m$ individuals, each
  representing a random weight vector of length $n$
$S$ ⟵ A 5-10% random sample of $D$
for $i = 1, 2, \ldots,$ max_Iteration do
    parents ⟵ Fitness_Calculation($P$, $S$)
    offspring ⟵ crossover(parents)
    offspring ⟵ mutation(offspring)
    population ⟵ parents ∪ offspring
end
return best fitted individual
```

selected from a collective dataset called BugHunter dataset [13]. Table 1 summarizes the data that we have used in our study.

5.1.1 BugHunter Dataset. This is an automatically created dataset containing code elements (files, classes, methods) and a wide variety of code metrics and bug information. This dataset includes buggy and fixed states of similar source code elements, which can be identified from the limited timeframe nevertheless the release version.

The dataset is comprised of 15 Java projects which are individually different. The dataset was created by connecting commits to bugs, analyzing the log, and identifying the bugs with the help of clues. To check the suitability of the dataset, authors have performed several filtering experiments on the raw dataset, and they are - Removal, Subtract, Single and GCF. Removal keeps the entries which are located in the class with larger cardinality, subtract reduces the entry number in the class which has larger cardinality, single filtering removes the entries of the class which has smaller cardinality and holding only one entry from the larger one and lastly, GCF divides the entry numbers of both classes by their greatest common factors [14]. It was found among the four filtering methods, Subtract filtering performed the best. Therefore, we decided to use the dataset obtained from Subtract filtering. We chose seven datasets with the highest number of Class entries. All of these datasets are high-dimensional, i.e, containing 98 features per data point.

5.1.2 Jenkins Dataset. Jenkins dataset was created from a project, named Jenkins which is a Java-based open-source project. Borg et al. [7] first extracted fixed bugs by implementing SSZ Algorithm [43] and connected those bugs to the respective bug fixing commit in the GitHub repository of Jenkins. The SSZ Algorithm used in the process found out the commits that have more impact in inducing the bugs. This dataset is a highly imbalanced one and this dataset has 44 features in total.

Table 1. Dataset Overview

Software Systems	Size	% of Bugs
BoardleafCommerce	2957	48%
elasticsearch	21657	50%
neo4j	3701	43%
netty	5677	38%
orientdb	4134	44%
ceylon-ide-eclipse	1275	33%
MapDB	899	48%
Jenkins dataset	28923	16%

5.2 Dataset Preparation

There were three parts in dataset preparation. At first, the dataset contains both categorical and numerical attributes. So to implement our algorithm, we have mapped the categorical attributes into numerical ones by using Label Encoder [17]. For the datasets obtained from BugHunter, the categorical attributes are 'Hash' and 'LongName', whereas, for the Jenkins dataset, the categorical attributes are 'commit' and 'classification'. Next, the feature containing the bug numbers has values starting from 0, where 0 indicates that the commit has no bug and any other integers indicate that the commit has bugs. So for our convenience, we have transformed all the other values except 0 to be 1, indicating the commit has bugs. This way, we present the bug and non-bug data into two groups of the dataset. Finally, we normalized the dataset to present the values into a standard scale ranging from 0 to 1.

5.3 Execution of Algorithms

The main algorithm used in our study is the Genetic Algorithm. This algorithm follows an iterative process that starts from a randomly generated population, and at each generation, a set of candidate solutions are generated and passed to the next generation. In our study, we selected 100 generations to execute. We also examined larger number of generations (i.e., 150) but did not find any noticeable difference in the resulting solution. We have taken a population of 100 to be considered in each generation. We did not notice any significant difference in the resulting solution by varying this parameter. Each individual is assigned *n* random weights ranging between 0 and 1, where *n* corresponds to the number of dimensions (features) of the input dataset. We have considered the single point type for crossover and bit flip type for mutation.

Next, we have used t-SNE for mapping the dataset from high-dimensional to low-dimensional space. t-SNE has been implemented in the fitness function of GA to help in calculating the fitness value of each population. We have used the Scikit-Learn implementation of the algorithm by tuning different parameters. The *n_components* was selected to 2 as we want our high-dimensional dataset to be converted to a two-dimensional dataset. The *random_state* was set to 0 and the *perplexity* value was tuned based on the dataset density. The perplexity value of the t-SNE algorithm is considered to be one of the most confusing parameters which require manual selection [9]. The perplexity value typically ranges from 5 to 50, and the denser dataset needs a larger perplexity. Therefore, based on our dataset density, we have chosen a perplexity range between 7 to 15. We did some implementations using different perplexity values, and observed this perplexity range to be better suited for our datasets to separate the cluster of bugs from the non-bugs. For calculating the fitness value, we have used the SVM algorithm on the t-SNE plot, and the recall value obtained from each population is considered to be an individual's fitness value.

The BugHunter dataset has been created considering the imbalanced ratio of bug and non-bug data. The authors [13] have implemented the random under sampling method to balance the ratio because without sampling, the classification algorithm showed high values for precision, recall and F-measure. But in the Jenkins dataset, the proportion of bug to non-bug data is significantly noticeable. Therefore, we have tried to find an optimal separating hyperplane for the imbalanced data and thus used SGDClassifier where the *max_iter* is set to 1000 and the stopping criterion, *tol*, is set to $1e^{-5}$.

Finally, to evaluate the performance of the transformed data, we have applied random forest, k nearest neighbors, logistic regression and naive bayes classifiers to investigate whether the transformed data performs better in detecting bugs than the original data. We have used the default parameter settings while implementing the algorithm. A 10-fold Cross Validation has been implemented to split the training and testing data. Random Forest also provided us with the most relevant feature values, and based on the feature scores, we have selected top 30 features. The details of the features are available in [13].

6 Results

In this section, we discuss the experimental results in the light of the research questions RQ1-RQ2.

6.1 Discussion on RQ1.

RQ1 asks about the possibility of data transformation that can visually cluster and separate bugs from non-bugs in low-dimensional space. If such a data transformations exist, then it also asks whether they can help machine learning models to better detect the bugs in software artifacts.

6.1.1 Analysis of Visual Plots. To answer this question, we transformed the original data by multiplying the features with the weights computed and showed the t-SNE plots for the transformed datasets. Fig. 2 illustrates for every dataset, the t-SNE plot of the original data and the t-SNE plot of the transformed data (i.e., the data obtained after multiplying the weight values to features). From Fig. 2 we can observe that the bug and non-bug data points are often better separated in the t-SNE plot of the transformed data compared to the original data. Such cases are indicated using oval shapes. There are also cases when the t-SNE plot of the transformed data appears to cluster the bugs and non-bugs better than plot for the original data (e.g., MapDB and Jenkins dataset).

6.1.2 Analysis of Model Performances. To investigate whether our data transformation approach can help the machine learning models to perform better in detecting bugs, we multiplied the features of the original dataset with the best weights computed using the genetic algorithm. We then trained four machine learning models on the original data and also on the transformed data.

The results are shown in Table 2, where we can observe that the random forest model and k-nearest neighbors model built on transformed dataset have a higher mean recall values than the recall value obtained from the model on original dataset (4.25% on average over all datasets). However, the same trend was not seen for the other two classifiers - logistic regression and naive Bayes. The recall values obtained were similar for both the original and transformed data across all subject systems. One potential reason for the logistic regression and naive Bayes not being benefited from the feature transformation may be that these model relies on properties that are not inherently geometric. Thus our approach that finds the weight based on the geometric separation between the bug and non-bug data at a low-dimensional embedding are expected to be less useful for these models.

Table 2 shows that k-nearest neighbors performed best for most datasets. For some datasets, random forest performed better than logistic regression and the for some other, we see

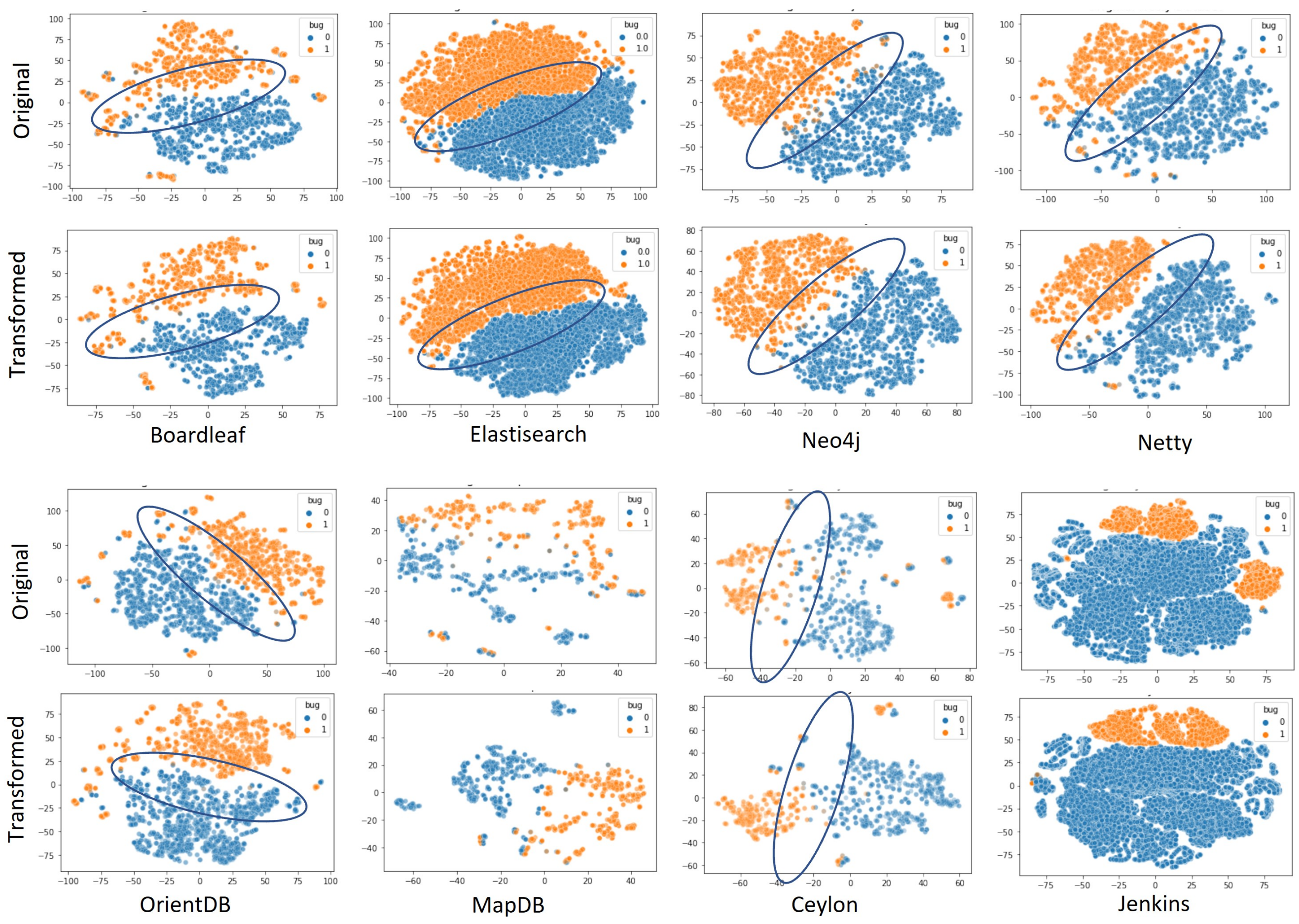

Fig. 2. The t-SNE plots of the datasets before and after transformation. The transformed dataset often shows better separation between the two classes as illustrated using oval shapes. For MapDB and Jenkins datasets, the transformed dataset show better clustering of the data points.

Table 2. Recall values before and after transforming the datasets (Here, Org=Original and Trans=Transformed)

Software Systems	Random Forest			K Nearest Neighbors			Logistic Regression			Naive Bayes		
	Org	Trans	% Improved	Org	Trans	% Improved	Org	Trans	% Improved	Org	Trans	% Improved
Boardleaf	0.36	0.43	7%	0.55	**0.58**	3%	0.52	0.50	0%	0.50	0.50	0%
Elastisearch	0.27	0.30	3%	0.45	0.47	2%	0.50	**0.51**	1%	0.29	0.29	0%
Neo4j	0.20	0.25	5%	0.35	**0.36**	1%	0.18	0.18	0%	0.23	0.23	0%
Netty	0.21	0.24	3%	0.33	**0.36**	3%	0.14	0.15	1%	0.25	0.24	0%
Orientdb	0.22	0.25	3%	0.35	**0.40**	5%	0.17	0.16	0%	0.23	0.23	0%
MapDB	0.36	0.41	5%	0.48	0.56	8%	0.55	**0.59**	4%	0.34	0.34	0%
Ceylon	0.14	0.19	5%	0.18	**0.24**	6%	0.09	0.12	4%	0.22	0.22	0%
Jenkins	0.22	0.25	3%	0.25	0.29	4%	0.30	0.30	0%	0.49	**0.50**	1%

the opposite trend. This is consistent also with the known literature. Random forest has been used widely for bug prediction in the literature [7] and k-nearest neighbors are known to work better when the number of features is large and the class values are skewed [32]. Naive Bayes assumes the features of a dataset as independent predictor but in real life data, it is not possible that all the features are mutually independent [24]. Prior studies show that logistic regression does not perform well when the number of noisy data is greater than number of explanatory variables, whereas random forest shows better performance as it better handles the noisy data [26].

Note that we have shown only the recall values to evaluate the model's performance following prior research. However, we checked the precision and F-measure values, which also appeared to be higher for the model built with transformed data. The reason for using primarily the recall values is that in software bug detection, the recall corresponds to the bugs which are correctly identified as bugs. A high recall value represents that the model is successful in finding most bugs. Recall is significant when we want to emphasize that we want to capture the positive cases only [20], i.e. which in our context are the software bugs. Precision indicates a powerful plausibility that the predicted bugs are actually bugs, but recall is directly related to the proportion of correctly predicted bugs from all the non-bug data. So increase in recall provides us with better detection of the data which are initially predicted as non-bugs [41].

Answering RQ1: Our results reveal that genetic algorithm based search are able to find data transformations that can better cluster and separate bug data from non-bug data when embedded in a low-dimensional plot using t-SNE. Furthermore, the k-nearest neighbors and random forest model for the transformed data may provide better recall values when compared with the models built on the non-transformed data. Given that we have secured a mean 4.25% higher recall value over eight datasets, it appears that the data transformation does help machine learning techniques to detect the bugs more accurately.

6.2 Discussion on RQ2.

RQ2 asks whether data transformation can help build confidence in explaining the model performance. Although a machine learning model's performance can be evaluated using quantitative evaluation metrics such as recall, precision, accuracy and F-measure, users may raise questions on why they should trust the model as it may learn from unrelated features or make decision based on parameters that are not well understood. Although in the previous section, we have seen machine learning models built on transformed data to have a better recall value, such quantitative measure alone cannot explain why these models are performing better than the others. However, we can again rely on the genetic algorithm, where each individual (i.e., weight vector) corresponds to a data transformation and the fitness of an individual represents how well the data transform separates the bug and non-bug data in a t-SNE plot. Since the fitness score is computed based on a SVM classifier's performance on the t-SNE plot, it provides us with some confidence that the data transformation separates the bug and non-bug classes in a visual plot which is human interpretable.

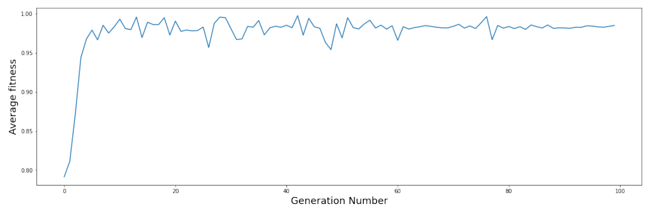

Fig. 3. Average fitness vs generation plot of Jenkins datatset

We have also found that the visual plot improves with the number of generations of the genetic algorithms. Fig. 3 illustrates the average fitness vs generation plot of the Jenkins dataset. From the figure, we see that as the generation number increases, the average fitness values are also improving. From this, we can obtain some confidence that the machine learning models on the transformed data may perform better in a low-dimensional space.

Answering RQ2: Our approach do not provide a clear pathway to explain the behaviour of machine learning models. However, in light of answering this question, we can state that the working of the models follows a transparent working mechanism that we can observe from the t-SNE plots. Furthermore, the improved performance of some models on transformed data strongly aligns with our intuition of visual separation of the bug and non-bug data in low-dimensions. Thus our feature transformation based approach to building a bug prediction model certainly helps to build some confidence in explaining the model's performance.

7 Limitations and Avenue for Future Research

Only two of our four machine learning classifiers showed improved performances with our feature transformation approach. Designing model specific feature transformation for bug datasets would be an interesting approach for future research. Moreover, we have normalized the data before applying our proposed algorithm. It is possible that data normalization can impact the result. Examining our approach with more models, state-of-the-art techniques and on a larger number of datasets can further strengthen our research. We might get some meaningful and interesting result if we can implement our proposed techniques with millions of data points. As our dataset amount was limited, that is why we could not apply deep neural networks (DNN) for learning feature transformation that could improve the accuracy. Future research can be conducted in this aspect for increasing the separability between the buggy and non-buggy commits. Moreover, the transformation approach could have been applied to different bug dataset to test the generalisability and that would certainly provide us significant insights about the features' natures. We did not investigate whether there is a relation between the weights and features, i.e., whether higher weights indicate importance. The reason is that intuitively, the weights in our approach only relate to better separation between bugs and non-bugs at the low-dimensional plot. However, it would be interesting to further examine why and how the features are being weighted.

Our genetic algorithm based approach selects the weight vector that correspond to highest fitness value (i.e., provides best separation in the t-SNE plot). The algorithm has been used in combination with support vector machine. Therefore, our model can impact the results when tuning the parameters. However, at the end of the algorithm, there are often many good weight vectors to choose from. Such an automatic selection based on fitness score ignores the rich structural information that can be seen in a t-SNE plot. It is thus natural to ask whether one can visually inspect a few candidate t-SNE plots and choose the best weight vector to be used. To investigate user's opinion, we conducted a pilot user study with 12 users who had experience in software development area. The result of the user study showed slight improvement of the accuracy result when humans are selecting the plots based on whether the clusters appear to be more separated when inspected visually. We believe a formal user study may be valuable to understand the scope of bringing humans in the loop to create better bug detection models.

8 Conclusion

We proposed a new approach for data transformation that helps to construct machine learning models with better bug prediction performances and with improved interpretability. Our approach runs a genetic algorithm to find the set of weights such that transforming the features based on these weights allows separating the bug data from the non-bug data in a t-SNE plot. We applied four machine learning classifiers on the transformed dataset and observed that some models show better bug prediction ability when built on feature-transformed data, compared to their counterparts that are built on the original data. Our inspection of the t-SNE plots obtained from the original and transformed datasets showed that the plots from the transformed datasets often provide better clustering and separation of the bug and non-bug data than the original dataset. We believe that our work on developing feature transformation based bug detection models will inspire future research that embodies visualization techniques to improve the performance and interpretability of machine learning models in software research.

Acknowledgments

This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grants, and by an NSERC Collaborative Research and Training Experience (CREATE) grant, and by two Canada First Research Excellence Fund (CFREF) grants coordinated by the Global Institute for Food Security (GIFS) and the Global Institute for Water Security (GIWS).

References

[1] [n. d.]. jenkinsci/jenkins: Jenkins automation server. https://github.com/jenkinsci/jenkins

[2] 2020. Novel XGBoost Tuned Machine Learning Model for Software Bug Prediction. In *Proceedings of International Conference on Intelligent Engineering and Management, ICIEM 2020*. Institute of Electrical and Electronics Engineers Inc., 376–380. https://doi.org/10.1109/ICIEM48762.2020.9160152

[3] Rahul Adhao and Vinod Pachghare. 2020. Feature selection using principal component analysis and genetic algorithm. *Journal of Discrete Mathematical Sciences and Cryptography* 23, 2 (feb 2020), 595–602. https://doi.org/10.1080/09720529.2020.1729507

[4] Miltiadis Allamanis, Henry Jackson-Flux, and Marc Brockschmidt. 2021. Self-Supervised Bug Detection and Repair. (may 2021). arXiv:2105.12787 http://arxiv.org/abs/2105.12787

[5] Anna C. Belkina, Christopher O. Ciccolella, Rina Anno, Richard Halpert, Josef Spidlen, and Jennifer E. Snyder-Cappione. 2019. Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. *Nature Communications* 10, 1 (dec 2019), 1–12. https://doi.org/10.1038/s41467-019-13055-y

[6] Joseph Berkson. 1944. Application of the Logistic Function to Bio-Assay. *J. Amer. Statist. Assoc.* 39 (1944), 357–365.

[7] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. 2019. SZZ unleashed: An open implementation of the SZZ algorithm-featuring example usage in a study of just-in-time bug prediction for the jenkins project. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, MaLTeSQuE*. Association for Computing Machinery, Inc, 7–12. https://doi.org/10.1145/3340482.3342742

[8] Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[9] Yanshuai Cao and Luyu Wang. 2017. Automatic Selection of t-SNE Perplexity. (aug 2017). arXiv:1708.03229

[10] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[11] Marco D'Ambros, Michele Lanza, and Martin Pinzger. 2007. "A Bug's Life" Visualizing a Bug Database. In *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007, Banff, Alberta, Canada, June 25-26, 2007*, Jonathan I. Maletic, Alexandru C. Telea, and Andrian Marcus (Eds.). IEEE Computer Society, 113–120.

[12] Binu Melit Devassy, Sony George, and Peter Nussbaum. 2020. Unsupervised clustering of hyperspectral paper data using T-SNE. *Journal of Imaging* 6, 5 (may 2020), 29. https://doi.org/10.3390/JIMAGING6050029

[13] Rudolf Ferenc, Péter Gyimesi, Gábor Gyimesi, Zoltán Tóth, and Tibor Gyimóthy. 2020. An Automatically Created Novel Bug Dataset and its Validation in Bug Prediction. *Journal of Systems and Software* 169 (2020). https://doi.org/10.1016/j.jss.2020.110691

[14] Rudolf Ferenc, Zoltán Tóth, Gergely Ladányi, István Siket, and Tibor Gyimóthy. 2020. A public unified bug dataset for java and its assessment regarding metrics and bug prediction. *Software Quality Journal* 28, 4 (dec 2020), 1447–1506. https://doi.org/10.1007/s11219-020-09515-0

[15] Evelyn Fix and Joseph L. Hodges. 1989. Discriminatory Analysis - Nonparametric Discrimination: Consistency Properties. *International Statistical Review* 57 (1989), 238.

[16] Denis Gracanin, Kresimir Matkovic, and Mohamed Eltoweissy. 2005. Software visualization. *Innov. Syst. Softw. Eng.* 1, 2 (2005), 221–230.

[17] Heena Gupta and V. Asha. 2020. Impact of Encoding of High Cardinality Categorical Data to Solve Prediction Problems. *Journal of Computational and Theoretical Nanoscience* 17, 9 (dec 2020), 4197–4201. https://doi.org/10.1166/jctn.2020.9044

[18] Maen Hammad, Somia Abufakher, and Mustafa Hammad. 2014. A visualization approach for bug reports in software systems. *International Journal of Software Engineering and its Applications* 8, 10 (2014), 37–46. https://doi.org/10.14257/ijseia.2014.8.10.04

[19] Awni Hammouri, Mustafa Hammad, Mohammad M. Alnabhan, and Fatima Alsarayrah. 2018. Software Bug Prediction using Machine Learning Approach. *International Journal of Advanced Computer Science and Applications* 9 (2018).

[20] Steffen Herbold, Alexander Trautsch, and Fabian Trautsch. 2020. On the feasibility of automated prediction of bug and non-bug issues. *Empirical Software Engineering* 25, 6 (nov 2020), 5333–5369. https://doi.org/10.1007/s10664-020-09885-w

[21] John H Holland. 1992. Genetic algorithms. *Scientific american* 267, 1 (1992), 66–73.

[22] Andre Hora, Nicolas Anquetil, Stephane Ducasse, Muhammad Bhatti, Cesar Couto, Marco Tulio Valente, and Julio Martins. 2012. BugMaps: A tool for the visual exploration and analysis of bugs. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*. 523–526. https://doi.org/10.1109/CSMR.2012.68

[23] Leif Jonsson. 2018. *Machine Learning-Based Bug Handling in Large-Scale Software Development.* Ph. D. Dissertation. Sweden.

[24] Neli Kalcheva, Maya Todorova, and Ginka Marinova. 2020. Naive Bayes Classifier, Decision Tree and AdaBoost Ensemble Algorithm–Advantages and Disadvantages. *KNOWLEDGE BASED SUSTAINABLE DEVELOPMENT* (2020), 153.

[25] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. 2021. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications* 80, 5 (feb 2021), 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

[26] Kaitlin Kirasich, T. Smith, and Bivin Sadler. 2018. Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets.

[27] Wentian Li, Jane E. Cerise, Yaning Yang, and Henry Han. 2017. Application of t-SNE to human genetic data. *Journal of Bioinformatics and Computational Biology* 15, 4 (aug 2017). https://doi.org/10.1142/S0219720017500172

[28] Firoz Mahmud, Md Enamul Haque, Syed Tauhid Zuhori, and Biprodip Pal. 2014. Human face recognition using PCA based Genetic Algorithm. In *Proc. of the 1st International Conference on Electrical Engineering and Information and Communication Technology, ICEEICT*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICEEICT.2014.6919046

[29] Ruchika Malhotra. 2015. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing Journal* 27 (feb 2015), 504–518. https://doi.org/10.1016/j.asoc.2014.11.023

[30] K. F. Man, K. S. Tang, and S. Kwong. 1996. Genetic algorithms: Concepts and applications. *IEEE Transactions on Industrial Electronics* 43, 5 (1996), 519–534. https://doi.org/10.1109/41.538609

[31] Aimin Miao, Jiajun Zhuang, Yu Tang, Yong He, Xuan Chu, and Shaoming Luo. 2018. Hyperspectral Image-Based Variety Classification of Waxy Maize Seeds by the t-SNE Model and Procrustes Analysis. *Sensors* 18, 12 (2018). https://doi.org/10.3390/s18124391

[32] Sushruta Mishra, Pradeep Kumar Mallick, Lambodar Jena, and Gyoo-Soo Chae. 2020. Optimization of Skewed Data Using Sampling-Based Preprocessing Approach. *Frontiers in Public Health* 8 (2020), 274. https://doi.org/10.3389/fpubh.2020.00274

[33] Debajyoti Mondal, Manishankar Mondal, Chanchal K. Roy, Kevin A. Schneider, Yukun Li, and Shisong Wang. 2019. Clone-World: A visual analytic system for large scale software clones. *Vis. Informatics* 3, 1 (2019), 18–26.

[34] Golam Mostaeen, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. 2018. On the Use of Machine Learning Techniques Towards the Design of Cloud Based Automatic Code Clone Validation Tools. In *18th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2018, Madrid, Spain, September 23-24, 2018*. IEEE Computer Society, 155–164.

[35] Golam Mostaeen, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. 2019. CloneCognition: machine learning based code clone validation tool. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE*. ACM, 1105–1109.

[36] Sakib Mostafa and Debajyoti Mondal. 2021. On the Evolution of Neuron Communities in a Deep Learning Architecture. *CoRR* abs/2106.04693 (2021). arXiv:2106.04693

[37] Sammar Moustafa, Mustafa Y. ElNainay, Nagwa El Makky, and Mohamed S. Abougabal. 2018. Software bug prediction using weighted majority voting techniques. *Alexandria Engineering Journal* 57, 4 (dec 2018), 2763–2774. https://doi.org/10.1016/j.aej.2018.01.003

[38] Behzad Soleimani Neysiani and Seyed Morteza Babamir. 2020. Automatic Duplicate Bug Report Detection using Information Retrieval-based versus Machine Learning-based Approaches. In *Proc. of the 6th International Conference on Web Research (ICWR)*. 288–293. https://doi.org/10.1109/ICWR49608.2020.9122288

[39] C. Lakshmi Prabha and N. Shivakumar. 2020. Software Defect Prediction Using Machine Learning Techniques. In *Proceedings of the 4th International Conference on Trends in Electronics and Informatics, ICOEI 2020*. Institute of Electrical and Electronics Engineers Inc., 728–733. https://doi.org/10.1109/ICOEI48184.2020.9142909

[40] Rakesh Rana, Miroslaw Staron, Christian Berger, Jörgen Hansson, Martin Nilsson, and Wilhelm Meding. 2014. The Adoption of Machine Learning Techniques for Software Defect Prediction: An Initial Industrial Validation. In *Knowledge-Based Software Engineering - 11th Joint Conference, JCKBSE 2014, Volgograd, Russia, September 17-20, 2014. Proceedings (Communications in Computer and Information Science, Vol. 466)*, Alla G. Kravets, Maxim Shcherbakov, Marina V. Kultsova, and Tadashi Iijima (Eds.). Springer, 270–285.

[41] Zeeshan Ali Rana, M. Awais Mian, and Shafay Shamail. 2015. Improving Recall of software defect prediction models using association mining. *Knowledge-Based Systems* 90 (dec 2015), 1–13. https://doi.org/10.1016/j.knosys.2015.10.009

[42] Stefan Strüder, Mukelabai Mukelabai, Daniel Strüber, and Thorsten Berger. 2020. Feature-oriented defect prediction. In *Proc. of the 24th ACM International Systems and Software Product Line Conference*, Roberto Erick Lopez-Herrejon (Ed.). ACM, 21:1–21:12.

[43] Jacek undefinedliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When Do Changes Induce Fixes? *SIGSOFT Softw. Eng. Notes* 30, 4 (may 2005), 1–5. https://doi.org/10.1145/1082983.1083147

[44] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605.

[45] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. 2010. Naïve Bayes. *Encyclopedia of machine learning* 15 (2010), 713–714.

[46] Shamima Yeasmin, Chanchal K. Roy, and Kevin A. Schneider. 2014. Interactive visualization of bug reports using topic evolution and extractive summaries. In *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*. Institute of Electrical and Electronics Engineers Inc., 421–425. https://doi.org/10.1109/ICSME.2014.66

[47] Yongli Zhang. 2012. Support vector machine classification algorithm and its application. In *Communications in Computer and Information Science*, Vol. 308 CCIS. Springer, Berlin, Heidelberg, 179–186. https://doi.org/10.1007/978-3-642-34041-3_27