Ingénierie des Systèmes d'Information
Vol. 31, No. 10, October, 2026, pp. 3545-3557
Journal homepage: http://iieta.org/journals/isi

Prediction of Cost-Sensitive Software Defect Based on Stacking Ensemble Learning

Asmaa Hadi Al Bayati*, Shaymaa A. Chyad, Ibrahim Ahmed Saleh

Department of Software, College of Computer Sciences and Mathematics, University of Mosul, Mosul 41002, Iraq

Corresponding Author Email: asmashade77@uomosul.edu.iq

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/isi.311025

Received: 12 November 2025
Revised: 15 September 2026
Accepted: 30 September 2026
Available online: 31 October 2026

Keywords:
software defect prediction, cost-sensitive learning, Shapley value, feature selection, boosting ensemble, class imbalance

ABSTRACT

Software defect prediction (SDP) is essential for improving software reliability and allocating testing resources efficiently. However, class imbalance and unequal misclassification costs remain major challenges, particularly when defective modules are incorrectly classified as non-defective. This study proposes a cost-sensitive ensemble learning (EL) method that integrates Shapley-based feature selection with boosting for SDP. First, a feature selection mechanism based on Shapley values is developed to estimate the marginal contribution of individual features to predictive performance. A cost-sensitive adjustment is then incorporated to account for the different consequences of false-negative and false-positive predictions, retaining features with positive cost-adjusted contributions. The selected feature subsets are subsequently incorporated into the boosting process, where sample weights are iteratively updated according to a predefined misclassification cost matrix. The proposed method, referred to as Cost-Sensitive Shapley Feature Selection (CSSFS), is evaluated on 20 benchmark datasets from NASA, AEEEM, and ReLink and compared with random forest (RF), decision tree (DT), AdaBoost, logistic regression (LR), and cost-sensitive boosting (CSBst). Performance is assessed using F1-score, recall, area under the receiver operating characteristic curve (ROC-AUC), and G-mean, with the Wilcoxon signed-rank test used to examine statistical differences. The results show that CSSFS achieves higher F1-scores on most datasets, with particularly clear improvements on highly imbalanced datasets. For example, it obtains F1-scores of 0.41 on MC1 and 0.45 on KC3. The method also achieves statistically significant improvements over RF on 18 of the 20 datasets. Although its training time is higher than that of several conventional classifiers, the additional computational cost remains moderate under the experimental conditions. These findings indicate that combining cost-sensitive feature selection with boosting can improve the identification of defective software modules, particularly in imbalanced datasets.

1. INTRODUCTION

Providing secure and reliable software is a fundamental requirement for development teams. Nevertheless, potential defects in the software can significantly impact its quality at different stages of development. Many defects inevitably arise due to human factors, communication problems, technical challenges, and other factors. For this purpose, it is essential to test software before release to detect and fix problems. In the actual development process, full software testing requires significant human resources, and testing units that are likely to have flaws must be thoroughly tested, as testing resources within a project are sometimes scarce [1].

Software defect prediction (SDP) creates predictive models that determine whether future instances (such as files, modules, or classes) will be defective or defect-free by using past data to train machine learning, statistical models, or data mining. Software testers can more efficiently allocate testing resources with the help of accurate prediction results. The process of SDP is shown in Figure 1. Researchers collected software development data, including code metrics, code base, bug reports, version control history, etc., and extracted features that help predict defects from the collected data. Next, divide the data into training and testing sets. Then, use the test set to evaluate the prediction model's performance and choose a suitable machine learning, data mining, or statistical model as a learner [2].

Since most software operates normally in reality, there are significantly fewer defect samples in the collected dataset than there are non-defect categories. As a result, the model tends to predict the majority non-defective class and performs less effectively when predicting the defective class.

There are two types of misclassifications: predicting defective units as non-defective and predicting non-defective units as defective. The first type of error leaves defective modules in the program. It has a major detrimental effect on program quality, while the second type of error results in an increased number of units tested and an increase in the cost of testing. When comparing the second type of error with the first one, the cost of the second type of error is much lower than the first type of error, so it is necessary to focus on improving the model's prediction performance for the defective class. Recent

relevant research shows that introducing the cost property into defect prediction models can effectively solve the problem of poor minority class prediction models caused by class imbalance [3, 4].

Software defect datasets may be obtained from various sources, such as defect tracking, software versioning, and software code review systems, which will often lead to datasets having high-dimensional and redundant features. The challenge of high-dimensional and redundant features makes data processing and model building more difficult, thus leading to poor model performance when tested on new data [5]. Dimensionality reduction is therefore employed to solve this problem. The methods currently used by researchers to predict software defects mainly include random forest (RF), support vector machines (SVM), Naive Bayes (NB), deep learning (DL), ensemble learning (EL), logistic regression (LR), etc. [6]. However, these methods cannot obtain good classification results when faced with the above-mentioned problems, such as class imbalance, different misclassification costs, high dimensions, and redundant features.

This study proposes a cost-sensitive stacking approach with integrated feature selection for SDP to address the aforementioned issues. The cost-sensitive approach is applied concurrently in the two phases of feature selection and integrated learning to address the problem of poor prediction performance for minority classes in the dataset. A new feature selection method is proposed.

This method obtains the contribution value of features based on the marginal contribution and the cost function of sample features and selects the features that produce positive contributions as the final feature set. Feature selection is integrated into the process of EL, and the most appropriate features are selected for each base classifier before training to improve its prediction performance, avoid overfitting, increase diversity among the base classifiers, and enhance the performance of the weighted aggregation stage.

While previous work has either applied cost sensitivity only during training (e.g., cost-sensitive boosting (CSBst)) or only at the decision threshold, the Cost-Sensitive Shapley Feature Selection (CSSFS) method presents two important contributions: (1) a cost-sensitive Shapley value, which imposes penalties on the contribution of features considering misclassification costs when selecting features; (2) a dual-stage cost-sensitive mechanism that incorporates misclassification costs during both feature selection and ensemble weighting. The CSSFS method is distinguished by its two-stage application of cost sensitivity: first, by utilizing the Shapley value accounting for misclassification costs during feature selection; and second, by incorporating these costs when determining the ensemble weights.

The rest of the article is organized as follows: Section 2 presents related work; Section 3 discusses cost-sensitive techniques; Section 4 explains feature extraction using the Shapley algorithm and the implementation of CSSFS; Section 5 presents the experimental setup and results; and Section 6 concludes the paper.

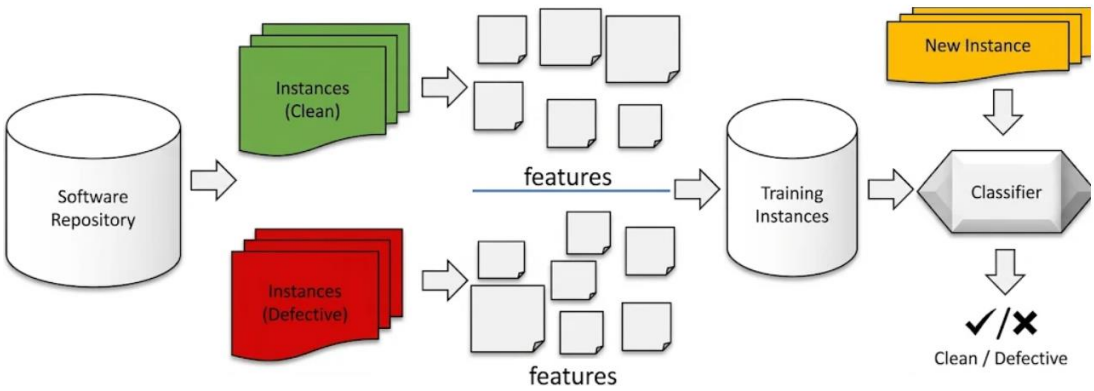


Figure 1. Software defect prediction (SDP) process

2. RELATED WORK

This section provides an overview and summarizes the previous research on cost-sensitive SDP using many machine learning methods.

Ibrahim et al. [7] suggested a technique for predicting software defects and tested it on very unbalanced method-level datasets from 23 open-source Java projects, where eight EL algorithms were tested: RF, Gradient Boost, Bagging, AdaBoost, Random Boost, Easy Ensemble, Balanced Bagging, and Balanced RF. According to the findings, the best recall and area under the receiver operating characteristic curve (ROC-AUC) values were obtained by the Balanced RF classifier.

Matloob et al. [8] utilized many machine learning modules and combined them to perform EL. Prior to training, Holdout normalizes and verifies the traditional metric features and calculates the area under the curve (AUC) value to gauge the impact of defect prediction. A stacking model is proposed based on this. The stacking model performs better than the conventional single machine learning model. In contrast, there is a significant chance that a single machine learning model would overfit and perform poorly.

Arar and Ayan et al. [9] proposed the CSBst algorithm, which uses a decision stump as the base classifier of the integrated model. During the model training process, the weight of samples that generate the first type of error is increased, and the threshold-moving method is utilized to increase the prediction accuracy of defect classes when the classifier is ultimately integrated. Although its approach of biasing the threshold towards defective classes in the integration stage will improve the recall rate of defective classes, the overall accuracy will be greatly reduced due to the data distribution in which non-defective classes account for the majority.

Ali et al. [10] contributed a research paper consisting of two

parts. The first part selects the most effective discriminating factors for accurately predicting software component defects. The second part applies decision-tree ensemble-based prediction and cost-sensitive LR models. The models that are suggested are evaluated using the most recent techniques in the literature regarding accuracy, AUC and recall. Following that, eleven datasets are used to evaluate the models, and the analysis and results indicate that the suggested prediction models perform better than the schemes found in the literature.

Niu et al. [11] introduced additional cost-sensitive discriminative regularization into the coding coefficients and a new cost-sensitive reconstruction loss function. An integrated cost-sensitive dictionary learning framework jointly optimizes both terms. By doing this, the reconstruction loss was as low as possible, leading to a more effective cost-sensitive dictionary for encoding test data features. Twenty-five software programs from four benchmark datasets were included in the experimental portion. NASA, AEEEM, ReLink, and Jureczko were the subject of comprehensive analyses. The outcomes show the efficacy of a trained cost-sensitive lexicon for SDP in comparison with ten cutting-edge SDP techniques.

Alsaeedi and Khan [12] used ten NASA datasets to conduct a study and compared these supervised machine learning and ensemble classifiers. According to the experimental findings, RF was the classifier that performed the best overall, most of the time when compared to the other models.

Hammouri et al. [13] evaluated decision tree (DT), NB, and artificial neural network (ANN) for Software Bug Prediction (SBP). In their investigation, three actual debugging datasets were employed. The outcomes were analyzed using metrics including root mean square error (RMSE), F1-score, recall, accuracy, and precision. Their study's findings demonstrated the strong performance of the evaluated classifiers. Table 1 illustrates a comparison among these related works.

Existing studies have typically addressed class imbalance or improving classification accuracy separately. However, little research has focused on simultaneously combining cost-sensitive feature selection with stacking EL. Furthermore, several methods suffer from reduced diversity among learners or increased computational complexity. Also, the literature indicates a scarcity of studies combining Shapley-based feature selection with cost-sensitive stacking for SDP, highlighting the need for broader study and comparison to confirm the novelty of this approach.

Unlike previous studies that apply cost-sensitive learning only during classification or ensemble training, the proposed CSSFS method integrates cost-sensitive mechanisms into both feature selection and stacking stages simultaneously. In addition, the proposed approach improves learner diversity by dynamically selecting different feature subsets for each base learner, which distinguishes it from conventional boosting and stacking approaches.

Table 1. Comparison among related works

Study	Method	Dataset	Strength	Limitation
Ibrahim et al. [7]	EL (RF, Gradient Boost, Bagging, AdaBoost, Random Boost, Easy Ensemble, Balanced Bagging, Balanced RF)	23 open-source Java projects	Balanced RF achieved the best recall and ROC-AUC on highly imbalanced datasets.	Performance depends on dataset imbalance and ensemble complexity.
Matloob et al. [8]	Stacking ensemble model	Traditional software metric datasets	Better performance than single machine learning models and reduces overfitting	Increased model complexity and training time
Li et al. [9]	CSBst	NASA	Improves recall of defective classes	Lower overall accuracy due to class imbalance
Ali et al. [10]	Cost-sensitive LR and DT Ensemble	11 datasets	Good discrimination and better prediction performance	Limited ensemble diversity
Niu et al. [11]	Cost-sensitive Dictionary Learning	NASA, AEEEM, ReLink, Jureczko	Handles imbalance and improves cost-sensitive prediction	Computational complexity
Alsaeedi and Khan [12]	RF and Ensemble Classifiers	10 NASA datasets	RF achieved the best overall performance	Limited evaluation of modern DL methods
Hammouri et al. [13]	DT, NB, ANN	3 real debugging datasets	Strong prediction performance using multiple evaluation metrics	Limited dataset size and diversity

Note: EL = ensemble learning; RF = random forest; DT = decision tree; NB = Naive Bayes; ANN = artificial neural network; LR = logistic regression; CSBst = cost-sensitive boosting; ROC-AUC = area under the receiver operating characteristic curve; DL = deep learning.

3. COST-SENSITIVE METHODS

The cost-sensitive method is a machine learning technique for estimating the costs of classification errors for different categories [14]. In many practical applications, different categories of errors may lead to different consequences, so traditional classification algorithms may not meet practical needs well. Cost-sensitive methods help machine learning models better adapt to this situation by considering different categories of costs. Cost sensitivity can be incorporated into certain algorithms, including LR and SVM, by changing the sample weights. Misclassified samples are given higher weights, thus paying more attention to these errors in model training [15].

The cost matrix is the core of the cost-sensitive classification method for binary classification problems; it is a two-dimensional matrix in which each element represents the cost associated with a particular combination of actual and predicted classes. Table 2 illustrates this by using rows to represent the actual classes and columns to represent the predicted classes. For clarity, 'NO' represents the negative class and 'YES' represents the positive class. $C_{\mathrm{NO,NO}}(x)$ represents the cost of a true negative (TN), $C_{\mathrm{YES,YES}}(x)$ represents the cost of a true positive (TP), $C_{\mathrm{NO,YES}}(x)$ represents the cost of misclassification for a false positive (FP), and $C_{\mathrm{YES,NO}}(x)$ represents the cost of misclassification for a false negative (FN). The cost matrix reflects the different consequences that can occur when misclassifying [16].

Table 2. Cost matrix

	Predicted Class	
Actual Class	**No**	**Yes**
No	$C_{\mathrm{NO,NO}}(x)$	$C_{\mathrm{NO,YES}}(x)$
Yes	$C_{\mathrm{YES,NO}}(x)$	$C_{\mathrm{YES,YES}}(x)$

For example, assume the following cost matrix for SDP:

•Correct classification costs are set to 0:

-($C_{\mathrm{NO,NO}} = 0$)

-($C_{\mathrm{YES,YES}} = 0$)

•Misclassifying a defective module as non-defective (FN) has a high cost:

-($C_{\mathrm{YES,NO}} = 10$)

•Misclassifying a non-defective module as defective (FP) has a lower cost:

-($C_{\mathrm{NO,YES}} = 1$)

Thus, the cost matrix becomes:

Actual Class	Predicted NO	Predicted YES
NO	0	1
YES	10	0

This means that failing to detect a defective module is considered ten times more costly than incorrectly predicting a clean module as defective.

3.1 Feature selection method

Feature selection refers to choosing a subset of the most representative and informative features from the original dataset [17]. The aim of feature selection is to reduce feature dimensionality while retaining the most useful information, thereby enhancing model performance, reducing the risk of overfitting, and speeding up training. In SDP, two primary feature selection techniques are commonly used: filter and wrapper, as shown in Figure 2.

•Filter-based feature selection filters features before model training. It usually uses statistics or correlations between features as evaluation criteria to evaluate the importance of features. Filtering methods usually do not involve model training, so they are computationally efficient [18].

•Wrapper-based feature selection uses a machine learning model in the feature selection process and continuously trains the model to evaluate the importance of different features. The advantage of the wrapper method is that it considers the relationship between features and the target; it can assess the significance of features more precisely [19].

Since the model needs to be trained multiple times, its computational cost is usually higher than that of the filtering method.

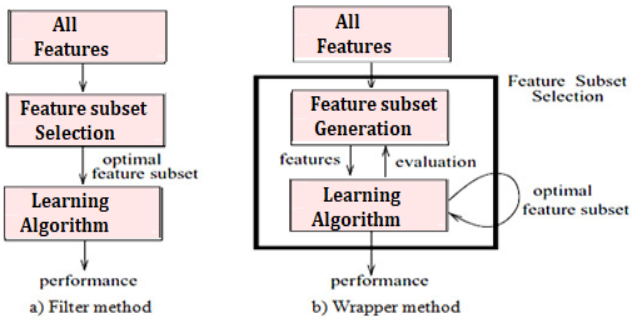


Figure 2. Workflow for filter and wrapper feature selection

3.2 Stacking ensemble learning

EL is a machine learning method that combines many weak learners into strong learners through a certain method, thereby improving the accuracy of model prediction [20]. The stacking method is a kind of EL. The benefit of stacking is that it learns how to effectively combine the predictions of multiple machine learning models. As illustrated in Figure 3, first initialize the weight of each sample to 1/M, where M is the total number of samples. Then use the current sample weights and samples to train the weak learner, calculate its error rate, and combine the predictions of the base learners to produce the final ensemble prediction.

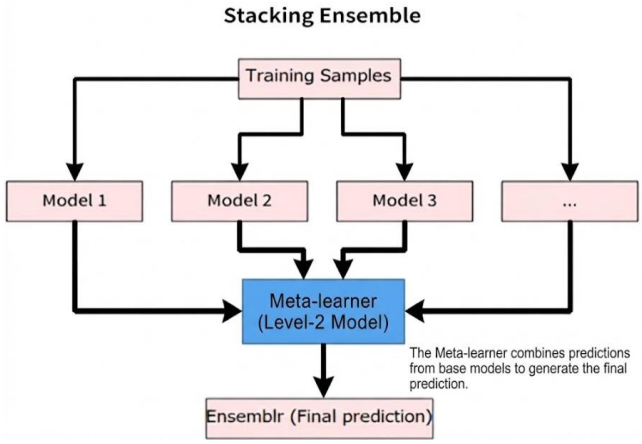


Figure 3. Cost-sensitive boosting (CSBst)-based ensemble process

4. COST-SENSITIVE SHAPLEY FEATURE SELECTION METHOD

The proposed cost-sensitive ensemble method integrates two components: a cost-sensitive feature selection algorithm and a cost-sensitive ensemble algorithm. Since the costs of different misclassifications are different in software engineering practice, it is necessary to use cost-sensitive methods in the feature selection and integration stages, respectively, and finally combine the two algorithms into the hybrid method.

4.1 Feature selection based on the Shapley algorithm

This paper proposes a cost-sensitive feature selection algorithm to effectively identify redundant and irrelevant features in software defect datasets and retain features that are strongly associated with the defect class. Cooperative game theory serves as its foundation and determines the marginal contribution of features by calculating the Shapley value [21] of each sample feature. On this basis, the cost-sensitive idea is added to reduce the corresponding contribution value according to the cost matrix for the sample features that produce Type I and Type II errors. Finally, the contribution values of each feature are added and averaged as the average contribution value, and the features that have a positive effect on the prediction results are selected as the final feature subset.

4.1.1 Calculate feature contribution value

Use the Shapley value to measure the contribution of features to prediction results. For each feature, add its average

marginal contribution under different feature orders to obtain the Shapley value of the feature as shown in Eq. (1):

$$\emptyset_i = \sum_{S \subseteq x\{x_i\}} \frac{|S|!\,(n-|S|-1)!}{n!} \left(val_x\,(S \cup \{x_i\}) - val_x(S)\right) \tag{1}$$

where, $\emptyset_i$ is the Shapley value of *i*-th feature, S is all feature subsets except x_i, x is a sample instance for calculating the contribution value $[x_1, x_2, \cdots, x_n]$, n is the total number of features $\frac{|S|!(n-|S|-1)!}{n!}$ is the weight of the feature subset S. $val_x\,(S)$ is the model prediction on the feature subset S, which can be computed using the equation that follows:

$$val(S) = \int \hat{f}\,(x) d\,\mathbb{P}_{x \notin S} - E_X\,(\int \hat{f}\,(x)) \tag{2}$$

where, $\hat{f}(x)$ is the prediction of x, $E_X\,(\int \hat{f}\,(x))$, $\int \hat{f}\,(x) d\,\mathbb{P}_{x \notin S}$ is to perform multiple integrations on features that do not belong to the feature subset S. The Shapley value φ_i obtained from Eq. (1) is an arbitrary real number, which can be a positive number, a negative number, or zero.

•A positive number indicates the feature has a positive contribution to the model prediction.

•A negative number indicates a negative contribution to the model prediction.

•Zero indicates that the feature has no impact on the prediction.

Since the Shapley value is not inherently bounded, and it will affect classification performance if it is too large or too small, its value is mapped to the range [−1, 1] through Eq. (3) to obtain the normalized Shapley value φ_i. At the same time, it facilitates subsequent cost-sensitive implementation.

$$\widetilde{\emptyset}_i = \frac{e^{\emptyset_i} - e^{-\emptyset_i}}{e^{\emptyset_i} + e^{-\emptyset_i}} \tag{3}$$

4.1.2 Introduce cost-sensitive methods

It can be seen from Table 2 that $C_{Yes,No(x)}$ represents the cost of the first type of classification error, $C_{No,Yes(x)}$ represents the cost of the second type of classification error. The cost of the second type of error is usually set to 1, while the cost of the first type of error is greater than 1. The specific value depends on the actual situation. The calculation of the cost-sensitive method can be implemented by Eqs. (4) and (5):

$$\text{cost}_x = \begin{cases} 1 & y_x = h(x) \\ e^{C_{YES,NO}\,(x)} & y_x = 1 \text{ and } h(x) = 0 \\ e^{C_{NO,YES}(x)} & y_x = 0 \text{ and } h(x) = 1 \end{cases} \tag{4}$$

$$\theta_i = \theta_i \times \text{cost}_x \tag{5}$$

Among them, y_x represents the actual value, $h(x)$ represents the model-predicted value, and $cost_x$ represents the cost adjustment function of the sample x. For correctly classified samples, $y_i = h(x_i)$, the θ_i value of its feature remains unchanged. For the first type of misclassification, $y_i = 1$ and $h(x_i) = 0$, e exponential term is less than zero, and its value is controlled by the cost matrix, reducing θ_i value of the corresponding feature. For the second type of misclassification, $y_i = 0$ and $h(x_i) = 1$, the exponential term of e is also less than zero. However, because the second kind of misclassification is less expensive than the first, the ϕ_i value of the corresponding feature decreases less in the first category. Taking the above three situations into consideration, the cost-sensitive normalized Shapley value ϕ_i is finally obtained.

4.1.3 Perform feature selection

Before selecting features, it is essential to consider each feature's contribution across all samples. The average contribution of the feature is calculated for Eq. (6), where m is the total number of samples, and ω_i is the contribution of *i*-th feature to the model in the current dataset.

$$\omega_i = \frac{1}{m} \sum_{j=1}^{m} \theta_{i,j} \tag{6}$$

Feature selection's objective is to determine the characteristics that best support prediction, so the final feature is selected based on the feature contribution obtained by Eq. (7). The feature selection procedure is as follows: select features that make a positive contribution to the prediction model and add them to the final feature subset R. The final R is the desired feature set.

$$\text{R} = \{\omega_i | \omega_i \in \{\omega_1, \omega_2, \ldots\ldots, \omega_n\}, \omega_i > 0\} \tag{7}$$

4.2 Cost-sensitive boosting-based ensemble algorithm

The proposed CSSFS method integrates cost-aware feature selection based on Shapley values with cost-aware EL based on boosting this algorithm for cost-aware based learning strategy which sample weights are repeated updated depended to predicate performance of each learner. The implemented algorithm relies on boosting mechanisms, specifically, updating sample weights and aggregating learner weights; consequently, the methodological terminology was revised to reflect the framework actually implemented.

The ensemble model consists of some base models trained sequentially. Initially, all samples are assigned equal weights; in each iteration, suitable feature subsets are selected using a cost-aware procedure based on Shapley values (Section 4.1). The base model is trained using the selected features and current weights, after which a cost-aware classification error is calculated based on a predefined cost matrix. The model's weight is determined based on this weighted error, and sample weights are updated to increase the importance of misclassified samples—particularly defective units misclassified as non-defective.

After iterations are complete, the predictions from the base learners are combined using their respective weights to produce the final prediction of the ensemble model. Therefore, the most accurate description is a CSBst model with feature selection, rather than traditional stacking. This clarification does not alter the algorithm or the results; rather, it ensures that the terminology aligns with the actual procedures.

In order to make the prediction model pay more attention to the first type of classification error problem, the cost-sensitive mechanism is introduced into the boosting-based ensemble algorithm by adjusting the distribution weight of data in the algorithm training. It merges many predictions from various base models to produce a final prediction with improved performance. It is sometimes referred to as stacking generalization or stacked ensembles. Multiple learners that

perform well in different ways are used, and each sample is initially assigned the same weight $D_1(x, y) = \frac{1}{M}$. The error rate of each learner $\sum_m D_n(x_m, y_m)\, y_m\, f_n(x_m)$ is then calculated, and the learner weight is determined based on the error rate $\beta_n = \frac{1}{2}\, ln\left(\frac{1+\gamma_n}{1-\gamma_n}\right)$.

The final prediction is obtained as a weighted average of the base learners. The difference is that in the next data weight update stage, the sample weight is updated according to the cost matrix. The weight update method is shown below:

$$G(x) = sign\left(\sum_{n=1}^{N} \beta_n\, f_n(n)\right) \tag{8}$$

The sample weight is updated according to the cost matrix. The weight update method is shown below:

$$D_{n+1}(x, y) = \frac{D_n(x, y)\exp\left(-cost\ \beta_n\, y_m\, f_n(x_m)\right)}{Z_n} \tag{9}$$

where, $D_{n+1}(x, y)$ is the updated data weight. The sample weight is adjusted in a cost-sensitive manner through the exponential term $-cost\ \beta_n\, y_m\, f_n(x_m)$. The cost parameter (cost) in this exponential term is obtained from Eq. (4). The cost matrix divides the sample data into three categories. The importance levels are the first type of error samples, the second type of error samples, and correctly classified samples according to the degree of importance, so that the learner can be trained according to these three importance levels in the next round of training. Iteration: After training all learners, integrate all weak learners into a strong classifier according to Eq. (10). During the integration, the weak learners are weighted according to error rate, and the sign function sign () is used on the weighted results to obtain the final prediction result.

$$G(x) = sign(\sum_{n=1}^{N} \beta_n\, f_n(x)) \tag{10}$$

4.3 Cost-Sensitive Shapley Feature Selection method implementation

CSSFS is composed of the feature selection algorithm in Section 4.1 and the integration algorithm in Section 4.2, which is illustrated in pseudo-code in Figure 4.

CSSFS integrates the cost-sensitive feature selection method into the EL algorithm. By applying feature selection before training each base learner, the most appropriate subset of features for the base learner is determined to improve its prediction performance. Figure 5 shows the CSSFS method process. Since different base learners may benefit from different subsets of informative features, feature selection can improve the diversity among base learners and ultimately improve the predictive performance of the ensemble model. The CSSFS method applies a cost-sensitive idea to both feature selection and integration phases to reduce the incidence of type I classification errors and improve prediction performance.

The CSSFS method has overall time $O(T{\cdot}n{\cdot}m{\cdot}C{+}K{\cdot}E{\cdot}m{\cdot}d)$, where n is the number of features, m is the number of samples, and K is the number of base learners. For typical NASA datasets ($n \leq 40$, $m \leq 10^4$), training is feasible within minutes on standard hardware (16 GB of random access memory (RAM)). Space complexity is $O(m{\cdot}n)$, acceptable for real-world use. However, datasets with $n > 200$ features require efficient Shapley approximation.

```
Input:
   Dataset D
   Cost Matrix C
   Feature Set F

Output:
Optimal Feature Subset S
  1. Compute Shapley value for each feature.
  2. Normalize feature contribution.
  3. Apply cost-sensitive adjustment.
  4. Compute average contribution.
  5. Select features with positive contribution.
  6. Train stacking ensemble.
  7. Update sample weights using cost matrix.
  8. Generate final prediction.
```

Figure 4. Pseudocode of Cost-Sensitive Shapley Feature Selection (CSSFS) algorithm

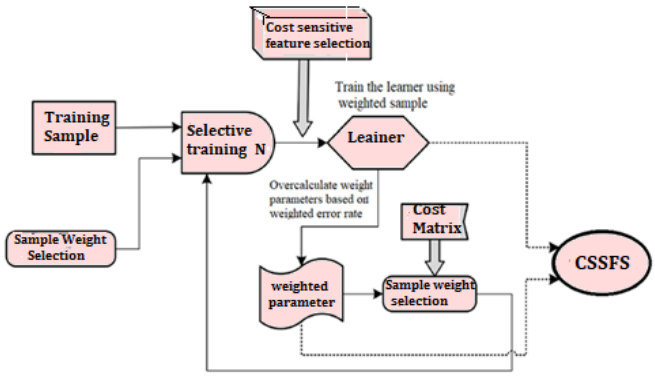


Figure 5. Cost-Sensitive Shapley Feature Selection (CSSFS) algorithm process

5. EXPERIMENTAL SETUP AND RESULTS

To ensure reproducibility, all experiments were implemented in Python using Scikit-learn libraries. The random seed was fixed to 42 for all experiments to ensure reliable and reproducible results. This seed controls randomly involved in generating the training and testing partitions during cross-validation. Missing values were removed before training, and min-max normalization was applied to all numerical features. The experiments were conducted using 5-fold cross-validation repeated 10 times, and the reported results represent the average performance across all runs. The number of estimators for RF and AdaBoost was fixed at 100, while the learning rate was set to 0.1. All experiments were executed on a Windows 10 machine equipped with an AMD Ryzen 5 5600 CPU and 16 GB RAM. The hyperparameters are shown in Table 3.

5.1 Introduction to datasets

This paper uses datasets from three sources [22-24]:
- NASA datasets: A collection of 12 datasets
- AEEEM dataset compiled by D'Ambros et al. [22]
- ReLink datasets reported by Zhao et al. [24]

Table 3. Parameters of the system

Parameter	Value
Random Seed	42
Cross Validation	5-Fold
Repetitions	10
AdaBoost Estimators	100
AdaBoost Base Estimator	DT
AdaBoost Learning Rate	0.1
RF Tree Depth	10
DT Splitting Criterion	Gini
CSSFS Cost Matrix	FN = 10, FP = 1
CSSFS Feature Selection	Positive Cost-Sensitive Shapley Contribution
RF Trees	100
Learning Rate	0.1
Normalization	Min-Max
Programming Language	Python
Framework	Scikit-learn

Note: RF = random forest; DT = decision tree; FN = false negative; FP = false positive; CSSFS = Cost-Sensitive Shapley Feature Selection.

NASA contains a total of 12 ,datasets including CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5. The lowest percentage of defective grades is 2.31%, and the highest is 27.53%, indicating class imbalance, and the phenomenon is more prominent. AEEEM has a total of 5 datasets: EQ, JDT, LC, ML, and PDE. Number of features: 61. The lowest percentage for defect categories is 9.26%, and the highest percentage is 39.81%. ReLink has three datasets: Apache, Safe, and Zxing. Number of features: 26. The lowest percentage for defect categories is 29.57%, and the highest is 50.52%. The class distribution is relatively balanced. The details of the dataset are shown in Table 4.

The Wilcoxon signed-rank test was used to compare CSSFS with the RF, DT, AdaBoost, LR, and CSBst classifiers using F1-score results across 20 datasets. A significance level of $p \leq 0.05$ was adopted. The results shown in Table 5 indicate that CSSFS obtained statistically significant improvements over the benchmark classifiers across several datasets, particularly those with high class imbalance (MC1, KC3, PC1, and PC2). CSSFS also achieved significant differences in the highly imbalanced datasets (MC1, KC3, PC1, and PC2), indicating that the effectiveness of the algorithm is more pronounced when there is a class imbalance problem.

Table 4. Dataset information

Source	Dataset	Granularity	No. of Features	Instances	Defective Modules	Defect Proportion (%)
NASA (12 datasets)	CM1	Function	37	327	42	12.84
	JM1	Function	21	7782	1672	21.49
	KC1	Function	21	1183	314	26.54
	KC3	Function	39	194	36	18.56
	MC1	Function	38	1988	46	2.31
	MC2	Function	39	125	44	35.2
	MW1	Function	37	253	27	10.67
	PC1	Function	37	705	61	8.65
	PC2	Function	36	745	16	2.15
	PC3	Function	37	1077	134	12.44
	PC4	Function	37	1287	177	13.75
	PC5	Function	38	1711	471	27.53
AEEEM (5 datasets)	EQ	Class	61	324	129	39.81
	JDT	Class	61	997	206	20.66
	LC	Class	61	691	64	9.26
	ML	Class	61	1862	245	13.16
	PDE	Class	61	1497	209	13.96
ReLink (3 datasets)	Apache	File	26	194	89	45.88
	Safe	File	26	56	22	39.29
	Zxing	File	26	399	118	29.57

Table 5. Signed-rank test *p*-values (CSSFS vs. baselines)

Comparison	No. of Datasets with $p < 0.05$	No. of Datasets with $p > 0.05$	*p*-Values Range
CSSFS vs RF	18/20	2/20	0.0001–0.041
CSSFS vs DT	17/20	3/20	0.0003–0.048
CSSFS vs AdaBoost	14/20	6/20	0.002–0.062
CSSFS vs LR	16/20	4/20	0.001–0.055
CSSFS vs CSBst	15/20	5/20	0.004–0.051

Note: RF = random forest; DT = decision tree; LR = logistic regression; CSBst = cost-sensitive boosting; CSSFS = Cost-Sensitive Shapley Feature Selection. The *p*-value range represents the minimum and maximum *p*-values across all 20 datasets.

5.2 Experimental setup

This paper uses a total of 20 datasets from three sources in Table 4 to conduct experiments, uses 5-fold cross-validation, and performs 10 experiments to obtain the average value. RF, DT, AdaBoost, CAdaboost, and CSBst methods were selected as baselines for comparison, and the results of each experiment were statistically analyzed. Different cost factors $C_{YES,NO(x)}$ are also selected to investigate the impact of the cost matrix on prediction results and determine the cost value required to achieve the best prediction performance.

Table 6. Confusion matrix

		Predicted	
		Negative (N) –	Positive (P) +
Actual	Negative –	True negative (TN)	False positive (FP) Type I error
	Positive +	False negative (FN) Type II error	True positive (TP)

Since SDP is a two-class problem, Table 6 displays the confusion matrix, which can be used to represent defect prediction results, where TN represents the total number of non-defective samples correctly predicted as non-defective, and TP represents the number of defective samples correctly predicted as defective. FP signifies the number of samples that are not flawed but expected to be defective, and FN represents the number of defective samples incorrectly predicted as non-defective. According to the confusion matrix, other evaluation indicators such as accuracy, recall, false positive rate (FPR), F1-score, and G-mean can be calculated. Among them, Precision represents the proportion of samples predicted as defective that are actually defective. The calculation formula is:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{11}$$

The recall, also called the true positive rate (TPR), represents the ratio of defective samples correctly predicted by the model to all actual defective samples. The calculation formula is:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{12}$$

FPR represents the percentage of non-defective samples that were incorrectly predicted as defective samples. The calculation formula is:

$$\text{FPR} = \frac{FP}{FP + TN} \tag{13}$$

The F1-score combines precision and recall into a single measure, and its value ranges from 0 to 1. The closer the score is to 1, the better the balance between precision and recall, and the better the model performs. The calculation formula is:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{14}$$

G-mean is a comprehensive assessment indicator that takes into account the ability to predict defective and non-defective categories at the same time. The G-mean ranges from 0 to 1, and values closer to 1 indicate better predictive performance across defective and non-defective classes. For unbalanced datasets, a higher value of the G-mean indicates that the model effectively identifies two categories, as shown in the calculation equation:

$$\text{G} - \text{mean} = \sqrt{\text{Precision} \times \text{Recall}} \tag{15}$$

AUC [6] represents the area under the ROC curve, which plots the TP rate against the FPR across classification thresholds. Value range: 0.5 to 1. When AUC is equal to 0.5, it indicates that random guessing is as effective as the model. The closer the AUC is to 1, the better the model will be able to categorize positive and negative categories. This experiment uses Python; the development environment is PyCharm 2023; the operating system used is Windows 10; the central processing unit (CPU) is an AMD Ryzen 5 5600 hexagonal processor at 3.50 GHz; and the memory is 16 GB.

Before training, all preprocessing steps, including missing value removal, min-max normalization, and cost-sensitive Shapley-based feature selection, were performed independently within each training fold of the 5-fold cross-validation. Similarly, feature selection was conducted using only training fold data, and the selected feature subset was subsequently used to transform the test fold. No information from the test fold was used at any stage of model training or feature selection. The random seed was fixed at 42 to ensure reproducibility. The number of estimators in AdaBoost and RF was set to 100. The learning rate was fixed at 0.1.

5.3 Result analysis

To verify whether the observed improvements of CSSFS are statistically significant, the Wilcoxon signed-rank test was conducted between CSSFS and the baseline classifiers using F1-score results across all datasets. The results demonstrated that CSSFS achieved statistically significant improvements at a significance level of $p < 0.05$ on most datasets.

5.3.1 Comparison of experimental results

The experimental comparison between the proposed CSSFS method and the baseline classifiers was conducted using 20 datasets with varying defect ratios and imbalance levels. Table 7 presents the F1-score results obtained by the five prediction algorithms.

Table 7. F1-score of 5 algorithms on 20 datasets

Dataset	RF	DT	AdaBoost	LR	CSBst
CM1	0.14	0.25	0.27	0.3	0.36
JM1	0.28	0.33	0.33	0.34	0.41
KC1	0.41	0.39	0.39	0.43	0.43
KC3	0.14	0.32	0.40	0.35	0.45
MC1	0.25	0.36	0.37	0.18	0.41
MC2	0.47	0.50	0.50	0.5	0.52
MW1	0.23	0.24	0.27	0.3	0.32
PC1	0.31	0.34	0.33	0.36	0.41
PC2	0.00	0.06	0.06	0.1	0.1
PC3	0.19	0.32	0.31	0.36	0.37
PC4	0.55	0.53	0.52	0.53	0.58
PC5	0.49	0.48	0.49	0.53	0.53
EQ	0.74	0.64	0.64	0.69	0.69
JDT	0.59	0.55	0.55	0.6	0.64
LC	0.35	0.35	0.35	0.31	0.38
ML	0.32	0.34	0.34	0.44	0.45
PDE	0.31	0.30	0.30	0.38	0.39
Apache	0.74	0.66	0.68	0.72	0.71
Safe	0.61	0.59	0.62	0.63	0.66
Zxing	0.42	0.41	0.44	0.49	0.53

Note: RF = random forest; DT = decision tree; LR = logistic regression.

•The F1-scores of the five prediction models across the 20 datasets are compared. The results demonstrate that CSSFS achieves the best or competitive F1 performance on most datasets, particularly on highly imbalanced datasets such as MC1, KC3, MW1, and PC1. This improvement indicates that integrating cost-sensitive learning with boosting-based EL and feature selection enhances the classifier's ability to identify defective modules while reducing the bias toward majority classes.

•The CSSFS F1-score is only slightly lower than the highest-performing classifier on two datasets and is equal to the highest score on four datasets. On the PC2 dataset, the defect proportion is only 2.15%. On the MC1 dataset, which contains only 2.31% defective modules, CSSFS achieved an F1-score of 0.41, outperforming RF (0.25), DT (0.36),

AdaBoost (0.37), and LR (0.18). Similarly, on the KC3 dataset, CSSFS obtained an F1-score of 0.45, representing the highest performance among all compared methods.

•The performance of all prediction algorithms is relatively low, and the RF F1-score reaches 0, indicating that it fails to identify the defective class in this case.

•CSSFS achieves higher F1-scores than the baseline methods on several datasets. In the KC3, MW1, and PC1 datasets, the CSSFS algorithm has the highest degree of improvement compared to other algorithms.

CSSFS achieves the highest F1-score on most datasets, particularly in KC3: CSSFS (0.45) outperforms RF (0.14) and DT (0.32).

MC1: CSSFS (0.41) outperforms LR (0.18) and RF (0.25); PC1: CSSFS (0.41) outperforms all others. For relatively balanced datasets (e.g., Apache, EQ, MC2), all algorithms perform similarly. For PC2, which has only a 2.15% defect rate, all algorithms perform poorly (F1 $\leq$ 0.1) due to severe imbalance, as shown in Figure 6.

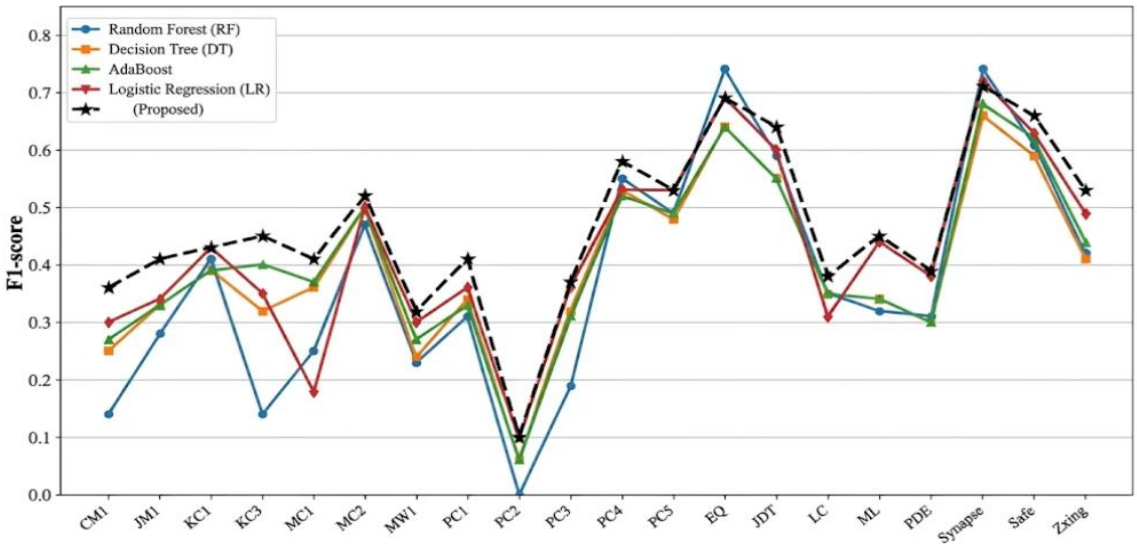


Figure 6. F1-score performance comparison across 20 datasets

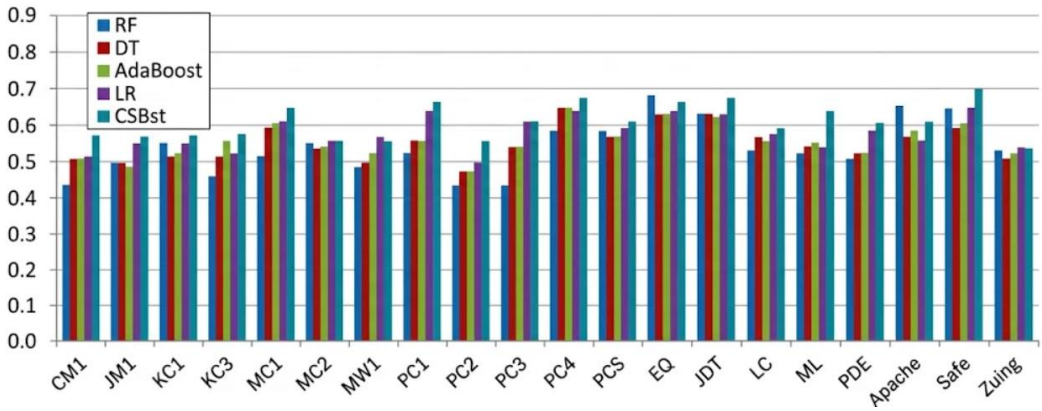


Figure 7. Recall of 6 algorithms on 20 datasets

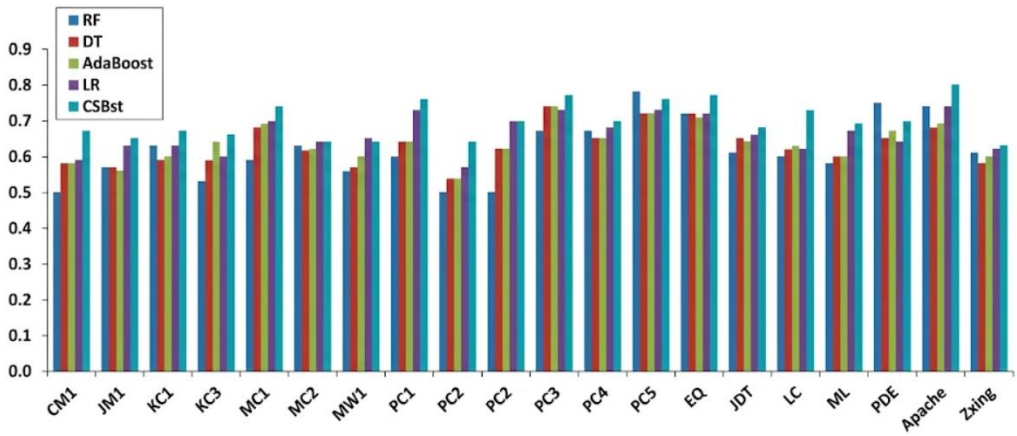


Figure 8. Area under the curve (AUC) of 6 algorithms on 20 datasets

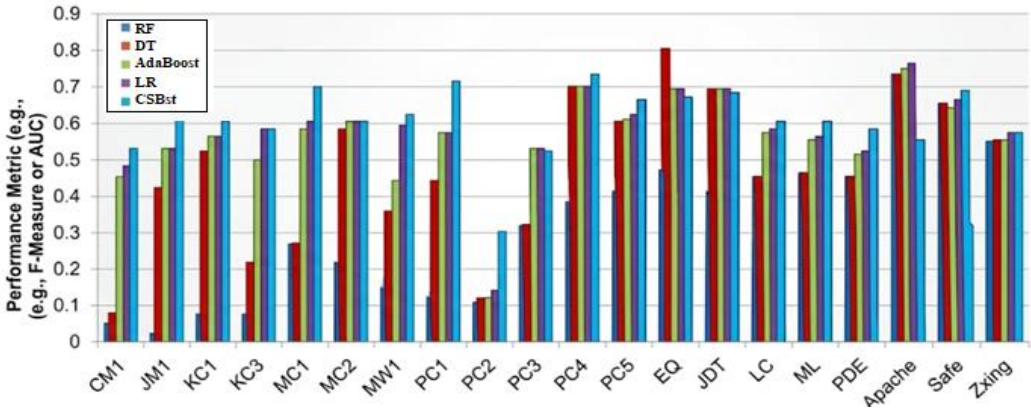


Figure 9. G-mean of 6 algorithms on 20 datasets

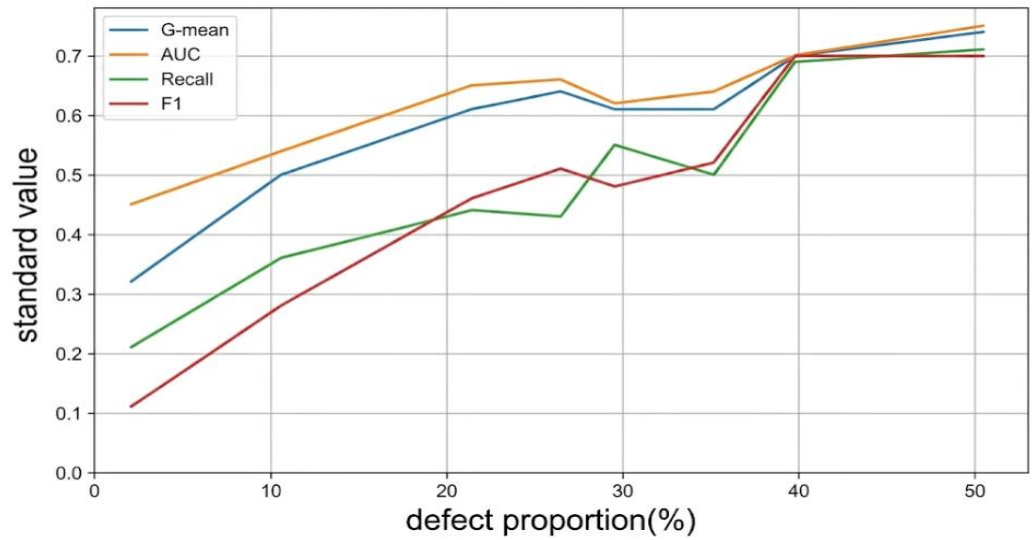


Figure 10. Performance of the Cost-Sensitive Shapley Feature Selection (CSSFS) algorithm under different defect proportions

On relatively balanced datasets such as Apache and Safe, the performance gap between CSSFS and other classifiers becomes smaller. This behavior suggests that the proposed method is particularly effective for handling class imbalance problems rather than balanced classification tasks. Figures 7–9 provide a visual comparison of recall, AUC, and G-mean across all datasets. Figure 7 shows that CSSFS consistently achieves higher recall values on most datasets, indicating improved detection of minority defective samples. This is especially important in SDP because missing defective modules may lead to significant software failures and maintenance costs.

Figure 8 illustrates that CSSFS achieves more stable and higher AUC values on most datasets, suggesting that it can effectively distinguish defective from non-defective samples under different classification thresholds. The graph shown in Figure 10 shows that the CSSFS algorithm has a higher AUC index than other algorithms in most datasets, suggesting that it can effectively distinguish defective and non-defective samples under different probability thresholds. The overall prediction performance of the CSSFS algorithm is better than that of other algorithms.

It can be seen in Figure 9 that CSSFS provides superior G-mean performance compared with baseline methods, indicating a better balance between minority and majority class predictions. This balance is particularly evident in highly imbalanced datasets, where traditional classifiers tend to favor the majority class.

It can be seen that, except for individual datasets, the CSSFS algorithm performs better than the other algorithms in various evaluation indicators on most datasets. Observations show that these datasets are MC2, EQ, Apache, Safe, and Zxing. The defect proportions of these datasets are 35.2%, 39.81%, 50.52%, 39.29%, and 29.57%, respectively, which are relatively balanced. It can be seen that the prediction performance of the CSSFS algorithm on balanced data is not much different from other algorithms, but it outperforms other algorithms on unbalanced data. Therefore, the CSSFS algorithm is more suitable for the prediction of unbalanced data.

To investigate the CSSFS algorithm's prediction ability under various defect ratios, the defect ratios in the dataset and their evaluation indicators are plotted in a line chart as shown in Figure 10, where the coordinates are the defect ratio and the value of each indicator. The figure shows that as the defect ratio continues to increase, the performance of the forecast model also improves. When the defect ratio is low, the performance of the forecasting model improves rapidly. After the defect ratio reaches 20%, the improvement in performance is relatively stable. The results indicate that CSSFS performs better when the defect proportion exceeds 20%.

5.3.2 Effect of dataset imbalance

The experimental results indicate that the proposed CSSFS

method performs significantly better on highly imbalanced datasets such as MC1, PC1, and KC3. This improvement can be attributed to the cost-sensitive weighting mechanism, which assigns higher importance to minority defective samples during both feature selection and ensemble training. In contrast, on relatively balanced datasets such as Apache and Safe, the performance gap between CSSFS and traditional classifiers becomes smaller because class imbalance has less influence on the learning process.

5.3.3 Analysis of minority and majority classes

To further evaluate classifier behavior, the prediction performance for minority defective classes and majority non-defective classes was separately analyzed. Results show that CSSFS achieves higher recall and G-mean values compared with baseline methods, indicating stronger capability in identifying minority defective modules while preserving balanced classification performance for majority classes. In particular, lower FN rates demonstrate that the proposed cost-sensitive mechanism effectively reduces missed defective samples.

5.3.4 Sensitivity analysis of cost matrix settings

The choice of cost ratio between FN and FP directly influences cost-sensitive learning behavior. In this paper, we assume that the cost of overlooking a defective software module is ten times the cost of misclassifying a non-defective module, based on the reference values $C_{\{YES, NO\}} = 10$ and $C_{\{NO, YES\}} = 1$.

To evaluate the robustness of CSSFS under different cost settings, we conducted a sensitivity analysis by varying the FN cost across {1, 5, 10, 20, 50} while keeping the FP cost fixed at 1. Experiments were performed on four representative imbalanced datasets (MC1, KC3, PC1, and PC2) using the same experimental protocol described in Section 5.2. As shown in Table 7 and Figure 10, the F1-score improved as the cost of FN increased, with marginal gains observed beyond the 10:1 ratio. This confirms the robustness of the CSSFS framework and the effectiveness of cost-sensitive learning in addressing class imbalance; it also indicates that the model is robust regarding the precise cost value, provided a reasonable penalty is imposed on FN. The 10:1 ratio offers a balanced and practical setting, while the cost matrix remains adjustable to suit specific application needs.

5.3.5 Computational cost and training time analysis

Although the theoretical time complexity of CSSFS is provided in Section 4.2, a practical comparison of training time and computational resources is necessary to evaluate its feasibility in real-world scenarios. Table 8 reports the average training time and peak memory usage of CSSFS compared with the baseline methods (RF, DT, AdaBoost, LR, and CSBst) across all 20 datasets, using the same hardware configuration and 5-fold cross-validation.

As shown in Table 8, CSSFS requires more training time than single-model methods such as DT and LR, mainly due to the cost-sensitive Shapley-based feature selection and iterative sample-weight updates. However, its computational overhead remains moderate compared with CSBst, which also incorporates cost-sensitive boosting. The slightly higher memory consumption of CSSFS results from storing feature contributions and cost-adjusted sample weights but remains practical for the datasets considered.

For high-dimensional datasets ($n > 200$), approximate or sampling-based Shapley estimation can further reduce the computational cost, as discussed in Section 6. Overall, the results indicate that CSSFS introduces an acceptable computational overhead while providing improvements in F1-score, recall, and G-mean.

Table 8. Training time and computational resource comparison

Method	Average Training Time (s)	Peak Memory (MB)	Time Complexity
RF	12.4	420	O(T.n.m.log m)
DT	4.8	180	O(n.m..log m)
AdaBoost	18.6	510	O(T.n.m)
LR	3.2	120	O(n.m)
CSBst	22.1	580	O(T.n.m.C)
CSSFS (proposed)	27.1	640	O(T.n.m.C+K.E.m.d)

Note: RF = random forest; DT = decision tree; LR = logistic regression; CSBst = cost-sensitive boosting; CSSFS = Cost-Sensitive Shapley Feature Selection; MB = megabytes.

6. CONCLUSIONS

This study proposes a cost-sensitive ensemble method with integrated feature selection for SDP to address high dimensionality and class imbalance. First, in order to solve high-dimensional problems, a cost-sensitive feature selection method was proposed for feature selection by calculating the marginal contribution value for each sample feature. At the same time, in order to retain more features related to defective category samples when selecting subsets of features, a cost-sensitive idea is introduced in the feature contribution calculation process, and the different feature contribution values are changed according to the cost matrix, thus reducing the occurrence of errors.

Secondly, the cost-sensitive feature selection method is embedded into EL, and the feature selection method is added to each base learner, which effectively improves the prediction performance of the base learner, reduces similarity among base learners, increases learner diversity, and increases the accuracy of the base learner. At the same time, the cost-sensitive idea is introduced in EL, and the sample weight is adjusted according to the sample misclassification situation and the cost matrix during sample iteration so that the first type of erroneous samples can receive more attention in the next round of training.

In addition, since some noise data inevitably exists in the software defect dataset, the boosting algorithm may treat the noise data as misclassified data during the iterative process, thus paying excessive attention to the noise data and affecting the final prediction performance. Therefore, future work will explore:

(1) Approximated Shapley kernels for high-dimensional

data,

(2) Cross-project defect prediction scenarios, and

(3) Noise-tolerant boosting mechanisms that distinguish between genuine misclassifications and label noise.

Experiments have shown that the CSSFS method requires a moderate amount of additional training time while maintaining efficiency on standard hardware, whereas using an approximate Shapley estimation is recommended for high-dimensional data to reduce computational cost.

ACKNOWLEDGMENT

The authors thank the University of Mosul's College of Computer Sciences and Mathematics for providing the facilities that improved the quality of this work.

REFERENCES

[1] Kumar, H., Saxena, V. (2024). Software defect prediction using hybrid machine learning techniques: A comparative study. Journal of Software Engineering and Applications, 17(4): 155-171. https://doi.org/10.4236/jsea.2024.174009

[2] Li, N., Shepperd, M., Guo, Y.C. (2020). A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, 122: 106287. https://doi.org/10.1016/j.infsof.2020.106287

[3] Zhao, W.D., Zhang, S.D., Wang, M. (2022). Software defect prediction method based on cost-sensitive random forest. In 12th International Conference on Intelligent Information Processing, Qingdao, China, pp. 369-381. https://doi.org/10.1007/978-3-031-03948-5_30

[4] Wang, S.S., Chi, G.T. (2024). Cost-sensitive stacking ensemble learning for company financial distress prediction. Expert Systems with Applications, 255: 124525. https://doi.org/10.1016/j.eswa.2024.124525

[5] Ali, U., Aftab, S., Iqbal, A., Nawaz, Z., Bashir, M.S., Saeed, M.A. (2020). Software defect prediction using variant based ensemble learning and feature selection techniques. International Journal of Modern Education and Computer Science, 12(5): 29-40. https://doi.org/10.5815/ijmecs.2020.05.03

[6] Catal, C., Sevim, U., Diri, B. (2011). Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. Expert Systems with Applications, 38(3): 2347-2353. https://doi.org/10.1016/j.eswa.2010.08.022

[7] Ibrahim, A.M., Abdelsalam, H., Taj-Eddin, I.A.T.F. (2023). Software defect prediction at method level using ensemble learning techniques. International Journal of Intelligent Computing and Information Sciences, 23(2): 28-49. https://doi.org/10.21608/ijicis.2023.189934.1251

[8] Matloob, F., Ghazal, T.M., Taleb, N., et al. (2021). Software defect prediction using ensemble learning: A systematic literature review. IEEE Access, 9: 98754-98771. https://doi.org/10.1109/ACCESS.2021.3095559

[9] Arar, Ö.F., Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. Applied Soft Computing, 33: 263-277. https://doi.org/10.1016/j.asoc.2015.04.045

[10] Ali, A., Khan, N., Abu-Tair, M., Noppen, J., McClean, S., McChesney, I. (2021). Discriminating features-based cost-sensitive approach for software defect prediction. Automated Software Engineering, 28: 11. https://doi.org/10.1007/s10515-021-00289-8

[11] Niu, L., Wan, J.W., Wang, H.Y., Zhou, K.W. (2020). Cost-sensitive dictionary learning for software defect prediction. Neural Processing Letters, 52: 2415-2449. https://doi.org/10.1007/s11063-020-10355-z

[12] Alsaeedi, A., Khan, M.Z. (2019). Software defect prediction using supervised machine learning and ensemble techniques: A comparative study. Journal of Software Engineering and Applications, 12(5): 85-100. https://doi.org/10.4236/jsea.2019.125007

[13] Hammouri, A., Hammad, M., Alnabhan, M., Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2): 78-83. https://doi.org/10.14569/IJACSA.2018.090212

[14] Araf, I., Idri, A., Chairi, I. (2024). Cost-sensitive learning for imbalanced medical data: A review. Artificial Intelligence Review, 57: 80. https://doi.org/10.1007/s10462-023-10652-8

[15] Nikolaou, N., Edakunni, N., Kull, M., Flach, P., Brown, G. (2016). Cost-sensitive boosting algorithms: Do we really need them? Machine Learning, 104: 359-384. https://doi.org/10.1007/s10994-016-5572-x

[16] Zhang, J.Y., Cai, Z.W., Yu, J., et al. (2025). Light field reflection matrix model for three-dimensional surfaces of isotropic materials. Optics Express, 33(16): 33768-33785. https://doi.org/10.1364/OE.568330

[17] Pudjihartono, N., Fadason, T., Kempa-Liehr, A.W., O'Sullivan, J.M. (2022). A review of feature selection methods for machine learning-based disease risk prediction. Frontiers in Bioinformatics, 2: 927312. https://doi.org/10.3389/fbinf.2022.927312

[18] Bolon-Canedo, V., Remeseiro, B. (2020). Feature selection in image analysis: A survey. Artificial Intelligence Review, 53: 2905-2931. https://doi.org/10.1007/s10462-019-09750-3

[19] Patel, D., Saxena, A., Wang, J. (2024). A machine learning-based wrapper method for feature selection. International Journal of Data Warehousing and Mining 20(1): 1-33. https://doi.org/10.4018/IJDWM.352041

[20] Mohammed, O.Y., Saleh, I.A. (2024). Prediction wireless network traffic evaluation potential based on ensemble algorithms. In 2025 IEEE 22nd International Multi-Conference on Systems, Signals & Devices (SSD), Monastir, Tunisia, pp. 914-921. https://doi.org/10.1109/SSD64182.2025.10989849

[21] Rozemberczki, B., Watson, L., Bayer, P., et al. (2022). The Shapley value in machine learning. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, Vienna, pp. 5572-5579, https://doi.org/10.24963/ijcai.2022/778

[22] D'Ambros, M., Lanza, M., Robbes, R. (2010). An extensive comparison of bug prediction approaches. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa, pp. 31-41. https://doi.org/10.1109/MSR.2010.5463279

[23] Iqbal, A., Aftab, S., Ali, U., et al. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. International Journal of Advanced Computer Science and Applications, 10(5): 300-308.

https://doi.org/10.14569/IJACSA.2019.0100538
[24] Zhao, Y., Zhu, Y., Yu, Q., Chen, X.Y. (2021). Cross-project defect prediction method based on manifold feature transformation. Future Internet, 13(8): 216. https://doi.org/10.3390/fi13080216