Received: February 29, 2024. Revised: March 25, 2024.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

FSDP: Frequent Software Defects Prediction Based on Defect Correlation Learning for Quality Software Development

Sareddy Shiva Reddy[1]* **Suresh Pabboju[2]**

[1]*Department of Computer Science and Engineering,*
JNTUH University College of Engineering, Science & Technology Hyderabad, India
[2]*Department of IT, CBIT, Osmania University, Hyderabad, India*
* Corresponding author's Email: reddyshiva996655@gmail.com

Abstract: Software has become an essential and important part of every domain system. Developing quality software is critical to maintaining a stable and secure system. Most of the existing software defect prediction tasks focus on the various kinds of defects leftover in the software system, but they do not focus on the most common and frequent software defects that developers most commonly do and which have a significant effect on the quality of software development. These unnoticeable defects have a considerable impact on the functionality and also on development time, effort, and cost of the software. This paper propose a frequent software defects prediction (FSDP) mechanism based on defects Correlation learning method (CLM) utilizing various defects metrics. The aim of this work is to assist developers in accurately identifying software defects and support project managers in ensuring software quality by minimizing the presence of defect-prone code during development. The evaluation of FSDP was performed using NASA datasets in comparison with the conventional Naive Bayes, Support Vector Machine, and Random Forest classifiers and also compare with the state-of-the-arts methods to measure accuracy, precision, recall and F-Score to understand the impact of the prediction accuracy. The proposed FSDP achieves 98.88% accuracy and 98.86% precision in compare to state-of-the-arts classifiers methods indicates the effectiveness of the proposed approach in defect prediction.

Keywords: Frequent defects, Correlation, Software defect prediction, Machine learning.

1. Introduction

The challenges of software development are considerable, and addressing defects early in the development cycle is crucial to mitigate potential consequences. The cost of fixing defects increases exponentially as the development process progresses. Early detection and correction are cost-effective compared to addressing issues later in the development or even post-deployment stages. Identifying defects in the early stages helps in preventing downstream issues that might require extensive rework and resources. So, addressing issues at the inception of development prevents the accumulation of defects, resulting in a more robust and reliable system [1].

Defect prediction methods based on regression techniques [2] aim to identify and quantify the relationship between various software metrics and the likelihood of defects. The primary goal is to build a predictive model that can estimate the number or density of defects in software based on the values of independent variables or attributes. It is useful for providing quantitative estimates of defect proneness [3]. Classification mechanisms [4-6] are commonly employed in these methods to categorize instances as either defective or non-defective based on various attributes.

The past studies methods [7, 8, 10] typically rely on historical data, including information about previously identified defects and corresponding software metrics, to train and validate the models. The goal is to build models capable of generalizing well to new, unseen data and effectively identify

Table 1. Notation and Description

Abbreviation	Description
CLM	Correlation Learning Method
CO-FSD	Correlation-FSD
DPM	Detection Profile Method
DT	Decision Trees
FSD	Frequent Software Defect
FSDP	FSD Prediction
LOC	Lines of Code
LR	Logistic Regression
LDA	Linear Discriminant Analysis
ML	Machine Learning
MLP	Multilayer Perceptron
NB	Naive Bayes
OO	Object Orientation
RF	Random Forest
SD	Software Defect
SDP	Software Defect Prediction
SDLC	Software development life cycle
SHL-MLP	Single hidden layer-MLP
SVM	Support Vector Machine

potential software defects during development or maintenance phases. It helps identify which software components are more likely to contain defects, allowing for targeted quality assurance efforts.

The effectiveness of the model depends on the quality and relevance of the selected attributes and the underlying assumption that the relationship between metrics and defects is captured accurately by the chosen regression method [11-13]. But, defect prediction in software development is a challenging task, and despite advancements in classifiers, there are still several issues and research areas that remain under exploration. In this work we focus on the problem of frequent software defects occurrences due to interdependency in the analysis and coding the various system blocks which are based on conditional statements, loops, or case statements, which may affect the performance of software applications. Identifying and rectifying these common issues can significantly improve the overall quality of software. But, the presence of unnoticeable FSDs can indeed have a significant impact on the functionality, development time, effort and cost implications and also the quality assurance challenges.

Early detection and resolution of defects can help minimize their impact on both the functionality and the overall development process [14]. Hence, in this paper we propose a mechanism for Frequent Software Defect Prediction (FSDP) in software development using a Correlation Learning Method (CLM) with a focus on defects metrics. The primary goal of the designed learning method is to improve the prediction performance of classifiers by recognizing and exploiting the close correlation among Attribute-Set Dependencies (FSD). FSDs are identified as the most interpretative defects, often occurring if prior defects exist in the software. A most common example FSD is observed software defect is, when a variable being declared but not initialized for a conditional check or a variable being initialized for a loop without proper increment for termination which effects the software quality development and can lead to performance instability. Here we contribute the following to achieve the benefit the attribute software development:

- Correlation Learning Method (CLM) to predict defects in software development, specifically targeting defects metrics dependencies, and highlights the potential benefits for project teams.
- The CLM is applied to NASA datasets to address the defect prediction problem by finding associations between defects using defect correlation. This approach aims to improve the overall quality of software development by identifying and addressing potential defects early in the development process.
- The outcome is expected to help software project teams in reducing costs and efforts by enhancing defect prediction accuracy. This, in turn, contributes to better software quality and stability.

A process for FSDP is performed using the WEKA tool and NASA datasets. The FSDP mechanism is trained and then empirically evaluated its performance on each dataset and measures the accuracy, precision, recall and F-score. The notation utilized in this paper is described in Table 1.

The following paper is organized in four sections. Section 2, discuss the related works related software defects metrics and predictions. Section 3, present the proposed FSDP methodology and function process. Section 4, discusses the evaluation measures and its results comparisons. Finally, Section 5, summarize the paper conclusion.

2. Related works

Software Defect Prediction (SDP) is crucial for enhancing the efficiency of the software development process, saving time, and minimizing the resources required for defect correction. By identifying potential defects early in the development process, SDP aims to reduce the time and resources spent on fixing issues later in the software development life cycle [15-17]. It aligns with the broader goal of improving the quality of software products and reducing the burden on both users and the companies dependent on the software.

Received: February 29, 2024. Revised: March 25, 2024.

Most of the SPD works activities that involve using statistical methods, Capture-Recapture (CR) models, and Detection Profile Method (DPM) for coding methods, auditing data, and ensuring the quality process in software development through the estimation of the number of defects remaining in the software system [18-20]. The estimated number of defects can be used as a key metric for managing software processes and evaluating the quality of the software system before its delivery.

The importance of early detection of defective software components in the software development life cycle (SDLC) implies that metrics and indicators can applied to assess and enhance the quality of the software being developed.

2.1 Software defects metrics

Software parameters and metrics are essential to measure and assess various aspects of software development to ensure its quality and effectiveness [11]. Metrics are indicators that describe specific attributes of the software Process Efficiency, Effort Assessment, Defect Reduction and Project Performance Evaluation [21, 22].

Lines of Code (LOC) are one of the traditional and widely used metrics in software engineering to measure various aspects of a software project [13]. However, it's important to note that while LOC can provide some insights, it has its limitations and should be used cautiously as a sole indicator for evaluating software quality or predicting defects. With increasing LOC the probability of defects might also increase and it can oversimplify the relationship between code length and defects. While LOC can be a useful metric in certain contexts, it's important to consider other metrics and qualitative aspects when assessing software quality and predicting defects.

The most widely used indicator in SDP software is the "cyclic complexity indicator" proposed by "McCabe" [23] and utilized to characterize the complication of software products. McCabe's metrics are calculated by considering the number of nodes, arcs, and related components following the code management scheme. Several studies in the SDP utilizes McCabe's metrics [5, 7, 11, 13] have shown an apparent correlation between LOC indicators which suggests that there is an apparent relationship between the complexity of the code (as measured by McCabe's metrics) and the identification of deficits in the software, and deficit identification for predicting the shortcomings, flaws, or issues in the software.

Halstead [24] also provides a collection of software indicators to quantify various aspects of source code measures. The metrics are based on counting the number of operands and operators in a program. It metrics are associated with program level vocabulary, program length, volume, difficulty, effort and time with the development cycle and can be used to assess code complexity and estimate development effort and time required in conjunction with other software metrics for a more comprehensive analysis [25].

Šikic et al. [11] presents a set of features designed to capture the evolution of software modules over time. These features take into account all changes made in the module's source code and are calculated by aggregating change metrics extracted from each modification. The chronological order of the changes is considered in this process, providing a more detailed and nuanced view of the module's development compared to existing metrics. This approach recognizes that the history of changes made to a software module can be valuable in understanding its development process. By aggregating change metrics over time, the proposed features aim to offer insights into the evolution of the module. These features have proven to be relevant for defect-proneness, indicating that they may be useful for identifying potential issues or vulnerabilities in the software. Even these metrics have been effective in improving model performance, particularly on data that is affected by class imbalance. It can be problematic because it may lead models to prioritize the majority class, resulting in suboptimal performance for the minority class, which in this case might represent defect instances.

Rhmann et al. [13] describes the software change metrics (SCM) for defect prediction in software and the use of machine learning and hybrid algorithms, specifically MLT (Machine Learning Techniques) and HBSA (Hybrid Binary Swarm Algorithm), has been explored for predicting faulty classes. A GFS-loogitboost hybrid algorithm is proposed for the best performance in terms of precision and recall indicates a successful for the application context. Even though, different techniques used for defect prediction are statistically similar in terms of their performance suggests a certain level of consistency among the methods but it also need to suggest the importance of generalizing results by suggesting the need for experiments on different types of large datasets.

Huda et al. [21] proposed a framework for finding significant metrics using a hybrid wrapper and filter approach appears to be innovative and promising. By integrating the training of metric selection and fault prediction into a single process, it aims to reduce complexity, making the approach more efficient. The use of a hybrid heuristic, combining intrinsic characteristics from both filters and wrappers, is an interesting strategy. It allows for a more computationally efficient metric selection process. The incorporation of ANNIGMA and SVM wrappers, along with maximum relevance filters, in two different hybrid models, adds diversity to the approach. The ability to identify a more compact set of significant metrics while achieving higher performance in fault prediction is a notable achievement. This study is limited to procedural metrics which need to extend to object-oriented (OO) metrics for the broader applicability in SDP.

Automated software defect prediction is a crucial aspect of software development, and the complexity of modern software systems poses challenges in building effective prediction models. The issue of dealing with a large number of correlated metrics and selecting a subset that improves model performance is indeed a significant challenge. By addressing the challenge of feature selection in software defect prediction models, researchers and practitioners can enhance the efficiency and accuracy of predicting defects in complex software systems.

2.2 Software defect predictions

Addressing software defects (SD) is crucial because they can lead to malfunctions, security vulnerabilities, or other issues that negatively impact user experience. The creations of SDs are due to the coding mistakes, incorrect conditions, or flawed design. The impact of these defects is substantial, affecting the **reliability and quality** of the software [1]. Reliable software is expected to perform consistently and correctly under various conditions. A testing team can analyze software test results to detect bugs, but testing the entire software module can be costly and time-consuming. Therefore, it is necessary to identify faulty modules at an early stage so that software testers can detect modules that require intensive testing [16].

The importance of Software Defect Prediction (SDP) is to identify and addressing defects in software before its release. It aims to save time and resources by anticipating and addressing defects early in the development process. The utilization of Machine Learning (ML) algorithms in SDP is highlighted as a means to predict and identify potentially defective modules within the software. The effectiveness of SDP relies on the choice and implementation of ML algorithms. These ML classification algorithms play a crucial role in predicting and identifying defective modules.

- *Naive Bayes (NB):* It is a popular classification algorithm that is commonly used in SDP. It is particularly well-suited for this task because it is simple, efficient, and performs well even with relatively small datasets [6]. It's important to note that while NB is effective for many classification tasks, it may not capture complex dependencies between features. In SDP, where the relationships between different metrics may be intricate, other ML algorithms might also be considered based on the characteristics of the dataset.
- *Support Vector Machine (SVM):* In the context of SDP the SVM can be applied to classify whether a software module or component is likely to be defective based on various features or metrics. It works well with a set of relevant features which include code complexity, code churn, developer experience, and various code metrics [17]. The selection of these features is crucial, and domain knowledge is often used to identify relevant metrics that may correlate with defect proneness and need to choose appropriate features with fine-tune for optimal performance.
- *Random Forest (RF)*: It is a popular ML algorithm in SDP which aims to identify potential defects or bugs in software early in the development process, helping developers and testers focus their efforts on high-risk areas [10, 25]. It is less prone to overfitting compared to individual decision trees, making it more robust on new and unseen data. But, the imbalanced datasets of SDs making it challenging to interpret the rationale behind SD predictions. So, it its success depends on defect metrics and the relevance of the chosen features.

Several researches works utilizing ML has proposed for SDP [3, 4, 6, 8, 14], and also demonstrated success in terms of accuracy, reliability, and performance improvements, there appears to be a lack of clear evidence for predicting the FSD in software developments.

Siswantoro et al. [4] propose a comprehensive evaluation study for SDP using various traditional ML models on NASA metrics datasets such as k-NN, DT, LR, LDA, SHL-MLP and SVM. Hyperparameter tuning was performed using random search for each model, improving their performance. Feature dimensionality reduction was carried out using PCA for reducing the number of features while retaining the most critical information. The analysis revealed that all classifiers were impacted by data imbalance and overfitting initially, but with hyperparameter tuning shows an effective in mitigating these issues. k-NN emerged as the top-performing classifier among the methods evaluated.

Khalid et al. [14] focuses on utilizing machine learning (ML) techniques, specifically feature selection and K-means clustering, for software defect prediction. The study involves examining various ML techniques and optimizing them on a freely available dataset to enhance accuracy compared to previous research. Additionally, the research employs the Particle Swarm Optimization (PSO) method and an ensemble approach to analyze results and improve accuracy performance on the CM1 dataset. The results indicate enhanced performance across all ML and optimized ML models, but there are still cases with error rates. The need for improving data correlation learning among features is highlighted as a potential avenue for further improvement. The improving data correlation learning suggests that future work could focus on refining the understanding and utilization of data features to further enhance accuracy.

Choudhary et al. [19] deployed machine learning methods for software change matrices-based software failure prediction. They compare their designed methods to existing classifier methods based on the labelling matrix for prediction defects. They use RF, J48 and k-NN for fault prediction. In this work, an SDP model is designed based on previous software change matrices by using an algorithm based on hybrid search ML methods.

Malhotra [26] uses SDP-based ML algorithms for designing an OO software metric. It did some testing on the Android app that they got from the online repository. The methodology was evaluated by comparing with MLP, LR and SVM. The analysis results show that MLP and LR show better enhancement in comparison, whereas SVM performs the lowest.

Jiang et al. [27] highlighted the advantages and disadvantages of the prediction evaluation technology and state the cost of the project should determine the best type of designed model. They used 13 datasets from the NASA MDP and six different classification algorithms to determine the optimal algorithm performance in RF, NB, LR, Bagging, J48, and IBK. This study strongly suggests using cost-effective curves to provide an accurate assessment of the types of programs, and randomization in the RF is the best.

To the best of our knowledge, the current literature has limited research and methods to determine key indicators for all automated monitoring and control, achieving high-quality products in all releases. The choice of high-quality metrics is essential to developing highly accurate SDP mechanisms. So, to address this gap in the literature, in consideration we contribute to fill by providing a more comprehensive understanding of the prevalence of SDs in the context of ML-based SDP in the next section.

3. Proposed FSDP methodology

The core objective of the FSDP classifier is to find the defect that most frequently occurs during development utilizing past historical data sets. To understand the FSD we utilized NASA datasets. For each data set, we initially perform the training process through the Correlation Learning Method (CLM). The CLM identifies the knowledge and constructs the rule patterns to classify the defects associated. The constructed rule patterns will be utilized by the FSD classifiers using rule-based association mining. In the following section, we discuss the mechanism of CLM and the process of FSD classification.

3.1 Correlation learning method (CLM)

Most existing defect learning approaches [3, 4, 8, 9] utilize the defect association through association rule mining by employing support-confidence methods. However, it was observed that many rules that are constructed with low support and confidence thresholds result in low deficit rules for defect prediction. The CLM supplements the defect association rule mining with added interestingness measures dependent on "statistical importance" and "correlation analysis".

An association rule based on the support-confidence method mostly estimates only the conditional probability of defects to a given set of defects. It is unable to measure the actual strength of the correlation and its implications between them. So, to enhance we supplement the correlation rules along with the support-confidence method to learn the best defect association rules for the classifier prediction.

The CLM implements *Lift* correlation measures to learn the strong relation of the FSD defects among the datasets. To identify a correlation among the defects, let's assume *F1* and *F2* are the two defects from an FSD set. The relation of *F1* with *F2* will be considered independent if $prob(F1 \bigcup F2) = prob(F1) \cdot prob(F2)$ among the probability of occurrence among *F1* with *F2*, it is identified as dependent and has an event of correlation.

So, to measure the correlation value among the *F1* and *F2* it computes the *Lift* using Eq. (1).

$$Lift(F1, F2) = \frac{prob(F1 \bigcup F2)}{prob(F1) \cdot prob(F2)} \tag{1}$$

If the value of *Lift* is *>1* then the correlation is considered as positive for the prediction, else if *<1* then it is considered as negatively correlated, and if

Lift =1, then both are completely independent of each other without any correlation. So, we only considered the defects whose *Lift* value is *>1* as it implies the occurrence in the defect data sets. The steps for processing of CLM are presented in the Algorithm 1.

The algorithm initially implements 4 functions to identify the of defect occurrence. These functions are:

- *findBoth_Occurance_Count(F_a, F_b, D)* - finds the count of feature F_a and F_b occurrence in dataset *D*.
- *findNo_Occurance_Count*(F_a, F_b, *D*) - finds the count of feature F_a and F_b are not found in dataset *D*.
- *find_Occurance_Count*(F_a, *D*) - finds the F_a count of occurrence in dataset *D*.
- *find_Occurance_Count*(F_b, *D*) - finds the F_b count of occurrence in dataset *D*.

Later it computes the probability of occurrence of F_a and F_b defect utilizing the occurrences values, and also the value of *Lift (L)* using Eq. (1). If the value of *L* is more than *1* then F_a *and* F_b *are correlated in the defect prediction, otherwise negatively correlated.* These identified correlated features are stored in an array of *CO-FSD[ ] as a correlation learning rules.*

Algorithm-1: CLM Method

```
D → is the collection of data sets.
FSD → is a set of frequent defects.

for i=0, each defect in FSD, i++ loop {
  $F_a$ = FSD[i];
  for j=1, each defect in FSD, j++ loop {
    $F_b$ = FSD[j];

    //--No. of occurrence where both $F_a$ and $F_b$ are
    identified in the datasets.
    $B_{val}$ = findBoth_Occurance_Count ($F_a$, $F_b$, D);

    //-- No. of occurrence where both $F_a$ and $F_b$
    not identified in the datasets.
    $N_{val}$ = findNo_Occurance_Count ($F_a$, $F_b$, D);

    //--No. of occurrence where $F_a$ identified but $F_b$
    not identified in the datasets.
    $X_{val}$ = find_Occurance_Count ($F_a$, D);

    //--No. of occurrence where $F_b$ identified but $F_a$
    not identified in the datasets.
    $Y_{val}$ = find_Occurance_Count ($F_b$, D);

    //--probability of occurrence of $F_a$ defect
    $prob(F_a) = \frac{(B_{val} + X_{val})}{(B_{val} + X_{val} + N_{val} + Y_{val})}$

    //--probability of occurrence of F2 defect
    $prob(F_b) = \frac{(B_{val} + Y_{val})}{(B_{val} + X_{val} + N_{val} + Y_{val})}$

    //-- According the Eq (1)

    $L = \frac{prob(F_a \cup F_b)}{prob(F_a) \cdot prob(F_b)}$

    If L > 1 then
      //--$F_a$ and $F_b$ are correlated then defect
      predicted
      CO-FSD[c] = {$F_a$, $F_b$};
    Else
      $F_a$ and $F_b$ are negatively correlated.
    End-If
  }
}
```

So, on completion of the CLM process, we can identify the most correlated FSD that can strongly impact the software development. Now, utilizing the learned CO-FSD rules we classify the real-time defects in a software product.

3.2 FSDP mechanism

The FSDP is perform through the classification of defects for the prediction will be performed using a rule-based classification approach by using the learned rule patterns generated by CLM. The rule of associations identifies a defect by predicting the relations among the different defect types, with an assumption that if a program has a defect *F1* and *F2* then the defect *F3* will also occur, which generally represents as, "*F1 ^ F2 → F3".* For example, in a situation where we have a rule "*F1 ^ F2 → F3"* for a defect *F1, F2,* and *F3* obtained from the datasets, the correlation of *F3* with other defects is not yet identified. So, the rule indicates that defect *F3* also needs to check whether the corresponding defect correlated or not to other defects. If the correlation occurs is positive, then the new rule will be "*F1 ^ F2 ^ F3 → F4",* and it will continue until we find all the defects correlated for the defect prediction.

The mechanism of FSDP is designed using a rule-based classification approach. It was learned in past studies [26, 28, 29] that rules can be the most prominent in providing the best knowledge to do classification. The FSDP utilizes the learned knowledge from CO-FSD learning rules to classify the defect.

To predict the possibility of defects in datasets we classify them with the learned CO-FSD rules. According to the rule-based classification theory if the set precondition rules are satisfying the rules of consequent then we can relate to a prediction class.

Let's assume dataset *D* consists of a collection of *x* data records, and CO-FSD learning has a set of rules as *R*. Now if each condition of *R* holds as true for a data record of *D*, then we say the rule is satisfied for the defect prediction. The FSDP prediction performance will be measured in terms of its coverage and accuracy.

Let's assume FSDP classifies n_{record} using the rules of CO-FSD learning, and $m_{correct}$ is the number of correctly classified by using rules from the dataset *D*. So, by using Eq.(2) and (3) we can compute the FSDP prediction coverage and accuracy.

$$\text{coverage}(R) = \frac{n_{records}}{|D|} \tag{2}$$

$$\text{accuracy}(R) = \frac{n_{corrects}}{n_{records}} \tag{3}$$

So, FSDP is being evaluated using the designed CLM process over attributes of the NASA datasets as presented next section.

4. Experiment evaluation

4.1 Datasets

We utilized the NASA Metrics Data Program (MDP) dataset [30] for the evaluation. The NASA MDP dataset is a valuable resource that provides information about software defects in various NASA projects. This includes details such as the number of defects found in each project, the size of the code base, and the effort required to address the software issues. The dataset is commonly employed by software engineers to analyze the relationship between different software metrics and software defects. A set 6 datasets from projects: MW1, PC1, PC2, PC3, PC4, and CM1 are considered which have different number of features variation, but they all have the same classes as "defective Y" and "defective N." The distribution of defective instances and defects percentage of these dataset are given in Table 2.

The NASA defect datasets are commonly used for this purpose, and they are publicly available for users to assess and validate various fault prediction techniques. By using NASA defect datasets for evaluating the FSDP method provides a standardized and accessible framework for assessing the performance of fault prediction techniques, promoting advancements in the field and contributing to the overall improvement of software reliability.

Table 2. Datasets Description

Dataset	No. of Attributes	Defective Instances	Total Instances	Defects (%)
MW1	37	27	253	10.67
PC1	37	61	705	8.65
PC2	36	16	745	2.14
PC3	37	134	1077	12.44
PC4	37	177	1287	13.75
CM1	22	49	498	9.83

4.2 Evaluation measures

In this section, we will consider various measurements for SDP using the WEKA tool utilizing the datasets of NASA. We run the training process initially to build the CO-FSD learning rules and then utilizing these rules in the classification to evaluate each dataset and measure the accuracy, precision, recall and F-score to analyse the improvement.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{4}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{5}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{6}$$

$$\text{F-Score} = \frac{(2 \times \text{ Precision} \times \text{Recall})}{(\text{Precision + Recall})} \tag{7}$$

where,

- $TP \rightarrow$ Number of defective instances correctly classified as nominal defects.
- $TN \rightarrow$ Number of clean instances correctly classified as clean.
- $FP \rightarrow$ indicates the number of clean instances that have been misclassified as faults.
- $FN \rightarrow$ indicates the number of faulty software instances that have been accidentally classified as clean.

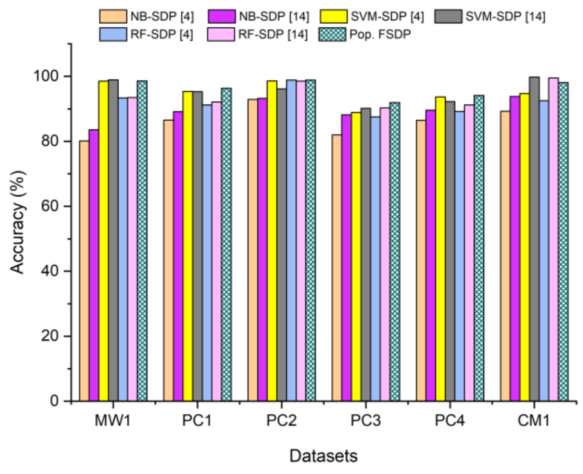

Figure. 1 Comparison of classifiers Accuracy

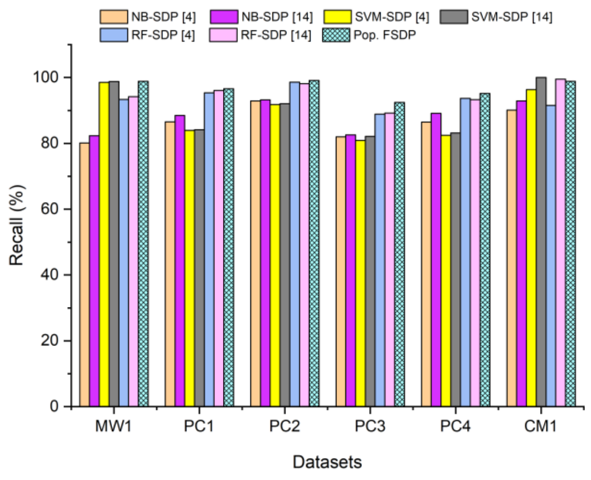

Figure. 3 Comparison of classifiers Recall

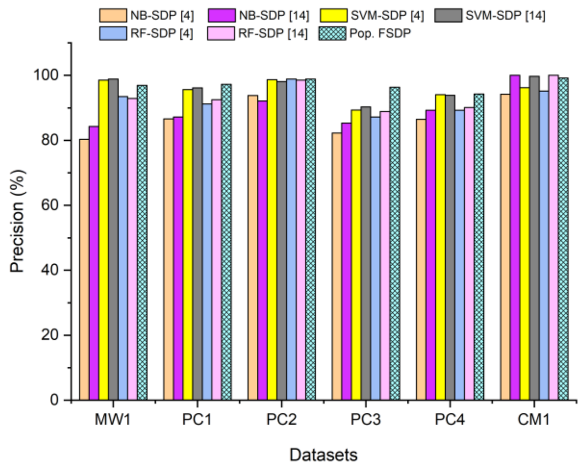

Figure. 2 Comparison of classifiers Precision

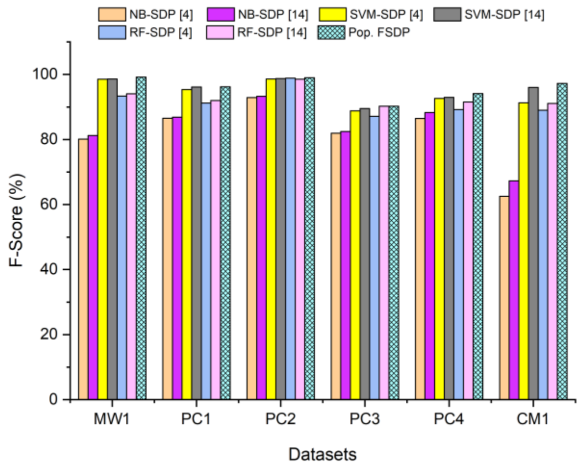

Figure. 4 Comparison of classifiers F-Score Scores

4.3 Result analysis

The evaluation of analytical enhancement is performed using four conventional classifiers: NB, RF, SVM and two state-of-the-art comparative studies on SDP analysis in [4, 14] in compared to the proposed FSDP. The classifier provides us with an opportunity to compare the performance of previous classification methods and techniques. Since these classifiers also use the ability to predict, it makes sense to investigate whether different sources were identified by each individual and whether these changes compare differences between these differences.

A. Accuracy Analysis

The performance of a proposed FSDP in the context of predicting defects in different datasets suggests that the FSDP achieved better accuracy in prediction compared to various classifiers. The calculated accuracy value using Eq. (4) of all the classifiers are presented in Table 3.

Fig. 1 illustrates the performance of different classifiers, particularly focusing on FSDP, SVM-SDP, and RF based SDP, in predicting defective instances across various datasets (PC1, PC2, PC3, MW1, and CM1). FSDP outperformed other classifiers in most cases, achieving the highest accuracy. Specifically, FSDP achieved an accuracy of 98.88% with PC2, 96.31% with PC1, and 91.9% with PC3 datasets. SVM-SDP achieved the highest accuracy of 98.91% in the MW1 dataset. The comparison between SVM-SDP and RF based SDP showed similar accuracies, while NB based SDP demonstrated the lowest accuracy. In the case of the CM1 dataset, SVM-SDP demonstrated the best result with an accuracy of 99.80%. The choice of the best classifier might depend on the specific dataset being used. In the MW1 dataset, for instance, SVM-SDP performed slightly better than FSDP.

B. Precision and Recall Analysis

The performance of precision of the proposed FSDP in comparison to other classifiers such as NB,

Table 3. Accuracy (%) Comparison

Datasets	NB-SDP [4]	NB-SDP [14]	SVM-SDP [4]	SVM-SDP [14]	RF-SDP [4]	RF-SDP [14]	Prop. FSDP
MW1	80.15	83.54	98.53	**98.91**	93.38	93.51	98.61
PC1	86.56	89.14	95.35	95.32	91.21	92.12	**96.31**
PC2	92.92	93.25	98.63	96.16	98.86	98.51	**98.88**
PC3	81.98	88.19	88.87	90.18	87.51	90.28	**91.90**
PC4	86.49	89.58	93.69	92.26	89.19	91.23	**94.11**
CM1	89.24	93.80	94.74	**99.80**	92.48	99.5	98.1

Table 4. Precision (%) Comparison

Datasets	NB-SDP [4]	NB-SDP [14]	SVM-SDP [4]	SVM-SDP [14]	RF-SDP [4]	RF-SDP [14]	Prop. FSDP
MW1	80.31	84.25	98.57	**98.85**	93.47	92.89	96.92
PC1	86.58	87.22	95.65	96.14	91.22	92.54	**97.26**
PC2	93.81	92.15	98.67	98.11	98.86	98.55	**98.86**
PC3	82.25	85.32	89.34	90.29	87.18	88.89	**96.34**
PC4	86.49	89.24	94.05	93.89	89.28	90.08	**94.26**
CM1	94.22	**100**	96.19	99.7	95.14	**100**	99.21

Table 5. Recall (%) Comparison

Datasets	NB-SDP [4]	NB-SDP [14]	SVM-SDP [4]	SVM-SDP [14]	RF-SDP [4]	RF-SDP [14]	Prop. FSDP
MW1	80.15	82.36	98.53	**98.89**	93.38	94.21	98.45
PC1	86.57	88.51	83.98	84.12	95.36	96.05	**96.58**
PC2	92.91	93.21	91.78	92.04	98.63	98.14	**99.12**
PC3	81.98	82.58	80.92	82.15	88.87	89.22	**92.47**
PC4	86.49	89.14	82.43	83.19	93.69	93.28	**95.14**
CM1	90.14	92.9	96.32	**100**	91.54	99.5	98.85

Table 6. F-Score (%) Comparison

Datasets	NB-SDP [4]	NB-SDP [14]	SVM-SDP [4]	SVM-SDP [14]	RF-SDP [4]	RF-SDP [14]	Prop. FSDP
MW1	80.12	81.22	98.53	**99.61**	93.38	94.05	99.21
PC1	86.56	86.89	95.34	96.14	91.21	91.98	**96.22**
PC2	92.89	93.29	98.63	98.35	98.86	98.51	**98.95**
PC3	81.94	82.45	88.84	89.54	87.12	90.26	**90.21**
PC4	86.49	88.29	92.61	92.95	89.18	91.51	**94.16**
CM1	62.57	67.3	91.28	96	88.98	91.1	**97.22**

RF and SVM on different datasets calculated using Eq. (5) is given in Table 4.

Fig. 2 illustrates the performance of different methods for prediction on precision values across various datasets. FSDP Method consistently achieves better precision in prediction across all datasets, except for MW1 where SVM-SDP performs better. It shows a highest precision of 98.86% for dataset PC2 and a lower but still strong precision of 94.26% for PC4. It outperforms NB, RF, and SVM-based SDP methods [4, 14] in terms of positive predictions and precision. The proposed FSDP also demonstrates a nearby precision of 99.21% for CM1, indicating competitive performance where NB-SDP and RF-SDP from reference [14] show 100% results with CM1. The lowest precision for FSDP is observed in PC4, with a still respectable value of 94.26%. The FSDP method seems to be a robust and competitive approach for prediction, consistently performing well across various datasets and outperforming other classifiers in terms of precision.

Received: February 29, 2024. Revised: March 25, 2024.

The recall results comparison analysis of various classifiers with the proposed FSDP method calculated using Eq. (6) is given in Table 5 across all datasets. It is a measure of sensitivity or true positive rate which indicates better performance in capturing all positive instances.

Fig. 3 demonstrates the performance of different classifiers, particularly focusing on the FSDP method. It shows improvements in accuracy and precision, leading to better recall scores across various datasets.

The recall scores vary across datasets, with the highest of 99.12% observed for PC2 and the lowest of 92.47% for PC3. In the case of MW1 dataset, SVM-SDP [14] outperforms FSDP with a higher recall of 98.89%. The NB based SDP is mentioned as one of the classifiers with the lowest performance across different datasets. Specifically for CM1 datasets, SVM-SDP [14] achieves a perfect 100% recall, while FSDP shows a slightly lower result, 3% below. The FSDP method generally performs well in terms of recall scores, but there are instances where other classifiers, such as SVM-SDP [14], may outperform it on specific datasets.

C. F1-Score Analysis

The performance comparison of F1-score on different datasets for the proposed FSDP and other classifiers is given in Table 6. It evaluates the class imbalance problem during classification using Eq. (7) based on the value of precision and recall.

Fig. 4 show the performance of different models, specifically the FSDP model, SVM-SDP, and NB based SDP, on various datasets such as PC1, PC2, PC3, CM1, and MW1, specifically focusing on the F1-Score metric. The FSDP method demonstrates superior F1-Score scores across all datasets compared to other methods. The improvement in F1-Score is attributed to better accuracy, precision, and recall. Specifically, FSDP achieved the best F1-Score with PC2 and CM1 datasets, showing 98.95% and 97.22%, respectively.

The FSDP method exhibited the lowest F1-Score with the PC3 dataset, reaching 90.21%. In contrast, SVM-SDP achieved the highest F1-Score with MW1, reaching 99.61%. The NB based SDP showed the lowest performance among the methods. The results suggest that FSDP generally outperforms other methods across the given datasets, while the NB classifier exhibits less consistent performance across different datasets. It's important to consider the specific characteristics of each dataset when choosing a classifier, as different algorithms may perform better or worse depending on the nature of the data.

5. Conclusion

This paper focuses on developing a FSDP system using a Correlation Learning Method (CLM). The main goal is to predict the most frequently occurring defects in software development. The FSDP is based on the CLM builds the attribute to generate a set of rules that aids in the accurate prediction of the probability of occurrence of a defect. The proposed FSDP is tested using five datasets from NASA repositories. The performance of the FSDP is evaluated through metrics such as accuracy, precision, recall, and F1-Score. The FSDP demonstrates high performance with 98.88% accuracy, 98.86% precision, 99.12% recall, and a F1-Score of 98.95%. These metrics indicate the effectiveness of the proposed approach in defect prediction. The paper suggests that in future work, the FSDP will explore attribute correlation using deep learning techniques for all defect-impacting input attributes, aiming to enhance SD prediction.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, methodology, software, validation, Sareddy Shiva Reddy; formal analysis, investigation, resources, data curation, Sareddy Shiva Reddy; writing-original draft preparation, writing-review and editing, Sareddy Shiva Reddy; visualization, supervision, Dr. Suresh Pabboju.

References

[1]. P. Afric, D. Vukadin, M. Silic, G. Delac, "Empirical Study: How Issue Classification Influences Software Defect Prediction", *IEEE Access*, Vol. 11, pp. 11732-11748, 2023.

[2]. A. Abdelaziz, N. R. Darwish, H. A. Hefny, "Multiple Linear Regression for Determining Critical Failure Factors of Agile Software Projects", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 3, 2019, doi: 10.22266/ijies2019.0630.24.

[3]. G. Esteves, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Understanding machine learning software defect predictions", *Automated Software Engineering*, Vol. 27, No. 3-4, pp. 369-392, 2020.

[4]. M. Z. F. N. Siswantoro, U. L. Yuhana, "Software Defect Prediction Based on Optimized Machine Learning Models: A Comparative Study", *TEKNIKA*, Vol. 12, No. 2, pp. 166-172, 2023.

[5]. D. Bowes, T. Hall, J. Petri, "Software defect prediction: do different classifiers find the same defects?", *Software Quality Journal*, Vol 26, pp. 525-552, 2018.

[6]. R. Marco, S. S. S. Ahmad, S. Ahma, "Empirical Analysis of Software Effort Preprocessing Techniques Based on Machine Learning", *International Journal of Intelligent Engineering and Systems*, Vol. 14, No. 6, 2021, doi: 10.22266/ijies2021.1231.49.

[7]. S. Amasaki, "Cross-version defect prediction: use historical data, cross-project data, or both?", *Empirical Software Engineering*, Vol. 25, No. 2, pp. 1573-1595, 2020.

[8]. S. Ali, M. Adeel, S. Johar, M. Zeeshan, S. Baseer, A. Irshad, "Classification and Prediction of Software Incidents Using Machine Learning Techniques", *Security and Communication Networks*, Article ID 9609823, Vol. 2021.

[9]. Y. Shao, B. Liu, S. Wang, G. Li, "Software defect prediction based on correlation weighted class association rule mining", *Knowledge-Based Systems*, Vol. 196, 105742, 2020.

[10]. A. Balaram, S. Vasundra, "Software Fault Detection using Multi-Distinguished-Features Sampling with Ensemble Random Forest Classifier", *International Journal of Intelligent Engineering and Systems*, Vol. 15, No. 5, 2022, doi: 10.22266/ijies2022.1031.43.

[11]. L. Šikic, P. Afric, A. S. Kurdija, M. ŠIlic, "Improving Software Defect Prediction by Aggregated Change Metrics", *IEEE Access*, Vol. 9, 2021.

[12]. J. Sohn, S. Yoo, "Empirical Evaluation of Fault Localisation Using Code and Change Metrics", *IEEE Transactions on Software Engineering*, Vol. 47(8), 2021.

[13]. W. Rhmann, B. Pandey, G. Ansari, D. K. Pandey, "Software fault prediction based on change metrics using hybrid algorithms: An empirical study", *Journal of King Saud University - Computer and Information Sciences*, 2019.

[14]. A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software Defect Prediction Analysis Using Machine Learning Techniques", *Sustainability*, 15, 5517, 2023.

[15]. C. Liu, S. Sanober, A. S. Zamani, L. R. Parvathy, R. Neware, and A. W. Rahmani, "Defect Prediction Technology in Software Engineering Based on Convolutional Neural Network", *Security and Communication Networks*, Article ID 5058461, Vol. 2022.

[16]. L. Yang, Z. Li, Dongsheng Wang, Hong Miao, Zhaobin Wang, "Software Defects Prediction Based on Hybrid Particle Swarm Optimization and Sparrow Search Algorithm", *IEEE Access*, Vol. 9, 2021.

[17]. L. Gong, S. Jiang, L. Bo, L. Jiang, and J. Qian, "A novel classimbalance learning approach for both within-project and cross-project defect prediction", *IEEE Transactions on Reliability*, Vol. 69, No. 1, pp. 40-54, 2020.

[18]. M. O. Elish, K. Elish, "An Empirical Comparison of Resampling Ensemble Methods of Deep Learning Neural Networks for Cross-Project Software Defect Prediction", *International Journal of Intelligent Engineering and Systems*, Vol. 14, No. 3, 2021, doi: 10.22266/ijies2021.0630.18.

[19]. G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, C. Catal, "Empirical analysis of change metrics for software fault prediction", *Computers & Electrical Engineering*, Vol. 67, pp. 15-24, 2018.

[20]. A. Alsaeedi, M. Z. Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study", *Journal of Software Engineering and Applications*, Vol. 12, pp. 85-100, 2019.

[21]. S. Huda, S. Alyahya, M. M. Ali, S. Ahmad, J. Abawajy, H. Al-dossari, J. Yearwood, "A Framework for Software Defect Prediction and Metric Selection", *IEEE Access*, Vol. 6, 2018.

[22]. X. Sun, W. Zhou, B. Li, Z. Ni, J. Lu, "Bug localization for version issues with defect patterns", *IEEE Access*, Vol. 7, 18811-18820, doi: 10.1109/ACCESS.2019.2894976, 2019.

[23]. T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, Vol. 2, pp. 308-320, 1976.

[24]. M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*, 1977.

[25]. T. Zhou, X. Sun, X. Xia, B. Li, X. Chen, "Improving defect prediction with deep forest", *Information and Software Technology*, Vol. 114, pp. 204-216, 2019.

[26]. R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with Android software", *Applied Soft Computing*, Vol. 49, pp. 1034-1050, 2016.

[27]. Y. Jiang, B. Cukic, and T. Menzies, "Fault prediction using early lifecycle data", In: *Proc. of 18th IEEE Int. Symp. Softw. Rel. (ISSRE)*, pp. 237-246, 2007.

[28]. K. Bashir, T. Li, C. W. Yohannese, Y. Mahama, "Enhancing software defect prediction using supervised-learning based framework", In: *Proc. of 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2017.

[29]. F. Hassan, S. Farhan, M. A. Fahiem, H. Tauseef, "A Review on Machine Learning Techniques for Software Defect Prediction", *Technical Journal*, Vol. 23, pp. 63-71, 2018.

[30]. M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets", *IEEE Transaction Software Engineering*, Vol. 39, No. 9, pp. 1208-1215, 2013.