IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 March 2024, accepted 3 April 2024, date of publication 11 April 2024, date of current version 19 April 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3387856*

RESEARCH ARTICLE

# An Evaluation of Multi-Label Classification Approaches for Method-Level Code Smells Detection

**PRAVIN SINGH YADAV[1], RAJWANT SINGH RAO[1], AND ALOK MISHRA[2], (Senior Member, IEEE)**

[1]Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya Bilaspur, Bilaspur 495009, India
[2]Faculty of Engineering, Norwegian University of Science and Technology (NTNU), 7034 Trondheim, Norway

Corresponding author: Alok Mishra (alok.mishra@ntnu.no)

**ABSTRACT** (1) Background: Code smell is the most popular and reliable method for detecting potential errors in code. In real-world circumstances, a single source code may have multiple code smells. Multi-label code smell detection is a popular research study. However, limited studies are available on it, and there is a need for a standardized classifier for reliably identifying various multi-label code smells that belong to the method-level code smell category. The primary goal of this study is to develop a rule-based method for detecting multi-label code smells. (2) Methods: Binary Relevance, Label Powerset, and Classifier Chain methods are utilized with tree based single-label algorithms, including some ensemble algorithms in this research paper. The chi-square feature selection technique is applied to select relevant features. The proposed model is trained using 10-fold cross-validation, Random Search cross-validation parameter tuning, and different performance measures are used to evaluate the model. (3) Results: The proposed model achieves 99.54% of the best jaccard accuracy for detecting method-level code smells using the Classifier Chain method with the Decision Tree. The Decision Tree model incorporating a multi-label classifier outperforms alternative approaches to multi-label classification. Single-label classifiers produced better results after considering the correlation factor. (4) Conclusion: This study will facilitate scientists and programmers by providing a systematic method for detecting various code smells in software projects and saving time and effort during code reviews by detecting multiple problems simultaneously. After detecting multi-label code smell, programmers can create more organized, easier-to-understand, and trustworthy programs.

**INDEX TERMS** Code smell, cross-validation, multi-label code smell detection, parameter tuning, random search cross-validation, Z-score.

## I. INTRODUCTION

Multi-label classification involves predicting the presence or absence of multiple labels for an instance of source code; unlike traditional multi-class problems, only one label is associated with a single instance [1]. Multi-label classification is more general because the real world often has multiple semantic attributes or categories. In recent years, multi-label classification has garnered significant attention across different research domains.

Various machine learning methods like Random Forest (RF), Extreme Gradient Boosting (XGB), Decision Tree (DT), Gradient Boosting (GB), and Artificial Neural Network (ANN) are utilized with multi-label classification methods such as Binary Relevance (BR), Label Powerset (LP), and Classifier Chain (CC). All machine learning methods with Binary Relevance (BR) multi-label classifiers are denoted with the "BR_" prefix, and similarly, the "LP_" and "CC_" prefixes have the same meaning.

During software maintenance, changes are made to the program so it can function in an entirely new environment. Suboptimal software design and bad implementation approaches are two common causes of increased maintenance effort [2], [3].

Manual detection strategies based on code smell recognition were utilized in multiple approaches. To the best of our

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

knowledge and the available literature, Metrics-based code smell detection [4] and machine learning-based code smell detection [5], [6] are a few examples of methods that employ automatic detection techniques.

It is also unclear what defines a ''code smell''; different studies have used varied threshold values for the metrics used to identify instances that demonstrate code smells.

### A. PROBLEM STATEMENT

Researchers have analyzed different code smells and reached different conclusions. It is common for instances to belong to more than one class in real-world datasets. However, multi-label code smell has not received much attention from researchers, as shown in Table 1. There are several factors that affect the difference between results, and a few significant factors are listed below:

- A dataset that needs to be correctly normalized may dominate the learning process and bias the model.
- Including irrelevant features in the model's training process may cause the building of an improper model.
- Data leakage is another factor that can have an impact on the performance of a model. To conduct an accurate performance analysis, the testing data must be completely unknown to the model.

### B. MOTIVATION

The research conducted by Fontana et al. [7] has established a significant basis for analyzing code smells. Nevertheless, the datasets these researchers utilize have demonstrated a restricted scope regarding their utility in real-life situations. To address this, Nucci et al. [8] updated the Fontana et al. [7] datasets to reflect the real-world situation better. On the other hand, their results raise issues about the generalizability of Fontana et al. [7] initial findings. This motivates further research in the area of code smell detection and classification. More specifically, it is necessary to study the generalizability of current code smell detection methods that better perform on real-world code. The effect of code smells on software quality and maintainability should also be investigated.

This study presents various approaches to utilizing label dependencies for multi-label classification tasks effectively. The proposed approach captures the relationships between labels and software metrics for multi-label prediction.

### C. THE CONTRIBUTIONS OF THIS STUDY

The contributions are as follows:

- A multi-label method-level code smell dataset is utilized for this experimental work, and a multi-label dataset shows a real-world scenario.
- The paper presents various decision tree-based algorithms with Binary Relevance, Label Powerset, and Classifier Chain methods designed to address the multi-label classification problem. The proposed method offers a systematic approach to capture the dependencies among labels.
- The output is shown in tabular form that can be easily interpreted. It is an easy-to-understand way that helps in decision-making.
- To enhance the quality of the dataset, Z-score normalization for standardization is employed; furthermore, 10-fold cross-validation is applied. This combination helps to achieve a smoother dataset, enabling more effective training and accurate predictions.
- The chi-square feature selection technique is applied to select relevant features from the dataset.
- Assess the practical effectiveness by performing a comprehensive range of experiments and employing various evaluation metrics. The experimental outcomes have shown that the proposed method yields superior outcomes.

### D. RESEARCH QUESTIONS

To study the multi-label code smell detection approach, the following research questions (RQ) have been identified:

**RQ1** *Which machine learning approach provides the best multi-label code smell detection results?*

**Motivation:** Tree-based algorithms perform better in code smell detection [9]. Alazba et al. [10], Reis et al. [11], and Aljamaan et al. [12] presented ensemble approaches having outstanding results in single-label code smell detection. This paper utilized BR_RF, BR_XGB, BR_DT, BR_GB, BR_ANN, LP_RF, LP_XGB, LP_DT, LP_GB, LP_ANN, CC_RF, CC_XGB, CC_DT, CC_GB, and CC_ANN method to check the effectiveness of machine learning approach in multi-label classification.

***RQ2*** *Does using a collection of software metrics selected by the feature selection/extraction technique improve the multi-label code smell detection performance?*

**Motivation:** In various studies, Alazba et al. [10] utilized the Gain Ratio, and Mhawish et al. [9] utilized Genetic Algorithm (GA) Naïve Bayes and GA-CFS and demonstrated the significance of feature selection/extraction techniques in software metrics selection for single-label code smell detection. This paper utilizes the chi-square feature selection method to check the effectiveness of the feature selection technique for detecting multi-label code smell.

**RQ3** *What would be the effectiveness of machine learning classifier performance in multi-label code smell detection while considering correlation?*

**Motivation:** Guggulothu et al. [13] presented some classifier performance after considering correlation and showed that they found good results after considering correlation. This paper utilized machine learning and ensemble classifiers to check the efficacy after considering correlation in multi-label code smell prediction.

The remaining sections are structured as follows. Section II introduces the multi-label classification-related work. Section III discusses the proposed methodology. Section IV discusses the results of the experiments. Section V presents threads and validity, and Section VI discusses the conclusion of this experimental work.

## II. LITERATURE REVIEW

Multi-label classification has gained popularity in numerous fields, including text classification [14] and image classification [15]. The field of software engineering also uses multi-label classification. Multi-label versus single-label classifications of software system failure reports were studied by Feng et al. [16].

There are numerous methods for identifying code smells. Our efforts focus on machine learning methods to identify suspicious code.

Using 27 design metrics derived from a tool, Maneerat et al. [17] gathered datasets to evaluate seven code smells and apply seven machine learning techniques for detection.

Fontana et al. [6] used a machine learning technique that rates the severity of code smells to assist developers in assigning values to classes and functions. They employed techniques from various classification methods.

The previously reported research only considered one label for code smell detection. In this respect, Guggulothu et al. [13] have employed multi-label categorization strategies.

Di Nucci et al. [8] addressed the limitations observed in a prior experiment. They used the same seventy-four systems and experimental setup to test their model; the main difference was in the training dataset, where they included more realistic examples of each code smell kind. However, their findings indicated that the model's performance could have been more accurate than the original study. As a result, the authors concluded that applying machine learning techniques for code smell detection requires further investigation and consideration.

The proposed research differs from the existing literature in two significant aspects: first, an extra step is added to the process using a feature selection strategy, and second, various machine learning algorithms are incorporated for detecting the code smells. Table 1 comprises a brief description of the approaches used.

## III. PROPOSED METHODOLOGY

The proposed work represents a method-level multi-label code smell prediction framework, a dataset by Guggulothu et al. [13] is utilized for this experiment. As represented in Figure 1, the two method-level code smells (Long method and Feature envy) are merged to build the dataset. The dataset is divided into two parts; some preprocessing is applied to both parts, and further feature selection, parameter tuning, and performance evaluation are applied, as discussed in detail below. Jupyter Notebook 6.5.2, Scikit-learn 1.2.2, skmultilearn 0.2.0, Pandas 1.5.3, NumPy 1.26.4, Keras 2.12.0, and tensorflow 2.12.0 packages are utilized to develop Python code.

### A. DATASET CHOICE AND ILLUSTRATION

The Long method and Feature envy datasets from Guggulothu et al. [13] were utilized in this study. The dataset is publicly available on https://github.com/thiru578/Multilabel-Dataset. The dataset is a multi-label dataset that represents real-world situations. Initially, the Fontana et al. dataset [7] had 420 instances, one label, and 82 software metrics. Later, Guggulothu et al. [13] modified the Fontana et al. [7] dataset and removed irrelevant software metrics such as classes, projects, and packages. As a result, the final dataset has 445 instances, two labels, and 46 software metrics. Out of these, 100 instances have both code smells, 102 instances have any one of the two code smells, and 243 instances have no code smell.

- **Long method**- A Long method is a method-level code smell that extensively uses data from other classes, has much code, is complicated, and is hard to recognize [9].
- **Feature envy**- Feature envy is a method-level code smell. In place of their data, these methods extensively use data from other classes. Utilizing characteristics entered via accessor methods, they lean toward utilizing features of other classes [9].

**Multi-label method-level code smell dataset statistics-** Table 2 provides an overview of essential parameters of the multi-label dataset that show the characteristics and are critical in measuring the imbalance ratio. There are extra metrics besides the traditional ones for multi-label datasets, such as the mean imbalance ratio (MeanIR) and coefficient of variation of the imbalance ratio per label (CVIR) [38]. Cardinality is the average number of active labels per instance; density is a dimensionless measure obtained by dividing the cardinality by the total number of labels. The dataset has four possible label combinations consisting of two labels. The MeanIR and the CVIR provide insights into the dataset's balancing; generally, an imbalance is indicated by a MeanIR greater than 1.5 and a CVIR greater than 0.2 Charte et al. [38]. The balanced nature of the utilized multi-label dataset is supported by its MeanIR of 1.079 (below 1.5) and CVIR of 0.102 (below 0.2).

$$\begin{aligned} \text{MeanIR} &= \frac{1}{|K|}\sum_{k=1}^{K}(\text{IRLbl}\,(k)) \\ &= \frac{1+(162/140)}{2} \\ &= 1.079 \end{aligned} \tag{1}$$

$$\begin{aligned} \text{CVIR} &= \frac{\text{IRLbl}\sigma}{\text{MeanIR}} \\ &= \frac{0.110}{1.079} \\ &= 0.102 \end{aligned} \tag{2}$$

$$\textit{where}\ \text{IRLbl}\sigma = \sqrt{\sum_{k=k1}^{K}\frac{(\text{IRLbl}\,(k)-\text{MeanIR})^2}{|K|-1}}$$

### B. TRAIN TEST SPLIT

Train test split is essential to separate training data from test data to save the model from data leakage. This experiment uses the hold-out method with an 80%-20% ratio to train test split. In the initial dataset, Guggulothu et al. [13] contain 445 instances and 47 features; after the training test

**TABLE 1.** Comparison of earlier approaches used to detect code smells.

| Reference | Year | Predicting multi-label code smells | Used feature selection/ feature extraction technique | Analysis of code smells includes correlation | Dataset relevant for use in the real-world case | Used ensemble technique (Boosting) |
|---|---|---|---|---|---|---|
| Maneerat et al.[17] | 2011 | No | No | No | No | No |
| Fontana et al.[18] | 2013 | No | No | No | No | No |
| Amorim et al.[19] | 2015 | No | Yes (Genetic Algorithm) | No | No | No |
| Fontana et al.[7] | 2016 | No | No | No | No | Yes |
| White et al. [20] | 2016 | No | No | No | No | No |
| Kim et al. [21] | 2017 | No | No | No | No | No |
| Fontana et al. [6] | 2017 | No | Yes | No | No | No |
| Kaur et al.[22] | 2017 | No | No | No | No | No |
| Nucci et al. [8] | 2018 | No | Yes(Gain Ratio Feature Evaluation) | No | Yes | Yes |
| Liu et al. [23] | 2018 | No | No | No | No | No |
| Pecorelli et al. [24] | 2019 | No | Yes, correlation-based feature selection (CFS) | No | No | No |
| Jesudoss et al. [25] | 2019 | No | No | No | No | No |
| Guggulothu et al.[26] | 2019 | No | No | No | No | No |
| Mhawish et al.[27] | 2019 | No | Yes (GA-based Naive Bayes, CFS) | No | No | No |
| Gupta et al. [28] | 2019 | No | Yes (Wilcoxon sign rank and cross-correlation analysis) | No | No | No |
| Dewangan et. al. [29] | 2021 | No | Yes (chi-square, Wrapper-based Feature Selection) | No | No | No |
| Kiyak et al.[1] | 2019 | No | Yes (a subset of the feature) | No | No | Yes |
| Barbez et al. [30] | 2020 | No | No | No | No | Yes |
| Mhawish et al. [9] | 2020 | No | Yes (GA-based Naive Bayes, CFS) | No | No | Yes |
| Guggulothu et al. [13] | 2020 | Yes | No | Yes | Yes | No |
| Kaur et al. [31] | 2021 | No | Yes (CFS, Info gain Attribute Evaluator, Relief Attribute Eval (RAE)) | No | No | No |
| Draz et al. [32] | 2021 | No | No | No | No | No |
| Gupta et al. [33] | 2021 | No | Yes (Cross-Correlation analysis and Wilcoxon Sign Rank Test) | No | No | No |
| Alazba et al. [10] | 2021 | No | Yes (Gain Ratio) | No | No | Yes |
| Aljamaan et al. [12] | 2021 | No | Yes (Gain Ratio) | No | No | Yes |
| Reis et al.[11] | 2022 | No | No | No | No | Yes |
| Khleel et al.[34] | 2022 | No | Yes (Wrapper-based methods, filter-based methods, embedded methods) | No | No | No |
| Abdou et al. [35] | 2022 | No | No | No | No | Yes |
| Dewangan et al. [36] | 2023 | No | Yes (chi-square) | No | No | Yes |
| Rao et al. [37] | 2023 | No | Yes, Principal component analysis (PCA) | No | No | No |
| **Proposed** | | **Yes** | **Yes (chi-square)** | **Yes** | **Yes** | **Yes** |

**TABLE 2.** Statistics of multi-label method-level code smell dataset.

| Number of instances | Number of features | Number of target labels | Cardi-nality | Density | Mean-IR | CVIR |
|---|---|---|---|---|---|---|
| 445 | 47 | 2 | 0.678 | 0.339 | 1.079 | 0.102 |

split, the training set has 356 instances, and the test set has 89 instances, as shown in Figure 1.

### C. NORMALIZATION

The feature ranges in the datasets used are not uniform, so feature normalization should be applied before applying the machine learning technique. A normalization method, the Z-score feature scaling methodology, was used in this paper. It sets the dataset mean to 0 and the standard deviation to 1 [39].

$$\text{Z} - \text{score} = (x - \mu)/\sigma \tag{3}$$

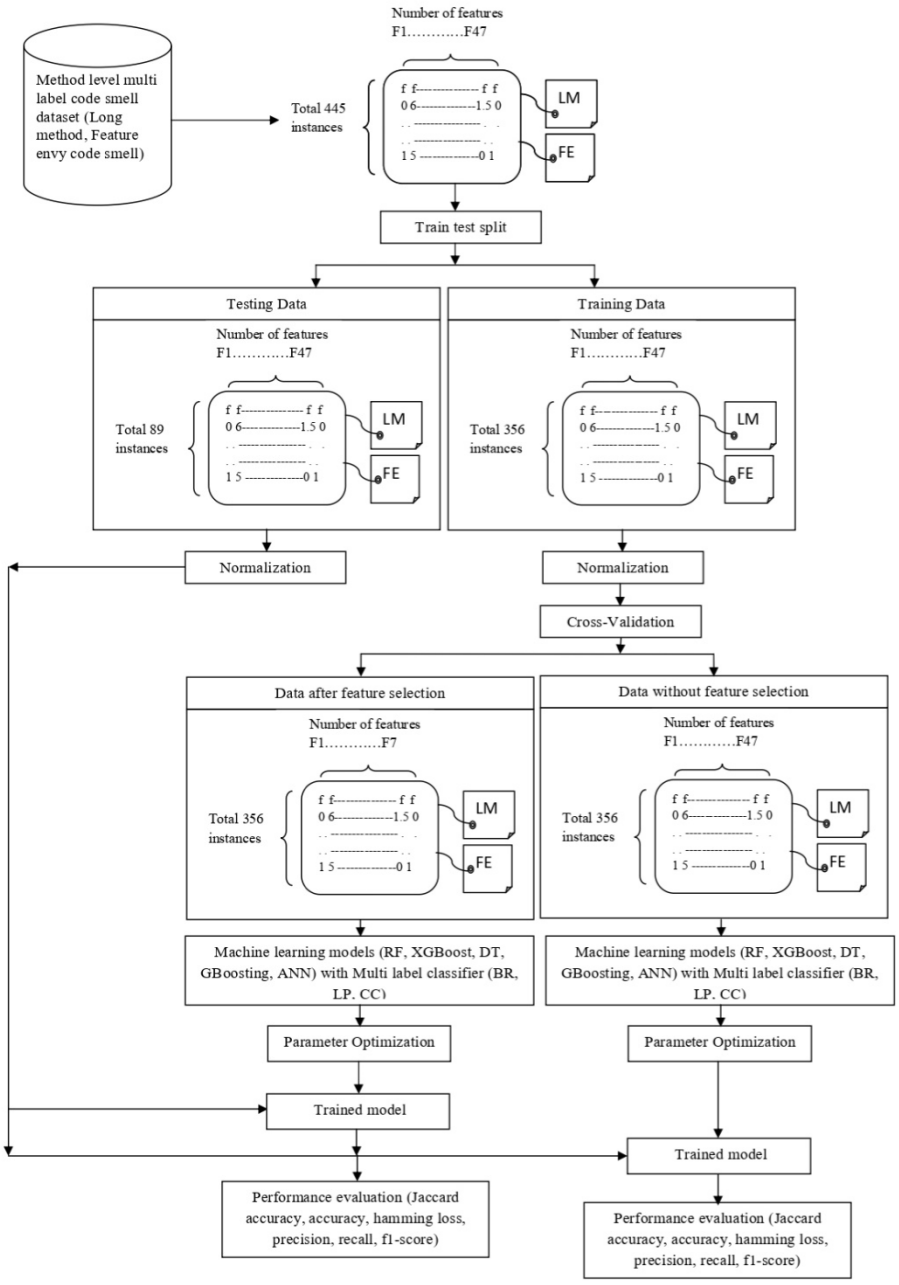


**FIGURE 1.** Proposed model for multi-label code smell detection.

where x = Original value, $\mu$ = Mean of data, $\sigma$ = Standard deviation of data.

### D. CROSS-VALIDATION

In this experimental work, 10-fold cross-validation is applied, which is essential to train the model correctly. In cross-validation, the model is trained with different combinations of training sets. The model is trained in ten iterations; during each cycle, a portion serves as test data, while the remaining portion serves as train data. The ultimate value of accuracy is determined by taking the mean of all iterations. The outcome is produced after ten runs of the 10-fold cross-validation procedure, which effectively takes the randomness into account.

### E. FEATURE SELECTION

Dewangan et al. [29], [36] utilized chi-square feature selection in software metrics selection for single-label code smell detection and found better results. Chi-square feature selection is crucial in multi-label code smell detection for its ability to identify the most discriminative features relevant to each label, aiding in the accurate classification of multi-label code smell. In this experimental work,

the chi-square feature selection method is applied; selecting relevant features while removing unnecessary ones is a crucial process in feature selection. By computing the chi-square statistic between each feature and each label, chi-square feature selection determines the significance of features for multi-label classification. The chi2 function from scikit-learn is used for this purpose. Features with higher chi-square values are more significant for multi-label code smell prediction. Seven features were selected using the feature selection method. Table 3 shows the selected features. The details about software metrics and custom metrics are shown in Table 4 and Table 5, respectively. Metrics definitions are available at https://github.com/bniyaseen/codesmell/blob/master/metricdefinitions.pdf. The general formula for chi-square [36]:

$$\text{Chi} - \text{square} = \sum \frac{(O-E)^2}{E} \tag{4}$$

where O = Observed frequency, and E = Expected frequency.

**TABLE 3. Selected features using the chi-square feature selection method.**

| Code smell | Selected software metrics(features) | Number of features | Percentage of selected features (In percentage) |
|---|---|---|---|
| Long method, Feature envy | 'ATFD_method', 'LOC_method', 'CYCLO_method', 'NOLV_method', 'NOAV_method', 'CFNAMM_method', 'CINT_method' | 7 | 14.894% |

### F. CLASSIFIER USED

Several methods deal with multi-label classification, such as problem transformation, algorithm adaptation, and ensemble methods. This experimental work uses three problem transformation methods of multi-label classification (BR, LP, CC) with single-label machine learning methods (RF, XGB, DT, GB, ANN) to predict multi-label method-level code smell.

**Multi-label classifier**-

- **Binary relevance** (BR) –The method converts the multi-label dataset into n-binary datasets, where n is the label count. Subsequently, a combined output is produced by the binary classifiers. Label correlation is not taken into account by this strategy [1].
- **Label powerset** (LP) - In this transformation, the multi-label dataset becomes a multi-class dataset, each representing a distinct set of labels [1].
- **Classifier chain** (CC) –N classifiers are involved in this method. The initial step is to train the classifier using the initial input. Iteratively, the input is supplemented with its prior binary prediction outcome [1].

**Machine learning method** –

- **Random forest** (RF) - The random forest approach is a machine learning tool for solving classification issues; it takes the average or mean value of the predictions made by each decision tree and uses it to generate the final output, which can enhance accuracy while reducing dataset overfitting [29] In this experiment, the models' parameters, Max Depth, Number of trees, Criterion, and Class weight are well-tuned for better results.
- **Extreme gradient boosting** (XGB) - This algorithm, also known as XGBoost, is a supervised machine learning algorithm based on trees. In order to ensure the intended result, it integrates the predictions of multiple weaker models. An optimization framework and regularized learning of goal functions are used in this process [40]. This experiment achieved Improved results by fine-tuning the following model parameters: Max Depth, Learning rate, and Number of trees.
- **Decision tree** (DT) - A decision tree is a hierarchical structure in which each leaf node usually represents a result or class label, and the inside nodes reflect decisions depending on the values of attributes. The

**TABLE 4. Software metrics with description [7], [9].**

| Quality Dimension | Metrics name | Metric Label | Granularity |
|---|---|---|---|
| Coupling | Access to Foreign Data | ATFD_method | Method |
| Size | Line of code | LOC_method | Method |
| Complexity | Cyclomatic Complexity | CYCLO_method | Method |
| Complexity | Number of Local Variable | NOLV_method | Method |
| Complexity | Number of Accessed Variables | 'NOAV_method' | Method |
| Complexity | Number of Parameters | 'NOP_method' | Method |
| Complexity | Maximum Nesting Level | 'MAXNESTING_method' | Method |
| Complexity | Access to Local Data | 'ATLD_method' | Method |
| Complexity | Called Local Not Accessor or Mutator Methods | 'CLNAMM_method' | Method |
| Coupling | Called Foreign Not Accessor or Mutator Methods | CFNAMM_method | Class, Method |
| Coupling | - | 'FANOUT_method' | Method |
| Coupling | Foreign Data Providers | 'FDP_method' | Method |
| Coupling | Changing Classes | 'CC_method' | Method |
| Coupling | Changing Methods | 'CM_method' | Method |
| Coupling | Number of Message Chain Statements | 'NMCS_method' | Method |
| Coupling | Maximum Message Chain Length | 'MaMCL_method' | Method |
| Coupling | Mean Message Chain Length | 'MeMCL_method' | Method |
| Coupling | Coupling Intensity | CINT_method | Method |

Note: In the above table, '-' indicate that no data available.

**TABLE 5.** Custom metrics with description [7], [9].

| Metrics name | Metric Label |
|---|---|
| Number of Final Attributes | NOFA |
| Number of Public Methods | NOPLM |
| Number of Protected Methods | NOPRM |
| Number of Private Methods | NOPM |
| Number of Abstract Methods | NOABM |
| Number of Final Methods | NOFM |
| Number of Protected Attributes | NOPRA |
| Number of not Final and non - Static Attributes | NONFNSA |
| Number of Private Attributes | NOPVA |
| Number of Final and Static Attributes | NOFSA |
| Number of Static Attributes | NOSA |
| Number of non - Final and Static Attributes | NONFSA |
| Number of Final and non - Static Attributes | NOFNSA |
| Number of Final and non - Static Methods | NOFNSM |
| Number of non - Final and non - Static Methods | NONFNSM |
| Number of Static Methods | NOSM |
| Number of Final and Static Methods | NOFSM |
| Number of non - Final and Static Methods | NONFSM |

values of attributes are evaluated at each internal node, which affects the variables [29]. The parameters of the models, including the Maximum depth of the tree, Criterion, the strategy used to choose the split, Minimum Number of samples required to split, Minimum Number of samples required to be a leaf node, and Minimum impurity split, have been fine-tuned in this experiment to achieve better outcomes.

- **Gradient boosting** (GB) - One of the most influential ensembles of machine learning approaches is the Gradient boosting (GB) algorithm. With its adaptability, GB can be used for both continuous target variables, as in regression, and categorical ones, as in classification tasks, helping to reduce algorithmic bias and improve accuracy [9]. In this model, Learning rate, Max Depth, and number of trees are essential parameters that are correctly tuned for better results.
- **Artificial neural network** (ANN) - An ANN, or neural network, is a mathematical model that takes design principles from the architecture and operation of natural networks in the brain. It manipulates data using interconnected artificial neurons, a technique in artificial neural computation [30]. This experiment uses the input layer, output layer, and two hidden layers; the relu activation function is utilized with adam optimizer and binary cross entropy as the loss function.

### G. PARAMETER OPTIMIZATION

This experiment utilizes the Random Search cross-validation (CV) method to tune the model better and find the best parameters. Table 6 shows the parameters with starting and end values.

**TABLE 6.** Parameter tuning using random search CV with range and step.

| Machine learning algorithm | Parameter | Start value | End value | Step |
|---|---|---|---|---|
| RF | Max Depth | 1 | 20 | 1 |
| | Number of trees | 1 | 20 | 1 |
| | Criterion | Information gain, Gini impurity | | |
| | Class weight | Balanced, Balanced subsample | | |
| XGB | Learning rate | 0 | 1 | 0.01 |
| | Max Depth | 1 | 10 | 1 |
| | Number of trees | 1 | 100 | 10 |
| DT | Maximum depth of the tree | 1 | 50 | 1 |
| | Criterion | Gini impurity, Information gain | | |
| | The strategy used to choose the split | best, random | | |
| | Minimum number of samples required to split | 2 | 25 | 1 |
| | Minimum number of samples required to be a leaf node | 1 | 10 | 1 |
| | Minimum impurity split | 0 | 5 | 1 |
| GB | Learning rate | 0 | 2 | 0.01 |
| | Max Depth | 1 | 10 | 1 |
| | Number of trees | 1 | 75 | 1 |
| ANN | Learning rate | 0 | 0.1 | 0.01 |
| | Activation function | Relu, Sigmoid | | |

Table 6 contains some parameters of the machine learning algorithm that do not have numerical values. These parameters include the criterion, class weight, strategy for choosing the split, and activation function. Instead of having start, end, and step values, these parameters have a set of possible values.

### H. PERFORMANCE EVALUATION

This experiment uses six evaluation approaches: jaccard accuracy, accuracy, hamming loss, precision, recall, and f1 score. Performance evaluation is essential to measure the correctness of the model. The performance evaluation metrics are-

**Jaccard accuracy** - The jaccard accuracy metric measures the percentage of correct predictions for a given set of labels. This metric is mainly used in multi-label classification [1], which is calculated as follows:

$$\text{Jaccard Accuracy} = \frac{1}{k}\sum_{i=1}^{k}\frac{|(Zi \bigcap Yi)|}{|(Zi \bigcup Yi)|} \tag{5}$$

where Zi = the predicted label set, Yi = the truth label set, and k = the total number of instances.

**Accuracy** -Accuracy quantifies how well a model does in making accurate predictions across all labels [29], which is calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total Number of predictions}} \tag{6}$$

**Hamming loss** - Hamming loss is used to quantify the typical mislabeled cases across all labels. This score takes

into account both correct and incorrect label predictions, as well as missed labels. This parameter's value should be decreased as classification performance improves [1]. The hamming loss can be expressed as the symmetric difference between the grounded truth label set and the predicted set.

$$\text{Hamming loss} = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{r}|Zi\Delta Yi| \tag{7}$$

where Zi is the predicted label set, Yi = the truth label set, k = the total number of instances, and r = the total number of labels.

**Precision** - Precision, sometimes known as a positive predictive value, is a metric for assessing a model's accuracy on positive predictive performance [29]; use the following formula:

$$\text{Precision (P)} = \frac{\text{Total positive correct predictions}}{\text{Total number of positive predictions}} \tag{8}$$

**Recall**- Recall, sometimes known as true positive rate, is a metric for assessing a model to identify positive instances correctly [41].

$$\text{Recall (R)} = \frac{\text{Total positive correct predictions}}{\text{Total Number of correct predictions}} \tag{9}$$

**F1 score** - The f1 score is applicable to assess the model's overall performance. It is the harmonic mean of precision and recall [1].

$$\text{F1 score} = \frac{2XPXR}{P+R} \tag{10}$$

## IV. EXPERIMENTAL RESULT

### A. PERFORMANCE OF MACHINE LEARNING APPROACH IN MULTI-LABEL CODE SMELL DETECTION

In order to answer RQ1, this paper utilized five different machine learning models (RF, XGB, DT, GB, ANN) along with a multi-label classifier (BR, LP, CC) to evaluate the efficacy of every method. Additionally, a chi-square feature selection method was utilized to determine the most suitable metrics, as listed in Table 3. Tables 7 to 15 provide a comprehensive overview of the experimental results for each method.

Table 7-9 presents the performance of five machine learning methods that detect multi-label code smell using BR, LP, and CC multi-label classifiers, respectively. The experiment compares the results obtained with and without a feature selection strategy, using three evaluation metrics: jaccard accuracy, accuracy, and hamming loss. The classifiers BR_DT, LP_DT, and CC_DT achieved the highest jaccard accuracy of 99.24%, 99.39%, and 99.54%, respectively, considering both with and without feature selection. The hamming loss values for the classifiers BR_DT, LP_DT, and CC_DT are 0.28%, 0.22%, and 0.22% respectively with feature selection.

Table 10-12 shows findings from the research that assessed the efficacy of five machine learning methods in detecting multi-label code smell. The study used BR, LP, and CC multi-label classifiers with and without feature selection. The tables compare the performance of the methods using three evaluation metrics: precision, recall, and f1 score, all using micro averaging. While considering micro averaging with an f1 score, it is observed that after feature selection, the performance of all classifiers increases except for the BR_GB and CC_DT.

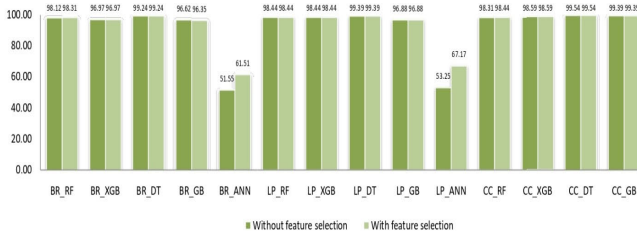


**FIGURE 2.** Result of code smell detection with and without feature selection.

Table 13-15 presents the results of five machine learning techniques applied to the BR, LP, and CC multi-label classifiers, both with and without feature selection methodology. The objective was to predict multi-label code smell, and three evaluation metrics based on macro averaging were used to compare performance: precision, recall, and f1 score. While considering macro averaging with an f1 score, it is observed that after feature selection, the performance of all classifiers increases except for the CC_RF and CC_DT.

The DT model with a multi-label classifier has proven to be more effective than other methods for multi-label classification in terms of performance. This is because the DT model has the ability to generate precise and easy-to-understand decision rules directly from the data. Additionally, DT is less likely to overfit and can capture complex relationships within the dataset, resulting in higher accuracy.

### B. EFFECT OF FEATURE SELECTION IN MULTI-LABEL CODE SMELL DETECTION

In response to RQ2, this experimental work applied the chi-square feature selection technique, identifying seven software metrics as more relevant than others. Figure 2 compares the jaccard accuracy of different classifiers with and without feature selection. It was observed that the performance of the XGB and DT classifiers remained unchanged after feature selection for all multi-label classifiers used in this paper. The LP_RF model showed no difference in response before and after feature selection, while the BR_RF and CC_RF models showed a positive impact after feature selection. The BR_ANN, LP_ANN, and CC_ANN models performed better after feature selection. Only the BR_GB model showed negative performance after feature selection, while the GB model showed constant performance in the rest of the cases.

Feature selection with a multi-label classifier has been shown to be more effective than using all available features in multi-label classification. This is because by focusing only on the most crucial features, computation is accelerated, and efficiency is enhanced. Prioritizing significant features simplifies

**TABLE 7.** BR multi-label classifier performance for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Jaccard accuracy (In percentage) | Accuracy (In percentage) | Hamming loss (In percentage) | Jaccard accuracy (In percentage) | Accuracy (In percentage) | Hamming loss (In percentage) |
| BR_RF | 98.12 | 98.65 | 0.67 | 98.31 | 98.76 | 0.62 |
| BR_XGB | 96.97 | 97.75 | 1.12 | 96.97 | 97.75 | 1.12 |
| BR_DT | 99.24 | 99.44 | 0.28 | 99.24 | 99.44 | 0.28 |
| BR_GB | 96.62 | 97.75 | 1.12 | 96.35 | 97.42 | 1.29 |
| BR_ANN | 51.55 | 47.19 | 32.02 | 61.51 | 68.54 | 17.98 |

**TABLE 8.** LP multi-label classifier performance for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Jaccard accuracy (In percentage) | Accuracy (In percentage) | Hamming loss (In percentage) | Jaccard accuracy (In percentage) | Accuracy (In percentage) | Hamming loss (In percentage) |
| LP_RF | 98.44 | 98.88 | 0.56 | 98.44 | 98.88 | 0.56 |
| LP_XGB | 98.44 | 98.88 | 0.56 | 98.44 | 98.88 | 0.56 |
| LP_DT | 99.39 | 99.55 | 0.22 | 99.39 | 99.55 | 0.22 |
| LP_GB | 96.88 | 97.75 | 1.12 | 96.88 | 97.75 | 1.12 |
| LP_ANN | 53.25 | 59.55 | 26.97 | 67.17 | 68.54 | 17.98 |

**TABLE 9.** CC multi-label classifier performance for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Jaccard accuracy (In percentage) | Accuracy (In percentage) | Hamming loss (In percentage) | Jaccard accuracy (In percentage) | Accuracy (In percentage) | Hamming loss (In percentage) |
| CC_RF | 98.31 | 98.88 | 0.56 | 98.44 | 98.88 | 0.56 |
| CC_XGB | 98.59 | 98.88 | 0.56 | 98.59 | 98.88 | 0.56 |
| CC_DT | 99.54 | 99.66 | 0.17 | 99.54 | 99.55 | 0.22 |
| CC_GB | 99.39 | 99.55 | 0.22 | 99.39 | 99.55 | 0.22 |
| CC_ANN | 53.25 | 59.55 | 26.97 | 67.17 | 68.54 | 17.98 |

**TABLE 10.** Microaveraging the performance of BR with various classifiers for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) |
| BR_RF | 100.00 | 99.05 | 99.05 | 99.38 | 99.14 | 99.14 |
| BR_XGB | 96.97 | 98.46 | 98.46 | 96.97 | 98.46 | 98.46 |
| BR_DT | 99.23 | 99.61 | 99.61 | 99.23 | 99.61 | 99.61 |
| BR_GB | 98.25 | 98.25 | 98.25 | 96.81 | 98.14 | 98.14 |
| BR_ANN | 54.05 | 67.80 | 67.80 | 77.42 | 75.00 | 75.00 |

research efforts, resulting in higher accuracy without excessive computing stress. This ultimately enables improved efficiency and better performance.

### C. IMPACT OF CORRELATION IN MULTI-LABEL CODE SMELL DETECTION

To address RQ3, various single-label machine learning methods and multi-label classifiers, including BR, LP, and CC, were employed. After analyzing the results in Tables 7 to 9, it was observed that LP and CC outperformed the BR method in terms of evaluation measures such as jaccard accuracy, accuracy, and hamming loss. LP and CC methods produced better results than BR due to their use of correlation factors in classification, which BR disregards. The classifier's ability to detect multi-label code smells is enhanced when correlations between code smells are considered.

**TABLE 11.** Microaveraging the performance of LP with various classifiers for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) |
| LP_RF | 100.00 | 99.21 | 99.21 | 100.00 | 99.21 | 99.21 |
| LP_XGB | 100.00 | 99.21 | 99.21 | 100.00 | 99.21 | 99.21 |
| LP_DT | 99.38 | 99.69 | 99.69 | 99.38 | 99.69 | 99.69 |
| LP_GB | 100.00 | 98.41 | 98.41 | 100.00 | 98.41 | 98.41 |
| LP_ANN | 58.89 | 68.83 | 68.83 | 68.60 | 78.67 | 78.67 |

**TABLE 12.** Microaveraging the performance of CC with various classifiers for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) |
| CC_RF | 100.00 | 99.15 | 99.15 | 100.00 | 99.21 | 99.21 |
| CC_XGB | 100.00 | 99.29 | 99.29 | 100.00 | 99.29 | 99.29 |
| CC_DT | 99.54 | 99.77 | 99.77 | 99.38 | 99.69 | 99.69 |
| CC_GB | 99.38 | 99.69 | 99.69 | 99.38 | 99.69 | 99.69 |
| CC_ANN | 58.89 | 68.83 | 68.83 | 68.60 | 78.67 | 78.67 |

**TABLE 13.** Macroaveraging the performance of BR with various classifiers for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) |
| BR_RF | 100.00 | 97.93 | 98.94 | 99.32 | 98.79 | 99.04 |
| BR_XGB | 96.94 | 100.00 | 98.45 | 96.94 | 100.00 | 98.45 |
| BR_DT | 99.31 | 100.00 | 99.65 | 99.31 | 100.00 | 99.65 |
| BR_GB | 98.08 | 98.08 | 98.08 | 96.80 | 99.48 | 98.12 |
| BR_ANN | 54.19 | 92.11 | 67.72 | 70.96 | 67.86 | 68.08 |

**TABLE 14.** Macroaveraging the performance of LP with various classifiers for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) |
| LP_RF | 100.00 | 98.28 | 99.12 | 100.00 | 98.28 | 99.12 |
| LP_XGB | 100.00 | 98.28 | 99.12 | 100.00 | 98.28 | 99.12 |
| LP_DT | 99.44 | 100.00 | 99.72 | 99.44 | 100.00 | 99.72 |
| LP_GB | 100.00 | 96.85 | 98.40 | 100.00 | 96.85 | 98.40 |
| LP_ANN | 58.89 | 82.51 | 68.58 | 68.60 | 91.38 | 78.21 |

### *D. PERFORMANCE COMPARISON WITH PREVIOUS PAPER RESULTS*

Table 16 compares different approaches to achieve jaccard accuracy in a method-level multi-label dataset. Kiyak et al. [1] and Guggulothu et al. [13] used the CC algorithm with the RF algorithm and LC with the B-J48 pruned method, respectively, and achieved 93.6% and 97.5% accuracy using all features. On the other hand, the proposed approach only used seven features and achieved 99.54% accuracy using the CC algorithm with the DT method. Comparing our method to others' reveals that it performs better.

Multi-label classifiers with feature selection techniques have been applied, and the combination of methods obtained better results than other authors. Feature selection reduces redundant features, and the selected feature may cause to give better accuracy. The proposed approach achieved superior

**TABLE 15.** Macroaveraging the performance of CC with various classifiers for detecting code smells with and without feature selection.

| Classifier | Without feature selection | | | With feature selection | | |
|---|---|---|---|---|---|---|
| | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) | Precision (In percentage) | Recall (In percentage) | F1 score (In percentage) |
| CC_RF | 100.00 | 98.44 | 99.21 | 100.00 | 98.28 | 99.12 |
| CC_XGB | 100.00 | 98.78 | 99.38 | 100.00 | 98.78 | 99.38 |
| CC_DT | 99.58 | 100.00 | 99.79 | 99.44 | 100.00 | 99.72 |
| CC_GB | 99.44 | 100.00 | 99.72 | 99.44 | 100.00 | 99.72 |
| CC_ANN | 58.89 | 82.51 | 68.58 | 68.60 | 91.38 | 78.21 |

**TABLE 16.** Performance comparison with previous paper results.

| Year | Reference | Technique used | Feature selection technique used | No. of feature used | Jaccard accuracy (In percentage) |
|---|---|---|---|---|---|
| 2019 | Kiyak et al.[1] | CC with RF | - | Not mentioned | 93.6 |
| 2020 | Guggulothu et al. [13] | LC with B-J48 pruned | - | 46 | 97.5 |
| | **Proposed approch** | **CC with DT** | **Chi-square** | **7** | **99.54** |

Note: In the above table, '-' represents no technique used.

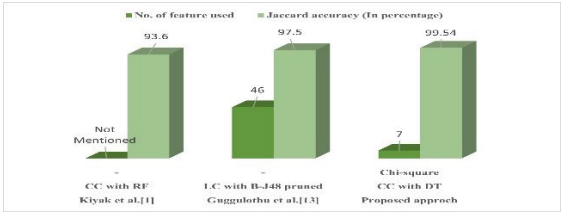


**FIGURE 3.** Performance comparison with other author results.

results using only seven features, thus significantly reducing computation time and saving valuable time and effort for researchers in this field.

Figure 3 shows a graphical representation of the proposed approach's results compared to other authors' results. Kiyak et al. [1] did not mention the number of features, and Guggulothu et al. [13] used all 46 features for their experiment. The proposed approach used only 7 features and got better results than others.

## V. THREATS OF VALIDITY

This study acknowledges that there may be some limitations to the proposed investigation but also suggests ways to overcome those restrictions. The study uses two code smells from Guggulothu et al. [13] dataset related to method-level code smells. It was created to represent a real-world use case by merging method-level datasets from Fontana et al. [7]. These datasets contain several features, mostly software metrics, which can display varying degrees of significance and correlation. The dataset was normalized using Z-score normalization, and the feature count was reduced using the chi-square feature selection methodology. However, there is potential for further enhancement in accuracy using other methods. Although the study's results may not apply to all industries, there is a need to work on multi-label datasets covering a wide range of code smells; the researchers plan to conduct similar studies on real-world industrial initiatives.

## VI. CONCLUSION

This study aims to address the challenge of detecting multi-label code smells in method-level code by using a combination of machine learning techniques, including ensemble techniques. There are very few studies in this field, and they do not include ensemble techniques. The study used five single-label machine learning techniques and three multi-label classifiers to identify the Long method and Feature envy code smells. The dataset was normalized using the Z-score approach, and the chi-square feature selection technique was applied to select essential features. The proposed model was trained using 10-fold cross-validation, applied Random Search CV parameter tuning, and assessed using various metrics such as jaccard accuracy, accuracy, hamming loss, precision, recall, and f1 score.

The Random Forest method combined with Label Powerset and Chain Classifier produced the jaccard accuracy of 98.44%. The Extreme Gradient Boosting approach with the Chain Classifier resulted in a jaccard accuracy of 98.59%. The Decision Tree method combined with the Chain Classifier achieved an impressive jaccard accuracy of 99.54%, while the Chain Classifier with the Gradient Boosting approach achieved a jaccard accuracy of 99.39%. The Artificial Neural Network method combined with Label Powerset, and Chain Classifier approaches demonstrated a jaccard accuracy of 67.17%.

The study found that feature selection positively impacted Binary Relevance and Classifier Chain with the Random Forest model. After considering feature selection, the Artificial Neural Network model showed improved performance. Additionally, all single-label classifiers produced better results after considering the correlation factor. The Label Powerset and Classifier Chain multi-label classifiers achieved better jaccard accuracy than the Binary Relevance classifier. The DT model with a multi-label classifier has

proven more effective than other methods for multi-label classification.

The study's findings can help software practitioners accurately classify multi-label code smells. The importance of feature selection was emphasized in this research, which provided a framework for future software development and contributed to existing literature. For this experiment, a freely available dataset was used that is limited to only two code smells occurring at the method-level. These code smells were selected based on their correlation to ensure reliable results. Although expanding the dataset to include additional code smells presents challenges, future research could involve exploring other multi-label datasets to validate further and extend this study's findings. However, additional research and scientific experiments are required to evaluate the effectiveness of these classifiers. The researchers plan to incorporate alternative algorithms into future research initiatives to enhance and improve the outcomes.

## ACKNOWLEDGMENT

The authors would like to thank the editors and the anonymous reviewers whose insightful remarks and ideas have enhanced the quality of the study.

## REFERENCES

[1] E. O. Kiyak, D. Birant, and K. U. Birant, "Comparison of multi-label classification algorithms for code smell detection," in *Proc. 3rd Int. Symp. Multidisciplinary Stud. Innov. Technol. (ISMSIT)*, Oct. 2019, pp. 1–6, doi: 10.1109/ISMSIT.2019.8932855.

[2] P. Rai, A. Pradhan, M. Pradhan, A. Chettri, and B. Limboo, "Comparative study on various techniques used in examination system a survey," *Int. J. Sci. Res. Comput. Sci. Eng.*, vol. 7, no. 2, pp. 24–28, Apr. 2019, doi: 10.26438/ijsrcse/v7i2.2428.

[3] F. N. Colakoglu, A. Yazici, and A. Mishra, "Software product quality metrics: A systematic mapping study," *IEEE Access*, vol. 9, pp. 44647–44670, 2021, doi: 10.1109/ACCESS.2021.3054730.

[4] S. Vidal, H. Vazquez, J. A. Diaz-Pace, C. Marcos, A. Garcia, and W. Oizumi, "JSpIRIT: A flexible tool for the analysis of code smells," in *Proc. 34th Int. Conf. Chilean Comput. Sci. Soc. (SCCC)*, Nov. 2015, pp. 1–6, doi: 10.1109/SCCC.2015.7416572.

[5] U. Azadi, F. A. Fontana, and M. Zanoni, "Poster: Machine learning based code smell detection through WekaNose," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, Gothenburg, Sweden, May 2018, pp. 288–289.

[6] F. Arcelli Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," *Knowl.-Based Syst.*, vol. 128, pp. 43–58, Jul. 2017, doi: 10.1016/j.knosys.2017.04.014.

[7] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1143–1191, Jun. 2016, doi: 10.1007/s10664-015-9378-4.

[8] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: Are we there yet?" in *Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2018, pp. 612–621, doi: 10.1109/SANER.2018.8330266.

[9] M. Y. Mhawish and M. Gupta, "Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics," *J. Comput. Sci. Technol.*, vol. 35, no. 6, pp. 1428–1445, Nov. 2020, doi: 10.1007/s11390-020-0323-7.

[10] A. Alazba and H. Aljamaan, "Code smell detection using feature selection and stacking ensemble: An empirical investigation," *Inf. Softw. Technol.*, vol. 138, Oct. 2021, Art. no. 106648, doi: 10.1016/j.infsof.2021.106648.

[11] J. P. D. Reis, F. B. E. Abreu, and G. D. F. Carneiro, "Crowdsmelling: A preliminary study on using collective knowledge in code smells detection," *Empirical Softw. Eng.*, vol. 27, no. 3, pp. 1–35, May 2022, doi: 10.1007/s10664-021-10110-5.

[12] H. Aljamaan, "Voting heterogeneous ensemble for code smell detection," in *Proc. 20th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2021, pp. 897–902, doi: 10.1109/ICMLA52953.2021.00148.

[13] T. Guggulothu and S. A. Moiz, "Code smell detection using multi-label classification approach," *Softw. Quality J.*, vol. 28, no. 3, pp. 1063–1086, Sep. 2020, doi: 10.1007/s11219-020-09498-y.

[14] *A Modular Deep Learning Approach for Extreme Multi-Label Text Classification*. Accessed: Dec. 4, 2023. [Online]. Available: https://www.researchgate.net/publication/332932101_A_Modular_Deep_Learning_Approach_for_Extreme_Multi-label_Text_Classification

[15] C. Li, C. Liu, L. Duan, P. Gao, and K. Zheng, "Reconstruction regularized deep metric learning for multi-label image classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 7, pp. 2294–2303, Jul. 2020, doi: 10.1109/TNNLS.2019.2924023.

[16] Y. Feng, J. Jones, Z. Chen, and C. Fang, "An empirical study on software failure classification with multi-label and problem-transformation techniques," in *Proc. IEEE 11th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2018, pp. 320–330, doi: 10.1109/ICST.2018.00039.

[17] N. Maneerat and P. Muenchaisri, "Bad-smell prediction from software design model using machine learning techniques," in *Proc. 8th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, May 2011, pp. 331–336, doi: 10.1109/JCSSE.2011.5930143.

[18] F. A. Fontana, M. Zanoni, A. Marino, and M. V. Mäntylä, "Code smell detection: Towards a machine learning-based approach," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2013, pp. 396–399, doi: 10.1109/ICSM.2013.56.

[19] L. Amorim, E. Costa, N. Antunes, B. Fonseca, and M. Ribeiro, "Experience report: Evaluating the effectiveness of decision trees for detecting code smells," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 261–269, doi: 10.1109/ISSRE.2015.7381819.

[20] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2016, pp. 87–98.

[21] D. K. Kim, "Finding bad code smells with neural network models," *Int. J. Electr. Comput. Eng.*, vol. 7, no. 6, p. 3613, Dec. 2017, doi: 10.11591/ijece.v7i6.pp3613-3621.

[22] A. Kaur, S. Jain, and S. Goel, "A support vector machine based approach for code smell detection," in *Proc. Int. Conf. Mach. Learn. Data Sci. (MLDS)*, Dec. 2017, pp. 9–14, doi: 10.1109/MLDS.2017.8.

[23] H. Liu, Z. Xu, and Y. Zou, "Deep learning based feature envy detection," in *Proc. 33rd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2018, pp. 385–396, doi: 10.1145/3238147.3238166.

[24] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia, "Comparing heuristic and machine learning approaches for metric-based code smell detection," in *Proc. IEEE/ACM 27th Int. Conf. Program Comprehension (ICPC)*, May 2019, pp. 93–104, doi: 10.1109/ICPC.2019.00023.

[25] A. Jesudoss, S. Maneesha, and T. Lakshmi naga durga, "Identification of code smell using machine learning," in *Proc. Int. Conf. Intell. Comput. Control Syst. (ICCS)*, May 2019, pp. 54–58, doi: 10.1109/ICCS45141.2019.9065317.

[26] T. Guggulothu and S. A. Moiz, "Detection of shotgun surgery and message chain code smells using machine learning techniques," *Int. J. Rough Sets Data Anal.*, vol. 6, no. 2, pp. 34–50, Apr. 2019, doi: 10.4018/ijrsda.2019040103.

[27] M. Y. Mhawish and M. Gupta, "Generating code-smell prediction rules using decision tree algorithm and software metrics," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 5, pp. 41–48, May 2019, doi: 10.26438/ijcse/v7i5.4148.

[28] H. Gupta, L. Kumar, and L. B. M. Neti, "An empirical framework for code smell prediction using extreme learning machine," in *Proc. 9th Annu. Inf. Technol., Electromech. Eng. Microelectron. Conf. (IEMECON)*, Mar. 2019, pp. 189–195, doi: 10.1109/IEMECONX.2019.8877082.

[29] S. Dewangan, R. S. Rao, A. Mishra, and M. Gupta, "A novel approach for code smell detection: An empirical study," *IEEE Access*, vol. 9, pp. 162869–162883, 2021, doi: 10.1109/ACCESS.2021.3133810.

[30] A. Barbez, F. Khomh, and Y.-G. Guéhéneuc, "A machine-learning based ensemble method for anti-patterns detection," *J. Syst. Softw.*, vol. 161, Mar. 2020, Art. no. 110486, doi: 10.1016/j.jss.2019.110486.

[31] I. Kaur and A. Kaur, "A novel four-way approach designed with ensemble feature selection for code smell detection," *IEEE Access*, vol. 9, pp. 8695–8707, 2021, doi: 10.1109/ACCESS.2021.3049823.

[32] M. M. Draz, M. S. Farhan, S. N. Abdulkader, and M. G. Gafar, "Code smell detection using whale optimization algorithm," *Comput., Mater. Continua*, vol. 68, no. 2, pp. 1919–1935, 2021, doi: 10.32604/cmc.2021.015586.

[33] H. Gupta, T. G. Kulkarni, L. Kumar, L. B. M. Neti, and A. Krishna, "An empirical study on predictability of software code smell using deep learning models," in *Proc. Int. Conf. Adv. Inf. Netw. Appl.*, in Lecture Notes in Networks and Systems, vol. 226, 2021, pp. 120–132, doi: 10.1007/978-3-030-75075-6_10.
[34] N. A. A. Khleel and K. Nehéz, "Deep convolutional neural network model for bad code smells detection based on oversampling method," *Indonesian J. Electr. Eng. Comput. Sci.*, vol. 26, no. 3, p. 1725, Jun. 2022, doi: 10.11591/ijeecs.v26.i3.pp1725-1735.
[35] A. Abdou and N. Darwish, "Severity classification of software code smells using machine learning techniques: A comparative study," *J. Softw., Evol. Process*, vol. 36, no. 1, p. e2454, Jan. 2024, doi: 10.1002/smr.2454.
[36] S. Dewangan, R. S. Rao, S. R. Chowdhuri, and M. Gupta, "Severity classification of code smells using machine-learning methods," *Social Netw. Comput. Sci.*, vol. 4, no. 5, pp. 1–20, Jul. 2023, doi: 10.1007/s42979-023-01979-8.
[37] R. S. Rao, S. Dewangan, A. Mishra, and M. Gupta, "A study of dealing class imbalance problem with machine learning methods for code smell severity detection using PCA-based feature selection technique," *Sci. Rep.*, vol. 13, no. 1, pp. 1–18, Sep. 2023, doi: 10.1038/s41598-023-43380-8.
[38] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," *Neurocomputing*, vol. 163, pp. 3–16, Sep. 2015, doi: 10.1016/j.neucom.2014.08.091.
[39] T. Tanaka, I. Nambu, Y. Maruyama, and Y. Wada, "Sliding-window normalization to improve the performance of machine-learning models for real-time motion prediction using electromyography," *Sensors*, vol. 22, no. 13, p. 5005, Jul. 2022, doi: 10.3390/s22135005.
[40] Y. Wang, Z. Pan, J. Zheng, L. Qian, and M. Li, "A hybrid ensemble method for pulsar candidate classification," *Astrophys. Space Sci.*, vol. 364, no. 8, pp. 1–13, Aug. 2019, doi: 10.1007/s10509-019-3602-4.
[41] L. Yu and A. Mishra, "Experience in predicting fault-prone software modules using complexity metrics," *Qual. Technol. Quant. Manage.*, vol. 9, no. 4, pp. 421–434, Jan. 2012, doi: 10.1080/16843703.2012.11673302.

**PRAVIN SINGH YADAV** received the M.C.A. degree from the Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur, Chhattisgarh, India, in 2013. He is currently a Research Scholar with the Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya. His research interests include code smell prediction and machine learning.

**RAJWANT SINGH RAO** received the Ph.D. degree from the Department of Computer Science, Institute of Science, Banaras Hindu University, Varanasi, India. He is currently an Assistant Professor with the Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya (a central university), Bilaspur, Chhattisgarh, India. His research interests include design patterns mining, code smell detection, and machine learning.

**ALOK MISHRA** (Senior Member, IEEE) is currently a Professor in data management and software engineering with Norwegian University of Science and Technology (NTNU), Norway. His research interests include software engineering, artificial intelligence, and cyber security. He is actively involved in editing special issues of reputed journals in his areas of research interest. In teaching, he has received Excellence in Online Education Award by U21Global Singapore, while in research, he has been awarded by Scientific and Research Council of Turkey and Board of Management of University for outstanding publications in science and social science citation indexed (Thomson Reuter) journals. He was a recipient of many scholarships, international awards, and research projects. He is an Editorial Board Member of many reputed journals, including *Computer Standards & Interfaces* (Elsevier), *ICT Express*, *Software Impacts*, and *Data Technologies and Applications*.