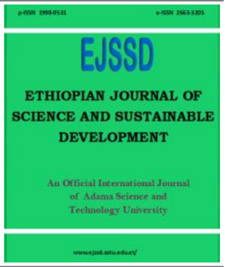
Ethiopian Journal of Science and Sustainable Development

e-ISSN 2663-3205 Volume 13 (1), 2026

Journal Home Page: www.ejssd.astu.edu.et



Research Paper

Enhanced Software Defect Prediction Using Ensemble Learning with Correlation-Based Feature Selection and SMOTE

Bahiru Shifaw Yimer[1*], Dita Abdujebar Abrahim[2]

[1]*Department of Computer Science & Engineering, Adama Science and Technology University, P.O. Box: 1888, Adama, Ethiopia*
[2]*Department of Software Engineering, Haramaya University, P.O. Box: 138, Haramaya, Ethiopia*

Article Info

Article History:
Received 20 June 2025
Received in revised form 28 October 2025
Accepted 07 November 2025

Keywords:
algorithms,
cross-validation,
extra tree,
grid search,
machine learning,
optimization

Abstract

Numerous studies have explored software defect prediction using machine learning algorithms; however, their performance on publicly available defect datasets often remains limited due to high feature dimensionality and class imbalance. This study addresses these issues using five AEEEM project datasets; namely, Eclipse Equinox (EQ), JDT, Apache Lucene (LC), Mylyn (ML), and PDE UI. Seven ensemble learning algorithms (AdaBoost, Gradient Boosting, XGBoost, Random Forest, Extra Trees (ET), Bagging, and Stacking) were implemented. To reduce dimensionality, three feature selection techniques, namely, Correlation-Based Feature Selection (CFS), Sequential Forward Selection (SFS), and Correlation-based Filter (CO), were applied, while the Synthetic Minority Oversampling Technique (SMOTE) method was employed to handle class imbalance. Experiments were conducted using 10-fold and nested cross-validation, and model performance was evaluated using accuracy, recall, precision, F-measure, and Area under ROC curve (AUC) metrics. The combination of CO feature selection with the ET ensemble algorithm outperformed all other models across the five datasets. Using nested cross-validation with grid search optimization, accuracies of 92.1, 97.3, 99.1, 98.2, and 98.5 % were achieved for the EQ, JDT, LC, ML, and PDE datasets, respectively. These findings demonstrate that integrating effective feature selection and data balancing significantly enhances defect prediction performance compared to models using default hyper-parameters.

1. Introduction

Software is a crucial technology used to solve societal problems across various domains of life. It is developed using different frameworks and programming languages. The software development process must be carefully controlled and monitored to ensure the production of high-quality software at an optimal cost. It is also essential to follow Software Quality Assurance practices such as code walkthroughs, software testing, code inspections, and software defect prediction (Adrion et al., 1982; Johnson & Malek, 1988; Rathore & Kumar, 2019) to achieve this goal.

Software companies and developers continue to face challenges in producing quality software due to the presence of defects throughout the development process. A software defect is an error, flaw, fault, or bug embedded in the requirements, design, or source code. These challenges negatively affect the functionality of the developed software and the experience of users interacting with the product. The problem also extends to the testing phase, where defects identified later in the cycle can significantly impact software quality,

*Corresponding author, e-mail: bahiru.shifaw@astu.edu.et

https://doi.org/10.20372/ejssdastu:v13.i1.2026.1111

reliability, and maintenance cost (Rawat & Dubey, 2012).

As software systems become increasingly complex and large, controlling, managing, fixing, reducing, and testing defects becomes more difficult (Alsawalqah et al., 2020). Therefore, attempting to control defects at any phase of the software development life cycle (SDLC), or detecting them early, helps developers, testers, and software companies deliver high-quality software (Rawat & Dubey, 2012). Software fault prediction, as stated by Alsawalqah et al. (2020) and Mehta & Patnaik (2021), is one of the most widely used approaches to reduce software development and maintenance costs during the testing phase of the SDLC. It is a technique designed to enhance software quality and efficiency by enabling timely detection of fault-prone modules before the actual testing process takes place (Rawat & Dubey, 2012; Alsawalqah et al., 2020). Therefore, to produce quality software, early detection of defects is essential before the testing phase begins. Software bug prediction approaches or models play a crucial role in this regard by classifying software modules into defective and non-defective categories.

Software metrics are also used to quantitatively describe the properties of specific software modules, thereby helping to identify defective software (Shatnawi & Li, 2008; Choudhary et al., 2018). Several researchers have investigated different approaches for building and evaluating software fault prediction models. These approaches include binary classification of faults (Li & Reformat, 2007; Mendes-Moreira et al., 2012; Vandecruys et al., 2008; Rathore & Kumar, 2019; Alsawalqah et al., 2020), bug or fault density prediction (Rathore & Kumar, 2019; Rathore & Kumar, 2016), and bug severity prediction (Sandhu et al., 2011; Shatnawi & Li, 2008). However, the most commonly implemented prediction scheme is binary classification, in which software modules are categorized as faulty or non-faulty.

Supervised learning approaches have also been widely used for defect prediction, including ensemble methods (Huda et al., 2018; Jiang et al., 2008), Decision Trees (Koprinska et al., 2007), Support Vector Machines (Elish & Elish, 2008), Naive Bayes (Menzies et al., 2007), Random Forest, Neural Networks, and Logistic Regression. In addition, unsupervised techniques such as fuzzy clustering (Yuan et al., 2002) and K-means clustering have been examined for bug prediction. Semi-supervised methods, such as Expectation Maximization (Seliya & Khoshgoftaar, 2007), have also been explored.

The performance of all the above approaches generally remains below 90% accuracy (Alsawalqah et al., 2020; Rathore & Kumar, 2019; Zhou et al., 2019). However, reviews of recent studies in software fault prediction show that ensemble learning methods are more effective and achieve better predictive performance (Aljamaan & Alazba, 2020; Matloob et al., 2021). Alsawalqah et al. (2017) conducted experiments using supervised ensemble classifiers such as Random Forest, Bagging, and AdaBoost, along with base classifiers including Multilayer Perceptron (MLP), C4.5 Decision Trees, and Naïve Bayes (NB), on NASA and PROMISE project datasets. Their findings indicated that AdaBoost combined with C4.5 Decision Trees outperformed the other models. Similarly, Zhou et al. (2019) applied supervised ensemble and deep learning approaches, including Cascade Deep Forest (DPDF), a hybrid ensemble and deep learning method, on 25 software projects from the NASA, PROMISE, AEEEM, and Relink datasets. Their results showed that DPDF achieved superior performance compared to other models.

Aljamaan & Alazba (2020) evaluated supervised ensemble classifiers such as Random Forest (RF), Extra Trees (ET), Adaptive Boosting (AdaBoost), Categorical Boosting (CatBoost), Extreme Gradient Boosting (XGBoost), Gradient Boosting (GB), and Histogram-Based Gradient Boosting (HGB) on NASA project datasets, finding that RF and ET outperformed the rest. Mehta (2021) applied several ensemble learning methods, including RF, ET, AdaBoost, Stacking, XGBoost, and Bagging, on NASA software project datasets, with XGBoost and Stacking achieving the best performance.

Generally, ensemble model offers improved predictive performance, enhanced stability and robustness, less overfitting, better handling of complex data, leveraging model diversity and flexibility. Mohammed & Kora (2023) assured that ensemble approach offers a state-of-the-art method and can bypass of the limitations in using a single model. Ensembles often achieve better accuracy in predictions as they combine multiple models. They are also less sensitive to outliers and noisy data leading to more consistent and reliable performance across different datasets. Combining diverse models helps to reduce the overall variance because of errors from model complexity as well as bias due to simplistic models making it less prone to overfitting the training data giving better generalization capability to new, unseen data (Mohammed & Kora, 2023). The rationale for using ensembles is their property of combining diverse base learners of different algorithms having different strengths and weaknesses on different training data obtained through adjusting weights thereby compensating each other's limitations to result in stronger overall predictive performance and better generalization.

The performance of the machine learning algorithms implemented by Balogun et al. (2020a) and Zhou et al. (2019) on the AEEEM software defect datasets was relatively low. For example, the study by Zhou et al. (2019) on software defect prediction using the Deep Forest (DPDF) model reported low accuracy due to high feature dimensionality and class imbalance. Feature selection technique can solve this problem by identifying most relevant set of features. Class balancing technique can also be used to remove class imbalance problem. However, software defect datasets are impacted by high feature dimensionality. The techniques of identifying best set of features resulting correct prediction were not extensively covered. Moreover, class balancing techniques to solve a biased classification problem well were not explored more. This research work proposes software defect prediction approaches that implement different ensemble algorithms with various base classifiers. The attempt is to increase the performance of prediction methods with SMOTE class balancing methods and extensive features selection mechanisms as preprocessing activity on AEEEM project datasets of software module's defect.

Therefore, this research is conducted to come up with a prediction model while selecting determinant attributes as well as finding effective ensemble learning algorithms through reducing bias and avoiding unbalanced classes. Hyper-parameter tuning was also done to get reliable and generalizable findings.

2. Materials and Methods

2.1 Data source and dataset description

Publicly available online datasets of AEEEM projects, gathered from different versions of software systems such as Eclipse and Apache and collected by D'Ambros et al. (2010), have been widely used in software defect prediction research (Balogun et al., 2020b; Zhou et al., 2019; You et al., 2016). The AEEEM dataset contains five software projects, and each project dataset includes a total of 62 attributes: one dependent (defect-proneness) attribute and 61 independent software metric attributes. These metrics are derived from a combination of CK (Chidamber and Kemerer) and object-oriented (OO) metrics, previous defect metrics, entropy of change metrics, churn-based source code metrics, and entropy of source code metrics (D'Ambros et al., 2010).

The CK_OO metric group consists of six CK metrics and eleven OO metrics, making a total of 17 metrics. The Previous Defects Metrics group contains five indicators of historical defect occurrences used to predict future defects: all bugs, non-trivial bugs (severity > trivial), major bugs (severity > major), critical bugs (critical or blocker), and high-priority bugs. The Entropy of Change Metrics group includes five measures that quantify the complexity of code changes across time. The Churn of Source Code Metrics group consists of code churn measures calculated using weighted churn deltas instead of simple lines-of-code churn. This group, known as Weighted Churn Metrics (WCHU), also contains 17 metrics. The Entropy of Source Code Metrics group computes entropy values directly from source code metrics rather than from change information, represented as Linearly Decayed Entropy metrics (LDHH), totaling another 17 metrics.

Each AEEEM project was developed for different purposes. Eclipse Equinox (EQ) is an implementation of the OSGi framework specification and provides core services and infrastructure for executing OSGi-based systems. JDT Core is used to support Java infrastructure within the Eclipse IDE, including the Java compiler, formatter, code assistance, and navigation support. Apache Lucene (LC) is a high-performance Java-based search engine library that provides full-text search, spell correction, structured search, and nearest-neighbor search capabilities. Mylyn (ML) is an Eclipse framework for task management and application lifecycle support, offering a task-focused interface and ALM integration tools. Finally, PDE UI is the Eclipse user interface framework used for developing, testing, debugging, and deploying Eclipse plug-ins. It includes OSGi tooling and provides project creation wizards, editors, launchers, conversion tools, user assistance tools, and integration with JDT.

Table 1 presents the detail description of AEEEM datasets of the five software projects with their number of attributes, number of modules (instances), programming language developed, number of defective and percentage of defective modules.

Table 1: Datasets of AEEEM projects developed with Java programming language and 61 features

AEEEM project purpose	№ of instances (Software modules)	Defective	% of defective
EQ	324	129	39.80
JDT	997	206	20.70
LC	691	64	9.30
ML	1862	245	13.20
PDE	1497	209	13.96
Total	5371	853	15.90

2.2 The proposed software defect prediction architecture

The proposed software defect prediction (SDP) architecture is designed to classify software modules into defective and non-defective categories using ensemble machine learning algorithms. As shown in Figure 1, the architecture integrates multiple techniques that support experimental framework of the SDP model.

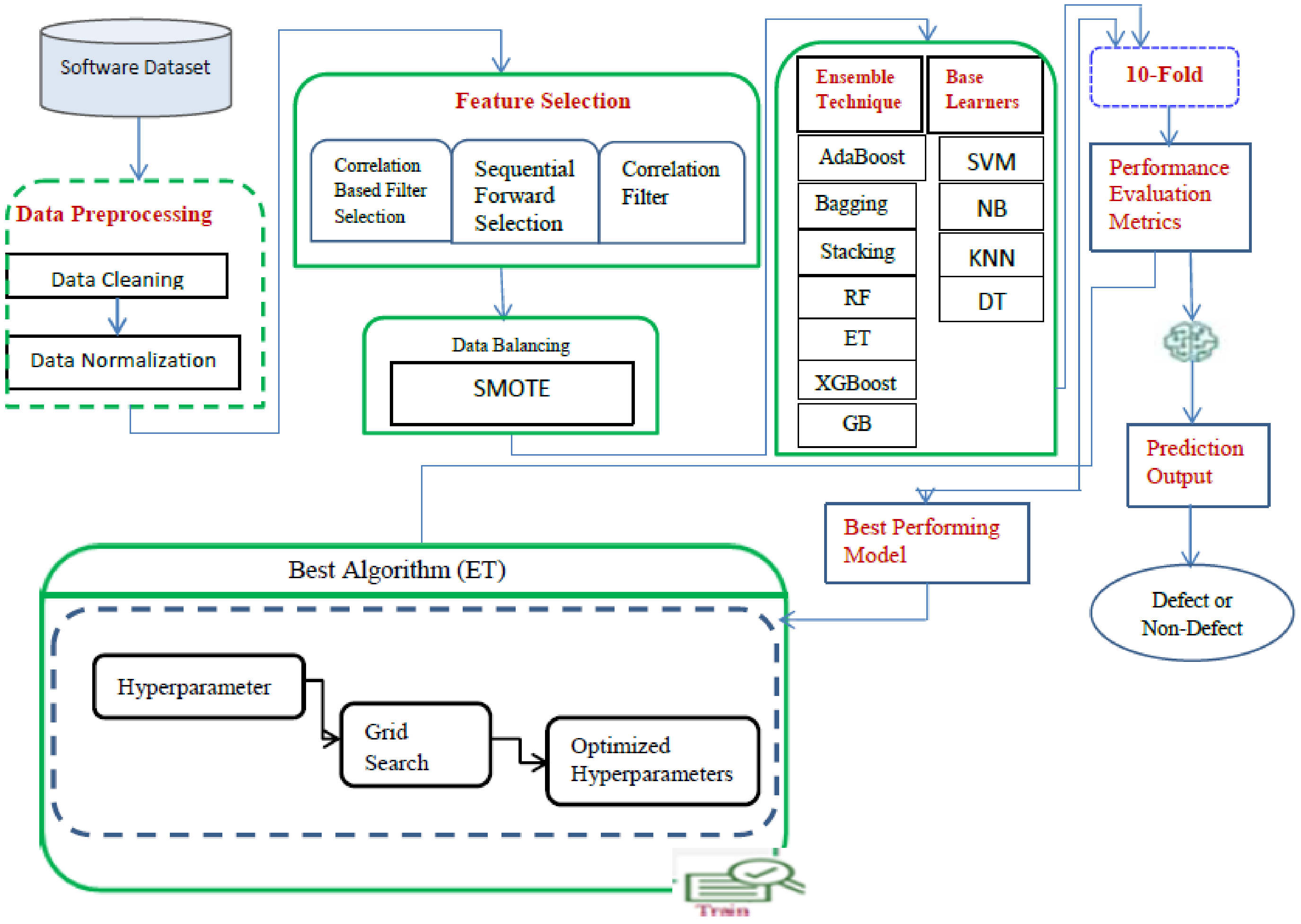


Figure 1: General architecture of Software defect prediction

The preprocessing phase includes data cleaning, data normalization using the Z-score method, dimensionality reduction through Correlation-Based Feature Selection (CFS), Sequential Forward Selection (SFS), and Correlation Filtering (CO), as well as data class balancing using the Synthetic Minority Oversampling Technique (SMOTE). CFS and CO are categorized under filtering methods while SFS is among wrapper methods used in data preprocessing steps of machine learning classifier. Hence, SFS was used with all the selected ensemble learning algorithms as well as with other selected base classifier in the study. CO feature selection technique uses brute force methods which can be implemented in terms of different threshold (0.7, 0.8, and 0.9) points which are experimented to extract substantial features. Feature selection techniques were applied to choose relevant set of attributes for enhancing prediction power of classifier algorithms and to minimize over fitting.

Ensemble learning algorithms such as AdaBoost, GB, XGBoost, RF, ET, Bagging, and Stacking are employed in the study. Furthermore, base classifiers including Support Vector Machine (SVM), Naive Bayes, Decision Tree, and K-Nearest Neighbor (KNN) are utilized. All the ensemble and base learning algorithms mentioned above are used to construct predictive models capable of identifying and classifying software modules as defective or non-defective.

The predictive performance of each model is assessed using standard performance evaluation metrics, namely, accuracy, recall, precision, F-measure, and Area under ROC curve (AUC) and the model demonstrating the highest accuracy and generalization capability was declared as the optimal one.

2.3 Hyper-performance optimization

Hyper-parameter tuning is the process of searching for and identifying the optimal parameter values of machine learning models to enhance their performance. Various hyper-parameter optimization techniques exist, including grid search, random search, genetic algorithms, and differential evolution. In software defect prediction, grid search, which systematically explores all possible combinations of parameter values within a defined search space, is often used (Mohammed & Kora, 2023). The search space consists of a set of hyper-parameters and their corresponding candidate values. For each combination, the grid search algorithm builds and evaluates a model, and the hyper-parameter values that yield the best-performing model are returned as the optimal settings. In this study, the hyper-parameters summarized in Table 2 were applied across the five datasets to obtain reliable and generalizable results.

3. Results and Discussion

3.1 Data preprocessing

All selected dataset features were normalized within the range of –3.00 to 3.00. As shown in Table 3, increasing the number of threshold points positively affected the model performance. The table summarizes the number of attributes selected by various feature selection methods. The datasets employed in this study suffer from class imbalance, causing the majority class to dominate the minority class during the machine learning training process. To address this issue, SMOTE was implemented to balance the dataset by oversampling the minority class instances.

Table 2: Hyper-parameters used for best performance ensemble learning (ET Approach)

Hyper-parameter name	Description value	Optimized value range
n_estimators	Number of trees in the forest	[10,50,200,300,400]
max_depth	Maximum depth of the tree.	[70,80,90]
min_samples_leaf	Minimum number of samples at a leaf node	[0.0001,1, 2] ['gini',
criterion	Measure the quality of a split.	['gini', 'entropy']
max_features	max number of features used for splitting a node	['auto', 'sqrt','log2']
min_samples_split	Minimum number of samples before the node is split	[2,3,5]

Table 3: The number of selected attributes for each algorithms in model building

Datasets	CFS	CO (threshold)			Ensemble learning algorithm						
		0.7	0.8	0.9	RF	ET	GB	XGBoost	BET	AE	SDF
EQ	17	14	26	39	11	7	47	7	16	7	9
JDT	12	16	27	45	31	5	50	12	23	26	6
LC	7	26	34	47	9	15	19	9	10	6	23
ML	16	24	34	47	34	26	47	18	14	38	15
PDE	15	20	28	45	11	20	30	16	13	28	15

BET: Bagging with Extra tree base learner, AET: Adaptive boosting with Extra tree base learner, SDF: Stacking by default final Meta learner with base weak learner (ET, RF and DT)

Figure 2 shows the sizes of both defective and non-defective classes for each of the five datasets, before and after data balancing. Before balancing, the dataset contained 4,518 non-defective and 853 defective modules, making a total of 5,371 instances. After applying the balancing technique, the total number of instances increased to 9,036, consisting of 4,518 non-defective and 4,518 defective modules. Specifically, the datasets EQ, JDT, LC, ML, and PDE contained 324, 791, 627, 1,617, and 1,288 non-defective modules, respectively, along with 129, 206, 64, 245, and 209 defective modules before balancing. After balancing, each dataset contained an equal number of defective and non-defective modules.

3.2 Performance prediction of the ensemble learning algorithms

The experimental results of the selected ensemble learning algorithms with the implementation of the feature selection techniques are depicted in Table 4.

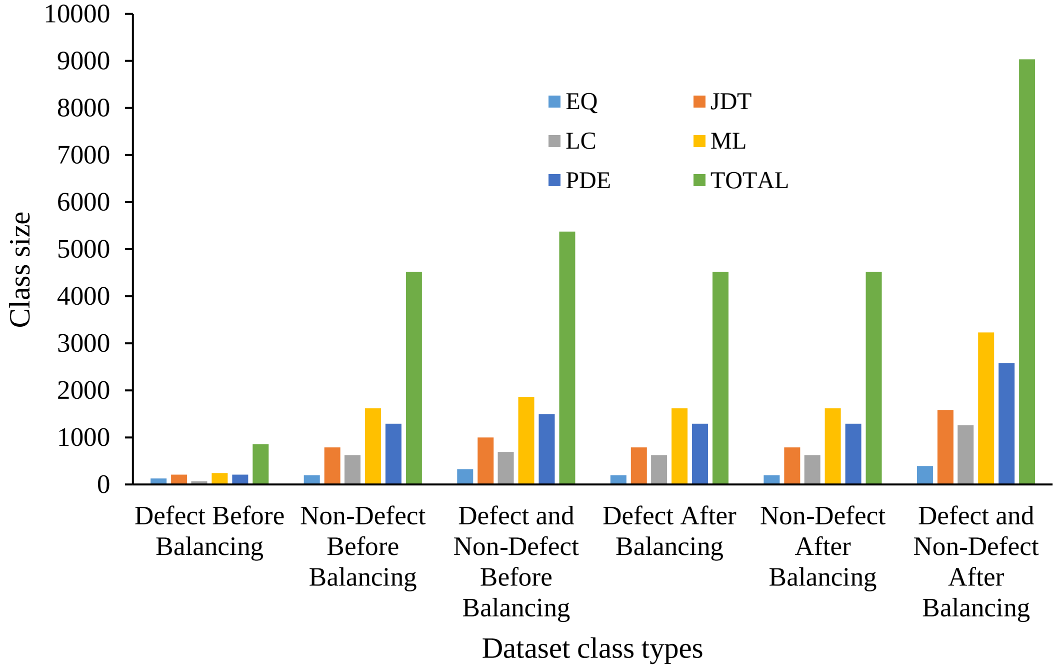


Figure 2: Size and class categories of datasets before and after balancing using SMOTE

Table 4: Accuracy result of the ensemble learning algorithms with the feature selection techniques

Algorithm	Datasets	CFS	CO (threshold)			SFS
			0.7	0.8	0.9	
Random Forest (RF)	EQ	0.903	0.915	0.915	0.913	0.944
	JDT	0.927	0.946	0.961	0.959	0.961
	LC	0.918	0.978	0.98	0.984	0.980
	ML	0.927	0.972	0.976	0.975	0.975
	PDE	0.949	0.966	0.971	0.975	0.963
Extra Tree (ET)	EQ	0.89	0.905	0.931	0.915	0.936
	JDT	0.937	0.958	0.963	0.97	0.963
	LC	0.927	0.994	0.992	0.989	0.988
	ML	0.93	0.977	0.982	0.982	0.982
	PDE	0.952	0.972	0.978	0.982	0.974
Gradient Boosting (GB)	EQ	0.897	0.879	0.91	0.91	0.923
	JDT	0.894	0.922	0.936	0.947	0.951
	LC	0.887	0.979	0.975	0.976	0.961
	ML	0.867	0.925	0.935	0.939	0.932
	PDE	0.847	0.923	0.926	0.937	0.932
Extreme Gradient Boosting (XGBoost)	EQ	0.905	0.923	0.908	0.905	0.928
	JDT	0.924	0.955	0.965	0.967	0.963
	LC	0.91	0.982	0.982	0.979	0.978
	ML	0.921	0.973	0.973	0.976	0.972
	PDE	0.929	0.965	0.97	0.97	0.97
Bagging of ET base learner	EQ	0.887	0.903	0.913	0.918	0.928
	JDT	0.934	0.948	0.96	0.965	0.958
	LC	0.919	0.99	0.99	0.987	0.977
	ML	0.926	0.975	0.981	0.981	0.975
	PDE	0.946	0.969	0.978	0.98	0.969
AdaBoost of ET base learner	EQ	0.882	0.903	0.918	0.918	0.954
	JDT	0.93	0.958	0.965	0.966	0.966
	LC	0.923	0.992	0.991	0.99	0.97
	ML	0.931	0.979	0.979	0.981	0.973
	PDE	0.953	0.972	0.981	0.981	0.983
Stacking of ET, RF and DT on base learner and default final estimator	EQ	0.903	0.905	0.918	0.908	0.933
	JDT	0.933	0.958	0.962	0.962	0.958
	LC	0.923	0.989	0.989	0.989	0.988
	ML	0.93	0.977	0.984	0.984	0.977
	PDE	0.953	0.974	0.981	0.981	0.966

For the Random Forest and Extra Trees algorithms, the Correlation Filter (CO) feature selection technique yielded better performance across all datasets except EQ. The SFS technique also demonstrated good performance on the EQ, JDT, and LC datasets. In the Gradient Boosting experiments, the SFS technique achieved superior performance on three datasets, namely, EQ, JDT, and PDE, while the CO technique performed well on the LC and ML models. For the Extreme Gradient Boosting algorithm, the CO feature selection method produced better results across all datasets except EQ, where SFS again performed well on EQ and PDE. All four ensemble learning algorithms employed the Decision Tree (DT) as their base classifier.

The remaining ensemble learning algorithms, Bagging, AdaBoost, and Stacking, utilized different base learners, including SVM, NB, DT, and KNN, among which the DT algorithm demonstrated superior performance. Both Bagging and AdaBoost employed the ET classifier as the base learner under different feature selection settings. In the Bagging experimental setup, the CO feature selection technique yielded better performance across all datasets except EQ, while the SFS technique performed well only on the EQ dataset. In contrast, for the AdaBoost experiments, the SFS technique showed better performance on the EQ, JDT, and PDE datasets, whereas the CO technique performed well only on the LC and ML datasets. For the Stacking experiments, where ET, RF, and DT were used as base learners with a default final estimator integrated with each feature selection technique, the CO method again produced better performance across all datasets except EQ. However, the SFS technique demonstrated good performance only on the EQ dataset.

Table 5 presents the performance comparison of three feature selection techniques, including the three CO threshold values, combined with seven ensemble machine learning algorithms across the five datasets. The results indicate that the CO feature selection technique achieved best performance compared to both SFS and CFS, with a 0.9 threshold value. The SFS technique also demonstrated good performance on some datasets.

Table 5: Performance values of all the ensembles learning algorithms

Datasets	Performance metrics	Ensemble Learning Algorithm						
		RF	ET	GB	XGBoost	BET	AET	SDF
EQ	Accuracy	0.913	0.915	0.91	0.905	0.918	0.918	0.908
	AUC	0.972	0.983	0.951	0.958	0.976	0.983	0.982
	F1-Measure	0.915	0.918	0.909	0.906	0.920	0.920	0.908
	Precision	0.900	0.900	0.900	0.889	0.897	0.898	0.902
	Recall	0.934	0.952	0.916	0.927	0.948	0.947	0.944
JDT	Accuracy	0.959	0.970	0.947	0.967	0.965	0.966	0.962
	AUC	0.994	0.996	0.985	0.990	0.995	0.996	0.995
	F1-Measure	0.959	0.97	0.946	0.967	0.966	0.965	0.962
	Precision	0.961	0.962	0.959	0.964	0.958	0.957	0.961
	Recall	0.957	0.979	0.935	0.971	0.974	0.974	0.964
LC	Accuracy	0.984	0.989	0.976	0.979	0.987	0.99	0.989
	AUC	0.999	1.000	0.995	0.998	0.999	1.000	0.999
	F1-Measure	0.984	0.989	0.976	0.979	0.987	0.991	0.989
	Precision	0.983	0.987	0.979	0.974	0.984	0.987	0.990
	Recall	0.986	0.998	0.973	0.985	0.990	0.995	0.987
ML	Accuracy	0.975	0.982	0.939	0.976	0.981	0.981	0.984
	AUC	0.996	0.997	0.984	0.995	0.998	0.996	0.997
	F1-Measure	0.975	0.982	0.939	0.975	0.981	0.979	0.981
	Precision	0.974	0.975	0.932	0.975	0.973	0.975	0.978
	Recall	0.976	0.990	0.947	0.976	0.990	0.984	0.99 0
PDE	Accuracy	0.975	0.982	0.937	0.970	0.98	0.981	0.981
	AUC	0.996	0.999	0.978	0.995	0.998	0.999	0.999
	F1-Measure	0.974	0.982	0.937	0.970	0.980	0.981	0.981
	Precision	0.975	0.973	0.933	0.969	0.972	0.972	0.976
	Recall	0.974	0.992	0.941	0.970	0.989	0.990	0.985

Further analysis of the CO feature selection technique across the seven ensemble algorithms was conducted to identify the best-performing ensemble learning models. Again, the CO method with a 0.9 threshold outperformed the other CO threshold settings. The experimental results show that ET ensemble algorithm achieved the best performance on most datasets, followed by AdaBoost and Stacking, which ranked second and third, respectively. In contrast, GB ensemble algorithm exhibited the lowest performance across all datasets.

3.3 Hyper-parameter of the best algorithm

As depicted in Table 6, hyper-parameters on the best performing ensemble learning model (ET approach) were further utilized to get the best performance of ET itself rather than taking default hyper-parameter. Here, Nested Cross Validation (10-fold Inner CV and 10-fold Outer CV) are applied. In other words, appropriate parameters on all five datasets are found using optimized value ranges with grid search algorithm where ET model resulted in the best output. Ten-fold inner loop CV and 10-fold outer loop CV are used during each iteration in order to the algorithm choose different combinations of the features from 5 * 3* 3 * 3 * 3 * 3 = 1,215 settings. In Table 6, results of all metrics of best hyper-parameters and default hyper-parameter are displayed with all datasets for the best algorithm.

Table 6: Software defect prediction using new deep forest (DPDF) (Zhou et al., 2019) and best performing ensemble learning (ET with best and default hyper-parameters)

Dataset	Performance Metrics	Hyper-parameter of ET		DPDF	Best hyper-parameters
		Best	Default		
EQ	Accuracy	0.926	0.915	0.78	*criterion='gini',max_depth=90,min_samples_leaf=1,min_samples_split=5,max_features='log2',n_estimators=10*
	AUC	0.970	0.983	0.85	
	F1-Measure	0.928	0.918	0.75	
	Precision	0.906	0.900	0.70	
	Recall	0.954	0.952	0.81	
JDT	Accuracy	0.973	0.970	0.85	*criterion='gini',max_depth=70,min_samples_leaf=1,n_estimators=300*
	AUC	0.997	0.996	0.86	
	F1-Measure	0.973	0.970	0.56	
	Precision	0.968	0.962	0.72	
	Recall	0.979	0.979	0.46	
LC	Accuracy	0.991	0.989	0.93	*criterion='gini',max_depth=70,min_samples_leaf=1,n_estimators=300*
	AUC	1.000	1.000	0.82	
	F1-Measure	0.991	0.989	0.37	
	Precision	0.985	0.987	0.81	
	Recall	0.998	0.998	0.24	
ML	Accuracy	0.982	0.982	0.87	*criterion='entropy',max_depth=80,max_features='log2',min_samples_leaf=1,n_estimators=400*
	AUC	0.997	0.997	0.82	
	F1-Measure	0.983	0.982	0.26	
	Precision	0.975	0.975	0.47	
	Recall	0.991	0.990	0.18	
PDE	Accuracy	0.985	0.982	0.87	*criterion='gini',max_depth=70,max_features='log2',min_samples_leaf=1,n_estimators=200*
	AUC	0.999	0.999	0.77	
	F1-Measure	0.985	0.982	0.31	
	Precision	0.979	0.973	0.59	
	Recall	0.991	0.992	0.21	

4. Conclusions

The goal of this research was to improve the performance of software defect prediction models, which has typically ranged between 70% and 90% in previous studies. To achieve this, five AEEEM datasets were used. Several preprocessing steps were performed to prepare suitable datasets for the selected algorithms. Z-score normalization was applied to reduce the effect of outliers, followed by feature selection and class balancing techniques to address high feature dimensionality and class imbalance, respectively. Finally, hyper-parameter optimization was conducted on the best-performing ensemble model to ensure reliable and generalizable findings.

The results show that the combination of the ET ensemble learning algorithm with CO feature selection and SMOTE-based data balancing achieved superior performance compared to previous studies across all five AEEEM datasets. All models developed in this study obtained accuracy values exceeding 90%, a level not achieved in earlier works. This improvement can be attributed partly to the normalization, SMOTE, and filtering techniques applied during data preprocessing, and partly to the use of grid search with nested cross-validation for hyper-parameter optimization of the ET algorithm, which was identified as the best-performing ensemble model through 10-fold cross-validation.

Overall, integrating ensemble learning algorithms with feature filtering mechanisms, data balancing techniques, and 10-fold resampling evaluation significantly enhanced the predictive accuracy of the software defect models used in this study. Nevertheless, future work should consider developing software defect prediction models using deep learning algorithms, which may further improve prediction accuracy. Software companies are also encouraged to prepare and release more software system datasets to support ongoing research in defect prediction. Additionally, conducting experiments on both private and publicly available datasets using alternative class balancing and feature selection methods may further strengthen software defect prediction performance.

Acknowledgements: The authors would like to thank Adama Science and Technology University and Haramaya University for their support during the course of the project related to this manuscript..

References

Adrion, W. R., Branstad, M. A., & Cherniavsky, J. C. (1982). Validation, Verification, and Testing of Computer Software. *ACM Computing Surveys*, 14(2), 159–192.

Aljamaan, H. & Alazba, A. (2020). Software Defect Prediction using Tree-Based Ensembles. In: Proceedings of the 16th ACM Int. Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2020).

Alsawalqah, H., Faris, H., Aljarah, I., Alnemer, L., & Alhindawi, N. (2017). Hybrid SMOTE-Ensemble Approach for Software Defect Prediction. In: Silhavy, R., Silhavy, P., Prokopova, Z., Senkerik, R., Kominkova Oplatkova, Z. (eds) Software Engineering Trends and Techniques in Intelligent Systems. CSOC 2017. Advances in Intelligent Systems and Computing, vol. 575. Springer, Cham.

Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Al Radaideh, A., Aljarah, I., & Alshamaileh, Y. (2020). Software defect prediction using heterogeneous ensemble classification based on segmented patterns. *Applied Sciences*, 10(5), 1745

Balogun, A. O., Basri, S., Mahamad, S., Abdulkadir, S. J., Almomani, M. A., Adeyemo, V. E., Al-Tashi, Q., Mojeed, H. A., Imam, A. A., & Bajeh, A. O. (2020b). Impact of Feature Selection Methods on the Predictive Performance of Software Defect Prediction Models: An Extensive Empirical Study. *Symmetry*, 12(7), 1147

Balogun, A.O., Lafenwa-balogun, F. B., Mojeed, H., & Adeyemo, V. E. (2020a). SMOTE-Based Homogeneous Ensemble Methods for Software Defect Prediction. In: Gervasi, O., et al. Computational Science and Its Applications - ICCSA 2020. Lecture Notes in Computer Science, Vol. 12254, pp. 615-631. Springer, Cham.

Choudhary, G.R., Kumar, S., Kumar, K., Mishra, A., & Catal, C. (2018). Empirical analysis of change metrics for software fault prediction. *Computers & Electrical Engineering*, 67, 15-24

D'Ambros, M., Lanza, M., & Robbes, R. (2010). An Extensive Comparison of Bug Prediction Approaches. Proceedings of 7th IEEE Working Conference on Mining Software Repositories (MSR), 2-3 May 2010, Cape Town, South Africa

Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81, 649–660.

Huda, S., Liu, K., Abdelrazek, M., Ibrahim, A., Alyahya, S., Al-Dossari, H., & Ahmad, S. (2018). An ensemble oversampling model for class imbalance problem in software defect prediction. *IEEE Access*, 6.

Jiang, Y., Cukic, B., & Ma, Y. (2008). Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13, 561–595.

Johnson, A. M., & Malek, M. (1988). Survey of software tools for evaluating reliability, availability, and serviceability. *ACM Computing Surveys*, 20(4), 227–269.

Koprinska, I., Poon, J., Clark, J., & Chan, J. (2007). Learning to classify e-mail. *Information Sciences*, 177 (10), 2167-2187

Li, Z. & Reformat, M. (2007). A practical method for the software fault-prediction. In: Proceedings of IEEE International Conference on Information Reuse and Integration, IEEE IRI-2007, 659–666.

Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., Abbas, S., & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. *IEEE Access*, 9, 98754–98771

Mehta, S., & Patnaik, K.S. (2021). Improved prediction of software defects using ensemble machine learning techniques. *Neural Computing and Applications*, 33, 10551–10562.

Mendes-Moreira, J., Soares, C., Jorge, A. M., & De Sousa, J. F. (2012). Ensemble approaches for regression: A survey. *ACM Computing Surveys*, 45(1), 10

Menzies, T., Greenwald, J., & Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, 33(1), 2–13.

Mohammed, A., & Kora, R. (2023). A comprehensive review on ensemble deep learning: Opportunities and challenges. *Journal of King Saud University - Computer and Information Sciences*, 35(2), 757-774

Rathore, S. S., & Kuamr, S. (2016). Comparative analysis of neural network and genetic programming for number of software faults prediction. *2015 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE)*, 328–332.

Rathore, S. S., & Kumar, S. (2019). A study on software fault prediction techniques. *Artificial Intelligence Review*, 51, 255–327.

Rawat, M. S., & Dubey, S. K. (2012). Software Defect Prediction Models for Quality Improvement: A Literature Study. *International Journal of Computer Science Issues*, 9(5), 288–296.

Sandhu P.S., Singh S., & Budhija N. (2011) Prediction of level of severity of faults in software systems using density based clustering. In: Proceedings of the 9th international conference on software and computer applications, IACSIT Press'11.

Seliya, N., & Khoshgoftaar, T. M. (2007). Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Quality Journal*, 15, 327–344.

Shatnawi, R., & Li, W. (2008). The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *Journal of Systems and Software*, 81(11), 1868–1882.

Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., & Haesen, R. (2008). Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software*, 81(5), 823–839.

You, G., Wang, F. & Ma, Y. (2016). An Empirical Study of Ranking-Oriented Cross-Project Software Defect Prediction. *Int. Journal of Software Engineering & Knowledge Engineering*, 26, (9 &10), 1511-1538

Yuan, X., Khoshgoftaar, T.M., Allen, E. B., & Ganesan, K. (2002). An Application of Fuzzy Clustering to Software Quality Prediction. Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 24-25 March 2000.

Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. *Information and Software Technology*, 114, 204–216.