Journal of Autonomous Intelligence (2023) Volume 6 Issue 1
doi: 10.32629/jai.v6i1.559

ORIGINAL RESEARCH ARTICLE

A hybrid software defects prediction model for imbalance datasets using machine learning techniques: (S-SVM model)

Mohd. Mustaqeem*, Tamanna Siddiqui

Department of Computer Science, Aligarh Muslim University (AMU), Aligarh, U.P, India. E-mail: mohdmus*taqeem34@gmail.com*

ABSTRACT

Software defect prediction (SDP) is an essential task for developing quality software, and various models have been developed for this purpose. However, the imbalanced nature of software defect datasets has challenged these models, resulting in decreased performance. To address this challenge, the author has proposed a hybrid machine learning model that combines Synthetic Minority Oversampling Technique (SMOTE) with Support Vector Machine (SVM)—SMOTE-SVM (S-SVM) model. The author has empirically examined SDP using multiple datasets (CM1, PC1, JM1, PC3, KC1, EQ and JDT) from the PROMISE and AEEEM repositories. The experimental study indicates that the S-SVM model involved training and compared with previously developed balanced and imbalanced test datasets using four evaluation metrics: Precision, Recall, F1-score, and Accuracy. For the balanced dataset, the S-SVM model achieved precision values ranging from 70 to 96, recall values ranging from 52 to 94, F1-score values ranging from 67 to 90, and accuracy values ranging from 69 to 98. For the imbalanced dataset, the S-SVM model achieved precision values ranging from 60 to 93, recall values ranging from 64 to 97, F1-score values ranging from 69 to 91, and accuracy values ranging from 67 to 87. The proposed S-SVM model outperforms other models' ability to classify and predict software defects. Therefore, the hybridisation of SMOTE and SVM improved the model's ability to categories and predict balanced and imbalanced datasets when sufficient defective and non-defective data is provided.

Keywords: Software Defect Prediction (SDP); SVM; SMOTE; Empirical Software Engineering; Software Quality; Balanced & Imbalanced Learning

ARTICLE INFO

Received: 3 March, 2023
Accepted: 7 May, 2023
Available online: 16 June, 2023

1. Introduction

Nowadays, the software industry is rising. The demand for software is very high in the market, and challenging to maintain software quality. The first focus should be the quality of the software[1]. In software development, there are many test factors. However, SDP is one of the most critical factors in measuring the software's quality; due to this, it developed an interest in researchers working on it. The SDP's primary goal is to anticipate defects in defect-prone modules. Detecting the software defects later may lead to considerable losses to the company and software users. Nevertheless, early detection of the defects and those modules which are less priority and risk-prone for the project may reduce the loss and other software quality testing expenses. To get better prediction results, SDP has to work on multiple portions, like enhancing the accuracy. Necessary data attributes include imbalanced & balanced datasets ratio, performance measurements, and classification algorithms[2]. Researchers have proposed various models of SDP over the years by using new technology, tools, and techniques. Various models are developed based on the same training and testing version of the project's dataset. Some are based on cross-project[1–6]. Predicting software defects by hybridising machine learning techniques has enhanced the SDP. Traditional and conventional defect prediction methods have only worked on a single version of the project, and model training is limited by the similarity of the trained and test datasets[3]. The empirical research is based on the hybridisation of machine learning technologies and dataset balancing techniques that can first resolve the imbalanced classes in the dataset. The output deviates much from the actual and accurate result. That is why balancing the data is very important to get real results. Secondly, after balancing data, the author has used a machine learning technique to predict the defects in the given datasets so that SDP has precise and high accuracy. The author has trained the model through seven datasets of the PROMISE repository (publically available) and compared the trained model with another SDP model that was developed earlier[7,8]. The trained model has a high rate of accuracy than others. After that, the author divided the given dataset into balanced and imbalanced parts in combined form. The trained model is implemented in both sections successfully. Four measurements of performance, precision, recall, F1-score, and accuracy, have been calculated. The trained model has shown deviation in the accuracy towards the balanced dataset and has provided many accurate results of SDP.

The manuscript can be categorised in the following manner. Section 2 represents related work in which conventional approaches are discussed. Section 3 describes the experimental work. Section 4 presents the results & discussions, and the final Section 5 includes a conclusion with a future work window.

2. Related work

Several traditional models have been developed for SDP in recent years, like just-in-time, software metrics defects prediction, optimisation techniques, and other hybrid approaches to defect prediction. The author has used an imbalanced dataset in their work so that accuracy is ultimately high, which is not a good approach. In his manuscript[7], the author mentioned using Naïve Bayes and Logistic Regression as a classifier to develop a defect prediction model. Other classification algorithms, Decision Tree (DT), Random Forest (RF), and ANN, can be used. Metrics assessment can be implemented to measure the performance of the classification metrics[7,9].

2.1 Traditional and conventional software defect prediction

Software Development Life Cycle (SDLC) is essential for developing software. The author has seen recently that the early detection of software defects is a much-needed and challenging task for the project coordinator[10]. In this modern world, new features have been included in domains of complex problems, enhancing the software's performance uncertainty. Despite checking all the documents carefully in an organised manner, a few warnings and bugs are inescapable, which can decrease the performance of the software.

According to Hassan *et al.*[11], software applications are tested in unstable situations for a particular period, called reliability probability. Traditional SDP models use software, process, and object-oriented metrics[4,12]. The dataset of SDP is divided into two categories. First part is training, and the second is testing. Within project, the single dataset is divided into train and test. Manjula and Florence[13] mentioned in their manuscript, defects can be predicted through a deep neural network (DNN) hybrid approach. The previous authors have experimented using NASA PROMISE datasets in their manuscript. Their accuracy is enhanced up to 98% (approx) because they have used imbalanced datasets even though they have not broken their dataset into the test and trained formats and calculated from the same data with which they train the model.

According to Jayanthi and Florence[3], an investigation of PROMISE NASA datasets repository using NN-based classification. They have claimed their performance enhancement up to 97.2% AUC based, but their dataset is imbalanced. According to Chidamer[14], software metrics predict defects. Whether object-oriented design metrics are suitable for defect prediction or not was examined by

Basili[15]. He has collected eight software applications to carry out his work. Alsawalqah *et al.*[16] said in their manuscript that defects can be predicted using the hybrid SMOTE-Ensemble technique using four datasets. They have used conventional methods to balance it and enhance the model's accuracy. He *et al.*[17] used their manuscript's adaptive synthetic sampling (ADASYN) approach. This approach improves the learning of data distribution through two methods: first, it reduces the bias that appears due to an imbalanced class, and second, it shifts adaptively towards the classification decision boundary. They have done their analysis on five evaluation metrics.

Nevertheless, this approach has some limitations, i.e., each region may only include one minority example for sparsely distributed minority cases. Because of its versatility, ADASYN precision may degrade[18]. According to Mirzaei *et al.*[19], their research has balanced the imbalanced dataset using the under-sampling technique DBSCAN algorithms. They have worked on 15 imbalanced datasets, compared them with six other algorithms, and got good results, but their model lacks performance with varying density clusters. It also suffers from high dimensionality data. The research of Hasanin and Khoshgoftaar[20] has shown the effects of random under-sampling with class imbalance big data and converting the data into a 50:50 ratio to enhance performance. However, this random under-sampling can discard the potentially helpful information that can be useful in classifiers. It may be that biased samples. Bach *et al.*[21] have proposed a method to balance the imbalanced dataset using under-sampling. They have removed high-density and low-density information to get better results, but their model can be lost various critical information that is much needed for classification. Moreover, Sawangarreerak and Thanathamathee[22] used random forest and sampling techniques to balance the imbalance university student depression dataset, but this model performs inferior to the complex dataset. It also requires significant memory storage for information retention.

Due to the highly imbalanced data distribution, it is challenging to tackle the complex problem. In this situation, many above-mentioned conventional algorithms can classify the significant ones and ignore the small ones in various cases of SDP problems. Therefore, it may lead to the poor performance of the classifier. This problem can be handled using the SMOTE algorithm for SDP in various imbalanced datasets. The experimental result of this hybrid model shows far better results in the performance and accuracy of defects prediction.

3. Experimental studies

The experimental investigations for SDP employing the suggested hybrid learning model are described in the following subsections. This paper uses a hybrid technique, SMOTE and SVM, for dataset balance and classification, using Python and Spyder, with SK learn as the coding library. The author has used the PROMISE software defect dataset repository for this research[23].

3.1 Dataset description and data processing details

CM1, JM1, PC3, and KC1 datasets represent what the author has extracted and divided into training and testing datasets used for SDP. Though various studies have been carried out on these datasets, the author has presented comparative studies of SDP on balanced and imbalanced datasets that show the true significance of the proposed hybrid model. Moreover, the datasets used for computation present the following features in **Table 1**.

The dataset description may include information on dataset names, modules, and defective and non-defective classes, with their percentage shown in **Table 2**. The other dataset AEEEM (Appraisal-Based Evaluation of Effort Models), is a collection of software projects annotated with information about the effort required to fix different defects. The author has used the EQ and JDT dataset to help researchers and practitioners evaluate software defect prediction models and techniques. The features included in the dataset are a combination of static and dynamic metrics, which provide information about various aspects of the code, including its complexity, size, and maintainability. **Table 2** shows the features description.

Table 1. PROMISE SDP features details

Features name	Description
LOC	Module total number of line count
Loblank	Number of total blank lines in the module
D	Difficulty measurement
v(g)	Cyclomatic complexity measurement (McCabe)
B	Effort's estimation
N	Module numeral operators
Iv(g)	(McCabe) complexity design analysis
V	Volume
Branchcount	Number of total branch in the software module
L	Length of program
E	Measurement
I	Intelligence measurement
total_op	Number of total operators
Locodeandcomment	Number of the total line of code and comments
Total_opnd	Number of total operators
T	Estimator of time
Defects/Problems	Defects regarding information, whether it is present or not
uniq_op	Number of total unique operators
uniq_opnd	Number of total unique operand
Ev(g)	McCabe complexity
Locomment	*Software module line of comment*

Table 2. Project AEEEM datasets attributes

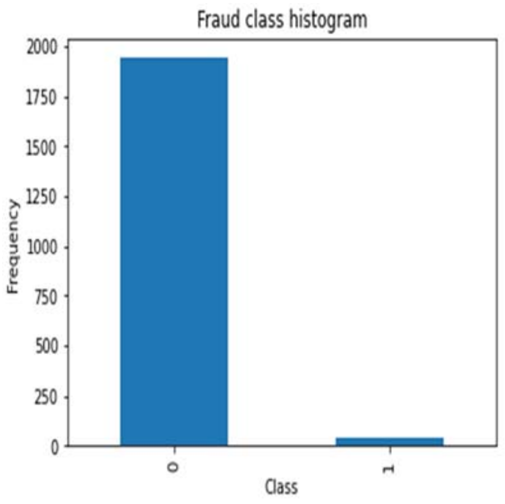
Dataset name	Total element	Non-defective	Defective	Percentage non-defective	Percentage defective
CM1	1,988	1,942	46	97.6	2.4
PC1	705	644	61	91.3	8.7
JM1	7,782	6,110	1,672	78.5	21.5
PC3	1,077	943	134	87.5	12.5
KC1	145	85	60	58.6	41.3
EQ	324	195	129	60.1	39.8
JDT	997	791	206	79.3	20.6

Table 3. Dataset detail division

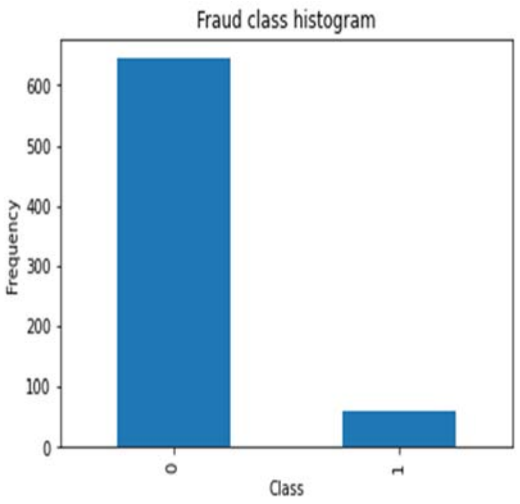
Metrics	Description	No. of attributes
Source Code Metric (SCM)	Source code computation	[17]
Churn of Source Code Metric (COSCM)	Analysis of SCM-based code churn as an artificial metrics	[17]
Entropy of Source Code Metric (EOSCM)	Entropy-based SCM computed on artificial metrics	[17]
Previous Defects Metric (PDM)	The revisions and defective metrics computation	[5]
Entropy of Changes Metric (EOCM)	Entropy-based changes for artificial metrics computation	[5]

Table 3 shows the characteristics of datasets used in the study on software defect prediction. The datasets are named CM1, PC1, JM1, PC3, KC1, EQ and JDT, each containing various elements. Each dataset's total number of elements is shown in the "Total element" column. The following two columns show the number of non-defective and defective elements in each dataset. The CM1 dataset has 1,942 non-defective and 46 defective features. The "Percentage non-defective" and "Percentage defective" columns show the proportion of non-defective and defective parts in each dataset, expressed as a percentage. For instance, the PC3 dataset has 943 non-defective elements, approximately 87.5% of the total

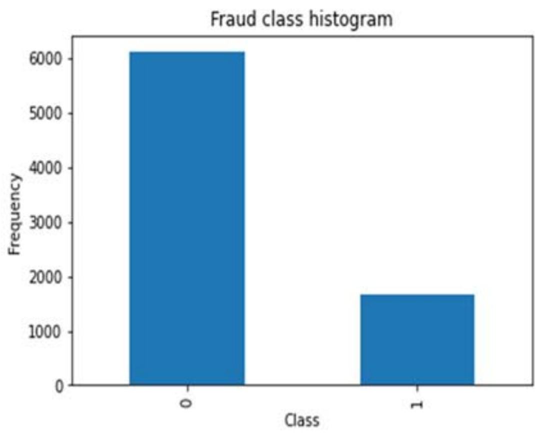
features, while the remaining 12.5% are defective (134). These statistics are essential in software defect prediction because the datasets are usually imbalanced, with a higher proportion of non-defective elements than defective elements. This imbalance can affect the performance of prediction models. Therefore, it is essential to consider the balance of the dataset when building and evaluating models. Defective and non-defective classes with graphs can be shown below. **Figure 1(a)** shows CM1, **Figure 1(b)** shows PC1 and **Figure 1** shows PC3 (c), **Figure 1(d)** shows JM1, **Figure 1(e)** shows KC1, **Figure 1(f)** shows EQ & **Figure 1(g)** shows JDT.



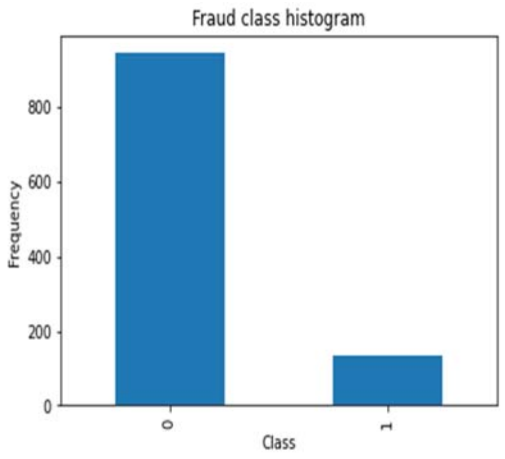
(a) CM1 defective & non-defective



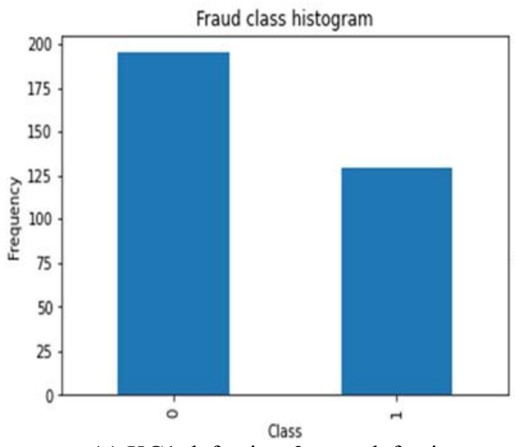
(b) PC1 defective & non-defective



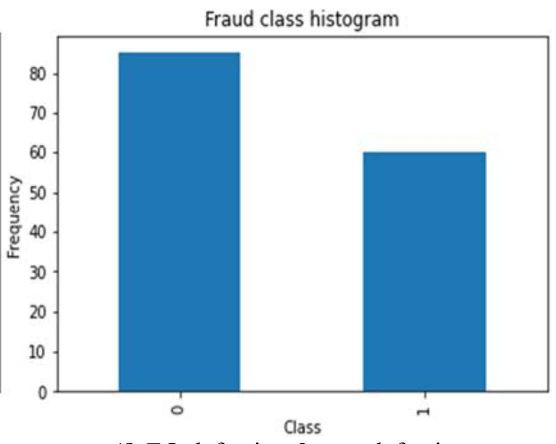
(c) JM1 defective & non-defective



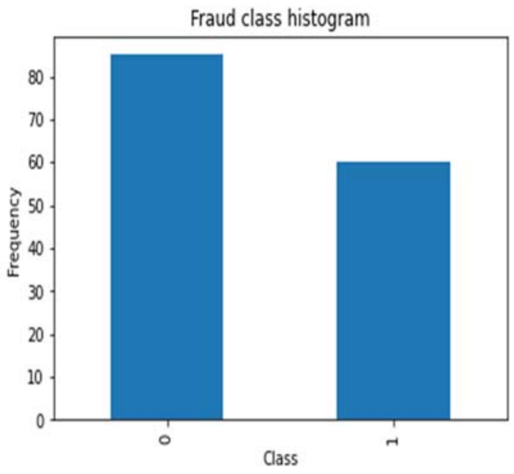
(d) PC3 defective & non-defective



(e) KC1 defective & non-defective



(f) EQ defective & non-defective



(g) JDT defective & non-defective

Figure 1. CM1, PC1, PC3, JM1, KC1, EQ and JDT defective & non-defective results.

3.2 Proposed model

The given section represents the proposed approach for SDP. The model has been divided into two sections. In the first section, the author has balanced the imbalanced dataset using SMOTE, and in the second section, SVM has been applied for classification.

3.2.1 SMOTE

Highly imbalanced datasets were previously used to predict software defects. Classification algorithms have great difficulty detecting small classes. As a result, the imbalanced dataset has to be balanced for precise SDP. There is a variety of balancing techniques. However, the proposed model implemented SMOTE, an oversampling strategy presented in the research of Kovács *et al.*[24]. It creates the synthetic data samples and modifies class allotment by oversampling the small class as a substitute of oversampling using replacement. These data samples are created using attribute space operations. The small class is oversampled by removing every data value and creating synthetic values with the line segments that connect all nearest neighbors small k classes.

Its generation begins with selecting k-nearest neighbors, followed by developing synthetic samples based on the variations between feature vectors of the value in question. It's the nearest neighbor. It adds the interpretation to the feature vector under consideration by multiplying it with a random value between OFF (0) & ON (1). A random location with a line segment between 2 distinct characteristics is chosen in the form of a result. As a result, SMOTE has expanded the data region of minority classes and pushed the class's decision region to become more generic.

When one class (the minority class) is greatly underrepresented relative to the other type in a binary categorisation task, SMOTE is a prominent strategy employed to tackle the issue of class disparity in artificial intelligence (the majority class). To rebalance the class distribution, SMOTE creates artificial representations of the minority class and adds them to the data.

Mathematically, SMOTE can be represented as follows:

Given:

A minority class example xi with features $xi_1, xi_2, \cdots, xi_n$,

A k-nearest neighbor of xi, denoted as xi_{nn}, with features $xi_{nn_1}, xi_{nn_2}, \cdots, xi_{nn_m}$,

A random number r between 0 and 1.

The synthetic example $xi_{synthetic}$ with features $xi_{synthetic_1}$, $xi_{synthetic_2}$, …, $xi_{synthetic_n}$ is generated as follows for a specific feature j:

$$xi_{synthetic_j} = xi_j + r * (xi_{nn_j} - xi_j)$$

where xi_j is the feature value of xi for feature j, and xi_{nn_j} is the feature value of xi_{nn} for feature j.

$$E\left(xi_{synthetic_j}\right) = E(xi_j) - E(r)E(xi_j) + E(r)E(xi_{nn_j})$$

$$E\left(xi_{synthetic_j}\right) = \frac{1}{2}\left(E(xi_j) + E\left(xi_{nn_j}\right)\right)$$

The author demonstrated that $E\left(xi_{synthetic_j}\right) = E(xi_j)$, indicating that the anticipated amount of the synthetic instances generated by stable SMOTE equals both the initial minority class instances and the artificial incidents produced by SMOTE, following Blagus and Elreedy's conclusion that $E\left(xi_j\right) = E\left(xi_{nn_j}\right)$. SMOTE is a simple yet effective technique for addressing the class imbalance in machine learning datasets.

3.2.2 SVM

Machine learning algorithms can be categorised as follows:

- Supervised
- Unsupervised
- Semi-supervised

It can also solve issues like regression, classification, and clustering. The SDP challenge problem is similar to a classification one. The data has two forms: defects and the other without.

There are numerous algorithms offered to attain classification problems as artificial intelligence progresses, such as Logistic Regression, Decision Tree, Random Forest, and so on; for work, the SVM[25] is the best-supervised learning paradigm with advantages:

- To prevent overfitting, L2 Regularization is used,
- Compatible with small datasets and give appropriate results,
- To fit the complex function and relationships among the features of various Kernel-Tricks,
- Handled the Non-linearity of the data,
- Model stability can be maintained by using the hyperplane dividing rule,
- High-dimensionality of the data can be managed.

SVM's goal is to maximise the classification decision boundaries. The hyperplane separates classes, like +1 (positive class) or –1 (negative class), rather than minimising prediction error. High-dimensional (n-dimensional) datasets that cannot be viewed can be employed in the SVM. As a result, processing data with n = 2 (2D) can be represented on a 2-Dimensional graph (**Figure 2**), with the hyperplane line that can separate classes. Furthermore, when the data is n-dimensional, the hyperplane is an (n – 1) vector function, which may be represented mathematically.

$$y = p_0 w_0 + p_1 w_1 + \cdots + p_{n-1} w_{n-1} + c \tag{1}$$

It can also show as:

$$y = P^T W + c \tag{2}$$

where W a weight vector, X an input feature vector, and b is bias. Once a hyperplane is found, the hypothesis based on SVM can be formulated below.

$$f(y) = \begin{cases} Class\ 1\ if y \leq 0 \\ Class\ 2\ if y > 0 \end{cases} \tag{3}$$

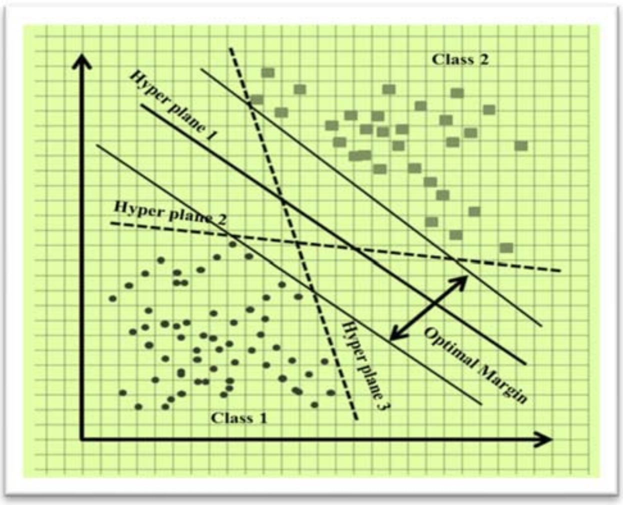


Figure 2. 2D graph Hyperplane.

Many hyperplanes can be drawn by adjusting p and c, but the hyperplane with the best margin will be chosen. The ideal margin is the maximum possible perpendicular distance between the hyperplane and each class. For example, hyperplane 1 in **Figure 2** has the best margin from classes 1 and 2. The optimal margin is established by minimising the cost or objective function. For example, the cost function is defined below:

$$J(p) = \frac{1}{2}\|p\|^2 + \frac{1}{n}\sum_{i=0}^{n} \max\left(0, (1 - y_i * (P^T w + c)\right) \tag{4}$$

However, the predictions are correct, and data is correctly categorised through hypothesis; SVM is fined for all. y_i close to the boundary$(0 < y_i < 1)$. Minimising $J(p)$ is the primary goal in terms of optimal P; therefore, the author has extracted the gradient of the cost function by differentiating equation 4 concerning P.

$$\nabla_p J(P) = \frac{\partial J(P)}{\partial P}$$
$$= \frac{1}{n}\sum_{i=0}^{n}\begin{cases} P if \max(0, (1 - y_i * (P^T W + C)) \\ P - y_i w_i otherwise \end{cases} \tag{5}$$

As far as the author has calculated $\nabla_p J(P)$, now it can be updated weights (P) using equation 3:

$$P_{new} = P_{old} - \alpha[J(P)] \tag{6}$$

Repeat till the smallest $J(P)$ found.

Because data is rarely linearly separable, the author must draw a decision boundary between the classes rather than separating them by a hyperplane. Dealing with the dataset's non-linearity, convert equation 2 in decision boundary.

$$y = P.\phi(W) + c \tag{7}$$

In equation 7, $\phi(W)$, the kernel function is what it's called. Various kernel functions are available to construct SVM, including linear, polynomial, exponential, and so on. However, the proposed model employs the Radial Basis Function (RBF)[26]. That is based on Euclidean distance, and the boundaries smoothness is defined by a parameter[27].

$$\phi(w) = \exp\left(-\frac{\|w - \overline{w}\|^2}{2\sigma^2}\right) \tag{8}$$

where, $\|w - \overline{w}\|^2$ is the Euclidean square, and the distance w between every single observation and the training sample's means $\overline{w}$.

```
Algorithm
Start
Step-1    Import Libraries // Import all the sufficient libraries and files to implementation
Step-2    Data = read (File) // Read the available data
Step-3    Initialise weights $W$ and bias b with any arbitrary number
Step-4    Feature Optimisation // Select the appropriate attributes for the prediction
Step-5    Train DataSet, Testing Data Sets = Data. Split (Ratio) // Splitting the data in two-part, one
          model training, another model testing in the suitable ratio
Step-6    Define $y$ // Equations 2 or 7
Step-7    Define $h(y)$ // Equation 3
Step-8    Define $J(P)$ // Equation 4
Step-9    Calculate $\nabla_p J(P)$ // Equation 5
Step-10   Repeat for minimum $J(P)$:
          Update $P$ // minute variations in weights to find the optimal margin (equation 6)
          Call Steps 6–9 // Re-calculate the $\nabla_P J(P)$ by making a prediction using updated $P$
          End
Step-11   Accuracy calculation // To check the efficiency of the model
              $Accuracy \leftarrow \frac{Totalno.\ of correct predictions}{Totalno. of predictions} * 100$
Step-12   Output // Accuracy of the model as output
Stop
```

3.3 Performance evaluation

Model performance testing evaluation is one of the significant tasks. It shows the operational performance with test data. For measuring performance, the confusion matrix technique is used. This technique stores predicted and actual class data to obtain the classification results. The confusion matrix helps compute the most common five evaluation metrics of SDP used in the proposed model.

Table 4. Confusion matrix

Actual value	**Predicted value**	
	Non-defective	**Defective**
Non-defective	False negative (FN)	True positive (TP)
Defective	True negative (TN)	False positive (FP)

1) Accuracy: The number of correct answers given in a classification. In a confusion matrix, there are two types of answers: True positive (TP) (where the defective value has been identified as defective) and True negative (TN) (where the defective value has not been determined as defective) (where non-defective value identified as non-defective). The following formula can be used to calculate accuracy:

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

2) Precision: Measure correctly answered defective values to the total predicted defective values in the given classification. Can be represented as:

$$Precision = \frac{(TP)}{(TP) + (FP)}$$

3) Recall: Measure truly answered defective values, all actual defective values in the classification. Can be represented as:

$$Recall = \frac{(TP)}{(TP) + (FN)}$$

4) F1-score: Weighted average Precision and Recall. It can be represented as:

$$F = \frac{2 * Precision * Recall}{Precision + Recall}$$

4. Implementation

The proposed model can be implemented in the following ways: the author has coded in Python and used the Spyder platform. The author has used the following datasets to test the suggested model.

4.1 Using CM1 dataset

In **Table 5**, confusion matrix CM1 datasets with the proposed model S-SVM.

4.2 Using PC1 dataset

In **Table 6**, confusion metrics PC1 dataset with S-SVM.

4.3 Using JM1 dataset

In **Table 7**, confusion metrics demonstration with JM1 dataset using S-SVM.

4.4 Using PC3 dataset

In **Table 8**, confusion metrics demonstration with the PC3 dataset using S-SVM.

Table 5. Confusion matrix for CM1 dataset using S-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	366	23
	Non-defective	91	297

Table 6. Confusion matrix for PC1 dataset using S-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	11	8
	Non-defective	66	73

Table 7. Confusion matrix for JM1 dataset using SMOTE-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	1,016	230
	Non-defective	430	768

Table 8. Confusion matrix for PC3 dataset using S-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	151	16
	Non-defective	69	142

The author examined merging both techniques sequentially to improve the SDP's performance in the current manuscript. A hybrid approach called SMOTE + SVM is used to achieve the best results, as shown in **Figure 3**. The SMOTE algorithm aims to generate synthetic values for a minority class that is a defect class to the dataset. It can distribute the number of occurrences in each class more evenly, allowing the classifier to consider the minority class. The SVM classifier will then be trained using a new oversampled dataset. The SVM classifiers' defect detection rates have produced the most promising and genuine findings.

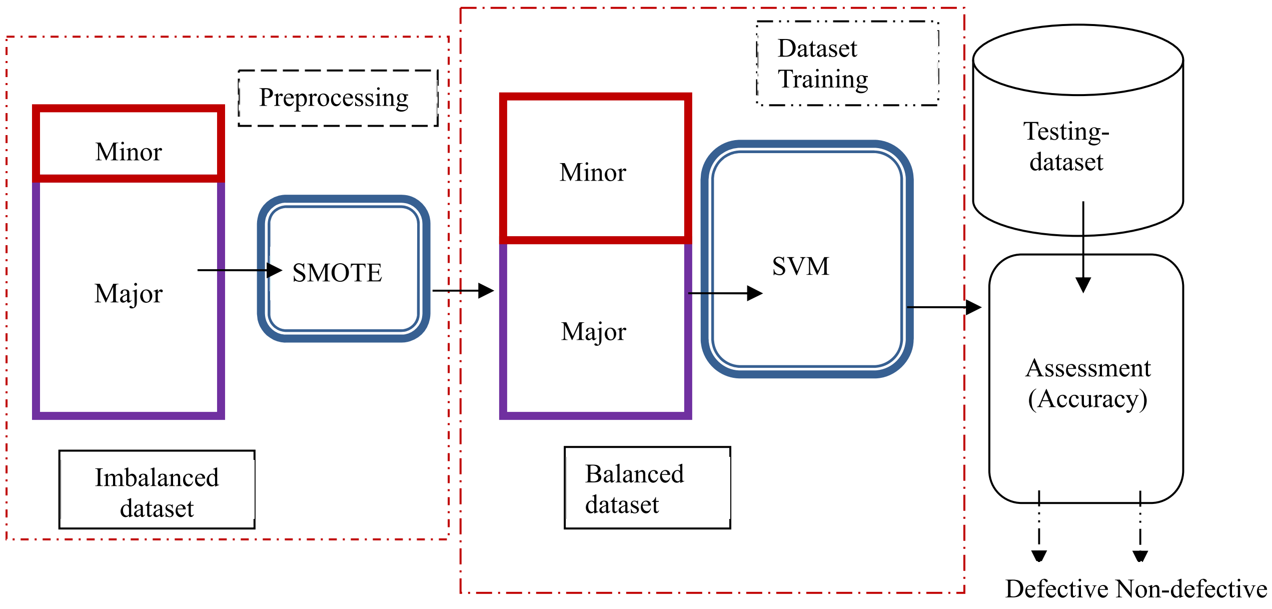


Figure 3. Hybrid S-SVM model for SDP.

4.5 Using KC1 dataset

In **Table 9**, confusion metrics demonstration with the KC1 dataset using SMOTE-SVM.

Table 9. Confusion matrix for KC1 dataset using SMOTE-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	4	9
	Non-defective	1	20

Table 9 represents the confusion matrix for the KC1 dataset using S-SVM for handling imbalanced data. In the context of binary classification, the confusion matrix summarises the performance of a machine learning model by comparing its predicted labels to the true labels of the dataset. The rows in the table represent the actual labels of the dataset, while the columns represent the predicted labels by the S-SVM model. The entries in the table represent the counts of instances that fall into each category: The top-left entry (4) represents the number of cases that are truly "Defective" in the dataset and were correctly predicted as "Defective" by the S-SVM model. The top-right entry (9) represents the number of instances that are actually "Defective" in the dataset but were mistakenly predicted as "Non-defective" by the S-SVM model. The bottom-left entry (1) represents the number of instances that are actually "Non-defective" in the dataset but were mistakenly predicted as "Defective" by the S-SVM model. The bottom-right entry (20) represents the number of instances that are true "Non-defective" in the dataset and were correctly predicted as "Non-defective" by the S-SVM model. From this table, the author can see that the model correctly predicted the "Non-defective" instances with high accuracy (20 out of 21 instances). The accuracy for "Defective" samples is lower (4 out of 13 instances). This suggests that the

S-SVM model may have difficulty accurately predicting the "Defective" instances in the KC1 dataset. Further analysis and tuning of the model may be necessary to improve its performance.

4.6 Using AEEEM project EQ dataset

In **Table 10**, confusion metrics are demonstrated with the EQ dataset using SMOTE-SVM.

Table 10. Confusion matrix for EQ dataset using SMOTE-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	29	17
	Non-defective	6	26

Table 10 represents the confusion matrix for the EQ dataset using the proposed model for handling imbalanced data. The EQ dataset is a project EQ dataset from the AEEEM repository, commonly used in software defect prediction research. In the context of binary classification, the confusion matrix summarises the performance of a machine learning model by comparing its predicted labels to the true labels of the dataset. The rows in the table represent the actual labels of the dataset, while the columns represent the predicted labels by the S-SVM model. The top-left entry (29) represents the number of instances that are truly "Defective" in the dataset and were correctly predicted as "Defective" by the S-SVM model. The top-right entry (17) represents the number of instances that are actually "Defective" in the dataset but were mistakenly predicted as "Non-Defective" by the S-SVM model. The bottom-left entry (6) represents the number of instances that are actually "Non-defective" in the dataset but were mistakenly predicted as "Defective" by the S-SVM model. The bottom-right entry (26) represents the number of instances that are true "Non-defective" in the dataset and were correctly predicted as "Non-defective" by the S-SVM model. From this table, the model correctly predicted the "Defective" instances with relatively high accuracy (29 out of 46). The accuracy for "Non-defective" samples is also relatively high (26 out of 32 instances). However, some instances were misclassified, with 17 instances of "Defective" being predicted as "Non-defective" and six instances of "Non-defective" being predicted as "Defective". Further analysis and fine-tuning of the model may be necessary to improve its performance on the EQ dataset.

4.7 Using AEEEM project JDT dataset

In **Table 11**, confusion metrics demonstration with the JDT dataset using SMOTE-SVM.

Table 11. Confusion matrix for EQ dataset using SMOTE-SVM

		Predicted	
		Defective	Non-defective
Actual	Defective	134	26
	Non-defective	5	152

Table 11 represents the confusion matrix for the JDT dataset using a proposed model for handling imbalanced data, referred to as SMOTE-SVM. The JDT dataset is a software defect prediction dataset. In the context of binary classification, the confusion matrix summarises the performance of a machine learning model by comparing its predicted labels to the true labels of the dataset. The rows in the table represent the actual labels of the dataset, while the columns represent the predicted labels by the S-SVM model. The entries in the table represent the counts of instances that fall into each category: The top-left entry (134) represents the number of instances that are truly "Defective" in the dataset and were correctly predicted as "Defective" by the S-SVM model. The top-right entry (26) represents the number of instances that are actually "Defective" in the dataset but were mistakenly predicted as "Non-defective" by the S-SVM model. The bottom-left entry (5) represents the number of instances that are actually

"Non-defective" in the dataset but were mistakenly predicted as "Defective" by the S-SVM model. The bottom-right entry (152) represents the number of instances that are true "Non-defective" in the dataset and were correctly predicted as "Non-defective" by the S-SVM model. Similar to the previous tables you provided, this confusion matrix helps evaluate the performance of the S-SVM model on the JDT dataset. From this table, the model correctly predicted the "Defective" instances with relatively high accuracy (134 out of 160). The accuracy for "Non-defective" samples is also relatively high (152 out of 157 instances). However, some instances were misclassified, with 26 instances of "Defective" being predicted as "Non-defective" and five instances of "Non-defective" being predicted as "Defective". Further analysis and fine-tuning of the model may be necessary to improve its performance on the JDT dataset. **Figure 3** shows the complete model demonstration. **Table 12** shows the imbalanced and balanced dataset of non-defective and defective classes. The balancing of the dataset has been experimentally performed by using SMOTE technique.

Table 12. Non-defective and defective classes measure

Dataset		Non-defective	Defective
CM1	Imbalanced	363	35
	Balanced	499	499
PC1	Imbalanced	555	65
	Balanced	144	144
JM1	Imbalanced	4,891	1,334
	Balanced	6,110	6,110
PC3	Imbalanced	659	94
	Balanced	943	943
	Imbalanced	85	60
KC1	Balanced	85	85
EQ	Imbalanced	195	129
	Balanced	195	195
JDT	Imbalanced	791	206
	Balanced	791	791

5. Results and discussion

SDP models developed earlier have broken their data into test and train sets. A non-defective majority dataset can perform well for the non-defective test dataset, but the model will not perform well for the defective majority dataset. If the imbalanced dataset is taken, the performance will be much high, but that will not be very objective. So, building a model with a balanced dataset is necessary. The author has used four PROMISE REPOSITORY and two AEEM project datasets (CM1, PC1, JM1, PC3, KC1, EQ and JDT). Initially, the datasets were imbalanced, but the author applied a balancing technique to ensure the performance was unbiased and genuine. The author has experimented in two parts for the model's comparative analysis and performance measurement. First, the author experimented with the imbalanced dataset and applied a hybrid technique to balance and classify the dataset. And the results show the significant differences between using balanced and imbalanced datasets. The results can be shown in **Table 12**. There are seven datasets, each broken into train and test sets. So, expected two major simulations can occur that are listed below.

5.1 For CM1 dataset

1) Balanced dataset: The model's performance is shown in **Table 12**, which has a precision of 80, a recall value of 94 with an F1-score is 87 and an accuracy of 98.

2) Imbalanced dataset: The model's performance can be achieved in **Table 12**, which has a precision of 93, a recall value of 77, an F1-score of 84, and an accuracy of 84.

5.2 For PC1 dataset

1) Balanced dataset: The model's performance is shown in **Table 12**, which has a precision of 92, a recall value of 52 with an F1-score is 67, and an accuracy is 92.

2) Imbalanced Dataset: The model's performance can be achieved in **Table 12**, which has a precision of 66, a recall value of 95, an F1-score of 78 and an accuracy of 74.

5.3 For JM1 dataset

1) Balanced Dataset: The model's performance is shown in **Table 12**, which has a precision of 70, a recall value of 82 with an F1-score is 75, and an accuracy is 69.

2) Imbalanced Dataset: The model's performance can be achieved in **Table 12**, which has a precision of 77, recall value of 64, F1-score of 70 and accuracy of 73.

5.4 For PC3 dataset

1) Balanced dataset: The model's performance is shown in **Table 12**, which has a precision of 69, a recall value of 90 with an F1-score is 78, and an accuracy is 78.

2) Imbalanced dataset: The model's performance can be achieved in **Table 12**, which has a precision of 90, recall value of 67, F1-score of 77 and accuracy of 79, respectively.

5.5 For KC1 dataset

1) Balanced dataset: The model's performance is shown in **Table 12**, which has a precision of 80, a recall value of 31 with an F1-score is 44, and an accuracy is 71.

2) Imbalanced dataset: The model's performance can be achieved in **Table 12**, which has a precision of 69, recall value of 95, F1-score of 80 and accuracy of 67, respectively.

5.6 For EQ dataset

1) Balanced dataset: The model's performance is shown in **Table 12**, which has a precision of 83, a recall value of 63 with an F1-score is 72, and an accuracy is 77.

2) Imbalanced dataset: The model's performance can be achieved in **Table 12**, which has a precision of 60, recall value of 81, F1-score of 69 and accuracy of 71, respectively.

5.7 For JDT dataset

1) Balanced dataset: The model's performance is shown in **Table 12**, which has a precision of 96, a recall value of 84 with an F1-score is 90, and an accuracy is 90.

2) Imbalanced dataset: The model's performance can be achieved in **Table 12**, which has a precision of 85, recall value of 97, F1-score of 91 and accuracy of 87, respectively.

In the last, the author can show that the performance measure of SDP with an imbalanced dataset is much higher than the balanced one. And the hybrid model performance decreases when the model is treated with the balanced dataset. That successfully shows the difference among the performance with different measurable metrics.

Table 13. Evaluation measures using considered metrics with imbalanced & balanced datasets

Hybrid model		The performance measure with SMOTE-SVM hybrid model (S-SVM model)			
datasets		Precision	Recall	F1-score	Accuracy
CM1	Balanced	80	94	87	98
	Imbalanced	93	77	84	84
PC1	Balanced	92	52	67	92
	Imbalanced	66	95	78	74
JM1	Balanced	70	82	75	69
	Imbalanced	77	64	70	73
PC3	Balanced	69	90	78	78
	Imbalanced	90	67	77	79
KC1	Balanced	80	31	44	71
	Imbalanced	69	95	80	67
EQ	Balanced	83	63	72	77
	Imbalanced	60	81	69	71
JDT	Balanced	96	84	90	90
	Imbalanced	85	97	91	87

Table 13 shows the performance measures of a hybrid model (S-SVM model) on seven datasets: CM1, PC1, JM1, PC3, KC1, EQ, and JDT. The model is evaluated under two different scenarios: balanced and imbalanced datasets. In the balanced scenario, the dataset has an equal proportion of the positive and negative classes, while in the imbalanced method, one class is underrepresented compared to the other. The performance measures reported in the table include precision, recall, F1-score, and accuracy. On the CM1 dataset, in the balanced scenario, the S-SVM model achieves a precision of 80%, recall of 94%, F1-score of 87%, and accuracy of 98%. In the imbalanced scenario, the

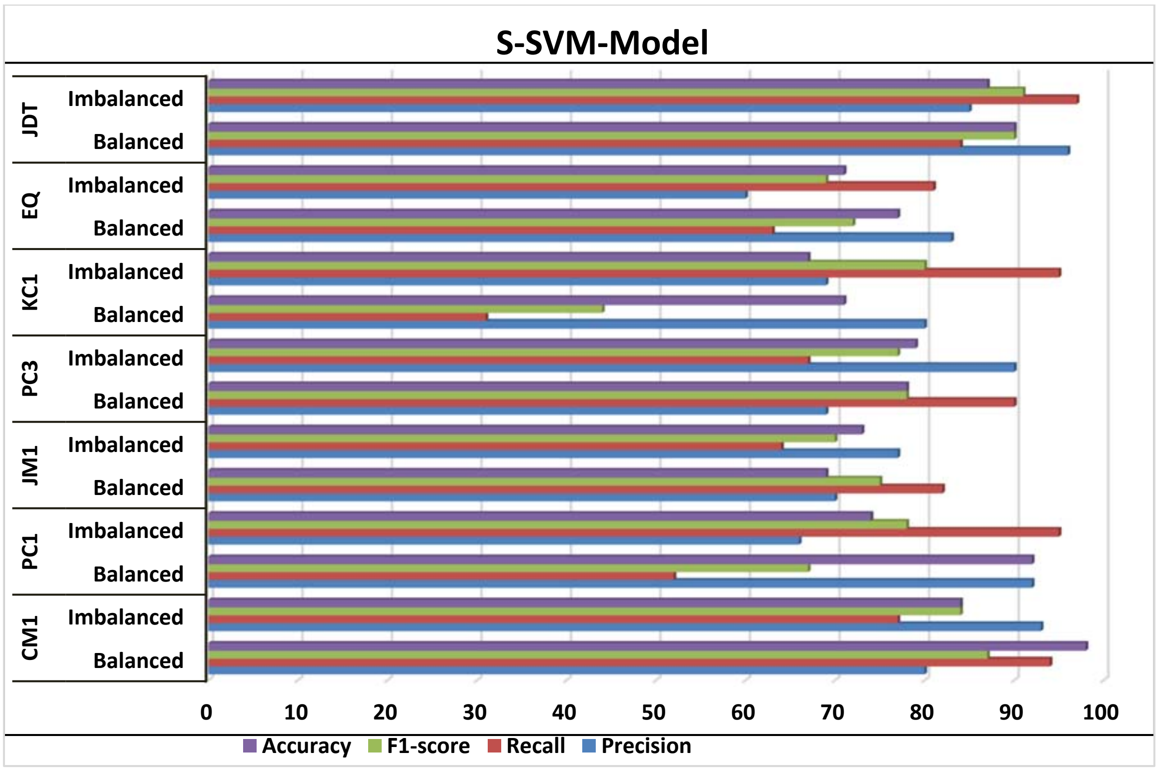


Figure 4. Comparative analysis with balanced & imbalanced dataset of S-SVM hybrid model.

model achieves a higher precision of 93% but a lower recall of 77%, resulting in an F1-score of 84% and an accuracy of 84%. On the PC1 dataset, the S-SVM model performs better in the imbalanced scenario, achieving a precision of 66%, recall of 95%, F1-score of 78%, and accuracy of 74%, compared to the balanced scenario, where it achieves a precision of 92%, recall of 52%, F1-score of 67%, and accuracy of 92%. Similar trends are observed on the JM1, PC3, KC1, EQ, and JDT datasets, where the performance of the S-SVM model varies depending on the dataset and the scenario. For example, on the JM1 dataset, in the balanced scenario, the model achieves a precision of 70%, recall of 82%, F1-score of 75%, and accuracy of 69%, while in the imbalanced scenario, it achieves a precision of 77%, recall of 64%, F1-score of 70%and accuracy of 73%. On the PC3 dataset, in the balanced scenario, the model achieves a precision of 69%, recall of 90%, F1-score of 78%, and accuracy of 78%, while in the imbalanced scenario, it achieves a precision of 90%, recall of 67%, F1-score of 77%, and accuracy of 79%. On the KC1 dataset, in the balanced scenario, the S-SVM model achieves a precision of 80%, recall of 31%, F1-score of 44%, and accuracy of 71%, while in the imbalanced scenario, it achieves a precision of 69%, recall of 95%, F1-score of 80%, and accuracy of 67%. On the EQ dataset, in the balanced scenario, the model achieves a precision of 83%, recall of 63%, F1-score of 72%, and accuracy of 77%, while in the imbalanced scenario, it achieves a precision of 60%, recall of 81%, F1-score of 69%, and accuracy of 71%. On the JDT dataset, in the balanced scenario, the model achieves a precision of 96%, recall of 84%, F1-score of 90%, and accuracy of 90%, while in the imbalanced scenario, it achieves a precision of 85%, recall of 97%, F1-score of 91%, and accuracy of 87%. Overall, the performance of the S-SVM hybrid model varies across different datasets and scenarios, with higher precision and recall generally observed in the imbalanced scenario. In contrast, higher accuracy is often achieved in a balanced scenario. The F1-score, which provides a balanced measure of precision and recall, also varies depending on the dataset and method. These results highlight the importance of considering the class distribution of the dataset and choosing appropriate evaluation measures when using the S-SVM hybrid model for classification tasks. **Figure 4** shows the graphical representation of the balanced and imbalanced dataset on the S-SVM model.

5.8 Comparative analysis of the previously developed models with the S-SVM model

The comparative analysis with previously developed models on the NASA dataset using the S-SVM (SMOTE-SVM hybrid) model reveals various findings. For the CM1 dataset, the S-SVM model achieved a precision of 80.00%, recall of 94.00%, F1-score of 87.00%, and accuracy of 98.00%. This performance is competitive with other models such as Naïve Bayes, Random Forest, C4.5 Miner, and ANN-ABC, outperforming them in precision, recall, and accuracy. Similarly, for the PC1 dataset, the S-SVM model achieved a precision of 92.00%, recall of 52.00%, F1-score of 67.00%, and accuracy of 92.00%. The S-SVM model's precision and accuracy are significantly higher than other models like SMOTE+Naïve Bayes, AdaBoost+SMOTE+Naïve Bayes, and Bagging+SMOTE+Naïve Bayes. These results suggest that the S-SVM model can be a viable option for classification tasks on the NASA dataset, especially in scenarios where precision and accuracy are essential. However, researchers should carefully consider the trade-offs between different performance measures and choose the appropriate model based on the specific requirements of their research or application.

Table 14. Performance analysis of previously developed models with the proposed model

NASA datasets	Techniques	Precision	Recall	F-m	Accuracy
CM1	Naïve Bayes[29,30]	86.20	78.65	34.09	64.57
	Random Forest[29,31]	71.10	71.29	32.17	60.98
	C4.5 Miner[32,33]	74.91	74.66	27.68	66.71
	Immunos[32,33]	73.65	75.02	30.99	66.03
	ANN-ABC[32,33]	75.00	81.00	33.00	68.00
	Hybrid self-organizing map[32,34]	70.12	78.96	30.65	72.37
	SVM[29,35]	81.20	79.08	31.27	78.69
	Majority vote[32,36]	79.80	80.00	30.46	77.01
	AntMiner+[32,36]	80.65	78.88	30.90	73.43
	ADBBO-RBFNN[32,37]	81.92	80.96	29.71	82.57
	NN GAPO+B[38]	-	-	-	74.40
	Decision Tree[29,35]	83.30	74.23	81.20	73.49
	KNN[31]	83.90	84.70	84.30	--
	MLP[31]	90.40	95.50	92.90	86.73
	SMOTE+Naïve Bayes[39]	20.22	71.42	31.51	60.24
	AdaBoost+SMOTE+Naïve Bayes[39]	14.28	64.28	23.36	45.87
	Bagging+SMOTE+Naïve Bayes[39]	20.00	71.42	31.24	59.63
	Proposed Model (S-SVM)	**80.00**	**94.00**	**87.00**	**98.00**
PC1	Naïve Bayes[31]	96.00	90.00	97.20	90.30
	Random Forest[31]	97.00	-	98.80	97.69
	C4.5 Miner[32]	76.58	81.76	34.05	62.18
	Immunos[32]	81.99	79.66	36.92	61.73
	ANN-ABC[32]	89.00	83.00	33.00	65.00
	Hybrid self-organizing map[32]	86.79	85.67	35.67	95.87
	SVM[32]	80.98	86.59	98.00	92.45
	Majority vote[32]	84.61	84.37	30.98	92.50
	AntMiner+[32]	89.34	87.12	26.11	91.85
	ADBBO-RBFNN[32]	90.89	89.33	20.24	-
	NN GAPO+B[31]	-	-	-	-
	Decision Tree[31]	97.00	-	98.00	-
	KNN[31]	95.00	90.00	98.00	95.71
	MLP[31]	97.00	99.00	98.00	96.00
	SMOTE+Naïve Bayes[39]	10.97	83.63	19.39	43.74
	AdaBoost+SMOTE+Naïve Bayes[39]	10.97	82.14	19.35	43.74
	Bagging+SMOTE+Naïve Bayes[39]	11.19	83.63	19.73	44.92
	Proposed Model (S-SVM)	**92.00**	**52.00**	**67.00**	**92.00**

5.9 Comparative analysis of SMOTE-Tomek and SMOTE-ENN with SMOTE-SVM model using CM1, PC1 and KC1 datasets experimental study

SMOTE is a popular data augmentation technique used to address the issue of class imbalance in machine learning datasets. It generates synthetic examples of the minority class by interpolating between minority class instances. SMOTE-SVM is an extension of SMOTE that incorporates information from an SVM classifier to generate synthetic samples.

On the other hand, SMOTE-Tomek and SMOTE-ENN are hybrid methods that combine SMOTE with Tomek links and Edited Nearest Neighbors (ENN) techniques, respectively, to improve further the effectiveness of SMOTE in handling class imbalance. CM1, PC1, and KC1 are datasets used for SDP, where the objective is to predict whether a software module is defective. Using these datasets, the author has now analysed the performance of SMOTE-Tomek, SMOTE-ENN, and SMOTE-SVM models.

Table 15. Performance analysis of various SMOTE techniques with the proposed model

Techniques	Datasets	Precision	Recall	F1-score	Accuracy
SMOTE-Tomek-SVM	CM1	99.0	99.0	99.0	99.0
	PC1	98.0	96.0	97.0	96.5
	KC1	89.0	83.0	90.0	89.0
SMOTE-ENN-SVM	CM1	99.0	100.0	99.0	99.0
	PC1	98.0	98.0	98.0	98.0
	KC1	92.0	89.0	90.0	89.0
SMOTE-SVM	CM1	97.0	98.0	97.0	97.0
	PC1	97.0	96.0	97.0	96.2
	KC1	91.0	92.0	89.0	90.9

SMOTE-Tomek-SVM, SMOTE-ENN-SVM, and SMOTE-SVM are evaluated on three datasets: CM1, PC1, and KC1. The evaluation metrics reported for each model on each dataset are precision, recall, F1-score, and accuracy. SMOTE-Tomek-SVM: CM1 dataset: The precision, recall, F1-score, and accuracy are reported as 99.0, 99.0, 99.0, and 99.0, respectively. PC1 dataset: The precision, recall, F1-score, and accuracy are reported as 98.0, 96.0, 97.0, and 96.5, respectively. KC1 dataset: The precision, recall, F1-score, and accuracy are reported as 89.0, 83.0, 90.0, and 89.0, respectively. SMOTE-ENN-SVM: CM1 dataset: The precision, recall, F1-score, and accuracy are reported as 99.0, 100.0, 99.0, and 99.0, respectively. PC1 dataset: The precision, recall, F1-score, and accuracy are reported as 98.0, 98.0, 98.0, and 98.0, respectively. KC1 dataset: The precision, recall, F1-score, and accuracy are reported as 92.0, 89.0, 90.0, and 89.0, respectively. S-SVM: CM1 dataset: The precision, recall, F1-score, and accuracy are reported as 97.0, 98.0, 97.0, and 97.0, respectively. PC1 dataset: The precision, recall, F1-score, and accuracy are reported as 97.0, 96.0, 97.0, and 96.2, respectively. KC1 dataset: The precision, recall, F1-score, and accuracy are reported as 91.0, 92.0, 89.0, and 90.9, respectively. From the reported metrics, the author can see that SMOTE-Tomek-SVM and SMOTE-ENN-SVM models have high precision, recall, F1-score, and accuracy values, indicating good performance on the datasets. However, it's important to note that the recall for SMOTE-ENN-SVM is very high (100.0) on the CM1 dataset, which may indicate an issue with class imbalance or data quality. Based on the information in the table, the models SMOTE-Tomek-SVM and SMOTE-ENN-SVM may have been overfitted, while S-SVM may not have been overfitted. The reason is that SMOTE-Tomek and SMOTE-ENN are variants of the SMOTE that combine oversampling (SMOTE) with undersampling (Tomek links or Edited Nearest Neighbors, respectively) to address the class imbalance. Therefore, "SMOTE" in the model names suggests that oversampling may have been applied in these models. On the other hand, S-SVM has no additional undersampling technique mentioned in its name, indicating that only SMOTE might have been used in this model without any undersampling. Oversampling is an approach to address the class imbalance, and its effectiveness depends on various factors, such as the dataset, the algorithm used, and the specific implementation.

6. Conclusion

In conclusion, the author presents a solution to the issue of dataset balancing in SDP by formulating it as a classification problem and proposing a hybrid SMOTE+SVM model. The author evaluated their model on different NASA and AEEM repository datasets: CM1, PC1, JM1, PC3, KC1, EQ and JDT and compared their performance using balanced and imbalanced datasets. The result shows that the performance of the SMOTE-SVM model is highly dependent on the dataset and scenario. Generally, the model performs better on balanced datasets than on imbalanced ones. For instance, on the CM1 dataset, the proposed model achieved an accuracy of 98% in the balanced scenario compared to 84% in the imbalanced scenario.

Similarly, on the PC1 dataset, the model achieved an accuracy of 92% in the balanced system and 74% in the imbalanced system. The author's findings indicate that balancing the dataset using the SMOTE technique improves the authenticity and quality of defect prediction results. This technique helped us achieve better quality and authentic results of defect prediction, even though it may reduce the accuracy in some cases. In future work, the authors plan to apply their proposed model to other datasets further to verify its performance and effectiveness. Overall, the hybrid SMOTE+SVM model can be considered a promising solution to the issue of imbalanced datasets in SDP.

Acknowledgments

Not applicable.

Authors' contributions

Each author contributed equally to this work.

Availability of supporting data

PROMISE software engineering repository[28] (http://promise.site.uottawa.ca/SERepository/datasets-page.html).

Conflict of interest

The authors declare no conflict of interest.

Consent for publication

All authors agreed.

Ethical approval and consent to participate

Not applicable.

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

References

1. Amasaki S. On applicability of cross-project defect prediction method for multi-versions projects. In: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering; 2017 Nov 8; Toronto Canada. New York: Association for Computing Machinery; 2017. p. 93–96. doi: 10.1145/3127005.3127015.
2. Shukla S, Radhakrishnan T, Muthukumaran K, Neti LBM. Multi-objective cross-version defect prediction. Soft Computing 2018; 22(6): 1959–1980. doi: 10.1007/s00500-016-2456-8.
3. Jayanthi R, Florence L. Software defect prediction techniques using metrics based on neural network classifier. Cluster Computing 2019; 22(1): 77–88. doi: 10.1007/s10586-018-1730-1.
4. Liu M, Miao L, Zhang D. Two-stage cost-sensitive learning for software defect prediction. IEEE Transactions on Reliability 2014; 63(2): 676–686. doi: 10.1109/TR.2014.2316951.
5. Herbold S, Trautsch A, Grabowski J. Global vs. local models for cross-project defect prediction. Empirical Software Engineering 2017; 22(4): 1866–1902. doi: 10.1007/s10664-016-9468-y.
6. Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE); 2016 May 14–22; Austin, TX. New York: IEEE; 2017. p. 309–320. doi: 10.1145/2884781.2884839.
7. Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering 2019; 45(2): 111–147. doi:

10.1109/TSE.2017.2770124.
8. Wahono RS. A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks. Journal of Software Engineering 2007; 1(1): 1–16. doi: 10.3923/jse.2007.1.12.
9. Malhotra R. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing 2015; 27: 504–518. doi: 10.1016/j.asoc.2014.11.023.
10. Benediktsson O, Dalcher D, Thorbergsson H. Comparison of software development life cycles: A multiproject experiment. IET Software 2006; 153(3): 87–101. doi: 10.1049/ip-sen:20050061.
11. Hassan MM, Afzal W, Blom M, *et al*. Testability and software robustness: A systematic literature review. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications; 2015 Aug 26–28; Madeira, Portugal. New York: IEEE; 2015. p. 341–348. doi: 10.1109/SEAA.2015.47.
12. Shepperd M, Bowes D, Hall T. Researcher bias: The use of machine learning in software defect prediction. IEEE Transactions on Software Engineering 2014; 40(6): 603–616. doi: 10.1109/TSE.2014.2322358.
13. Manjula C, Florence L. Deep neural network based hybrid approach for software defect prediction using software metrics. Cluster Computing 2019; 22: 9847–9863. doi: 10.1007/s10586-018-1696-z.
14. Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 1994; (6): 476–493. doi: 10.1109/32.295895.
15. Basili VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering 1996; 22(10): 751–761. doi: 10.1109/32.544352.
16. Alsawalqah H, Faris H, Aljarah I, *et al*. Hybrid SMOTE-ensemble approach for software defect prediction. Advances in Intelligent Systems and Computing 2017; 575: 355–366. doi: 10.1007/978-3-319-57141-6_39.
17. He H, Bai Y, Garcia EA, Li S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In: 2008 IEEE International Joint Conference on Neural Networks; 2008 Jun 1–8; Hong Kong. New York: IEEE; 2008. p. 1322–1328. doi: 10.1109/IJCNN.2008.4633969.
18. Nian R. Fixing imbalanced datasets: An introduction to ADASYN (with code!) [Internet]. San Francisco: Medium; 2018 [published 2018 Dec 23]. Available from: https://medium.com/@ruinian/an-introduction-to-adasyn-with-code-1383a5ece7aa.
19. Mirzaei B, Nikpour B, Nezamabadi-Pour H. An under-sampling technique for imbalanced data classification based on DBSCAN algorithm. In: 2020 8th Iranian Joint Congress on Fuzzy and intelligent Systems (CFIS); 2020 Sept 2–4; Mashhad, Iran. New York: IEEE; 2020. p. 21–26. doi: 10.1109/CFIS49607.2020.9238718.
20. Hasanin T, Khoshgoftaar TM. The effects of random undersampling with simulated class imbalance for big data. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI); 2018 Jul 6–9; Salt Lake City, UT. New York: IEEE; 2018. p. 70–79. doi: 10.1109/IRI.2018.00018.
21. Bach M, Werner A, Palt M. The proposal of undersampling method for learning from imbalanced datasets. Procedia Computer Science 2019; 159: 125–134. doi: 10.1016/j.procs.2019.09.167.
22. Sawangarreerak S, Thanathamathee P. Random forest with sampling techniques for handling imbalanced prediction of university student depression. Information 2020; 11(11): 1–13. doi: 10.3390/info11110519.
23. Software Defect Dataset. Promise software engineering repository [Internet]. Ottawa: University of Ottawa; 2004. Available from: http://promise.site.uottawa.ca/SERepository/datasets-page.html.
24. Kovács B, Tinya F, Németh C, Ódor P. Unfolding the effects of different forestry treatments on microclimate in oak forests: Results of a 4-yr experiment. Ecological Applications 2020; 30(2): 321–357. doi: 10.1002/eap.2043.
25. Wilson MD. Support vector machines. In: Encyclopedia of ecology. Amsterdam, Netherlands: Elsevier Science; 2008. p. 3431–3437. doi: 10.1016/B978-008045405-4.00168-3.
26. Zoppis I, Mauri G, Dondi R. Kernel methods: Support vector machines. Ranganathan R, Gribskov M, Nakai K, *et al.* (editors). Oxford: Academic Press; 2019. p. 503–510.
27. Chang YW, Hsieh CJ, Chang KW, *et al*. Training and testing low-degree polynomial data mappings via linear SVM. The Journal of Machine Learning Research 2010; 11(48): 1471–1490.
28. Sayyad Shirabad J, Menzies TJ. The PROMISE repository of software engineering databases [Internet]. Ottawa: University of Ottawa; 2005. Available from: http://promise.site.uottawa.ca/SERepository.
29. Alkhasawneh MS. Software defect prediction through neural network and feature selections. Applied Computational Intelligence and Soft Computing 2022; 2022: 2581832. doi: 10.1155/2022/2581832.
30. Mustaqeem M, Saqib M. Principal component based support vector machine (PC-SVM): A hybrid technique for software defect detection. Cluster Computing 2021; 24(3): 2581–2595. doi: 10.1007/s10586-021-03282-8.
31. Abualigah L. Group search optimiser: A nature-inspired meta-heuristic optimisation algorithm with its results, variants, and applications. Neural Computing and Applications 2021; 33: 2949–2972. doi: 10.1007/s00521-020-05107-y.

32. Abualigah L. Multi-verse optimiser algorithm: A comprehensive survey of its results, variants, and applications. Neural Computing and Applications 2020; 32: 12381–12401. doi: 10.1007/s00521-020-04839-1.
33. Rahim A, Hayat Z, Abbas M, *et al*. Software defect prediction with naïve bayes classifier. In: 2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST); 2021 Jan 12–16; Islamabad, Pakistan. New York: IEEE; 2021. p. 293–297. doi: 10.1109/IBCAST51254.2021.9393250.
34. Soe YN, Santosa PI, Hartanto R. Software defect prediction using random forest algorithm. In: 2018 12th South East Asian Technical University Consortium (SEATUC); 2018 Mar 12–13; Yogyakarta, Indonesia. New York: IEEE; 2019. p. 1–5. doi: 10.1109/SEATUC.2018.8788881.
35. Wang J, Shen B, Chen Y. Compressed C4.5 models for software defect prediction. In: 2012 12th International Conference on Quality Software; 2012 Aug 27–29; Xi'an, China. New York: IEEE; 2012. p. 13–16. doi: 10.1109/QSIC.2012.19.
36. Haouari AT, Souici-Meslati L, Atil F, Meslati D. Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction. Applied Soft Computing 2020; 96: 106686. doi: 10.1016/j.asoc.2020.106686.
37. Arar ÖF, Ayan K. Software defect prediction using cost-sensitive neural network. Applied Soft Computing 2015; 33: 263–277. doi: 10.1016/j.asoc.2015.04.045.
38. Abaei G, Selamat A, Fujita H. An empirical study based on semi-supervised hybrid self-organising map for software fault prediction. Knowledge-Based Systems 2015; 74: 28–39. doi: 10.1016/j.knosys.2014.10.017.
39. Saifudin A, Hendric SWHL, Soewito B, *et al*. Tackling imbalanced class on cross-project defect prediction using ensemble SMOTE. IOP Publishing 2019; 662(6): 062011. doi: 10.1088/1757-899X/662/6/062011.