Open Access Original Article

Feature Engineering of Sentiment Polarities for Bias/Variance Tradeoff Consideration in Software Defect Prediction Modelling

Review began 12/29/2024
Review ended 01/16/2025
Published 01/27/2025

DOI: https://doi.org/10.7759/s44389-024-02852-2

Taiwo O. Olaleye [1] , Oluwasefunmi T. Arogundade [1] , Adebayo Abayomi-Alli [2, 1] , Dada A. Aborishade [1] , Olusola J. Adeniran [3]

1. Computer Science, Federal University of Agriculture, Abeokuta, Abeokuta, NGA 2. HumanISE, Institute for Systems and Computer Engineering, Technology and Science, Porto, PRT 3. Mathematics, Federal University of Agriculture, Abeokuta, Abeokuta, NGA

Corresponding author: Taiwo O. Olaleye, olaleyeto@funaab.edu.ng

Abstract

Software quality assurance has been accorded high priority as one of the germane phases of the software development life cycle. Various automated approaches for ensuring quality assurance have evolved over the years, with the predictive analytics methodology gaining prominence. Existing studies in empirical software engineering research have failed to appreciate the prominence of quality data metrics and the subsequent statistical treatments of training sets prior to predictive modelling. This has resulted in suboptimal models with internal and external validity threats. Predictive analytics modelling could easily surpass state-of-the-arts if relevant feature engineering-based functionalities are incorporated into the proposed methodologies in literature. This study conducts multicollinearity test on sentiment polarities extracted from software defect reports, prior to a bias/variance tradeoff-based vote ensemble modelling of the training set. Experimental results show minimal bias, and the model generalizes well on unseen data. The combination of high performance across the performance metrics confirms that both bias and variance have been successfully minimized. The model achieved 2.79%, 21.84%, 5.73%, and 2.72% improvements on accuracy, precision, recall, and F1-measure, respectively, when compared with benchmark studies. The Unified Theory of Acceptance and Use of Technology model, employed for the perception evaluation of industry experts' opinion, indicates model's perceived ease of use and usefulness. The study establishes the importance of informed feature engineering techniques for a data science-based software defect prediction modelling.

Categories: Natural Language Processing (NLP), Software Quality Assurance (SQA), Data Mining
Keywords: software quality, sentiment analysis, word embedding, feature engineering, bias/variance tradeoff, software quality assurance

Introduction

Software quality assurance is a germane consideration in software development [1] and a global best practice in the software engineering industry due to the need to comply with the system requirements outlined in the software requirements specification document. A defect in software metrics indicates a breach of these system requirements, resulting in vulnerable software [2] if not addressed. Software defect prediction (SDP) models aim to improve software quality by examining software metrics and tracing potential faults [3] prior to deployment, as SDP is one of the software quality assurance activities [4]. The software development cycle is a procedure for designing, creating, and maintaining information systems [5], applied to a wide range of hardware and software systems. The testing phase of the development comes after the analysis, design, and implementation phases, before the maintenance phase in the waterfall model, and it is a crucial quality assurance step [1] because both testing and debugging must be thoroughly executed to qualify a software product as a high-quality system [6]. As today's software market rapidly grows in size and functionality, software testing plays a critical role in system development, especially in detecting defects. Meanwhile, defect prediction is a proactive approach to ensuring software quality and is better implemented to identify defect severity levels [7], in order to ensure efficient allocation of human and material resources for correctional measures.

Compliance with system requirements necessitates the importance of a predictive system during the development process of a software system, especially after the testing stage to address severities that contravene system requirements. Moreover, nowadays, industries and institutions require a high degree of compliance at different levels of commercial enterprise development to meet clearly specified standards [8]. The severity level of software defects likewise differs with varying degrees of flaws usually identified in execution codes with different impacts on the software. Some defects only slow down the progression, while others may cause system failure, hence the need to classify the severity extent of each defect [9]. The severity levels will assist developers in prioritizing the defects and forestall likely damages to the entire system. Depending on the depth of the identified anomaly, steps are taken to address the defect after testing. Defects detected after deployment affect reliability, but defect tracking systems allow developers to report identified anomaly, which improves the entire process as reports of the identified defects are analyzed by developers to determine the weight and severity and also to prevent duplication across modules of the software codes [10]. The reports are usually textual description of faults for immediate or scheduled attention, depending on the severity level of the defect, as unintentionally introduced by the software developer or other stakeholder [11,12]. The generated and analyzed reports are then assigned to relevant programmers to resolve issues raised for a continued development process. Severity levels could be binary classification of defects into severe and non-severe, number of defect density, multi-class classification, and classification based on severity levels, all of which specify faulty software modules in their predictive models [13-15]. Data science techniques, including machine learning (ML) and natural language processing (NLP), have been useful for severity detection modelling and software testing. While studies employ diverse data preprocessing techniques to clean the training set before ML, some germane feature engineering considerations are often missing, which constitute threats to the internal and external validity of existing models. Some of the identified drawbacks include multicollinearity of predictive attributes (which constitutes and internal validity threat), deployment of within-project software metrics (which creates a non-generalizability problem) [16], and the non-consideration of bias/variance tradeoff in existing methodologies, which is a severe internal validity threat weakness [17]. A summary of such studies is presented in Table *1*.

S/N	Author(s)	ML methodology	Strength(s)	Weakness(es)
6	[4]	Decision Tree	K-fold cross-validation reduced overfitting. Feature redundancy reduction through information gain	Imbalanced data sampling. Limitation to closed-source projects
3	[13]	Clustering	Robust feature engineering. Automated ground truth establishment	Limitation to within-project modelling. Imbalanced class sampling
9	[14]	Ensemble	Ground truth establishment through K-means automation. Parameter optimization	Limited software corpus for modelling. Non-feature engineering limited model performance
2	[15]	Ensemble	Minority oversampling to address overfitting. Ensemble modelling	Limitation of model to within-project metrics modelling
1	[17]	KT-SVM	Deployment of cross-project software metrics	No bias variance tradeoff consideration. Manual class labelling
4	[18]	Multilayer Perceptron	Multicollinearity mitigation	Undersampled defect class
5	[19]	MP+DT+MM-LR	Dimensionality reduction through information gain	Huge class representation gap due to class imbalance. Within-project approach
7	[20]	Boot strapping ensemble	Data resampling for equal class representation. Feature engineering of the model	Limitation to within-project approach
8	[21]	Clustering	Enhanced feature selection methodology	Prone to under-fitting. Class imbalance
10	[22]	Interactive machine learning	Cross-project approach. Model's capability over small data corpus	Class imbalance. Widespread feature redundancy
11	[23]	RNN-LSTM	Automated ground truth labelling. Robust feature engineering	Class imbalance. Limited to within-project
12	[24]	SVM	Adoption of sentiment analysis for ground truth establishment. Feature engineering enhancement	Imbalance in class representation. Limitation to within-project modelling
13	[25]	LMT	Sentiment analysis improved the establishment of ground truth. Hybrid approach of unsupervised and supervised modelling	Class imbalance. One-factor attribute limited model robustness
15	[26]	MFFE+MLP	Random oversampling reduced the ratio of class imbalance. Parameter optimization of ANN improved performance	Limitation to within-project corpus modelling. Restriction to single learner modelling instead of ensemble
16	[27]	CNN	BM25 approach enhances the assemblage methodology	Class imbalance
17	[28]	Boosting Ensemble	Six-attribute corpus has high discriminative ability. Adoption of report phrase summary	Limitation to within-project. Class imbalance

TABLE 1: Summary of reviewed related studies

KT-SVM: Kernel Trick-Support Vector Machine; MP+DT+MM-LR: Multilayer Perceptron+Decision Tree+Mixed Mode+Logistic Regression; RNN-LSTM: Recurrent Neural Network-Long Short-Term Memory; LMT: Logistic Model Tree; MFFE+MLP: Multi-Feature Fusion Embedding+Multilayer Perceptron; ML: Machine Learning; CNN: Convolutional Neural Network

This study is, therefore, motivated by the need to improve on existing ML frameworks designed for the prediction of software defects, which is imperative for the ever-evolving software industry, which grapples to meet the demands of today's information and communications technology-driven global economy. Desire for such improvements is further buoyed by the capabilities offered by sentiment analysis and opinion mining, considering the huge communication text data usually generated by software developers during software development life cycle. Sentiment analysis indeed offers an opportunity to establish a common ground truth for defect reports in repositories because labelling styles differ per data source. The novelty of this study actually lies in the sentiment polarity-based labelling, owing to the presence of emotions in defect reports, which could offer better actionable insights into the severity levels of defects identified in blocks of code. The main aim of this study, therefore, is to employ feature engineering and NLP use cases for optimizing existing SDP approaches in literature. This will help to address prevalent internal and external validity threats, while appraising the impact of the experimentation through perception evaluation of industry experts and traditional performance metrics.

Specific objectives are as follows:
1) Establishment of a sentiment-aware ground truth in the training set with a clean bill of health from multicollinearity.
2) Implementation of a bias variance tradeoff using synthetic minority oversampling and vote ensemble modelling.
3) Evaluation of the model using traditional performance metrics, industry experts' perception evaluation of the model, and validity threat evaluation.

The aforementioned is believed to have addressed prevalent internal and external validity threats in literature and as well as ensured a thoroughly evaluated model for validity analysis.

Materials And Methods

The feature engineering techniques employed in the various aspects of the research methodology are discussed in this section. The experimental approach of this study, especially with robust feature engineering, which usually requires textual data preprocessing [29], is better approached in an established methodological approach. The methodology adopted is the build, process and experiment [30] approach, which offers an improved ML model for the prediction of software defect severity levels, through the NLP, by mining software defect reports in cross-project benchmarks, across public and private sources. The software defect reports benchmark repositories, including primary data from private software development companies, are used in this study. The experimental process of the BPE ensures a robust training set, hence the inclusive defect reports from both private and benchmark repositories.

Experimental activities including data preprocessing carried out on the unstructured defect reports make the corpus suitable for subsequent phases of the framework. Classification of the software defect corpus into unique priority levels, prior to the ensemble supervised ML, is an experimental approach that fully automates the labelling of ground truths (severity levels) through the unsupervised ML technique of sentiment analysis. The sentiment analysis would be implemented on the acquired text-corpus to categorize the software testing reports into their severity levels (ground truths establishment) together with word embedding, which will all form attributes that make up the training dataset. In order to avoid the inherent widespread class imbalance that usually characterizes software defect corpus, a minority oversampling technique will be implemented as part of text preprocessing, to address the prevalent overfitting problem. The feature engineering techniques would be deployed with the ensemble ML methodology, to address the variance problem side of the tradeoff. The ensemble methodology will be trained on the attributes generated from the sentiment analysis and word embedding phases, which already established the severity ground truths for each software defect report document contained in the entire training set. The activity diagram presented in Figure *1* actualizes the aim of this study.

The sentiment analysis approach aims to analyze the sentiment of defect report to predict their severity levels. In particular, the compound sentiment score of a given defect report expresses its severity level; therefore, this proposed model assumes that a defect report is classified into high severity when its compound sentiment score is negative, i.e. $\leqslant -0.05$. It is instructive that negative sentiment terms conveyed by the reporters signify the exigency and the need to fix the defect instantly. On the other hand, a defect report is assigned a lower severe level when the compound sentiment score of the defect report is positive, i.e. $\geqslant 0.05$. This is on the premise that a defect reporter defines and summarizes the reported defect using positive emotions; hence, the defect report can be addressed later. The resulting predictive model would be evaluated using traditional metrics as well as through a perception evaluation survey of industry experts.

Data preprocessing of software defect reports

Lemmatization is likewise imperative for an efficient NLP prediction model. Usually, words used in the summary field of software defect reports written by reporters are expressed differently in several styles, whereas they depict the same meaning. Therefore, remaining words from the two previous steps are reduced into their root words (stemming) in order to avoid duplication of words with similar roots. For instance, words like 'failings', 'failures', 'fails', 'failure', and 'failed' are transformed into the root word 'fail'.

In this stage, NLP technicalities are applied to the defect reports, which is an unstructured text dataset. NLP techniques are used to process and transform the textual defect report into a collection of unique words or terms, called tokens. The text preprocessing stages include stop-word removal, lowercase conversion, lemmatization, and tokenization tasks. In tokenization, the summary description of each defect report is split into a set of terms, while stop-word removal involves elimination of needless words, which do not play any role in predicting the severity of reported defects [30]. Constructive words such as 'on', 'an', 'in', 'that', 'the', etc., are widespread in reports but do not convey specific information in NLP and could reduce the precision of the predictive model.

Given a software defect report *sdr* (from each public repositories) described in Equations (1), (2), and (3),

$$sdr = \langle dd, sl \rangle \qquad (1)$$

$$sdr' = \langle dd', sl \rangle \qquad (2)$$

$$dd' = \langle t_1, t_2, \ldots, t_n \rangle \qquad (3)$$

where

sl ε {severity level$_1$, level$_2$, level$_3$, ..., level$_n$},

sdr is the software defect report,

sdr′ is the preprocessed defect report,

dd is the defect description,

dd′ is the preprocessed defect description,

sl is the severity level, and

t_1, t_2, ..., t_n represent the output words from the above three preprocessing stages.

Only sdr′ file that contains all n-instances of dd′ from all public repositories consulted, which likewise contain t_1, t_2, ..., t_n tokenized words, makes the training set for this study. This is because the sl (severity levels) accompanying each defect report from their respective repositories would be discarded for the proposed automated labelling conceptualized for this study.

VADER-based sentiment analysis of defect report

Sentiment analysis defines polarity to distinguish emotions expressed in texts into positivity, neutrality, and negativity states. VADER approach calculates the three polarities for each word-token by allocating certain sentimentality values known as a lexicon whose value determines the severity levels.

In its computations, the word ‘*w*’ (in each defect report) is assigned numeric value 1, 0, or −1 for positive, neutral, or negative emotion, respectively. Therefore, polarity of a text (defect report) ‘*T*’ is as described in Equation (4):

$$T = \{w_1, w_2, w_3, \ldots, w_n\} \qquad (4)$$

while Equation (4) is computed on the frequency of words ‘ *w*’ in ‘*T*’, which occurs in ‘*z*’. The pos(*T*, *z*) and neg(*T*, *z*) are positive and negative words from ‘ *T*’ that occur in ‘*z*’ with the adds:

$$\mathrm{sum}(T, z) = \mathrm{pos}(T, z) - \mathrm{neg}(T, z) \qquad (5)$$

hence, sentiment $s_1(z)$ of a feature ‘*z*’ under polarized lexicon ‘ *T*’ is derived by Equation (6):

$$s_1(z) = \begin{cases} T & \text{if } \mathrm{sum}(T, z) > 0 \\ 0 & \text{if } \mathrm{sum}(T, z) = 0 \\ -T & \text{if } \mathrm{sum}(T, z) < 0 \end{cases} \qquad (6)$$

The compound score is, therefore, computed as presented in Equation (7):

$$x = \frac{x}{\sqrt{x^2 + x}} \qquad (7)$$

using the sum of valence scores (*x*) of each word in the lexicon, which is rejigged with the rules and normalized between −1 (high severity level) and +1 of low severity and α is the normalization constant with default value 15. Severity levels are then calculated as in Equations (8), (9), and (10):

$$\text{(low severity) when compound score} \geq 0.05 \qquad (8)$$

$$\text{(intermediate severity) level with compound score } -0.05 < \text{score} < 0.05 \qquad (9)$$

$$\text{while (high severity) level occurs when compound score} \leq -0.05 \qquad (10)$$

The resulting numeric values of positive, negative, neutral, and compound granularities, computed for each preprocessed text-data representing defect reports, thus make the four attributes generated in this phase on each defect report. The granularities establish the ground truth for this study, which describes the severity levels, as presented in Table *2*.

Word2vec-based word embedding on defect report

In addition to the 4-no polarity attributes (positivity, negativity, neutrality, and compound score values achieved from Vader) computed for each of the defect report above, additional 300-no attributes are to be generated for each defect report through the word2vec NLP algorithm. Each tokenized word from the preprocessing phase is transformed into numerical vectors of fixed length by the word2vec skip-gram model, which is an efficient method for learning distributed vector representations with high quality proven across studies to be better than the bag of words option. The neural network model, as illustrated in Figure *2* by Ramay et al. [26], captures a large number of accurate syntactic and semantic word connections to return k-dimensional vector by predicting the surrounding context words given the central target word [6]. For each defect tokenized report sdr', preprocessed words $w_1, w_2, w_3, \ldots, w_n$ from Equation (1) are transformed into a fixed-length vector presented in Equation (11):

$$w_i = \langle w_1, w_2, \ldots, w_n \rangle = \langle v_1, v_2, \ldots, v_n \rangle \qquad (11)$$

where v_i transforms word w_i into a fixed-length numerical vector, and the resulting vectors $v_i, \ldots, v_n$, for each of the report instance, add to the 4-no predictor polarity attributes earlier derived, to make up the total 304-no attributes for this study. Each 304 attributes will be generated for n-number of defect report instances acquired from the five software defect public repositories. The 304 per instance ratio directly goes into the next phase for subsequent data sampling.

Synthetic minority oversampling (SMOTE) of defect report

Instances of numeric reports from the preceding phases, each containing 304 attributes, are expected to exhibit imbalanced sampling. This is because the representations of *lsl* and *hsl*, described in Table *2*, cannot be equally represented (traditionally, there are more non-defective reports than defective ones in repositories) with grievous consequences on the predictive accuracy of any ML algorithm. Instances of any of the two severity levels with lower representations will affect the precision of the proposed ensemble model due to the biased composition of the three independent variables contained in the dataset [29]. This imbalance will result in a data mining problem called over-fitting (for the highly represented severity level(s) and under-fitting (for the lowest represented severity level(s)). Hence, due to the proliferation of defect reports with lower severities, which is rampant in defect repositories, the resulting model is likely to assign lower severity levels to report with high severity due to pattern over-familiarity. SMOTE is, therefore, implemented to address the under-sampling of the minority severity class by generating synthetic instances to scale up their representations. SMOTE is optimized to search for the 5-k nearest neighbor of any member of the smaller class and then compute characteristic attributes along the route formed between the neighbors in an efficient manner inheriting the exact distinctive nature and feature of the parent minority. The model is as described in the SMOTE illustration captured in Figure *3*. Upon SMOTE application to scale up classes with minority representations, the ensemble ML analytics phase follows to design the proposed predictive model. The SMOTE technique would result in an increase in the number of numeric defect instances and not on the 304-no attributes. The likelihood of the oversampling introducing noise would be mitigated by the ensemble modelling of vote, and even random forest (RF), in the subsequent section. This is because RF is an ensemble model in itself (made up of decision trees).

Software defect report acquisition

The supervised ensemble learning approach of this study would require a numeric predictive data attribute. Numeric data type would be achieved through the processing of software defect report acquired from public repositories and private software projects [30]. The data would be defect reports generated during software testing phase of the software development life cycle. The defect reports are textual phrases describing the nature of defects detected in each software modules, which are captured in different software repositories, sourced from private, public, and cross-software development projects across different use cases. The defect reports are traditionally labelled by the software testing reporters according to different severity levels. While some labels adopt binary approaches, others are in four- to five-level labelling according to the discretion of reporters and company severity classification policies. The concatenation of software defect reports across repositories adopted in this study is to ensure a robust big data across projects, which will efficiently address the most prominent threats to validity indicated in primary studies. The textual dataset deployed for this study is acquired from the five most prominent software engineering repositories recommended in primary studies, including NASA, Promise, Eclipse, Mozilla, and Apache. They are all acquired from Kaur and Jindal [31], as reports from enterprise. The software defect reports captured in this repositories have different attributes, including report title, report summary, the report severity level, etc., but the actual report is the most relevant for the objectives set out in this study. Defect reports from private repositories are also acquired. Data source is a mix of enterprise-level and open-source projects within the aforementioned repositories, spanning over 10 years for some of the sources. Unlike static code metrics, defect reports are actually presented in lumpsum, for most of the sources, covering both within or cross-projects

Ensemble predictive analytics of numeric software defect attributes

Ensemble ML models stem from the concept of synergy, where two heads think better than one. The basic idea of an ensemble is that having more than one ML algorithm, each producing marginally different results on the dataset, where some algorithms learn certain patterns well and others learn different patterns better and combining their expertise results in a significantly better model than any of the base learner individually. The ensemble employed in this study classifies by voting, such that for three different base learner algorithms in the ensemble, if two label a defect report as 'high severity' and one labels it as 'low severity', the vote ensemble will label the defect report as 'high severity' for having the highest vote for the particular defect report instance. However, an averaging method is adopted in this study, where the results of the base learners are averaged to determine the output of the vote ensemble. The most widely recommended base learner algorithms in the literature [32, 33] selected for the ensemble in this study include RF, C4.5 decision tree learner, multi-layer perceptron (MLP) neural network, and sequential minimal optimization of the improved version of a support vector machine [18]. The ensemble modelling would help to solve the variance problem towards a bias variance tradeoff consideration for a reliable predictive analytics. The MLP neural network has proven to be highly effective for SDP modelling [19], and will therefore contribute significantly to the efficiency of the ensemble in this study.

Mathematical modelling of the bias/variance tradeoff methodology

Let the compound score of the defect report di be denoted as $S(d_i)$, where $S(d_i) \in [-1,1]$. The severity level class for each defect report is defined based on the compound score:

$$\text{Severity}(d_i) = \begin{cases} \text{lsl (Low Severity Level)} & \text{if } S(d_i) \geq 0.05 \\ \text{hsl (High Severity Level)} & \text{if } S(d_i) \leq -0.05 \end{cases} \qquad (12)$$

for scores between $-0.05 < S(d_i) < 0.05$, they are treated as neutral.

where *lsl* indicates 'low severity level' and *hsl* indicates 'high severity level' class labels.

After sentiment analysis, each defect report d_i is converted into a 300-dimensional feature vector $v_i \in$ R300 using word2Vec. Thus, for each defect report, the final input feature vector x_i is

$$x_i = [v_{i1}, v_{i2}, \ldots, v_{i300}, \text{Severity}(d_i)] \in \mathbb{R}^{301} \qquad (13)$$

where the severity class (lsl or hsl) is appended as the 301st attribute.

To balance the severity level classes (low vs. high severity), SMOTE is applied. For minority class C_{min}, new synthetic instances $x_i^{'}$ are generated as linear interpolations between existing minority samples:

$$x_i^{'} = x_i \dot{+} \lambda\,(x_j - x_i) \qquad (14)$$

where $\lambda \sim Uniform(0,1))$

for the ensemble predictive modelling, average opinion is employed using base classifiers of RF (h_1), MLP (h_2), decision tree (h_3), and sequential maximum optimization (h_4). The final prediction $H(x)$ for a defect report instance X is

$$H(x) = \text{Mode}(h_1(x), h_2(x), \ldots, h_k(x)) \qquad (15)$$

where each $hj(x)$ is a prediction from the j-th base classifier, and the final decision is made based on majority voting.

To test for multicollinearity among the 301 attributes, the correlation coefficient matrix R is computed for the feature set $X = \{x_1, x_2, \ldots, x_n\}$:

$$R_{ij} = \frac{Cov(x)_i, x_j)}{\delta_{x_i}\delta_{x_j}} \qquad (16)$$

for $i, j \varepsilon [1, 301]$

where $Cov(x)_i, x_j)$ represents the covariance between attributes x_i and x_j , and δ_{x_i} and δ_{x_j} are the standard deviations. Multicollinearity is flagged if $[R_{ij}]$ is close to 1 for non-diagonal elements, indicating high correlation between features.

Using the SMOTE-balanced dataset and ensemble learning, the bias-variance tradeoff is evaluated by analyzing both training and test errors. Models with high bias (simple models) will exhibit both high training and test errors, while models with high variance (complex models) will show low training error but high test error. The goal of this study is to minimize both.

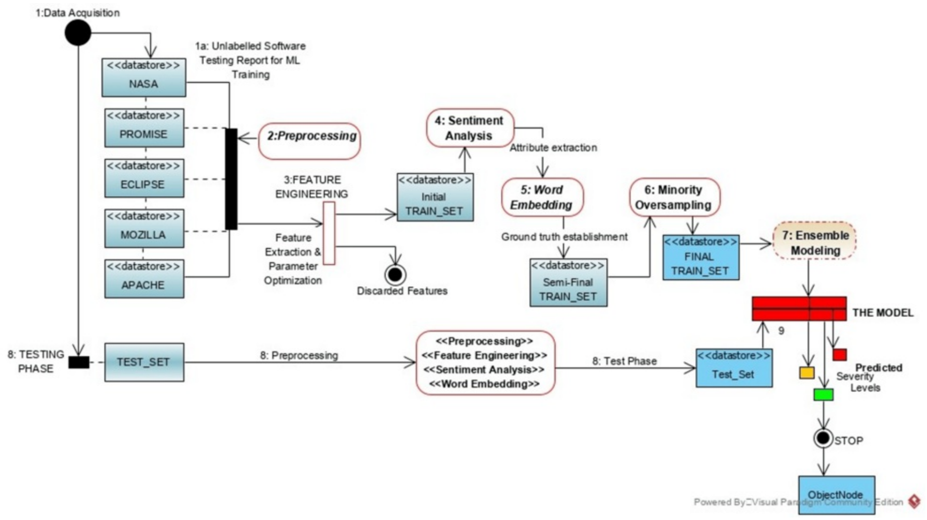


FIGURE 1: The internal framework of the methodological approach

S/N	Granularity	Sentiment range	Severity level
1	Positivity	Compound score ≥0.05	lsl: low severity level
2	Negativity	Compound score ≤−0.05	hsl: high severity level

TABLE 2: Derived ground truth labelling

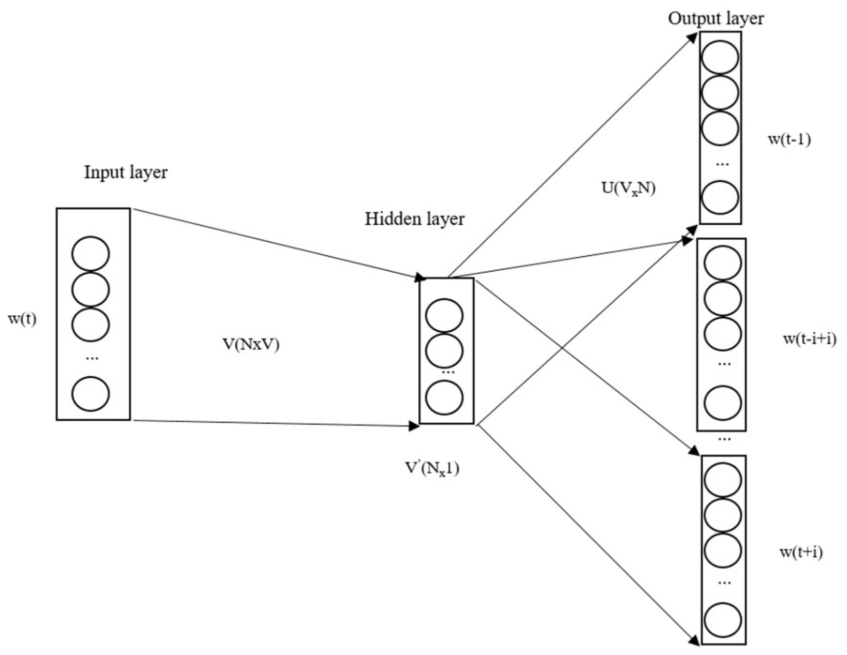


FIGURE 2: Illustration of the Word2vec model

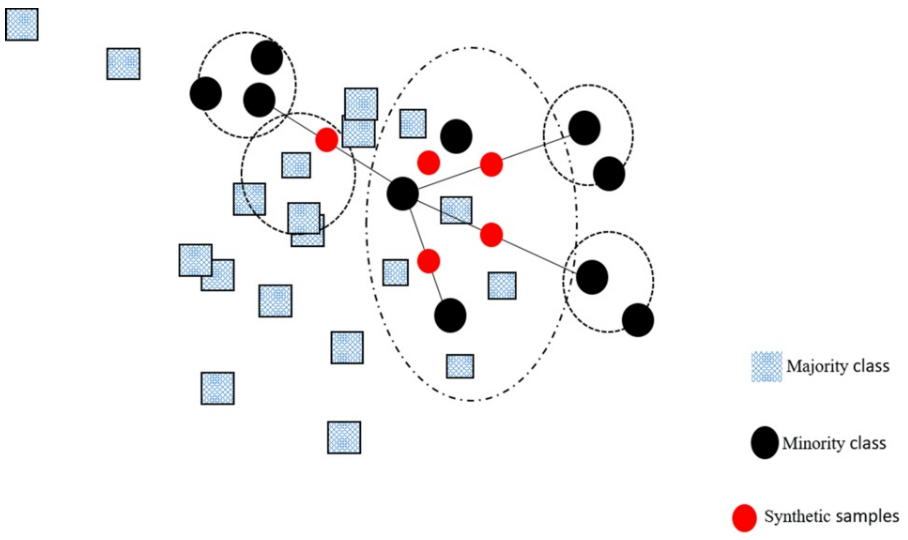


FIGURE 3: Description of the minority oversampling technique

Results

One of the specific objectives of this study is to acquire a heterogeneous software defect corpus from different software engineering use case industries, providing the proposed ensemble model with a robust training set. The results of this model could then be representative of a wide spectrum of software development use cases.

Experimental setup

The defect reports used in this study were acquired from publicly available software repositories, including Mozilla, PROMISE, and NASA. The Orange data mining toolkit was employed for preprocessing, sentiment analysis, and word embedding phases. Preprocessing included tokenizing the defect reports into n-gram structured tokens and computing sentiment polarities using the VADER sentiment analysis tool. The training set comprising 301 attributes (300 numeric feature vectors generated via word2Vec and a class label) was prepared for ML tasks.

Python programming in Jupyter Notebook (Anaconda Navigator IDE) was used for multicollinearity testing, SMOTE resampling to address class imbalance, and vote ensemble modelling. Model training, testing, and performance evaluation were performed on a Lenovo laptop (4GB RAM, 8GB HDD) running Windows 10 OS.

The severity prediction application was developed using Python libraries, featuring an interface for uploading defect reports and predicting their severity levels. A perception evaluation survey was conducted using a Google Form, shared with software development experts for application assessment.

The distribution of defect reports across repositories as contained in the final corpus deployed for this study is presented in Table *3*. A total of 6,789 defect reports were acquired in a comma-separated version (csv) file format. The defect preprocessing phase processes the defect reports into n-gram structured tokens, resulting in a structured software defect document containing 37,904 n-gram tokens. The VADER-based sentiment analysis is executed on the preprocessed tokens for each of the tokenized 6,789 defect report documents to compute their sentiment scores across the positive, negative, neutral, and compound polarities. Upon successful computations, the sentiment scores of the 6,789 defect reports are used to establish the ground truth in line with the structure presented in Table *2*. This is arrived at from the various emotions expressed in each defect reports, which is indicative of the severity level of each defect that is being reported. New attributes are generated for each of the defect reports, including the length of the defect title (T_len), the length of the defect report (R_len), and the duration it takes to resolve a reported defect (R_Duration). These are also included in the training set in order to determine the implications of the attributes on the severity level of each defect, alongside the feature vectors, towards approximating to either of the dual severity levels. Table *4* shows the summary statistics of the introduced attributes with the computed sentiment polarities. In the analysis, the minimum defect title length is four words, while the maximum is 122 words, indicating a wide range of defect titles. The defect report length ranges from 1 to 30,000, with a standard deviation of 2,097. The minimum positive polarity recorded by Vader in the entire corpus is 0, with a maximum of 0.574. This indicates that the majority of the positive sentiments expressed in the defect reports are not as strong, climaxing at an average polarity range. The interquartile range for the positive polarities shows that the maximum value of 0.574 earlier reported must be an outlier, since no data points are spotted along the three quartiles. The negative polarity follows a similar trend, just that polarity scores of 0.179 or less are discovered at the quartile 3 of the range. The neutral polarity cluster, however, shows a divergent outlook, indicative of the fact that the majority of the defect reports are of neutral sentiments. The mean neutral polarity across the acquired reports is 0.873495, with a deviation of 0.16582, indicative of less deviation. The interquartile range shows more positive polarities across the points, with the median and the maximum points returning a 1.0 polarity.

S/N	Repository	Initial size	Final train set
1.	Eclipse	66,304	2,021
2.	Apache	5,127	443
3.	NASA	1,020	501
4.	Promise	5,201	601
5.	Mozilla	6,276	912
6.	Private Repository	400	400

TABLE 3: Acquired defect report distribution across repositories

	T_len	R_len	R_Duration	pos	neg	neu	comp
mean	57.89918	1025.211	9365.405	0.046932	0.079571	0.873495	-0.03528
std	15.21265	2097.252	15828.52	0.107784	0.137003	0.16582	0.243282
min	4	1	0	0	0	0.262	-0.7964
25%	47	242	819	0	0	0.744	-0.1027
50%	58	423	2,729	0	0	1	0
75%	68	914	7,864	0	0.179	1	0
max	122	30,000	1,07,214	0.574	0.738	1	0.8271

TABLE 4: Data distribution of numeric data attributes in the training set

T_len: Length of the defect title; R_len: Length of the defect report; R_Duration: Duration taken to resolve a reported defect; pos: Positive; neg: Negative; neu: Neutral; comp: Compound; std: Standard deviation

Another major objective of this study is the test of multicollinearity, which constitutes a threat to the internal validity of an ML model if not addressed. None of the primary studies conducted this test on their predictive attributes for SDP, which renders their result questionable, as a high multicollinearity of attributes would return an unreliable prediction. The heat map of Figure 4 shows the result of the multicollinearity test conducted as part of the methodological approach of this study. The coefficients of a pair of attributes show their level of multicollinearity status, and it can be observed that none of the predictive attributes show a high collinearity status with the ground truth (S_L) severity level. Therefore, the state of the training set is healthy enough for the subsequent ensemble modelling.

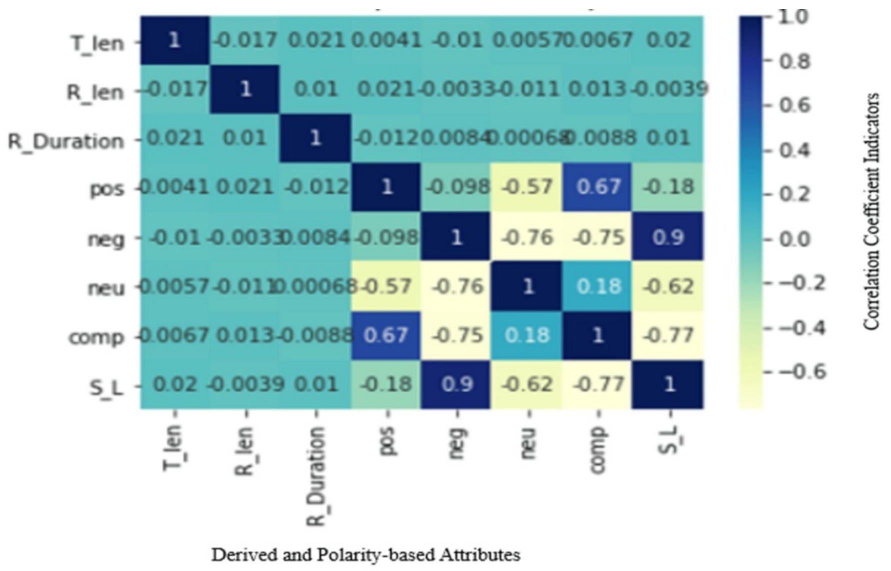
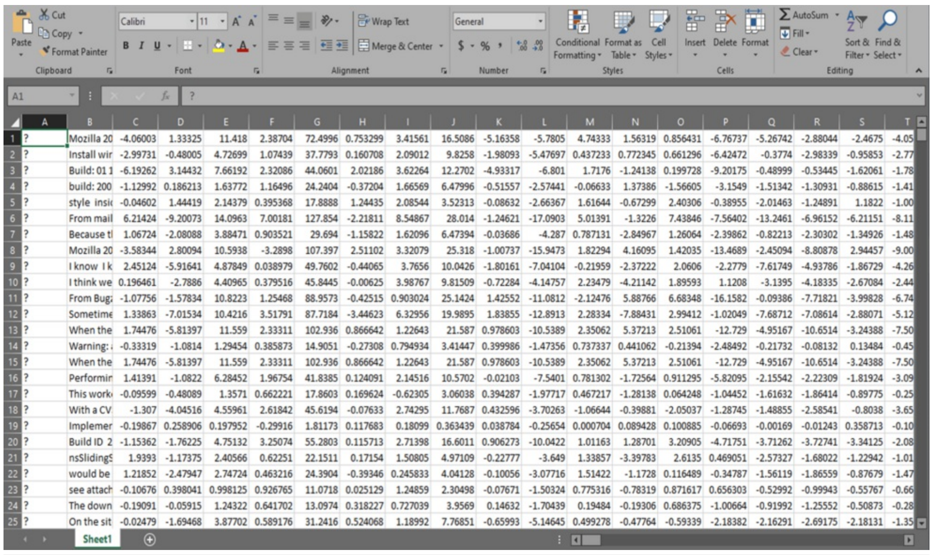


FIGURE 4: Correlation coefficient plot of the predictive attributes for multicollinearity test

The subsequent 300-no feature vector attributes for each 6,789 defect reports are then computed through the word2vec transfer learning methodology. The 301-attribute columns comprise the 300-no numeric feature vectors alongside the 1-no class column that labels each defect report as either of high severity or low severity for the subsequent supervised ML predictive analytics. The extracted numeric vectors, together with the polarity-based ground truths, are presented in the csv file of Figure 5, which is the actual training set of this study. The minority oversampling is next on the methodological approach and is achieved by the SMOTE technique. The pre- and post-SMOTE states of the training set are plotted and presented in Figure 6. With the obviously high imbalanced training set, SMOTE is implemented to oversample the minority class representation in the training set.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	?	Mozilla 20	-4.06003	1.33325	11.418	2.38704	72.4996	0.753299	3.41561	16.5086	-5.16358	-5.7805	4.74333	1.56319	0.856431	-6.76737	-5.26742	-2.88044	-2.4675	-4.05
2	?	Install wir	-2.99731	-0.48005	4.72699	1.07439	37.7793	0.160708	2.09012	9.8258	-1.98093	-5.47697	0.437233	0.772345	0.661296	-6.42472	-0.3774	-2.98339	-0.95853	-2.77
3	?	Build: 01 1	-6.19262	3.14432	7.66192	2.32086	44.0601	2.02186	3.62264	12.2702	-4.93317	-6.801	1.7176	-1.24138	0.199728	-9.20175	-0.48999	-0.53445	-1.62061	-1.78
4	?	build: 200	-1.12992	0.186213	1.63772	1.16496	24.2404	-0.37204	1.66569	6.47996	-0.51557	-2.57441	-0.06633	1.37386	-1.56605	-3.1549	-1.51342	-1.30931	-0.88615	-1.41
5	?	style insi	-0.04602	1.44419	2.14379	0.395368	17.8888	1.24435	2.08544	3.52313	-0.08632	-2.66367	1.61644	-0.67299	2.40306	-0.38955	-2.01463	-1.24891	1.1822	-1.00
6	?	From mail	6.21424	-9.20073	14.0963	7.00181	127.854	-2.21811	8.54867	28.014	-1.24621	-17.0903	5.01391	-1.3226	7.43846	-7.56402	-13.2461	-6.96152	-6.21151	-8.11
7	?	Because tl	1.06724	-2.08088	3.88471	0.903521	29.694	-1.15822	1.62096	6.47394	-0.03686	-4.287	0.787131	-2.84967	1.26064	-2.39862	-0.82213	-2.30302	-1.34926	-1.48
8	?	Mozilla 20	-3.58344	2.80094	10.5938	-3.2898	107.397	2.51102	3.32079	25.318	-1.00737	-15.9473	1.82294	4.16095	1.42035	-13.4689	-2.45094	-8.80878	2.94457	-9.00
9	?	I know I k	2.45124	-5.91641	4.87849	0.038979	49.7602	-0.44065	3.7656	10.0426	-1.80161	-7.04104	-0.21959	-2.37222	2.0606	-2.2779	-7.61749	-4.93786	-1.86729	-4.26
10	?	I think we	0.196461	-2.7886	4.40965	0.379516	45.8445	-0.00625	3.98767	9.81509	-0.72284	-4.14757	2.23479	-4.21142	1.89593	1.1208	-3.1395	-4.18335	-2.67084	-2.44
11	?	From Bug:	-1.07756	-1.57834	10.8223	1.25468	88.9573	-0.42515	0.903024	25.1424	1.42552	-11.0812	-2.12476	5.88766	6.68348	-16.1582	-0.09386	-7.71821	-3.99828	-6.74
12	?	Sometime	1.33863	-7.01534	10.4216	3.51791	87.7184	-3.44623	6.32956	19.9895	1.83855	-12.8913	2.28334	-7.88431	2.99412	-1.02049	-7.68712	-7.08614	-2.88071	-5.12
13	?	When the	1.74476	-5.81397	11.559	2.33311	102.936	0.866642	1.22643	21.587	0.978603	-10.5389	2.35062	5.37213	2.51061	-12.729	-4.95167	-10.6514	-3.24388	-7.50
14	?	Warning:	-0.33319	-1.0814	1.29454	0.385873	14.9051	-0.27308	0.794934	3.41447	0.399986	-1.47356	0.737337	0.441062	-0.21394	-2.48492	-0.21732	-0.08132	0.13484	-0.45
15	?	When the	1.74476	-5.81397	11.559	2.33311	102.936	0.866642	1.22643	21.587	0.978603	-10.5389	2.35062	5.37213	2.51061	-12.729	-4.95167	-10.6514	-3.24388	-7.50
16	?	Performin	1.41391	-1.0822	6.28452	1.96754	41.8385	0.124091	2.14516	10.5702	-0.02103	-7.5401	0.781302	-1.72564	0.911295	-5.82095	-2.15542	-2.22309	-1.81924	-3.09
17	?	This work	-0.09599	-0.48089	1.3571	0.662221	17.8603	0.169624	-0.62305	3.06038	0.394287	-1.97717	0.467217	-1.28138	0.064248	-1.04452	-1.61632	-1.86414	-0.89775	-0.25
18	?	With a CV	-1.307	-4.04516	4.55961	2.61842	45.6194	-0.07633	2.74295	11.7687	0.432596	-3.70263	-1.06644	-0.39881	-2.05037	-1.28745	-1.48855	-2.58541	-0.8038	-3.65
19	?	Implemer	-0.19867	0.258906	0.197952	-0.29916	1.81173	0.117683	0.18099	0.363439	0.038784	-0.25654	0.000704	0.089428	0.100885	-0.06693	-0.00169	-0.01243	0.358713	-0.10
20	?	Build ID 2	-1.15362	-1.76225	4.75132	3.25074	55.2803	0.115713	2.71398	16.6011	0.906273	-10.0422	1.01163	1.28701	3.20905	-4.71751	-3.71262	-3.72741	-3.34125	-2.08
21	?	nsSlidingS	1.9393	-1.17375	2.40566	0.62251	22.1511	0.17154	1.50805	4.97109	-0.22777	-3.649	1.33857	-3.39783	2.6135	0.469051	-2.57327	-1.68022	-1.22942	-1.01
22	?	would be	1.21852	-2.47947	2.74724	0.463216	24.3904	-0.39346	0.245833	4.04128	-0.10056	-3.07716	1.51422	-1.1728	0.116489	-0.34787	-1.56119	-1.86559	-0.87679	-1.47
23	?	see attach	-0.10676	0.398041	0.998125	0.926765	11.0718	0.025129	1.24859	2.30498	-0.07671	-1.50324	0.775316	-0.78319	0.871617	0.656303	-0.52992	-0.99943	-0.55767	-0.66
24	?	The down	-0.19091	-0.05915	1.24322	0.641702	13.0974	0.318227	0.727039	3.9569	0.14632	-1.70439	0.19484	-0.19306	0.686375	-1.00664	-0.91992	-1.25552	-0.50873	-0.28
25	?	On the sit	-0.02479	-1.69468	3.87702	0.589176	31.2416	0.524068	1.18992	7.76851	-0.65993	-5.14645	0.499278	-0.47764	-0.59339	-2.18382	-2.16291	-2.69175	-2.18131	-1.35

FIGURE 5: Eventual training set of 300-no word vector attributes and the sentimental ground truth

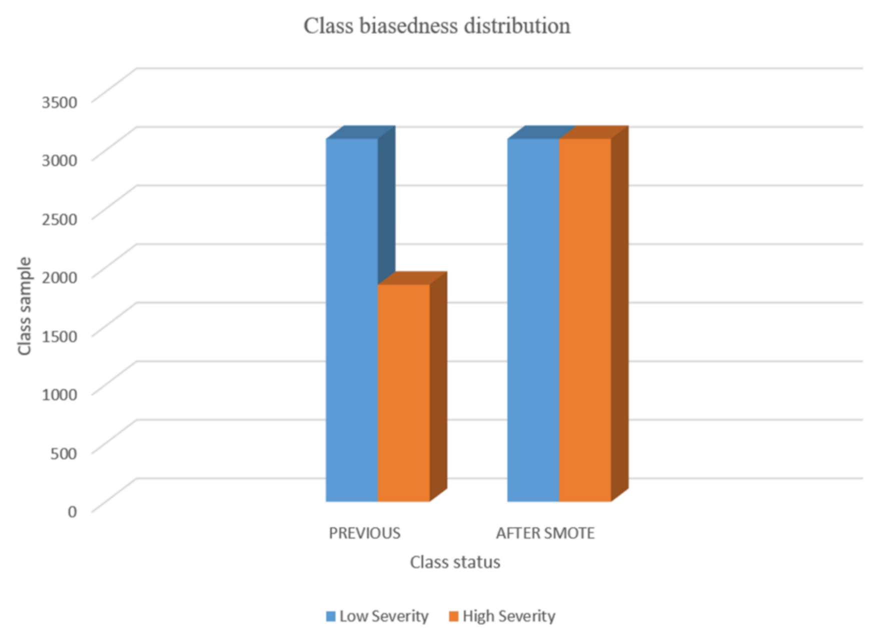
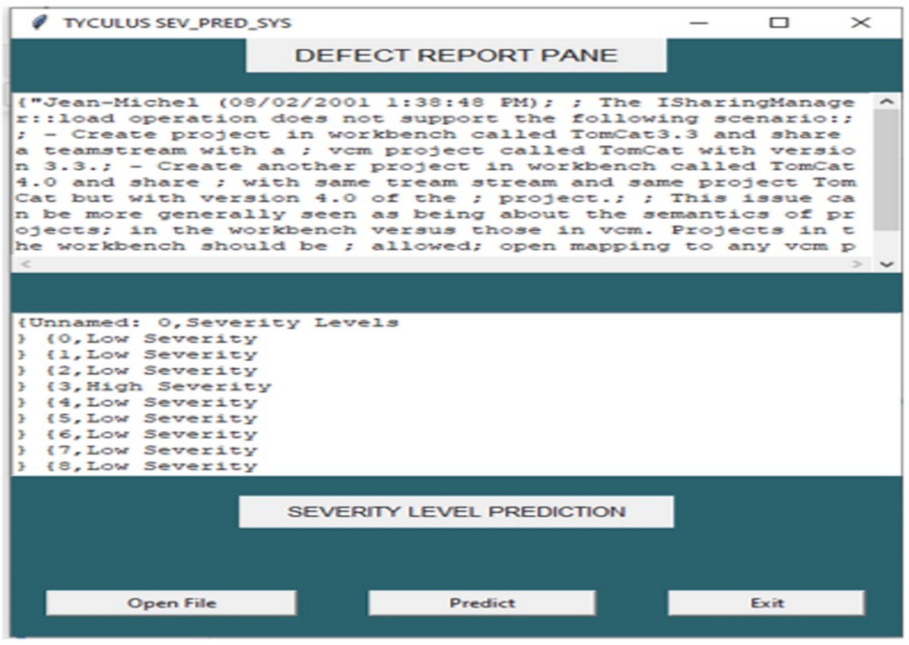


FIGURE 6: Distribution of ground truth before and after synthetic minority oversampling

The ML phase implemented by vote ensemble follows after the successful implementation of the synthetic minority oversampling. The vote ensemble base learners are fitted on the 6,789:301 dependent and independent attributes for the ML. Upon the successful training, the intelligent software defect severity level prediction model is saved for subsequent predictions by the defect predictor. The Python-implemented defect predictor has two segments, including the Defect Report and the Severity Level Prediction segments. The Defect Report Pane uploads software defect reports whose severity level is to be ascertained. Upon the click of the Open File button, the selection window comes up from where the location of the defect report could be selected on the host computer. Upon selection of the choice defect report, the content of the file is displayed on the Defect Report Pane. The Predict button then loads the initially saved prediction model for testing and evaluation, and the result of the prediction is displayed on the Severity Level Prediction pane, as shown in Figure 7.



```
{"Jean-Michel (08/02/2001 1:38:48 PM); ; The ISharingManage
r::load operation does not support the following scenario:;
; - Create project in workbench called TomCat3.3 and share
a teamstream with a ; vcm project called TomCat with versio
n 3.3.; - Create another project in workbench called TomCat
4.0 and share ; with same tream stream and same project Tom
Cat but with version 4.0 of the ; project.; ; This issue ca
n be more generally seen as being about the semantics of pr
ojects; in the workbench versus those in vcm. Projects in t
he workbench should be ; allowed; open mapping to any vcm p
```

```
{Unnamed: 0,Severity Levels
} {0,Low Severity
} {1,Low Severity
} {2,Low Severity
} {3,High Severity
} {4,Low Severity
} {5,Low Severity
} {6,Low Severity
} {7,Low Severity
} {8,Low Severity
```

FIGURE 7: The vote ensemble-based software defect severity predictor interphase

Performance evaluation through confusion matrix

The performance evaluation on the severity predictor model was carried out in terms of how well the system was able to rightly classify software defect reports into their respective severity levels through the computation of F1 measure, alongside other performance metrics. The metrics were calculated using the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) elements of the confusion matrix. The confusion matrix is used to ascertain the performance of a classification model on given test data, which is divided into two dimensions, including the predicted values and true values, alongside their total number of predictions in each category. The 2*2 confusion matrix in Figure 8 shows the predicted values of the model and the true values (actual values) of the given observations of the training set. The performance metrics are discussed below with their computations.

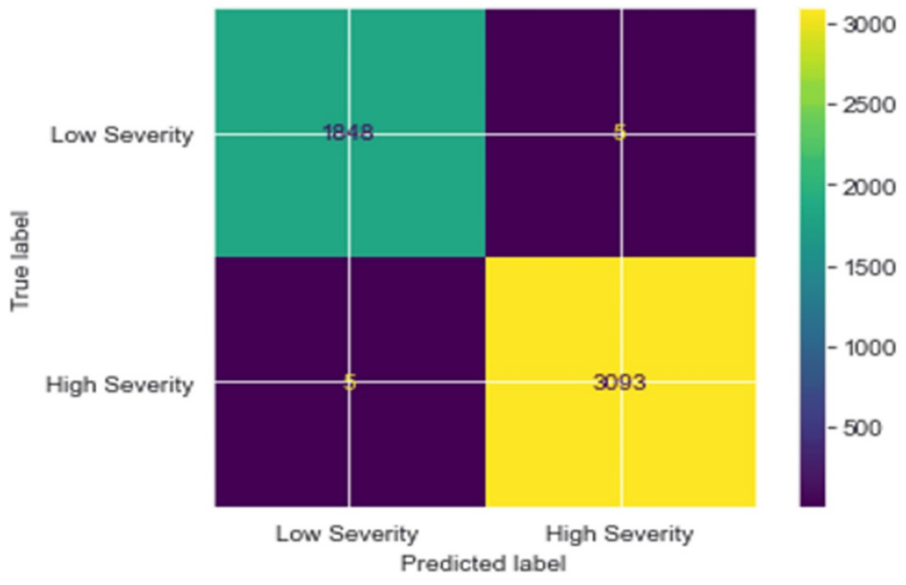


FIGURE 8: Confusion matrix of the severity predictor model

Accuracy returns the overall correctness of the model, which is indicative of the fraction of total defect reports that were correctly predicted by the classifier, and is as computed in Equation (16):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (16)$$

Precision measures the fraction of predictions (as computed in Equation (17)) that are of high severity:

$$Precision = \frac{TP}{TP + FP} \qquad (17)$$

Recall (sensitivity, computed in Equation (18)) tells of the fraction of positive samples that are actually predicted as positive:

$$Recall = \frac{TP}{TP + FN} \qquad (18)$$

Specificity tells of the fraction of negative samples that are actually predicted as negative, as in Equation (19):

$$Specificity = \frac{TN}{TN + FP} \qquad (19)$$

F1 score computes precision and recall into one singular metric as more useful performance evaluation metric than accuracy. It is presented in Equation (20):

$$F1 = 2 \times \frac{Recall \times Precision}{Recall + Precision} \qquad (20)$$

where TP is the measure of outcome of the model that is correctly predicted as positive, TN as the measure of model's outcome correctly predicted as negative, FP as the measure of the model's outcome incorrectly predicted as positive, and FN as the measure of model's outcome incorrectly predicted as negative.

Weighted averages of the performance metrics are presented in Table *5*, and as can be observed, the performance of the model is state-of-the-art with respect to the weighted averages of the various metrics. The class of low severities correctly predicted as low severities, and the class of high severity defect reports, which are correctly predicted as high severity, is on the high side, hence the reliability of the model. The F1 measure that finds the harmonic mean of precision and recall, which is a more reliable evaluation metric than accuracy, likewise returned a state-of-the-art performance with an insignificant misclassification error rate of 0.003.

S/N	Evaluator	Weighted average
1.	Accuracy	0.9979
2.	Precision	0.9973
3.	Recall	0.9973
4.	F1 measure	0.9972
5.	Misclassification rate	0.003
6.	Specificity	0.9983

TABLE 5: Performance metrics of the vote ensemble-based severity predictor model

On the specific objective to minimize the bias/variance tradeoff, the evaluation results provided in Table *5*, which stems from Figure *8*, suggest good model performance, indicating that both bias and variance have been effectively minimized. The breakdown of how these metrics reflect low bias and low variance is presented below:

Accuracy close to 1 indicates that the model makes very few classification errors. A high accuracy suggests that the model is capturing the underlying patterns well, with low bias. Given the accuracy is nearly perfect, it also indicates that the model generalizes well, pointing to low variance. Precision measures how often the positive predictions are correct, and a high precision of 0.9973 suggests that the model is not overly sensitive, meaning it is unlikely to misclassify negative examples as positives, which is indicative of a low bias and a good balance in classification, with minimal false positives. Recall measures how often actual positive instances are correctly identified, and a high recall of 0.9973 indicates that the model is correctly capturing the majority of positive examples, which means low bias in identifying the positive class. The F1 measure score balances precision and recall, and with a high F1 score close to 1, the model performs well on both precision and recall, minimizing both types of errors (false positives and false negatives), which reflects a good tradeoff between bias and variance. The low misclassification rate of 0.003 confirms that only a tiny fraction of instances are incorrectly classified, reinforcing that both bias (errors due to wrong assumptions) and variance (errors due to overfitting) are minimized. The specificity measures how well the model correctly identifies negative instances and at a weighted average of 0.9983. The figure indicates that the model accurately detects the negative class and avoids false positives, pointing to a low bias.

To elaborate, TPs indicate the model’s effectiveness in capturing defect reports that genuinely belong to the positive class, ensuring critical issues are accurately flagged. Conversely, FNs highlight instances where severe defect reports are overlooked, which could lead to unresolved critical issues in practice. The high recall observed in the results demonstrates that the FN rate is minimal, emphasizing the model's reliability in identifying positive cases. Furthermore, these metrics directly influence downstream evaluations like the F1 score, reinforcing the model's ability to balance precision and recall for robust classification performance.

Perception evaluation through UTAUT model

The opinions of domain experts are sought on the severity predictor model to gain their perception on the likelihood of deploying the predictor model in the software development industry. Their opinions were gathered through the digital survey form upon their successful interaction with the solution. The Unified Theory of Acceptance and Use of Technology (UTAUT) theoretical framework is, therefore, employed to evaluate the performance of the solution based on their observations and perceptions. The UTAUT construct for the solution was designed to gauge the operational simplicity of the solution (perceived ease of use), the essence (perceived usefulness), job fitness of the solution, the outcome expectations, and its social influence. In all, Table 6 shows the status of responding stakeholders as per their designation, and it is evident that responses from direct end users (Software testers) outweigh those of other technical experts, majority of which have over 10 years of experience. The perception evaluation survey design and outcome (based on UTAUT constructs), with respect to responses received from industry experts, is depicted in Figure 9, which shows the Likert scoring per construct.

Designation	Total
System/Program Analyst	6
Computer Programmer	4
Software Developers	5
Software Testers	11
Data Scientists	4
Other IT Professionals	3

TABLE 6: Professional experience and distribution of respondents

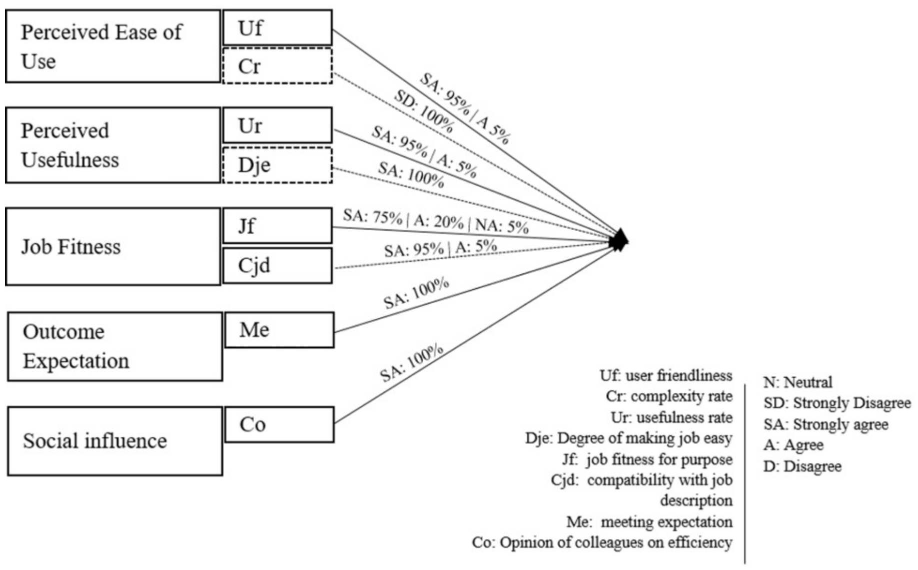


FIGURE 9: Depiction of experimental result from the UTAUT-based perception evaluation of the severity predictor model

UTAUT: Unified Theory of Acceptance and Use of Technology

Perceived Ease of Use

The degree to which respondents believe that the model would be less difficult to use is the main thrust of this construct. Its user friendliness (Uf) and complexity rate (Cr) are investigated. Experimental result shows that majority (95%) of respondents strongly agree to its user friendliness for software defect severity prediction (as 5% agree), while 100% of them strongly disagree to its complexity perspective.

Perceived Usefulness

The extent to which the solution is expected to improve end users' job performance is the main thrust of this construct. Its usefulness rate (Ur) and its degree of making jobs easy (Dje) are tested. Result shows majority (95%) of respondents strongly agree that the solution is useful for the purpose it is meant for, while 100% of respondents likewise strongly believe that the solution makes defect severity detection job easier.

Job Fitness

This construct was incorporated to address the perception of respondents on job fitness (Jf) and its compatibility for job description (Cjd). About 75% of respondents strongly agree to the job fitness of the predictor model, while 20% agree to the fact with 5% that do not agree. For the compatibility with job description, 95% of respondents strongly agreed, while 5% agree. On the compatibility with job description, 95% strongly agree and 5% agree.

Outcome Expectation

The outcome expectation, measured by the meeting expectation label (Me), speaks to expectation of the respondents with respect to the functional requirements. As may be observed, the entire respondents believe that the model meets the expectations of end users.

Social Influence

The social influence (Co) of the model is tested in this construct. The opinion of experts is sought on the efficiency of the solution as a social construct. In the end, the entire respondents agree on the social influence opportunity of the model.

Threat to Validity Evaluation

Validity risks have the possibility of limiting the anticipated ability of the study to achieve reliable outcomes that should remain true across studies and organizations. The general threat to framework validity is composed of four major classes, which will be evaluated in this study.

Conclusion validity: This looks at the measure of how reasonable a research conclusion is within the scope of the training set as well as the predictive methodological approach. In this research work, conclusion validity evaluation through benchmarks would ascertain the weight of the conclusion, with respect to studies that deploy same heterogeneous set, as well as the same feature extraction tools. Benchmarks likewise deploy predictive analytics with similar base learners. Figure *10* shows performance metrics of the model as compared with benchmarks. The basis for conclusion comparison is on the premise that all benchmarks deploy similar defect reports from Mozilla, Eclipse, Promise, Bugzilla, and Jira and likewise either employ sentiment analysis, word embedding, or both for feature extraction. They likewise deploy predictive analytics for classification and confusion matrix for evaluation purposes. The conclusion reached in this study is. therefore, valid based on the similarity in methodological approach across the studies. However, variations in the choice of preprocessing techniques such as parameter optimization, feature selection, and minority oversampling, to resolve one challenge or the other, could necessitate variations in their experimental results.

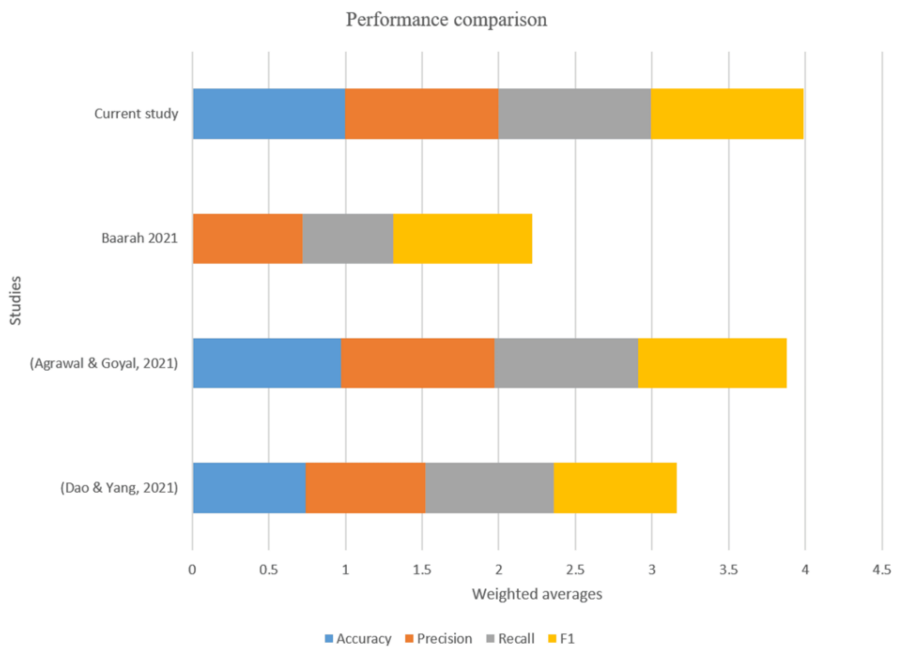


FIGURE 10: Conclusion validity of the research with benchmark studies

Internal validity: It considers the measurement of soundness of the research work with respect to the robustness of the methods used in the research. Experimental conditions and treatment deployed in this study indeed make a huge difference as evident from the performance metrics evaluation aforementioned owing to the bias-variance tradeoff consideration employed in the study. The synthetic minority oversampling of less-represented class of low severity resolved the usual biasedness of defect training set, while the ensemble predictive analytics addressed the variance problem. The threat to internal validity that could probably affect the result presented is in the area of training set employed in this study, which comes with a manual class labelling but which was jettisoned with the use of the sentiment polarity to determine severity levels. This, in addition to the feature vectors extracted through word2vec, which constitute the independent variables, could spin a threat owning to the fact that defect reports description is at the discretion of software developers who may not exhibit enough sentiments in their descriptions, which would ultimately determine their polarity cluster.

Construct validity: This checks for the extent to which the system measures up with the claims made by looking at the level of effectiveness of the proposed system, in terms of the performance metrics chosen for evaluation. Employment of confusion matrix for evaluating predictive analytics is a standard practice, hence its deployment in this study. Well-known metrics including accuracy, precision, recall, F-measure, etc., are used by majority of empirical software engineering studies and are likewise used in this study. Another possibility of construct validity threat is the usage of VADER-based sentiment analysis for feature extraction, owing to the availability of other polarity extractors including SentiWordNet, SentiS4D, etc., that may impact performances of emotion-based approaches. Choice of hyper-parameters could likewise vary experimental results; hence, this study retains default parameters for all preprocessing tools.

External validity: This considers the performance of proposed system in addressing real-life practical problems, which is in line with this study, which designs an intelligent model that predicts the severity level of software defects, in an attempt for quality assurance, during the software testing phase of the software development life cycle. The possibility of threats emanating from the use of a particular defect report, for the modelling, is eliminated owing to the heterogeneous nature of the dataset used in this study, which ensures a cross-project training set, thereby ensuring the possibility of generalization of experimental result.

Discussion

The need to ensure quality assurance during the software development life cycle has necessitated a robust software testing phase that identifies software defects and their corresponding reports. Several studies have aimed at predicting the severity levels of software defects through the use of defect reports, which are textual corpora that describe, in natural language, the anomalies noticed in software products. These studies have returned results with validity threats, including a high rate of multicollinearity in predictive variables, no consideration for the bias/variance tradeoff in training sets, etc. The predictive analytics deployed in this study, using NLP and feature engineering, have produced state-of-the-art results in terms of performance metrics. A precision rate of 0.9973 and a misclassification rate of 0.003 are rated as reliable by stakeholders who tested the solution with industry-based data. Compared with existing benchmarks, the results of this study surpass current performance in the literature, with high true positive and true negative indices. With accuracy metrics of 0.9979 and a recall value of 0.9983, the vote ensemble ML approach proves to be a reliable model that leverages the various techniques employed in this study, including a robust text preprocessing module, synthetic minority oversampling, VADER-based sentiment analysis, and word2vec embedding through transfer learning.

Sentiment analysis provided labelled data with high contextual accuracy, while word2Vec generated dense feature representations, capturing semantic relationships within defect reports. This synergy enhanced the model's ability to distinguish between severity classes, directly contributing to the improvements observed in the performance metrics and the perception evaluation survey through UTUAT.

The likelihood of threats in the study is also an important evaluation tool for an empirical software engineering study. The conclusion reached based on the confusion metrics was validated alongside existing studies with similar data and methodology. The accuracy and precision also surpass benchmarks, while the study shares precision with an existing study, thereby establishing its conclusion validity. Other validity threats are also discussed, attesting to the reliability of the proposed model in this study. The model showed improvements of 2.79%, 21.84%, 5.73%, and 2.72%, respectively, when compared with benchmark studies, further establishing its conclusion validity. The perception evaluation by industry experts returned highly commendable results for the model. Its perceived ease of use, usefulness, job fitness, outcome expectations, etc., all received positive feedback, making it suitable for industry use. Consequently, experimental results show that the combination of synthetic minority oversampling and ensemble ML efficiently addresses the bias-variance tradeoff consideration in predictive analytics and further establishes the importance of a multivariate training set through the dual methods of sentiment analysis and word embedding.

Conclusions

This study developed an NLP-based approach for predicting software defect severity levels using a vote ensemble machine learning model. The model was trained on 300 feature vectors generated through word2Vec embedding and labelled via sentiment analysis with VADER. Feature engineering techniques, including text preprocessing and synthetic minority oversampling, addressed gaps in prior studies, yielding improved performance metrics such as accuracy (+2.79%), precision (+21.84%), recall (+5.73%), and F1-score (+2.72%) compared to benchmarks. The integration of sentiment analysis, word embedding, and a bias-variance tradeoff scheme enhanced the system’s robustness, establishing a reliable multivariate training set for predictive analytics. Future work could explore alternative embedding techniques and incorporate closed-source defect reports to enrich the training set.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Taiwo O. Olaleye, Oluwasefunmi T. Arogundade, Dada A. Aborishade, Olusola J. Adeniran, Adebayo Abayomi-Alli

Acquisition, analysis, or interpretation of data: Taiwo O. Olaleye, Oluwasefunmi T. Arogundade, Adebayo Abayomi-Alli

Drafting of the manuscript: Taiwo O. Olaleye

Critical review of the manuscript for important intellectual content: Taiwo O. Olaleye, Oluwasefunmi T. Arogundade, Dada A. Aborishade, Olusola J. Adeniran, Adebayo Abayomi-Alli

Supervision: Oluwasefunmi T. Arogundade, Dada A. Aborishade, Olusola J. Adeniran, Adebayo Abayomi-Alli

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. Agrawal R, Goyal R: Developing bug severity prediction models using word2vec . International Journal of Cognitive Computing in Engineering. 2021, 2:104-115. 10.1016/j.ijcce.2021.08.001
2. Alsawalqah H, Hijazi N, Eshtay M, Faris H, Radaideh AA, Aljarah I, Alshamaileh Y: Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Applied Sciences. 2020, 10:1745. 10.3390/app10051745
3. Azeema MI, Palomba F, Shi L, Wang Q: Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. Information and Software Technology. 2019, 108:115-138. 10.1016/j.infsof.2018.12.009
4. Baarah A, Aloqaily A, Salah Z, Alshdaifat E: Sentiment-based machine learning and lexicon-based approaches for predicting the severity of bug reports. Journal of Theoretical and Applied Information Technology. 2021, 99:1386-1401.
5. Bani-Salameh H, Sallam M, Al shboul B: A deep-learning-based bug priority prediction using RNN-LSTM neural. e-Informatica Software Engineering Journal. 2021, 15:29-45. 10.37190/e-inf210102
6. Bassil Y: A simulation model for the waterfall software development life cycle . International Journal of Engineering & Technology. 2012, 2:2049-3444. 10.48550/arXiv.1205.6904
7. Benaddy M, Wakrim M: Simulated annealing neural network for software failure prediction . International Journal of Software Engineering and Its Applications. 2012, 6:35-46.
8. Bishnu PS, Bhattacherjee V: Software fault prediction using quad tree-based K-means clustering algorithm . IEEE Transactions on Knowledge and Data Engineering. 2012, 24:1146-1150. 10.1109/tkde.2011.163
9. Choeikiwong T, Vateekul P: Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets. Proceedings of the International MultiConference of Engineers and Computer Scientists. 2016, I:1-5.
10. Conklin WA: Software assurance: The need for definitions . 2011 44th Hawaii International Conference on System Sciences. 2011, 1-7. 10.1109/HICSS.2011.380
11. Dao A-H, Yang C-Z: Severity prediction for bug reports using multi-aspect features: A deep learning approach. Mathematics. 2021, 9:1644. 10.3390/math9141644
12. El-Shorbagy SA, El-Gammal WM, Abdelmoez WM: Using SMOTE and heterogeneous stacking in ensemble learning for software defect prediction. ICSIE '18: Proceedings of the 7th International Conference on Software and Information Engineering. 2018, 44-47. 10.1145/3220267.3220286
13. Gray D, Bowes D, Davey N, Christianson B: The misuse of the NASA Metrics Data Program data sets for automated software defect prediction. 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011). 2011, 96-103. 10.1049/ic.2011.0012
14. Ha DA, Chen TH, Yuan SM: Unsupervised methods for software defect prediction. SoICT '19: Proceedings of the 10th International Symposium on Information and Communication Technolog. 2019, 49-55. 10.1145/3368926.3369711
15. Han Z, Li X, Xing Z, Liu H, Feng Z: Learning to predict severity of software vulnerability using only vulnerability description. 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2017, 125-136. 10.1109/ICSME.2017.52
16. Huda S, Liu KL, Abdelrazek M, Ibrahim A, Alyahya S, Al-Dossari H, Ahmad S: An ensemble oversampling model for class imbalance problem in software defect prediction. IEEE Access. 2018, 6:24184-24195. 10.1109/access.2018.2817572
17. Iqbal A, Aftab S: A classification framework for software defect prediction using multi-filter feature selection technique and MLP. International Journal of Modern Education and Computer Science. 2020, 12:18-25. 10.5815/ijmecs.2020.01.03
18. Jin C: Cross-project software defect prediction based on domain adaptation learning and optimization . Expert Systems with Applications. 2021, 171:114637. 10.1016/j.eswa.2021.114637
19. Olaleye TO, Arogundade OT, Aborisade DO, Abayomi-Alli A, Olusola OJ: Multilayer perceptron of software complexity metrics for explainable multicollinearity mitigation and defect localization. Cureus Journal of Computer Science . 2025, 17:1-17. 10.7759/s44389-024-02871-z
20. Jindal R, Malhotra R, Jain A: Prediction of defect severity by mining software project reports . International Journal of System Assurance Engineering and Management. 2016, 8:334-351. 10.1007/s13198-016-0438-y
21. Nawaz A, Rehman A, Abbas M: A novel multiple ensemble learning models based on different datasets for software defect prediction [PREPRINT]. arXiv:2008.13114. 2020, 1-17. 10.48550/arXiv.2008.13114
22. Wang F, Ai J, Zou Z: A cluster-based hybrid feature selection method for defect prediction . 2019 IEEE 19th

International Conference on Software Quality, Reliability and Security. 2019, 1-9. 10.1109/QRS.2019.00014
23. Mehta S, Patnaik KS: Improved prediction of software defects using ensemble machine learning techniques . Neural Computing and Applications. 2021, 33:10551-10562. 10.1007/s00521-021-05811-3
24. Wu X, Zheng W, Chen X, Zhao Y, Yu T, Dejun M: Improving high-impact bug report prediction with combination of interactive machine learning and active learning. Information and Software Technology. 2021, 133:106530. 10.1016/j.infsof.2021.106530
25. Park B-J, Oh S-K, Pedrycz W: The design of polynomial function-based neural network predictors for detection of software defects. Information Sciences. 2013, 229:40-57. 10.1016/j.ins.2011.01.026
26. Ramay WY, Umer Q, Yin XC, Zhu C, Illahi I: Deep neural network-based severity prediction of bug reports . IEEE Access. 2019, 7:46846-46857. 10.1109/access.2019.2909746
27. Rathore SS, Kumar S: Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowledge-Based Systems. 2017, 119:232-256. 10.1016/j.knosys.2016.12.017
28. Song Q, Jia Z, Shepperd M, Ying S, Liu J: A general software defect-proneness prediction framework . IEEE Transactions on Software Engineering. 2011, 37:356-370. 10.1109/tse.2010.90
29. Tuteja M, Dubey G: A research study on importance of testing and quality assurance in software development life cycle (SDLC) models. International Journal of Soft Computing and Engineering. 2012, 2:251-257.
30. Umer Q, Liu H, Sultan Y: Emotion based automated priority prediction for bug reports . IEEE Access. 2018, 6:35743-35752. 10.1109/access.2018.2850910
31. Kaur A, Jindal SG: Text analytics based severity prediction of software bugs for apache projects . International Journal of System Assurance Engineering and Management. 2019, 10:765-782. 10.1007/s13198-019-00807-8
32. Lamkanfi A, Pérez J, Demeyer S: The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information. 2013 10th Working Conference on Mining Software Repositories (MSR). 2013, 203-206. 10.1109/MSR.2013.6624028
33. Olaleye TO, Arogundade OT, Aborishade DA, Adeniran OJ, Ajayi O, Ahiara A: Semantic graph analytics for research trends in software defect prediction. Cureus Journal of Computer Science. 2024, 1:es44389-024-02851-3. 10.7759/s44389-024-02851-3