Enhancing Software Defect Prediction with Ensemble Models based on Defect Relations Rule Learning

Sareddy Shiva Reddy
Assis. Professor, Department of CSE
JNTUH University College of Engineering, Science & Technology Hyderabad, Telangana

Suresh Pabbojur, PhD
Professor, Department of IT
CBIT, Osmania University, Hyderabad, Telangana

ABSTRACT

Software defect prediction (SDP) is a crucial aspect of software quality assurance, aiming to identify potential defects early in the development process to enhance reliability and reduce maintenance costs. This paper presents a defect relations rule learning (DRRL) to enhance the defect classification models. It discovers the rules based on the defect relation association and applies a rule-ranking mechanism to perform a two-stage prediction model for accurate defect prediction software modules. In the first stage Random Forest, Support Vector Machine, and Naïve Bayes — are employed to analyze their prediction accuracy. In the second stage, an Ensemble Voting Model (EVM) with classifiers prediction outcome for enhancing the accuracy and reliability of defect detection is proposed. The EVM was implemented and evaluated further to validate previous models for effectiveness. The EVM with the proposed DRRL exhibited superior performance of 99.2% accuracy for the CM11 dataset, 88.2% accuracy for the JM1 dataset, and 100% accuracy for the PC1 and PC4 datasets. These findings underscore the model's potential to significantly improve software defect prediction.

General Terms

Software Engineering, Defect Prediction, Machine Learning, Classification

Keywords

Software Defect Prediction, Defect Relations Rule, Machine Learning, Ensemble Model

1. INTRODUCTION

The success of a software system is determined not only by its cost and adherence to schedule but also significantly by its quality. Among the various characteristics that define software quality, the presence of residual defects has emerged as the industry benchmark [1]. Predicting software defects—defined as deviations from specifications or expectations that could cause operational failures—has been a pivotal research area in software engineering for over three decades [2]. These defects serve as indicators of reliability; however, assessing reliability accurately before the full deployment of the software remains a challenging task.

Current efforts in defect prediction emphasize estimating the remaining number of defects in software systems. This is achieved using a variety of data sources, including code metrics, inspection data, and process-quality data, and applying statistical approaches to analyze them [3], [4]. Notably, capture-recapture (CR) models and detection profile methods (DPM) are widely utilized in this domain [x]. These methodologies leverage different types of data and statistical techniques to provide insights into software quality and defect prevalence, thereby guiding maintenance and improvement efforts.

The predicted number of remaining defects in a software system serves as a crucial metric for developers. These metrics aid in controlling the software process by helping decide whether further inspections are necessary or if the software artifacts can proceed to the next development phase. It also helps in assessing the likely quality of the delivered software. On the other hand, the author [5] suggests that defects found during production reflect underlying process deficiencies. They illustrate this through a case study employing a defect-based method for software process improvement. Specifically, they utilize an attribute-focusing method to uncover associations among defect attributes, such as defect type, source, phase introduced, phase found, component, and impact. By identifying events leading to these associations, they pinpoint process issues and implement corrective actions.

Several approaches [6] support enabling a project team to enhance its development process by focusing specifically on predicting defect types and their associated correction efforts. By accurately identifying and categorizing defects, the team can allocate resources more efficiently and implement targeted interventions to address issues. This proactive strategy not only improves overall software quality but also optimizes the correction effort, leading to a more streamlined and effective development process. This work aims to find what the related defect(s) may occur for the given defect(s), by predicting defect types and their relation.

Using defect-type data to predict software defect relations allows us to identify relationships among different defect types. These relationships can be expressed as rules, such as: If defects A and B occur, then defect C is also likely to occur, formally written as $A \wedge B \rightarrow C$. These defect relations serve three primary purposes:

1. *Enhancing Defect Detection and Correction*: By identifying related defects to those already detected, it can make more effective corrections to the software. For instance, if it has a historical rule $A \wedge B \rightarrow C$, and defects A and B have been detected together but defect C has not yet been discovered, the rule suggests that defect C is likely present. This prompts us to inspect the corresponding software artifact to confirm its existence. If confirmed, then it will continue the search using additional rules, such as $A \wedge B \wedge C \rightarrow D$, further guiding the testing efforts and optimizing the use of limited testing resources.

2. *Evaluating Reviewers' Results*: During inspections, these rules can help assess the thoroughness of reviewers. For example, if the rule $A \wedge B \rightarrow C$ holds, but a reviewer has only found defects A and B, it is possible they missed

defect *C*. In such cases, a recommendation can be made to re-inspect the work for completeness, ensuring a more comprehensive review process.

3. *Improving Software Processes*: By analyzing why certain defects frequently occur together, managers can gain insights into potential process problems. If a recurring relation between defects is identified, managers can investigate and implement corrective actions to address the underlying issues, ultimately enhancing the overall software development process.

These applications of defect relations not only improve defect detection and correction but also enhance the inspection process and support continuous improvement in software development practices.

This paper presents a defect relations rule learning (DRRL) method for defect classification by discovering the rules based on the defect relation association. It is to discover software defect associations from historical software engineering data sets and help determine whether or not a defect is accompanied by other defects. The methods for predicting defect relations are grounded in the principles of association rule (AR) learning [1]. It seeks to uncover patterns of co-occurrence among attributes within a database. However, it is important to note that such associations do not imply a relationship. An AR is typically expressed in the form $A \rightarrow C$, where *A* (Antecedent) and *C* (Consequent) represent sets of items. The interpretation of these rules is straightforward for example, if a given database *D* of transactions, where each transaction *T* in *D* is a set of items, then the rule $A \rightarrow C$ indicates that whenever a transaction *T* includes *A*, it is likely to also include *C*, with a certain level of confidence. This confidence is defined as the proportion of transactions that contain both *A* and *C* relative to the total number of transactions that contain *A*.

The DRRL will uncover the patterns of co-occurring attributes in databases, and it has been observed that classification based on AR will achieve higher accuracy than other classification techniques. This is largely due to the use of heuristic and greedy search methods in building classifiers, which result in a representative subset of rules. Unlike decision-tree induction methods, which evaluate one variable at a time, AR identifies high-confidence relations among multiple variables, potentially overcoming some limitations of other techniques. Consequently, it will utilize the most effective rule(s) from the constructed set for accurate classification. A data source from the NASA repository is utilized to obtain four datasets for the learning, prediction, and experimental evaluation.

The remainder of this paper is structured as follows: Section 2 outlines the methodology employed in this study. Section 3 details the methods used for defect relation learning and prediction for defects. Section 4 presents the results of the experiments. Lastly, Section 5 provides a summary of the work and key findings.

2. RELATED WORKS

In software development, identifying and fixing software bugs—errors and flaws—is crucial, especially as these issues are addressed in subsequent software updates. Recent research has delved into this topic, building on existing literature. Thota et al. [6] conducted a significant study on software defect prediction (SDP), underscoring its importance in maintaining high-quality software amidst rapid technological advancements. The researchers proposed an effective approach that leverages soft computing-based machine learning techniques to optimize feature prediction. This strategy addresses challenges faced by industries with high software development costs, particularly in safety-critical systems, offering valuable insights to enhance testing strategies and improve overall software reliability.

Classification techniques play a vital role in this process by categorizing data into specific classes or labels, thereby identifying potential software defects. Commonly employed methods include decision trees (DT), logistic regression (LR), and support vector machines (SVM), all of which proactively assess and address software quality concerns. Historical studies [7-9] have utilized these techniques to enhance defect prediction model accuracy. Despite their effectiveness, these approaches face limitations, including challenges with the classification methods. These challenges highlight the need for ongoing research and innovation in defect prediction methodologies.

In addition to classification, Ensemble Modeling (EM) has become a prominent technique in machine learning (ML), known for its ability to enhance predictive performance by combining outputs from multiple models [10]. Techniques such as bagging, boosting, stacking, and RF have significantly advanced this field by addressing challenges like overfitting, underfitting, and biases that typically affect individual classifiers[11], [12]. By aggregating predictions from various base models, ensemble methods improve the accuracy and robustness of predictions, particularly in the context of defect prediction models. This aggregation helps to minimize the biases inherent in single classifiers. Despite these benefits, researchers have identified that ensemble techniques themselves can be prone to biases, which may impact their overall effectiveness [13], [14].

2.1 Classification Approaches in SDP

Machine learning (ML)-based classification algorithms have gained immense popularity and interest in recent literature on SDP. These approaches have proven to be highly effective in identifying defect-prone modules, offering significant benefits in software quality assurance and maintenance. Matloob et al. [15] conducted a systematic review of the literature focusing on the use of classification learning in SDP, and Daoud et al. [16] performed a comparative analysis of four classifiers using NASA's datasets to improve the accuracy and robustness of defect prediction models.

In [17], researchers developed an advanced cloud-based SDP system that utilized data fusion and decision-level machine learning fusion techniques. This innovative system combined the predictive outputs of three classifiers—naïve Bayes (NB), artificial neural network (ANN), and DT—through a fuzzy logic-based fusion method. When tested with NASA datasets, the system demonstrated superior performance compared to other techniques, aiming to enhance software quality while reducing costs. A comprehensive comparative analysis of various classifiers was undertaken to explore their effectiveness in software defect prediction, as detailed in [18]. The authors evaluated ten machine learning algorithms, including DT, NB, K-NN, SVM, RF, Extra Trees, Adaboost, Gradient Boosting, Bagging, and Multi-Layer Perceptron. This analysis was conducted using benchmark NASA datasets from the PROMISE repository, specifically CM1, KC1, KC2, JM1, and PC1. The experimental results revealed that these algorithms attained higher average accuracy rates on the PC1 dataset. Notably, the RF models, when combined with Principal Component Analysis (PCA), demonstrated significantly improved average performance across all datasets.

In [19], researchers introduced a novel algorithm combining SVM and Extreme Learning Machines (ELM) for predicting software reliability. They examined various factors affecting prediction accuracy, such as the utilization of historical failure data and selecting the appropriate type of failure information. To enhance feature selection, they proposed a model that leverages both ELM and SVM, addressing dataset imbalance through resampling methods and applying their approach to NASA Metrics datasets. In [20], the researchers tackled the issue of handling the extensive volume of software defect reports encountered during software development and maintenance. They developed an SDP model integrated feature selection through the Least Absolute Shrinkage and Selection Operator (LASSO) with the Support Vector Machine (SVM) algorithm to boost prediction accuracy. In [23], researchers introduced a cloud-based framework designed for real-time SDP, in which they evaluated four back-propagation training algorithms. Among these, Bayesian Regularization (BR) proved to be the most effective. The framework also incorporated a fuzzy layer to dynamically select the optimal training function based on performance metrics. Using publicly available NASA datasets for evaluation, the framework's performance was assessed through various measures.

Akimova et al. [25] conducted a comprehensive review of the application of deep learning techniques for SDP, a critical area aimed at improving software quality and reliability by identifying defective source code. The study focused on the latest advancements in ML, especially deep learning, and examined methods for the automatic extraction of semantic and structural features from code. This survey provided an in-depth analysis of recent research in the field, identified existing challenges, and discussed new trends in software defect prediction using deep learning.

Goyal [26] focused on SDP and the effective use of SVM to address issues related to imbalanced datasets, such as the uneven distribution of faulty and non-faulty modules. The study introduced a novel filtering technique called FILTER to improve defect prediction accuracy. The research involved developing various SVM-based classifiers, including linear, polynomial, and radial basis function models, and applying FILTER to five different datasets. The results indicated significant enhancements in model performance.

2.2 Ensemble Learning in SDP

Ensemble learning (EL) is an ML technique that integrates predictions from multiple weak classifiers to create a robust classifier that outperforms individual models [27]. EL encompasses various homogeneous methods, such as bagging, boosting, etc., as well as heterogeneous approaches like voting and stacking [28]. Voting, in particular, is a heterogeneous ensemble method that aggregates predictions from different base classifiers to improve overall performance.

Abbas et al. [29] introduced an intelligent system for predicting defective software modules, leveraging feature selection and ensemble machine learning techniques. Their approach incorporates a novel metric selection technique to identify the most relevant features and employs a three-step nested methodology for precise prediction. Initially, DT, SVM, and NB are utilized to identify faulty modules. In the subsequent step, the predictive accuracy of these methods is enhanced through ensemble techniques such as bagging, voting, and stacking. Finally, fuzzy logic is applied to further refine the predictive accuracy of the ensemble methods. The experiments, conducted on a combined software defect dataset from five NASA datasets, demonstrated that the proposed system outperforms other advanced techniques, achieving a notable accuracy rate.

Soe et al. [30], proposed a SDP model utilizing multi-layer feed-forward neural networks combined with stacking as an ensemble technique. To enhance the model's performance, six different search methods were employed for feature selection, with the multilayer perceptron used as the subset evaluator. The model achieved better accuracies on NASA's datasets using the best-first search, greedy stepwise search, and GS methods, respectively. Unlike previous studies that often rely on individual classifiers and face issues such as overfitting, lack of robustness, and algorithm-specific biases, this model addresses these limitations through its innovative ensemble approach.

Standalone classifiers often fail to capture the diverse patterns in complex software datasets, resulting in suboptimal predictive performance. While some studies have explored ensemble techniques, they mostly focus on homogeneous classifiers within their ensembles [31]. The proposed framework, however, introduces a paradigm shift by integrating the predictive accuracy of heterogeneous classifiers—RF, SVM, and NB —through a voting ensemble classification technique. This innovative approach addresses the limitations of both individual classifiers and conventional homogeneous ensembles. By leveraging the strengths of diverse classifiers, the proposed model enhances interpretability, generalizability, and predictive accuracy, offering a more comprehensive and effective solution for software defect prediction.

3. PROPOSED METHODOLOGY

This section introduces the basic concepts of defect relation rule learning (DRRL). Then, it presents the rule-ranking scheme used for defect relations and predictions. After that, it respectively gives the methods of defect relation prediction based on the DRRL method.

3.1 Defect Relation Rule Learning (DRRL)

DRRL was searching for interesting relationships, such as frequent patterns, associations, correlations, or potential causal structures, among sets of objects in databases or other information repositories. The approach was data-driven rather than hypothesis-driven. The interestingness of an association rule was measured by both support and confidence, which respectively reflect the usefulness and certainty of the rule. It had to be stressed that even rules that were being discovered with high levels of support (or relevance) and high confidence were not necessarily implying causality. However, such rules stimulated further research through the postulation of models that could be empirically evaluated.

The mechanisms of DRRL for generating rules are given in a series of procedural steps below:

1. Create the set of defect Items, $X = \{x_1, x_2, ..., x_m\}$ with a set of attribute values.
2. Create an item attribute set, *F* for each defect as a set where $F \subseteq X$.
3. Create a database, *Z* having multisets of *X*, where each transaction records $T \in Z$.
4. Process for Defect Relation Rule Learning (DRRL):
    - A relation rule, *R* is for a defect class *D* is expression as, $\mathrm{R} : F \Rightarrow D$, where

$F \subset X$, $D \subset X$ and $F \cap D = \phi$.

- F is referred to as the antecedent of the rule, and D is the consequent of the rule R.

5. Calculate the Support Value of the Rule, R:
   - The support of the rule $F \Rightarrow D$ in Z is defined as, $Sup(F \Rightarrow D) = Sup(F \cup D)$.
   - This means that $Sup(F \Rightarrow D)$ is the percentage of transactions in Z that contain $F \cup D$.
   - The support of a defect item set R is defined as
   $$Sup(F) = \frac{|(\mathrm{T} \in Z \mid F \subseteq \mathrm{T})|}{|Z|} .$$
   - This is the fraction of transactions T which supports the defect item attribute set F for the database Z.
   - It gives the minimum count of transactions required for a defect item set to satisfy the minimum support.
   - A transaction $T \in Z$ supports a defect item set $F \subseteq X$ if $F \subseteq T$ holds.

6. Calculate the Confidence Value of the Rule, R:
   - The rule $\mathrm{R} : F \Rightarrow D$ holds in Z with confidence
   $$Conf\,(F \Rightarrow D) = \frac{Sup(F \cup D)}{Sup(F)} .$$
   - This means that $Conf\,(F \Rightarrow D)$ the percentage of defect transactions in Z that contain attributes F and also contain defect class D. It provides the measure of rules relation strength and usefulness of the rule.

The process DRRL constructed rules might have a similar length of attributes, support, and confidence values. So, to normalize the rule selection a rule ranking scheme is employed to organize the rules as their priority.

3.2 Rule-Ranking Mechanism

The length-first (LF) methodology was employed to rank discovered rules before making predictions. This approach prioritized longer rules for defect relation prediction, enabling the identification of as many defects as possible that aligned with known defects. The utilization of the LF in the rule-ranking scheme is to prioritize longer rules based on their condition lengths. In case two or more rules are of the same length, they will be re-ranked based on their confidence values. If these rules also have identical confidence values, then they will be re-ranked according to their support values. The process of the rule-ranking scheme is described in Algorithm 1.

Algorithm-1: Rule-Ranking Mechanism

Input: Set of defect relation rules created, R.
Output: Ranking of each rule based on LF mechanism.

```
Loop for (each rule $r_a \rightarrow R$ ) do {
  Loop for (each rule $r_b \rightarrow R$ ) do {
    If $length(r_a) < length(r_b)$ {
        $r_a \leftrightarrow r_b$ ; //-- Interchange of priority.
    }
    Else if $length(r_a) \equiv length(r_b)$ {
      If $Conf\_Value(r_a) < Conf\_Value(r_b)$ {
        $r_a \leftrightarrow r_b$ ; //-- Interchange of priority.
      }
      Else if $Conf(r_a) \equiv Conf(r_b)$ {
        If $Sup\_Value(r_a) < Sup\_Value(r_b)$ {
            $r_a \leftrightarrow r_b$ ; //-- Interchange of priority.
        }
      }
    }
    $r_a \prec r_b$ ; // has higher priority
  }
}
```

3.3 Defect Relation Prediction

The prediction of the defect relationships, employed the DRRL rules to identify defect relations within the defect dataset initially. Although constructing these rules is relatively straightforward, implementing the DRRL rules for predicting defect relationships can be challenging, especially when dealing with datasets that often contain single defects. To address these challenges and ensure accurate predictions a "*NULL*" value is added for transactions with only one defect. This helps in differentiating between single-defect transactions and those with multiple defects. An illustration of this example is given in Figure 1.

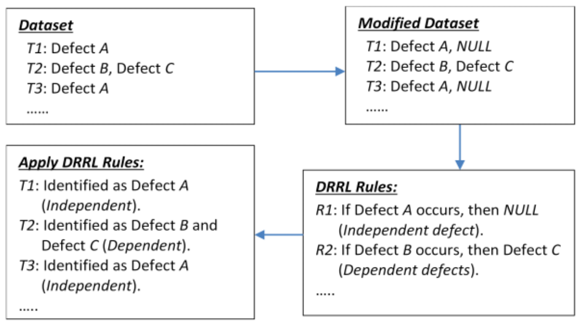

Dataset
T1: Defect A
T2: Defect B, Defect C
T3: Defect A
......
Modified Dataset
T1: Defect A, NULL
T2: Defect B, Defect C
T3: Defect A, NULL
......
DRRL Rules:
R1: If Defect A occurs, then NULL (Independent defect).
R2: If Defect B occurs, then Defect C (Dependent defects).
.....
Apply DRRL Rules:
T1: Identified as Defect A (Independent).
T2: Identified as Defect B and Defect C (Dependent).
T3: Identified as Defect A (Independent).
.....

Fig. 1: Illustration of Data Relation Prediction

To predict whether a *k-defect* will occur with others, it needs to follow a systematic approach that leverages the ranked rules and iteratively generates potential defect combinations. For a given *k-defect*, scan through the ranked rules to find rules whose antecedent (*i.e.*, the if-part of the rule) contains the *k-defect*. Identify the rule where the antecedent includes the *k-defect* and then merge the consequent (*i.e.*, the then-part of the rule) with the *k-defect* to form a *(k+1)-defect*. Now, for the newly formed *(k+1)-defect*, repeat the process to scan the ranked rules to find the next rule whose antecedent contains the *(k+1)-defect* and merge its consequent with the *(k+1)-defect* to form a *(k+2)-defect*. This will continue this process until no more applicable rules are available. The final defect set $\{(k+n)\text{-}defect\} \Theta \{k\text{-}defect\}$ represents the defect(s) that are predicted to occur with the original *k-defect.*

3.4 Classification using DRRL

To classify defective and non-defective modules, using rules to train models is an effective approach. In this research, four heterogeneous supervised machine learning classifiers have been implemented: Random Forest (RF), Support Vector Machine (SVM), and Naïve Bayes (NB).

- **Random Forest (RF)**: This is an ensemble learning method that constructs multiple decision trees during training and outputs the class which is the mode of the classes of the individual trees. RF is known for its high accuracy, robustness to overfitting, and ability to handle large datasets with higher dimensionality.

- **Support Vector Machine (SVM)**: SVM is a powerful classifier that works by finding the hyperplane that best separates the data into different classes. It is effective in high-dimensional spaces and is versatile in handling both linear and non-linear classification tasks using different kernel functions.
- **Naïve Bayes (NB)**: This is a probabilistic classifier based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. It is particularly useful for large datasets and is known for its simplicity, efficiency, and often surprisingly good performance.

These classifiers are trained using rules that differentiate between defective and non-defective modules. After training, these classifiers are evaluated to identify potentially faulty modules in new, unseen data. The selection of RF, SVM, and NB as base classifiers leverages their diverse and complementary strengths. RF excels in capturing complex relationships within data, making it particularly useful for software defect prediction [30]. SVM, with its capability to handle non-linear data through kernel functions, offers robust classification capabilities [32]. NB, based on probabilistic principles, provides simplicity and efficiency in processing large datasets, assuming conditional independence [33]. This strategic combination aims to harness the unique advantages of each classifier, enhancing the overall robustness and versatility of the classification system.

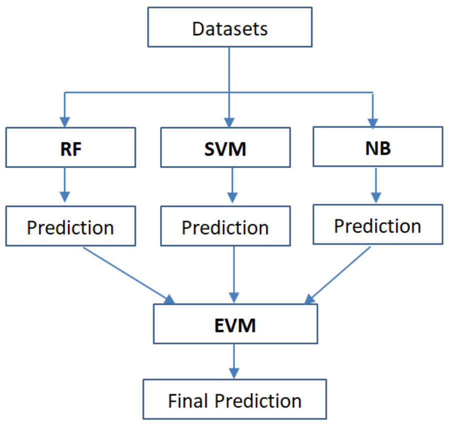


Fig. 2: Ensemble Voting Model (EVM) workflow

The ensemble model is a powerful technique in ML that combines the predictions of multiple individual models, to enhance the accuracy and robustness of the predictions. This paper employs an ensemble voting model (EVM) using RF, SVM, and NB to enhance the accuracy of software defect prediction. The EVM leverages the unique strengths of each base model, promoting a robust and reliable prediction system [34]. The predictive accuracy of three heterogeneous base classifiers—RF, SVM, and NB is utilized as input to the voting ensemble model through DRRL rules as shown in Figure 2. This integration harnesses the distinct capabilities of each classifier, leading to an overall improvement in accuracy. The experiment's evaluation demonstrates that the EVM the precision and reliability in the software defect prediction.

4. EXPERIEMENT EVALUATION

To perform the experiment evaluation, four publicly accessible NASA datasets (CM1, JM1, PC1, and PC4) were sourced from the MDP repository [36]. The obtained undergoes cleaning, normalization, and splitting initially to create two sub-sets, namely training, and testing in 80:20 ratios using the class-based splitting rule [35]. Three potential classifiers — RF, SVM, and NB iteratively evaluated based on the DRRL rules to obtain the highest possible accuracy for the datasets. Based on the predictive accuracy from individual classifiers is integrated using the voting ensemble technique, which further boosts the performance of the proposed model.

4.1 Performance Measures

To evaluate the effectiveness of the defect prediction method, precision, recall, accuracy, and F-measure (F1) are commonly used. These metrics can be derived from the confusion matrix, which summarizes the results of the predictions compared to the actual outcomes. A confusion matrix is a table used to evaluate the performance of a classification model. In the context of predictive modeling and classification, four key metrics are used to evaluate the performance of a model. True Positives (*TP*) represent the number of actual defects that the model correctly identifies as defects; False Positives (*FP*) refer to instances where the model incorrectly labels non-defects as defects, True Negatives (*TN*) denote the number of non-defects that the model accurately predicts as non-defects and False Negatives (*FN*) is the actual defects that the model fails to identify, incorrectly predicting them as non-defects. A comparison with three techniques proposed by Azam et al. [21], Iqbal et al. [22], and Alsaeedi et al. [24] is performed to assess the accuracy and reliability of a predictive model for software defect prediction.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

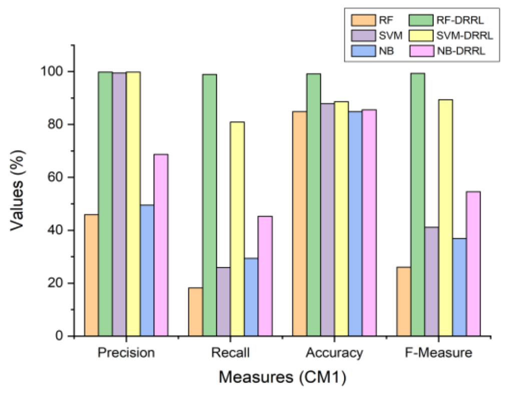
$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

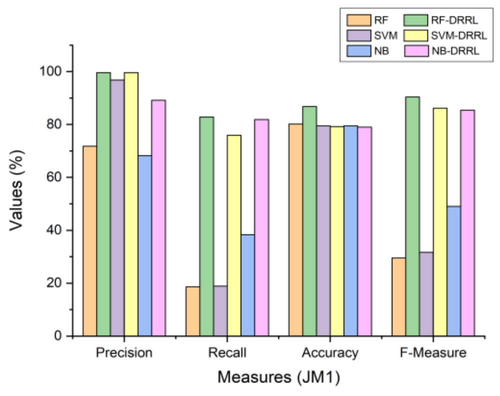
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

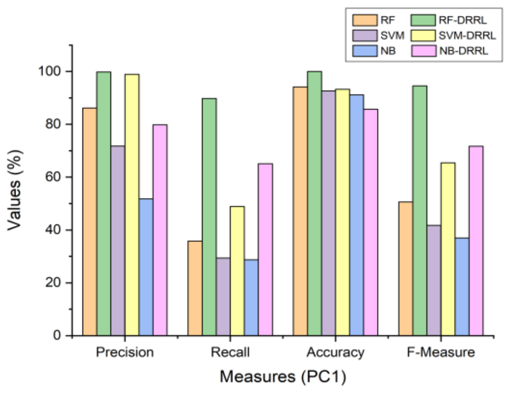
$$F1 = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})} \tag{4}$$

4.2 Results

Figure 3(a) illustrates the performance of the CM1 dataset, highlighting that the RF algorithm excelled in both precision and recall when combined with DRRL. The SVM demonstrated high precision but suffered from lower recall, indicating a tendency to reduce false positives. Naive Bayes (NB) achieved good overall accuracy with a balanced approach to precision and recall. Notably, with DRRL, NB exhibited impressive precision, decent recall, and an excellent F-measure, proving its effectiveness. Despite a slight decrease in recall, it maintained solid precision with a reasonable F-measure and competitive accuracy.


(a)


(b)


(c)

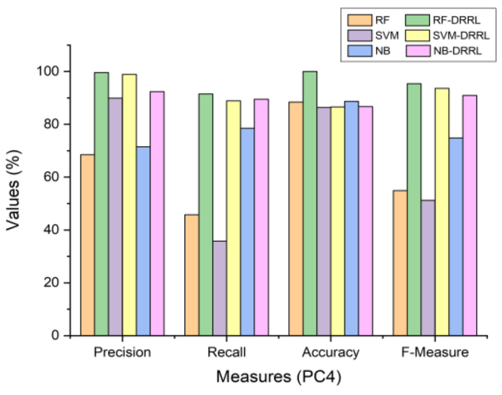

(d)

Fig. 3: Performance Measure of CM1, JM1, PC1 and PC4 Dataset

In contrast, Figure 3(b) presents the performance on the JM1 dataset, where RF achieved perfect precision with DRRL but had lower recall. SVM again showed high precision but very low recall, prioritizing the reduction of false positives. NB displayed moderate precision and recall, indicating a more balanced performance. These results underscore the varying performance characteristics of different models and datasets, with RF showing strong precision in both datasets but struggling with recall in JM1. SVM's focus on high precision often comes at the expense of recall, while NB offers a balanced approach across metrics.

Figure 3(c) illustrates the performance of various classifiers on the PC1 dataset, highlighting the impact of DRRL. The RF classifier achieved perfect precision and recall, indicating an optimal balance between identifying true positives and avoiding false positives. The SVM, while achieving perfect precision, suffered from lower recall, suggesting it missed some relevant instances. The NB classifier showed a balanced performance with notable improvements in recall, making it a reliable choice.

In contrast, Figure 3(d) depicts the performance on the PC4 dataset, where the RF classifier also achieved perfect scores with DRRL, albeit with potential over-fitting. Despite this, RF maintained commendable precision and recall, effectively identifying defective modules. The SVM's performance was marked by low recall, indicating challenges in defect recognition. Meanwhile, the NB classifier maintained a balanced and effective performance in both phases, showcasing its suitability for software defect prediction and its potential to enhance software quality assurance. The Random Forest (RF) classifier is prone to overfitting when applied to small and imbalanced datasets, such as those from NASA. This issue arises because the limited amount of data leads the RF model to capture noise rather than the underlying patterns, resulting in poor generalization of unseen data. An effective way to mitigate this overfitting is by employing ensemble models, which combine predictions from multiple algorithms. These ensemble approaches leverage the strengths of each model, thereby improving overall performance and accuracy.

• Performance of EVM

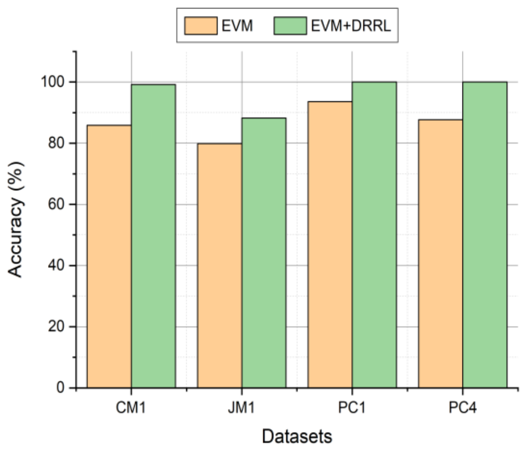


Fig 4: Comparison Analysis of EVM with the datasets

Figure 4 presents a comparative analysis evaluating the accuracy of the EVM with and without DRRL. The results demonstrate that incorporating DRRL significantly enhances prediction accuracy, particularly for the CM1, PC1, and PC4 datasets, where accuracy reaches up to 100%. This improvement underscores the EVM's effectiveness in identifying and retaining the most informative features, potentially reducing computational costs. However, in the case of the JM1 dataset, a marginal decrease in accuracy to 88.2% is observed with DRRL. This highlights the trade-off between the benefits of DRRL and the practical efficiency of classifiers in real-world applications.

- **Performance with existing techniques**

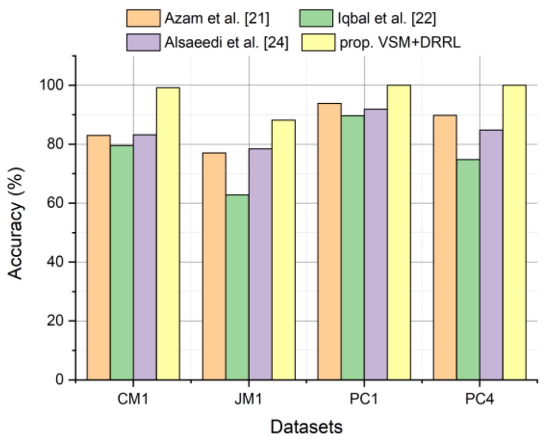


Fig 5: Comparison Analysis with existing technical works

Figure 5 presents a comparative analysis between the approach and existing technical works, demonstrating superior performance across all datasets. The EVM model achieves an accuracy of 99.2% with the CM1 dataset, 88.2% with the JM1 dataset, and a perfect 100% with both the PC1 and PC4 datasets. These results highlight the efficacy of the proposed approach in enhancing the effectiveness of the DRRL. The findings suggest that the individual classifier is highly effective in selecting rules that optimize the predictive performance of the EVM, thereby contributing significantly to improved overall accuracy.

5. CONCLUSION

This paper presented a Defect Relations Rule Learning (DRRL) approach designed to enhance software defect classification models. Unlike traditional decision-tree induction methods that evaluate one variable at a time, DRRL identifies high-confidence relationships among multiple variables, thereby utilizing the most effective rules for accurate classification. The study employs RF, SVM, and NB in the initial stage to evaluate prediction accuracy. The DRRL uncovers co-occurring attribute patterns in databases, demonstrating that classification based on association rules (AR) achieves higher accuracy compared to other techniques. In the second stage, an Ensemble Voting Model (EVM) is proposed to combine classifier prediction outcomes, further enhancing defect detection accuracy and reliability. The EVM, integrated with DRRL, shows superior performance, achieving 99.2% accuracy on the CM11 dataset, 88.2% on the JM1 dataset, and 100% on both the PC1 and PC4 datasets. These results highlight the DRRL's potential to significantly improve the precision of software defect prediction models. In the future it can be integrated with feature selection and ensemble methods can offer a means to improve prediction reliability through the combination of multiple models.

6. REFERENCES

[1] Mohandas R., Southern M., O'Connell E., Hayes M., "A Survey of Incremental Deep Learning for Defect Detection in Manufacturing," Big Data and Cognitive Computing, vol. 8, no. 1, pp. 7, 2024.

[2] Kumar H., Saxena V., "Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study," Journal of Software Engineering and Applications, vol. 17, pp. 155-171, 2024.

[3] Yuan Z., Liu C., Yu L., Zhang L., "ChangeChecker: A Tool for Defect Prediction in Source Code Changes Based on Incremental Learning Method," Proceedings of the 3rd International Conference on Computer Science and Network Technology, pp. 349-354, 2013.

[4] Wang S., Li Y., Mi W., Liu Y., "Software Defect Prediction Incremental Model Using Ensemble Learning," International Journal of Performability Engineering, vol. 16, no. 11, pp. 1771-1780, 2020.

[5] Bhandari I. S., Halliday M. J., Chaar J., Chillarenge R., Jones K., et al., "In Process Improvement through Defect Data Interpretation," IBM Systems Journal, vol. 33, no. 1, pp. 182-214, 1994.

[6] Thota M. K., Shajin F. H., Rajesh P., "Survey on Software Defect Prediction Techniques," International Journal of Applied Science and Engineering, vol. 17, pp. 331-344, 2020.

[7] Husin T. F., Pribadi M. R., Yohannes, "Implementation of Least Squares Support Vector Machines in Classification of Software Defect Prediction Data with Feature Selection," 9th International Conference on Electrical Engineering, Computer Science and Informatics, pp. 126-131, 2022.

[8] Richards J. A., "Supervised Classification Techniques," In Remote Sensing Digital Image Analysis, pp. 263-367, 2022.

[9] Odejide B. J., Bajeh A. O., Balogun A. O., Alanamu Z. O., Adewole K. S., Akintola A. G., Salihu S. A., "An Empirical Study on Data Sampling Methods in Addressing Class Imbalance Problem in Software Defect Prediction," in Proceedings of Computer Science, pp. 594-610, 2022.

[10] Wu X., Wang J., "Application of Bagging, Boosting and Stacking Ensemble and EasyEnsemble Methods for Landslide Susceptibility Mapping in the Three Gorges Reservoir Area of China," International Journal of Environmental Research and Public Health, vol. 20, no. 6, pp. 4977, 2023.

[11] Jiang F., Yu X., Gong D., Du J., "A Random Approximate Reduct-Based Ensemble Learning Approach and Its Application in Software Defect Prediction," Information Science, vol. 609, pp. 1147-1168, 2022.

[12] Chen H., Jing X.-Y., Zhou Y., Li B., Xu B., "Aligned Metric Representation Based Balanced Multiset Ensemble Learning for Heterogeneous Defect Prediction," Information and Software Technology, vol. 147, art. no. 106892, 2022.

[13] Balogun A. O., Bajeh A. O., Orie V. A., Yusuf-Asaju A. W., "Software Defect Prediction Using Ensemble Learning: An Analytic Network Process Based Evaluation Method," FUOYE Journal of Engineering and Technology, vol. 3, no. 2, pp. 50-55, 2018.

[14] Balogun A. O., Lafenwa-Balogun F. B., Mojeed H. A., Adeyemo V. E., Akande O. N., Akintola A. G., Bajeh A. O., Usman-Hamza F. E., "Synthetic Minority Over-sampling Technique-Based Homogeneous Ensemble Methods for Software Defect Prediction," in Computational Science and Its Applications, vol. 12254, pp. 615-631, 2020.

[15] Matloob F., Ghazal T. M., Taleb N., Aftab S., Ahmad M., Khan M. A., Soomro T. R., "Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review," IEEE Access, vol. 9, pp. 98754-98771, 2021.

[16] Daoud M. S., Aftab S., Ahmad M., Khan M. A., Iqbal A., Abbas S., Iqbal M., Ihnaini B., "Machine Learning Empowered Software Defect Prediction System," Intelligent Automation and Soft Computing, vol. 31, no. 2, pp. 1287-1300, 2022.

[17] Aftab S., Abbas S., Ghazal T. M., Ahmad M., Hamadi H. A., Yeun C. Y., Khan M. A., "A Cloud-Based Software Defect Prediction System Using Data and Decision-Level Machine Learning Fusion," Mathematics, vol. 11, no. 3, pp. 632, 2023.

[18] Cetiner M., Sahingoz O. K., "A Comparative Analysis for Machine Learning Based Software Defect Prediction Systems," in Proceedings of the 11th International Conference on Computing, Communication and Networking Technologies, pp. 1-7, 2020.

[19] Rath S. K., Sahu M., Das S. P., Bisoy S. K., Sain M., "A Comparative Analysis of Support Vector Machines and Extreme Learning Machine Classification on Software Reliability Prediction Model," Electronics, vol. 11, no. 17, pp. 2707, 2022.

[20] Wang K., Liu L., Yuan C., Wang Z., "Software Defect Prediction Model Based on Least Absolute Shrinkage and Selection Operator-Support Vector Machines," Neural Computing and Applications, vol. 33, no. 14, pp. 8249-8259, 2021.

[21] Azam M., Nouman M., Rehman G. A., "Comparative Analysis of Machine Learning Techniques to Improve Software Defect Prediction," KIET Journal of Computing and Information Sciences, vol. 5, no. 2, 2022.

[22] Iqbal A., Aftab S., Ali U., Nawaz Z., Sana L., Ahmad M., Husen A., "Performance Analysis of Machine Learning Techniques on Software Defect Prediction Using NASA Datasets," International Journal of Advanced Computer Science and Applications, vol. 10, no. 5, 2019.

[23] Mohammadi M., Nucci D. D., Tamburri D. A., "Bayesian Meta-Analysis of Software Defect Prediction With Machine Learning," IEEE Transactions on Industrial Cyber-Physical Systems, vol. 1, pp. 147-156, 2023.

[24] Alsaeedi A., Khan M. Z., "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study," Job Safety and Environmental Analysis, vol. 12, no. 5, pp. 85-100, 2019.

[25] Akimova E. N., Bersenev A. Y., Deikov A. A., Kobylkin K. S., Konygin A. V., Mezentsev I. P., Misilov V. E., "A Survey on Software Defect Prediction Using Deep Learning," Mathematics, vol. 9, art. no. 1180, 2021.

[26] Goyal S., "Effective Software Defect Prediction Using Support Vector Machines," International Journal of System Assurance Engineering and Management, vol. 13, pp. 681-696, 2022.

[27] Mehta S., Patnaik K. S., "Improved Prediction of Software Defects Using Ensemble Machine Learning Techniques," Neural Computing and Applications, vol. 33, no. 16, pp. 10551-10562, 2021.

[28] Wu X., Wang J., "Application of Bagging, Boosting and Stacking Ensemble and EasyEnsemble Methods for Landslide Susceptibility Mapping in the Three Gorges Reservoir Area of China," International Journal of Environmental Research and Public Health, vol. 20, no. 6, pp. 4977, 2023.

[29] Abbas S., Aftab S., Khan M. A., Ghazal T. M., Hamadi H. A., Yeun C. Y., "Data and Ensemble Machine Learning Fusion Based Intelligent Software Defect Prediction System," Computers, Materials & Continua, vol. 75, no. 3, pp. 6083-6100, 2023.

[30] Soe Y. N., Santosa P. I., Hartanto R., "Software Defect Prediction Using Random Forest Algorithm," in Proceedings of the 12th South East Asian Technical University Consortium, pp. 1-5, 2018.

[31] Cetiner M., Sahingoz O. K., "A Comparative Analysis for Machine Learning Based Software Defect Prediction Systems," in Proceedings of the 11th International Conference on Computing, Communication and Networking Technologies, pp. 1-7, 2020.

[32] Alsghaier H., Akour M., "Software Fault Prediction Using Particle Swarm Algorithm with Genetic Algorithm and Support Vector Machine Classifier," Software Practice and Experience, vol. 50, no. 4, pp. 407-427, 2020.

[33] Tua F. M., Sunindyo W. Danar, "Software Defect Prediction Using Software Metrics with Naïve Bayes and Rule Mining Association Methods," in Proceedings of the 5th International Conference on Science and Technology, pp. 1-5, 2019.

[34] Kim S. Y., Upneja A., "Majority Voting Ensemble with Decision Trees for Business Failure Prediction During Economic Downturns," Journal of Innovation and Knowledge, vol. 6, no. 2, pp. 112-123, 2021.

[35] Kaur I., Kaur A., "Comparative Analysis of Software Fault Prediction Using Various Categories of Classifiers," International Journal of System Assurance Engineering and Management, vol. 12, no. 3, pp. 520-535, 2021.

[36] PROMISE Software Engineering Repository, "http://promise.site.uottawa.ca/SERepository/datasets-page.html".