



Severity Classification of Code Smells Using Machine-Learning Methods

Seema Dewangan¹ · Rajwant Singh Rao¹ · Sripriya Roy Chowdhuri² · Manjari Gupta²

Received: 27 February 2023 / Accepted: 30 May 2023
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2023

Abstract

Code smell detection can be very useful for minimizing maintenance costs and improving software quality. Code smells help developers/programmers, researchers to subjectively interpret design defects in different ways. Code smells instances can have varied size, intensity or severity which needs to be focused upon as they affect the software quality accordingly. Therefore, this study aims to detect the severity of code smells from code smell datasets. The severity of code smells is significant for reporting code smell detection performance, as it permits refactoring efforts to be prioritized. Code smell severity also describes extent of effort required during software maintenance. In our work, we have considered four code smells severity datasets to detect the severity of code smell. These datasets are data class, god class, feature envy and long method code smells. This paper uses four machine-learning and three ensemble learning approaches to identify the severity of code smells. To improve the models' performance, we used fivefold cross-validation method: Chi-square-based feature selection algorithm and parameter optimization techniques. We applied two-parameter optimization techniques, namely grid search and random search and also compared their accuracy. The conclusion of this study is that the XG Boost model obtained an accuracy of 99.12%, using the Chi-square-based feature selection technique for the long method code smell dataset. In this study, the results show that ensemble learning is best as compared to machine learning for severity detection of code smells.

Keywords Severity of code smells · Code smells · Machine learning · Feature selection · Parameter optimization

Introduction

Code smells indicate issues with software design. In addition to making the code more difficult to comprehend, a code smell may make modifications and mistake proneness more likely. Software engineers may learn the code more efficiently than ever by identifying and removing code smells from the program.

The software is getting more and more complicated because there are more and bigger modules, more complicated requirements, and code smells, among other things. Challenging requirements are complex to assess and comprehend, making development difficult and beyond the scope of developers, but code smells may be recognized and refactored to make the software simpler, straightforward, and easier to produce and maintain [1]. The software engineering principles are required to develop better quality software [2]. In general, developers concentrate on functional requirements and overlook nonfunctional requirements such as maintainability, development, verifiability, reprocessing ability, and comprehensibility [3].

The severity of code smells is a significant consideration when reporting outcome of code smell detection, as it permits refactoring efforts to be prioritized. High-severity code smells can become a significant and complex problem for software's maintainability process. Thus, detecting code smells as well as their severity are very useful for software developers to minimize maintenance costs and improve software quality [4].

This article is part of the topical collection "Research Trends in Computational Intelligence" guest edited by Anshul Verma, Pradeepika Verma, Vivek Kumar Singh and S. Karthikeyan.

✉ Rajwant Singh Rao
rajwantrao@gmail.com

- ¹ Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur, Chhattisgarh, India
- ² Computer Science, DST-Centre for Interdisciplinary Mathematical Sciences, Institute of Science, Banaras Hindu University, Varanasi, India

The purpose of the research is to detect the severity of code smells to help the software developers minimize the maintenance charges and improve the software quality.

In the literature, a number of code smell severity detection techniques have been developed [5–8]. Each technique gives different outcomes because smells can be interpreted subjectively and, therefore, can be described in various ways [4]. To the best of our knowledge and available literature, only Fontana et al. [4] and Abdou [27] have found the code smell severity on the code smell dataset. They used ordinal and multinomial machine-learning algorithms (MLA) for code smell severity detection. In addition, they used ranking-based correlation to find the best algorithm. The Fontana et al. [4] and Abdou [27] approach has the following limitations: they have not presented class-wise accuracy in their studies. They did not consider other performance metrics such as precision, recall, and *F*-measure. They used different MLA; the ensemble learning methods were not applied.

This study hypothesizes detecting the severity of code smells using machine-learning and ensemble learning methods and presenting class-wise outcomes.

Contributions: In this work, we have applied seven MLA/ensemble learning models (logistic regression (LR), Random Forest (RF), KNN, decision tree (DT), AdaBoost, XGBoost, and Gradient Boost) with the Chi-square feature selection approach on each dataset. In addition, we applied grid search and random search-based parameter optimization techniques (POT) to see the effect of parameters optimization on the classification results of code smell severity detection. We achieved the highest severity classification accuracy (SCA), 99.12%, in the XG Boost model for the LM dataset.

The advantage of this research is to detect the severity from the code smell severity dataset to make our code or source code more effective, accurate, and easily understandable by the programmer and users.

The following is the outline of this paper: the next section explains related works. The third section describes the dataset's structure and proposed models. The fourth section describes the experimental results of our proposed models. The fifth section discusses our results and compares these with baseline results and the final section concludes the work.

Related Work

Various machine learning-based algorithms and techniques have been used by the researchers for code smell detection and also for code smell severity classification. The related work section is divided in two parts: the first part discusses the research works done for code smell detection using machine-learning techniques, and the second part discusses research works related to the severity of code smells.

Machine-Learning Techniques for Code Smell Detection

Fontana et al. [9] proposed a comparison-based observation among 16 MLAs on 4 code smell datasets from 74 java systems with manually validated examples on the training dataset for code smell detection. In addition, boosting techniques were used for four code smells datasets.

Mhawish et al. [10] suggested an MLA for detecting code smells in source code. They used the two-feature selection approach based on the genetic algorithm (GA) and a POT based on a grid search. Using the GA_CFS approach, they obtained the highest accuracy in the data class (DC), god class (GC), and long method (LM) smells by 98.05%, 97.56%, and 94.31%, respectively. They also obtained the highest accuracy of 98.38% in the LM using the GA-Naïve Bayes feature selection method. Mhawish et al. [11] presented code smell testing of prediction with the help of MLA and software metrics. They also used feature selection methods based on GA to increase the efficiency of these MLA by identifying the appropriate features in every dataset. Furthermore, they used the grid search algorithm based on POT to improve the performance of the approaches. The RF model obtains the highest accuracy of 99.71% and 99.70% in predicting the DC in the original refined datasets.

Kaur et al. [12] proposed a correlation metrics selection strategy and an ensemble learning method for detecting code smells in three publicly available Java datasets. They used bagging and RF classifier to examine each method with four occurrence measures accuracy (P1), *G*-mean 1 (P2), *G*-meam2 (P3), and *F*-measure (P4). Pushpalatha et al. [13] suggested that the severity of bug reports for closed-source datasets could be predicted. For this, they used the PROMISE Repository to get the dataset (PITS) for NASA projects. They improved the accuracy by employing ensemble learning strategies and two-dimensional reduction methods, such as Chi-square analysis and information gain, respectively. Alazba et al. [14] proposed 14 MLA and stacking ensemble learning algorithms on the six code smell datasets. The results of MLA were compared, and they found that the best accuracy was 99.24% applying the Stack-SVM algorithm for the LM Dataset. A search-based technique to enhance the code smell detection using Whale optimization method as a classifier was proposed by Draz et al. [15]. They researched five open-source software projects that detected the nine code smell types and established an average of 94.24% precision and 93.4% recall.

Dewangan et al. [16] presented six MLAs, two Chi-square, and Wrapper-based FSTs to pick the significant

feature from each dataset; the grid search technique was then applied to improve the model's outcome and achieved 100% accuracy using the LR model for the LM dataset. Reis et al. [17] proposed a Crowd smelling approach using collective knowledge in code smells detection with the LM, GC, and Feature envy (FE) datasets. They applied six MLA to detect the code smells. They obtained the highest outcome of 0.896% ROC using the Naive Bayes algorithm for GC detection and 0.870% ROC using AdaBoostM1 algorithm for LM detection. The worst performance was 0.570% using the RF algorithm for FE detection. Oort et al. [18, 19] presented a study to examine the occurrence of code smells in machine-learning projects. They collected the 74 machine-learning projects, and then applied the Pylint tool to those projects. After this, they assembled the delivery of Pylint messages per group per project, the best 10 code smells in these projects in general, and the best 20 code smells per group. They originate that the PEP8 rule for the identifier naming method may not forever be appropriate in machine-learning code due to its similarity with mathematical notation. They also detected severe problems with the measurement of needs that present the main threat to the maintainability and reproducibility of Python machine-learning projects.

Boutaib et al. [20] proposed a bi-level multi-label detection of smells (BMLDS) tool to reduce the population of classification series for detecting multi-label smells. They implemented a bi-level scheme in which the higher-level part is to discover the best classification for each measured series, and the lower level part is to construct the series.

Abdou et al. [21] proposed three ensemble methods (bagging, boosting, and rotation forest) with a resample technique to detect the software defects. They used seven code smell datasets given in the PROMISE repository. They found that the ensemble method gives better accuracy as compared to single learning methods. They found the 93.40% highest accuracy using Random Forest with resample technique model for KC1 dataset.

Dewangan et al. [22] introduced ensemble and deep learning algorithms as a method for identifying the code smell. They applied Chi-square-based FST to pick the significant features from each code smell dataset and a SMOTE technique is used to balance the dataset. They were able to achieve an accuracy of 100% by utilizing all of the ensemble approaches for the LM dataset.

Dewangan et al. [23] proposed five classification models to detect the code smell. They used four code smell datasets DC, GC, FE, and LM. They obtained 0.9912% best accuracy using Random Forest model for FE dataset.

Dewangan et al. [24] proposed three ML algorithms to detect the code smell. A principle component analysis (PCA)-based FST is used to pick the significant features from each code smell dataset. They obtained 99.97% best

accuracy using principal component analysis-based logistic regression (PCA_LR) model for DC dataset.

There is a notable difference between these works and the technique we used in this paper. The majority of the above studies are focused on the identification of code smells as described by Flower et al. [25]. Most of the previous research papers have examined only a few systems and applied the MLAs. However, they have not mentioned the aspect of severity of code smells in their work.

Machine-Learning Techniques for Code Smell Severity Detection

Vidal et al. [5] proposed a tool for detecting code smells using textual analysis. For this purpose, they conducted two separate experiments. They started by performing a software repository mining study to examine how engineers spotted code smells through textual or structural cues. They then carried out a user research with industrial developers and quality experts to qualitatively examine how they examined the detection of code smells using two different sources of details. They discovered that textual code smells are simpler to pick up.

Liu et al. [6] propose severity prediction of bug reports based on feature selection methods. They establish a ranking-based policy to enhance existing feature selection methods and intend an ensemble learning feature selection method by merging existing ones. They applied eight-feature selection methods. They found that the ranking-based strategy gets the highest *F1* score, 54.76%. Tiwari et al. [7] proposed a method to find the LM and their severity that shows the importance of refactoring LMs. They find that it matches the expert's severity evaluation for half the approaches within a tolerance level one. In addition, they identified a high severity; this evaluation is more or less equivalent to an expert's judgment.

MLAs were suggested by Baarah et al. [8] for closed-source software bug severity detection. They assembled a dataset from past bug information stored in the JIRA bug tracking system connected to different closed-source projects built by INTIX Company in Amman, Jordan. They evaluate eight MLAs such as Naive Bayes, Naive Bayes Multinomial, SVM, Decision Rules (JRip), Logistic Model Trees, DT (J48), RF, and KNN in the measurement of accuracy, *F*-measure, and area under the curve (AUC). The DT model obtained the highest performance accuracy, AUC, and *F*-measure with 86.31%, 90%, and 91%, respectively.

Gupta et al. [26] proposed a hybrid method to examine the severity of the code smell intensity in the Kotlin language and found which code smells are equivalent in the Java language. They used five types of code smells to examine the research work: complex method, long parameter list, large class, LM, and string literal duplication.

They applied various MLAs, where they found the JRip algorithm achieved the best performance with 96% precision and 97% accuracy. Abdou et al. [27] proposed the classification of code smell severity using MLAs based on regression, multinomial, and ordinal classifiers. In addition, they applied the local interpretable model agnostic explanations (LIME) approach to explaining the MLAs and prediction rules produced by the PART algorithm to find the efficiency of the feature. They employed the LIME algorithm to help us gain a deeper knowledge of the model's decision-making process and the characteristics that affect the model's decision. They found the highest accuracy of 92–97% with correlation measurement of the Spearman algorithm. Hejres et al. [28] proposed the detection of code smell severity using MLAs. They applied three models, sequential minimal optimization (SMO), artificial neural network (ANN), and J48, to detect the code smell severity from four datasets. They obtained the best result for GC and FE datasets using the SMO model, while the LM dataset obtained the best accuracy using adaptive neural network ensemble (ANNE) with the SMO model.

Nanda et al. [29] proposed a combination of SMOTE and Stacking model to classify the severity of GC, DC, FE, and LM datasets. They improved their performance from 76 to 92%. Fontana et al. [4] proposed MLA for classifying the severity of code smell. They implemented different MLAs, spanning from multinomial classification to regression and a binary classifier for ordinal classification. They found the

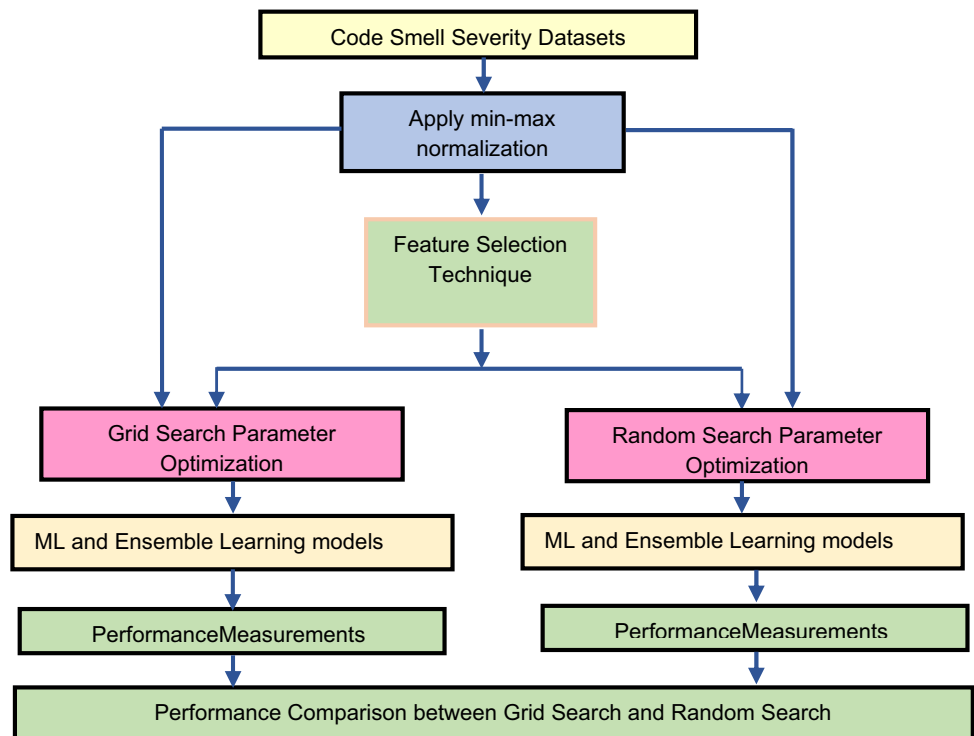
correspondence between the actual and predicted severity for the top techniques and achieved 88–96%, calculated by Spearman's p .

To the best of our knowledge and available literature, it is observed that most of the authors used different code smell datasets and code smell severity datasets using different types of MLAs, multinomial, and regression techniques to find the code smell and severity of code smell from datasets. It has been observed that the effect of grid search and ensemble learning algorithms on the severity datasets has not been applied earlier. Therefore, to study and analyze the effect of MLA and ensemble learning approaches, we have used MLAs and ensemble learning methods with grid search, random search, and Chi-square feature selection techniques to find the severity of code smell from the code smell severity datasets.

Proposed Model and Dataset Description

The severity of code smells is significant to study when describing code smell detection results since it categorizes refactoring efforts. This research work builds a model for detecting the severity of code smell using machine-learning methods. A step-by-step research framework designed for code smell severity detection is shown in Fig. 1. First, we collected the code smell severity datasets from Fontana et al. [4]. Then, we applied the min–max normalization technique to find the various values in the datasets. After that, we used

Fig. 1 Proposed model



the Chi-square feature selection algorithm to extract the best features from the datasets. Then, we applied grid search and random search POTs. Then, we applied MLAs with fivefold cross-validation, and finally, we obtained the performance measurements.

The following research queries are resolved in this paper.

RQ1: Which MLA/ensemble learning algorithm detects code smell severity best?

Motivation: Baarah et al. [8] and Fontana et al. [4] proposed various MLAs, such as Naive Bayes, Naive Bayes Multinomial, SVM, and Decision Rules (JRip), and also multinomial classification, regression, etc. Alazba et al. [14] applied both MLA and stacking ensemble learning algorithms and compared the performances of MLAs and ensemble learning algorithms. They discovered that ensemble learning algorithms were more accurate than MLAs in terms of performance. Therefore, to investigate and observe the effect of MLA and ensemble learning algorithms to code smell severity detection, we applied both MLA and ensemble learning algorithms.

RQ2: What is the effect of applying the feature selection method in code smell severity detection?

Motivation: Liu et al. [6], Mhawish et al. [10, 11], Kaur et al. [12], and Dewangan et al. [16] introduced the influence of various feature selection algorithms on the performance measures. They found an enhancement in the performance accuracy by applying feature selection methods. Therefore, to examine the effect of the feature selection method on the method's accuracy and extract software severities that play an essential task in the code smell severity detection process, we used Chi-square-based feature selection method for our work.

RQ3: Does the hyper-parameter optimization algorithm enhance the performance of the detection of the code smell's severity?

Motivation: Mhawish et al. [10, 11] applied grid search-based parameter optimization and found improvement in their results. Therefore, to study and analyze more effectively, tuning in MLA and ensemble learning algorithm parameters in this work have been applied.

Dataset Description

In this study of severity detection of code smell, we have taken four datasets from Fontana et al. [4], which consist of two class level datasets named data class (DC) and god class (GC) and two method level datasets named feature envy (FE) and long method (LM). These datasets can all be found at <http://essere.disco.unimib.it/reverse/MLCSD.html>. Fontana et al. [4] have selected 76 systems out of 111, computed by different sizes and a large set of object-oriented features. They considered the Qualitas Corpus of systems collected by Tempero et al. [30] for the system selection.

Table 1 Automatic detector tools (advisors)

Code smell	Reference, tool/detection rules
DC	iPlasma, Fluid Tool [33], Anti-pattern Scanner [31]
GC	iPlasma (GC, Brain Class), PMD [32]
FE	iPlasma, Fluid Tool [33]
LM	iPlasma (Brain Method), PMD, Marinescu detection rule [34]

Table 2 Dataset configuration [4]

Code smell severity datasets	Severity			
	1	2	3	4
DC	151	32	113	124
GC	154	29	110	127
FE	280	23	95	22
LM	280	11	95	34

They employed a variety of tools and methods to find code smell severity called advisors: iPlasma (GC, Brain Class), Anti-pattern Scanner [31], PMD [32], iPlasma, Fluid Tool [33] and Marinescu detection rules [34]. Table 1 shows the automatic detection tools.

Code Smells Classification of Severity

After manual assessment of code smells, a severity value between 1 and 4 is allocated to each assessed occurrence. The values with their meanings are describes as follows:

1. Allocated for no smell: a method or class that is unaffected by the code smell.
2. Allocated for non-severe smell: a method or class that is only slightly impacted by the code smell.
3. Allocated for smell: the class or method possesses all the properties of a smell.
4. Allocated for severe smell: there is a smell that is extremely strong in terms of size, complexity, or coupling.

Each code smell dataset has 420 instances (class or method), where 63 instances are selected for DC and GC datasets, and 84 instances are selected for FE and LM datasets. Details as shown by Fontana et al. [4] are given in Table 2.

The code smell datasets used in this paper are defined as follows:

Data class (DC): It mentions classes that maintain data with basic functionality and are heavily relied upon by other classes. A DC reveals a lot of features; it is not complicated and data are exposed via accessor methods [4].

God class (GC): It describes those classes that concentrate on the system's intelligence. The GC is assumed to be the utmost convoluted code smells for many causes, actions, and tasks that arise there. It causes problems with big-size code, coupling, and complexity [4].

Feature envy (FE): It mentions techniques that make extensive use of data from classes other than their own. It prefers to use other classes' features, considering features entered through accessor methods [4].

Long method (LM): It mentions techniques that prefer to centralize a class's functionality. An LM has a lot of code, is complicated, difficult to understand, and heavily relies on data from other classes [4].

Dataset Composition (Structure)

There are 420 manually calculated classes or methods in each dataset. Table 2 shows the dataset configuration, including the number of instances distributed to each severity level. The least repeated severity level in the datasets is 2, and the two class-based smells (DC and GC) have a dissimilar balance to the two method-based smells (FE and LM) with respect to severity levels 1 and 4 [4].

Normalization Technique

The datasets have a wide range of features; therefore, in such a case the features should be normalized before MLAs are used. The min-max feature scaling strategy was used in this paper to rescale the datasets' range of feature or observable values between 0 and 1 [35]. The min-max formula is shown in Eq. 1 with X' denoting the normalized value and X representing the initial real value. The feature's X_{min} and X_{max} values are altered to "0" and "1", respectively, while every other value is modified to a decimal between "0" and "1":

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

Feature Selection Algorithm

Feature selection is used to find the most significant feature in a dataset to refine results by improving the understanding of the instances that contribute in making a distinction among parallel roles in features [36]. We used a Chi-square-based feature selection technique in this study to extract the best instances from each dataset.

Typically, Chi-square feature selection is used in categorical datasets. By examining the relationships between features, Chi-square aids in choosing the optimal features. Equation 2 provides the formula for the Chi-square test [36]:

$$\chi^2 = \frac{(\text{Observed frequency} - \text{Expected frequency})^2}{\text{Expected frequency}} \quad (2)$$

We can calculate observed frequency and expected frequency for response and independent variables. The Chi-square calculates the difference between these two numbers. The more the difference, the more the response, and variables that are not related to each other are dependent.

The top 10 instances were extracted from each dataset. Table 3 shows these instances extracted by the Chi-square feature selection method from each dataset. The detailed explanation of selected metrics is described in Table 21 (Appendix section).

Hyper-parameter Tuning/Optimization

A hyper-parameter is a parameter value that controls the learning process. In machine-learning and ensemble learning methods, hyper-parameter optimization or tuning is used to select the best parameters for each algorithm. The best parameters for every method differ based on the learning dataset. The various parameter value combinations for each algorithm should be tried in order to find the exact parameters that will allow the predictive algorithm to successfully forecast the test dataset [11].

In this study, the grid search and random search-based POTs are applied to select the optimal parameter values for every method. Parameter optimization is necessary to

Table 3 Chi-square feature selection method's extracted feature

Dataset	No. of feature	Feature extracted by Chi-square
DC	10	LOC_project, LOC_package, NOMNAMM_project, LOC_type, LOCNAMM_type, NOCS_project, NOMNAMM_package, NOCS_package, Complextype, NIM_type
GC	10	LOC_project, LOC_package, NOMNAMM_project, LOC_type, LOCNAMM_type, NOCS_project, NOMNAMM_package, NOCS_package, Complextype, NOMNAMM_type
FE	10	Method, ATFD_type, Project, AMW_type, package, AMWNAMM_type, complextype, CBO_type, LOC_method, NOAV_method
LM	10	Method, ATFD_type, project, package, complextype, LOC_method, CBO_type, NOAV_method, CYCLO_method, CINT_method

select an algorithm’s optimal hyper parameters, which help outcomes for the utmost correct calculations. It depends on an inclusive search for the number of criteria that yields the best prediction model performance value [38]. To avoid over fitting the algorithm on the test dataset, a fivefold cross-validation is performed. It is applied to determine the efficiency of the procedure for every conceivable parameter combination.

We identified grid search and random search algorithm for each machine-learning and ensemble learning algorithms. We allowed the value range and several steps to the numeric parameters. As indicated in Table 5, the numbers which must be checked are allocated inside the range’s upper and lower boundaries according to the given step count allocated to every parameter. The best-selected parameters and allotted step count for nominal parameters for the grid search and random search are shown in Tables 4 and 5.

Table 4 Selected best parameters for each model

Model	Parameter	Parameter option
DT	Criteria	Gini index, gain ratio, information gain, accuracy
Gradient Boost	Criteria	friedman_mse, learning rate, N_estimators, max depth

Table 5 Number of allotted steps to every parameter (parameter tuning)

Algorithm	Parameters	Start value	End value	No. of steps
LR	C	0.1	100	100
	Penalty	11	12	N/A
RF	Number of trees	1	50	15
	Maximum depth	1	50	15
DT	Maximum depth	1	20	15
	Bootstrap	True/false		
	Minimum size of split	1	15	15
	Minimum gain	0	10	10
AdaBoost	N_estimators	10	1000	100
	Learning rate	0.1	100	100
	Base_estimator	None		
XG Boost	N_estimator	1	100	100
	Learning rate	0.01	100	100
	N_estimator	100	1000	100
Gradient Boost	Objective	Multi:softmax		
	Learning rate	0.01	100	100
	Max depth	1	10	10
	N_estimators	100	1000	100

Validation Methodology

This study has used a validation methodology to calculate each experiment’s performance. MLAs are calculated using fivefold cross-validation to divide the datasets into five segments, five times for the training of the algorithm. In each replication dataset, one part is evaluated as the test set, and the other is evaluated for the training set.

Performance Capacity

To evaluate the performance capacity of all the models, a set of experimental measurements such as true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are computed with the confusion matrix. In this study, we have evaluated various experiments.

- TP (true positive) stands for the outcomes in which the system correctly calculates the positive class.
- TN (true negative) stands for the outcomes in which the system correctly calculates the negative class.
- FP (false positive) stands for the outcomes in which the system calculates positive class but the class is actually a negative class. This means the system wrongly identifies a positive class.
- FN (false negative) stands for the outcomes where the system calculates the negative class which is actually a positive class which means that system wrongly identifies the negative class.

To compute the performance capacity of MLAs, four parameters: positive predictive value, true positive rate, *F*-measure, and SCA, are calculated using TP, TN, FP, and FN. The detailed description and equations of all parameters are given as follows:

Positive predictive value (PPV): PPV deals with the number of code smell instances precisely recognized by the MLA. Equation (3) is applied to calculate the PPV:

$$PPV = \frac{TP}{TP + FP} \quad (3)$$

True positive rate (TPR): TPR deals with the number of code smell instances precisely recognized by the MLA. Equation (4) is applied to compute TPR:

$$TPR = \frac{TP}{TP + FN} \quad (4)$$

***F*-measure:** The harmonic mean of PPV and TPR is the subject of *F*-measure, which is calculated for a balance between their respective values. The *F*-measure is calculated using Eq. (5):

$$F - \text{measure}(F) = 2 \times \frac{PPV \times TPR}{PPV + TPR} \quad (5)$$

Severity classification's accuracy (SCA): SCA deals with the organization of PPV and TPR. It depicts the exact classification of cases in the positive and negative classifications. Equation (6) is applied to calculate SCA:

$$SCA = \frac{TP + TN}{TP + TN + FP + FN} \quad (6)$$

Results of Proposed Model

In order to provide a response to RQ1, we made use of a total of three ensemble learning methods and four MLAs. In this paper, we have used four code smell severity datasets (DC, GC, FE, and LM), and each dataset has four types of severities. We have shown individual results for each severity of each dataset. In addition, the average result of all severity classes is also shown. We implemented the severity classification accuracy (SCA) in two ways: (1) SCA with the grid search algorithm and (2) SCA with the random search algorithm. The following subsection shows the obtained results from every MLA and ensemble learning algorithm in tabular form.

The following subsection uses seven MLAs and ensemble learning methods: logistic regression, Random Forest, *K*-nearest neighbor, decision tree, AdaBoost, XG Boosting, and Gradient Boosting. The following “Logistic Regression”–“Gradient Boosting (GB Algorithm)” shows

the experimental results of the MLA and ensemble learning methods in table form.

Logistic Regression

Logistic regression (LR) is a classification algorithm that simplifies LR to multi-class problems, i.e., greater than two promising distinct results. It is a technique for calculating the possibility of the unusual feasible results of classically dispersed associated variables, specified a set of particular variables [38].

Table 6 presents the SCA of each severity class and the average SCA of all severity classes for each dataset using the LR method. This study observed that the LR algorithm got the highest SCA, 97.34% using the grid search algorithm for the LM dataset for an average of all severities, and 91.42% SCA obtained using the Random search algorithm for the LM dataset for the average of all severities.

Random Forest

Random Forest (RF) is a classification method applied in regression and ensemble learning methods. When used for the MLA, the outcome of the RF is the class selected by a large number of trees.

Table 7 shows the RF algorithm got the highest SCA, 97.34% using the grid search algorithm for the LM dataset for an average of all severities, and 96.20% SCA obtained using the random search algorithm for the LM dataset for the average of all severities.

K-Nearest Neighbor

The *K*-nearest neighbor (KNN) algorithm is an MLA used to anticipate classification and regression issues. It is, however, generally applied for classification prediction problems. The KNN method assigns a value to a new data point based on how similar it is to the training point [40].

Table 8 presents that the KNN algorithm got the highest SCA, 90.95% using the grid search algorithm for the LM dataset for an average of all severities, and 91.64% SCA obtained using the Random search algorithm for the LM dataset for the average of all severities.

Decision Tree

A decision tree (DT) constructs the classification techniques in the structure of a tree. It divides a dataset into smaller subsets while, at the same time, a connected DT is incrementally developed. The last outcome is a tree with leaf nodes or results. The leaf node corresponds to a classification [39].

Table 9 presents that the DT algorithm got the highest SCA, 99.08% using the grid search algorithm for the LM

Table 6 Severity classification results of LR algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	85.00	85.00	85.00	84.62	37.50
	Severity 2	00	00	00	66.67	50.00
	Severity 3	47.00	58.00	52.00	58.34	57.20
	Severity 4	57.00	73.00	64.00	72.73	46.20
	Average result of all class	54.00	62.00	58.00	71.70	68.09
GC	Severity 1	93.00	88.00	90.00	87.50	94.12
	Severity 2	00	00	00	100	66.67
	Severity 3	62.00	67.00	64.00	66.67	63.64
	Severity 4	64.00	69.00	67.00	69.24	54.55
	Average result of all class	73.00	74.00	73.00	70.11	67.20
FE	Severity 1	83.00	100	91.00	100	68.75
	Severity 2	00	00	00	60.00	100
	Severity 3	73.00	73.00	73.00	72.73	57.20
	Severity 4	00	00	00	100	100
	Average result of all class	69.00	79.00	73.00	88.09	84.05
LM	Severity 1	100	100	100	100	75.00
	Severity 2	100	100	100	100	50.00
	Severity 3	80.00	100	89.00	100	90.00
	Severity 4	100	50.00	67.00	50.00	100
	Average result of all class	96.00	95.00	96.00	97.34	91.42

Table 7 Severity classification results of RF algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	93.00	82.00	87.00	87.00	82.36
	Severity 2	00	00	00	66.67	100
	Severity 3	40.00	67.00	50.00	85.00	66.67
	Severity 4	73.00	73.00	73.00	79.20	72.73
	Average result of all class	65.00	67.00	65.00	76.49	70.47
GC	Severity 1	93.00	93.00	93.00	93.34	93.34
	Severity 2	00	00	00	100	50.00
	Severity 3	69.00	100	82.00	88.89	100
	Severity 4	100	88.00	93.00	75.00	87.5
	Average result of all class	86.00	88.00	86.00	76.98	78.06
FE	Severity 1	100	100	100	100	100
	Severity 2	00	00	00	100	100
	Severity 3	80.00	100	89.00	100	100
	Severity 4	100	33.00	50.00	66.67	66.67
	Average result of all class	92.00	93.00	91.00	89.42	91.43
LM	Severity 1	100	100	100	100	100
	Severity 2	100	100	100	50.00	100
	Severity 3	100	90.00	95.00	95.24	90.00
	Severity 4	67.00	100	80.00	50.00	100
	Average result of all class	98.00	98.00	98.00	97.34	96.20

Table 8 Severity classification results of KNN algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	75.00	86.00	80.00	84.62	85.72
	Severity 2	00	00	00	50.00	66.67
	Severity 3	50.00	73.00	59.00	50.00	72.73
	Severity 4	100	71.00	33.00	61.54	71.43
	Average result of all class	71.00	71.00	70.00	71.62	70.82
GC	Severity 1	67.00	86.00	75.00	85.72	83.34
	Severity 2	00	00	00	57.14	100
	Severity 3	32.00	55.00	40.00	54.54	70.00
	Severity 4	60.00	30.00	40.00	70.00	63.16
	Average result of all class	45.00	50.00	45.00	64.72	69.05
FE	Severity 1	82.00	98.00	89.00	98.00	92.00
	Severity 2	50.00	25.00	33.00	75.00	78.22
	Severity 3	68.00	65.00	67.00	65.22	67.98
	Severity 4	00	00	00	85.72	82.56
	Average result of all class	70.00	77.00	73.00	68.34	71.57
LM	Severity 1	96.00	96.00	96.00	96.15	100
	Severity 2	00	00	00	100	92.67
	Severity 3	67.00	100	80.00	100	80.00
	Severity 4	100	20.00	33.00	80.00	100
	Average result of all class	87.00	86.00	83.00	90.95	91.64

Table 9 Severity classification results of DT algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	76.00	97.00	85.00	90.63	96.67
	Severity 2	00	00	00	100	100
	Severity 3	85.00	46.00	59.00	70.37	45.84
	severity 4	70.00	92.00	79.00	66.67	92.00
	Average result of all class	72.00	75.00	71.00	73.80	75.00
GC	Severity 1	92.00	100	96.00	95.24	100
	Severity 2	75.00	75.00	75.00	57.14	75.00
	Severity 3	88.00	78.00	82.00	66.67	77.78
	severity 4	92.00	88.00	90.00	61.54	88.46
	Average result of all class	90.00	90.00	90.00	66.67	86.00
FE	Severity 1	95.00	100	97.00	87.50	100
	Severity 2	100	86.00	92.00	50.00	85.72
	Severity 3	100	88.00	93.00	95.24	87.50
	severity 4	100	100	100	60.00	100
	Average result of all class	97.00	96.00	96.00	86.90	96.00
LM	Severity 1	100	98.00	99.00	100	98.36
	Severity 2	00	00	00	100	100
	Severity 3	81.00	100	89.00	100	100
	severity 4	100	100	100	100	100
	Average result of all class	93.00	95.00	94.00	99.08	95.00

dataset for an average of all severities, and 96.00% SCA obtained using the Random search algorithm for the FE dataset for the average of all severities.

AdaBoost (Adaptive Boosting)

The first well-doing boosting algorithm built for binary classification was the AdaBoost algorithm. Yoav Fried and Robert Schapire were the ones who found it [41]. A popular boosting technique turns several “poor classifiers” into one “strong classifier.”

Table 10 presents that the AdaBoost algorithm obtained the highest 98.18% SCA using the grid search method for the LM dataset for the average of all severities and 97.61% SCA using the Random search algorithm for the LM dataset for an average of all severities.

XGB Algorithm (XG Boosting)

The XGBoost is a tree-based MLA with superior presentation and speed, commonly known as the extreme gradient boosting algorithm. It is a straightforward algorithm that works with MLA and has grown in popularity since it produces effective outcomes for organized and tabular

data. Tianqi Chen developed XGBoost, which is primarily governed by the Distributed Machine-Learning Community (DMLC) organization. It is open-source software [42].

Table 11 presents that the XGB algorithm obtained the highest 99.12% SCA using the grid search method for the LM dataset for the average of all severities and 98.89% SCA using the random search method for the LM dataset for the average of all severities.

Gradient Boosting (GB Algorithm)

The gradient boosting (GB) algorithm is the most effective ensemble MLA. The errors in MLAs are mainly separated into two types, i.e., bias error and variance error. GB algorithm is a boosting technique which can be employed to decrease the bias error of the algorithm. The GB algorithm is used for both categorical and constant target variables, such as classifiers and regression [43].

Table 12 presents that the GB algorithm obtained the highest, 98.67% SCA using the grid search method for the LM dataset for the average of all severities and 98.32% SCA using the Random search algorithm for the LM dataset for the average of all severities.

Table 10 Severity classification results of AdaBoost algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	95.00	72.00	82.00	74.45	72.42
	Severity 2	25.00	20.00	22.00	81.00	80.00
	Severity 3	42.00	61.00	50.00	52.00	60.87
	Severity 4	68.00	63.00	65.00	66.67	62.96
	Average result of all class	68.00	63.00	64.00	70.37	63.09
GC	Severity 1	100	73.00	85.00	75.18	73.17
	Severity 2	00	00	00	52.00	50.00
	Severity 3	56.00	50.00	53.00	50.00	50.00
	Severity 4	60.00	71.00	65.00	72.00	71.43
	Average result of all class	77.00	65.00	70.00	66.12	65.48
FE	Severity 1	98.00	97.00	98.00	96.52	96.77
	Severity 2	00	00	00	76.00	66.67
	Severity 3	80.00	94.00	86.00	92.23	94.12
	Severity 4	67.00	100	80.00	100	100
	Average result of all class	90.00	93.00	91.00	91.98	92.16
LM	Severity 1	1.0	1.0	1.0	100	100
	Severity 2	00	00	00	87.24	100
	Severity 3	87.00	100	93.00	100	100
	Severity 4	100	100	100	98.25	100
	Average result of all class	96.00	98.00	97.00	98.18	97.61

Table 11 Severity classification results of XG Boosting algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	84.00	96.00	90.00	65.85	96.30
	Severity 2	40.00	29.00	33.00	80.00	57.14
	Severity 3	65.00	59.00	62.00	68.75	58.62
	Severity 4	68.00	71.00	70.00	81.82	71.43
	Average result of all class	70.00	71.00	70.00	78.80	80.24
GC	Severity 1	97.00	97.00	97.00	66.67	97.06
	Severity 2	25.00	14.00	18.00	85.72	71.43
	Severity 3	52.00	79.00	62.00	76.20	78.95
	Severity 4	82.00	58.00	68.00	69.56	58.34
	Average result of all class	77.00	75.00	74.00	78.80	80.00
FE	Severity 1	94.00	100	97.00	70.00	100
	Severity 2	00	00	00	83.34	60.00
	Severity 3	85.00	96.00	90.00	66.67	95.65
	Severity 4	75.00	60.00	67.00	100	60.00
	Average result of all class	85.00	90.00	88.00	91.90	92.14
LM	Severity 1	100	100	100	67.27	100
	Severity 2	100	100	100	100	100
	Severity 3	100	100	100	80.95	100
	Severity 4	100	100	100	66.67	100
	Average result of all class	100	100	100	99.12	98.89

Table 12 Severity classification results of gradient boosting algorithm

Datasets	Class name	PPV (%)	TPR (%)	F-measure (%)	Grid search SCA (%)	Random search SCA (%)
DC	Severity 1	93.00	100	96.00	100	92.98
	Severity 2	100	60.00	75.00	60.00	66.67
	Severity 3	68.00	83.00	75.00	83.00	84.00
	Severity 4	95.00	75.00	84.00	75.00	78.79
	Average result of all class	88.00	87.00	87.00	86.90	88.22
GC	Severity 1	93.00	84.00	88.00	83.87	85.86
	Severity 2	50.00	33.00	40.00	33.34	40.00
	Severity 3	59.00	77.00	67.00	77.28	78.89
	Severity 4	78.00	72.00	75.00	72.00	76.65
	Average result of all class	76.00	75.00	75.00	75.00	76.89
FE	Severity 1	92.00	98.00	95.00	98.22	99.00
	Severity 2	100	20.00	33.00	80.00	84.56
	Severity 3	94.00	89.00	92.00	89.47	90.00
	Severity 4	80.00	100	89.00	100	100
	Average result of all class	92.00	92.00	90.00	91.67	92.12
LM	Severity 1	100	100	100	100	100
	Severity 2	100	100	100	100	100
	Severity 3	100	100	100	100	100
	Severity 4	100	100	100	100	100
	Average result of all class	100	100	100	98.67	98.32

Table 13 SCA comparison between grid search and random search of all machine-learning methods

	Datasets	Grid search SCA	Random search SCA
LR	DC	71.70	68.09
	GC	70.11	67.20
	FE	88.09	84.05
	LM	97.34	91.42
RF	DC	76.49	70.47
	GC	76.98	78.06
	FE	89.42	91.43
	LM	97.34	96.20
KNN	DC	71.62	70.82
	GC	64.72	69.05
	FE	68.34	71.57
	LM	90.95	91.64
DT	DC	73.80	75.00
	GC	66.67	86.00
	FE	86.90	96.00
	LM	99.08	95.00
Ada Boost	DC	70.37	63.09
	GC	66.12	65.48
	FE	91.98	92.16
	LM	98.18	97.61
XGB	DC	78.80	80.24
	GC	78.80	80.00
	FE	91.90	92.14
	LM	99.12	98.89
GB	DC	86.90	88.22
	GC	75.00	76.89
	FE	91.67	92.12
	LM	98.67	98.32

SCA Comparison Between Grid Search and Random Search of All Machine-Learning Methods

Table 13 shows the comparison among the SCA of all the machine-learning methods obtained by Grid search and Random search algorithms. It is observed that some of the machine-learning methods perform better using the grid search algorithm, and some perform better using the random search algorithm.

From our work, we observed the following: (1) for the DC dataset, the highest severity detection accuracy is 88.22% using the gradient boosting algorithm for the random search method; (2) for the GC dataset, the highest severity detection accuracy is 86.00% using the DT algorithm for the random search method; (3) for the FE dataset, the highest severity detection accuracy is 96.00% using the DT algorithm for the random search method; and (4) for the LM dataset, the highest severity

Table 14 Comparison among all algorithms

Algorithms	Dataset	DC				GC				FE				LM			
		P (%)	T (%)	F (%)	SCA (%)	P (%)	T (%)	F (%)	SCA (%)	P (%)	T (%)	F (%)	SCA (%)	P (%)	T (%)	F (%)	SCA (%)
LR		54.00	62.00	58.00	71.70	73.00	74.00	73.00	70.11	69.00	79.00	73.00	88.09	96.00	95.00	96.00	97.34
RF		65.00	67.00	65.00	76.49	86.00	88.00	86.00	78.06	92.00	93.00	91.00	91.43	98.00	98.00	98.00	97.34
KNN		71.00	71.00	70.00	71.62	45.00	50.00	45.00	69.05	70.00	77.00	73.00	71.57	87.00	86.00	83.00	91.64
DT		72.00	75.00	71.00	75.00	90.00	90.00	90.00	86.00	97.00	96.00	96.00	96.00	93.00	95.00	94.00	99.08
AB		68.00	63.00	64.00	70.37	77.00	65.00	70.00	66.12	90.00	93.00	91.00	92.16	96.00	98.00	97.00	98.18
XGB		70.00	71.00	70.00	80.24	77.00	75.00	74.00	80.00	85.00	90.00	88.00	92.14	100	100	100	99.12
GB		88.00	87.00	87.00	88.22	76.00	75.00	75.00	76.89	92.00	92.00	90.00	92.12	100	100	100	98.67

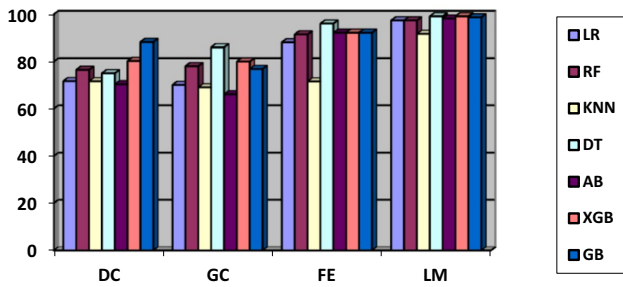


Fig. 2 Comparison bar chart of all algorithms

detection accuracy is 99.12% using the XG Boost algorithm for the grid search method.

Comparison Among All the Algorithms Used in this Work

A comparison among all machine-learning methods are shown in Table 14, and Fig. 2 shows the comparative analysis among all the algorithms using a bar graph. The following results are obtained here:

1. The gradient boosting (GB) algorithm achieved the highest SCA of 88.22% for the DC dataset.
2. The DT approach achieved the maximum SCA of 86.00% for GC and 96.00% for the FE dataset.
3. XGB approach achieved the highest SCA of 99.12% for the LM dataset.

Table 15 Result comparison with and without applying feature selection technique

Datasets	Algorithms	Results with feature selection technique		Results without feature selection technique	
		<i>F</i> -measure (%)	SCA (%)	<i>F</i> -measure (%)	SCA (%)
DC	LR	58.00	71.70	80.00	72.23
	RF	65.00	76.49	90.00	75.98
	K-nearest neighbor	70.00	71.62	70.00	71.12
	DT	71.00	75.00	73.00	73.00
	AdaBoost	64.00	70.37	69.00	70.27
	XG Boost	70.00	80.24	85.00	81.00
	Gradient boosting	87.00	88.22	90.00	86.67
GC	LR	73.00	70.11	70.00	65.67
	RF	86.00	78.06	86.00	76.76
	K-nearest neighbor	45.00	69.05	52.00	61.10
	DT	90.00	86.00	85.00	82.08
	AdaBoost	70.00	66.12	86.00	65.78
	XG Boost	74.00	80.00	77.00	79.56
	Gradient boosting	75.00	76.89	82.00	75.00
FE	LR	73.00	88.09	79.00	89.14
	RF	91.00	91.43	92.00	93.65
	K-nearest neighbor	73.00	71.57	72.00	68.78
	DT	96.00	96.00	95.00	95.32
	AdaBoost	91.00	92.16	90.00	90.26
	XG Boost	88.00	92.14	83.00	89.78
	Gradient boosting	90.00	92.12	88.00	93.65
LM	LR	96.00	97.34	99.00	97.00
	RF	98.00	97.34	99.00	98.14
	K-nearest neighbor	83.00	91.64	97.00	95.87
	DT	94.00	99.08	98.00	93.98
	AdaBoost	97.00	98.18	100	98.00
	XG Boost	100	99.12	100	98.74
	Gradient boosting	100	98.67	100	98.82

Influence of Feature Selection Technique

This section focuses on the influence of feature selection techniques to enhance the results and identify the features which play an essential role in code smell severity detection. The Chi-square feature selection technique is constructed in response to research question 2 (RQ2). The SCA (%) and *F*-measure (%) of all the algorithms with and without applied feature selection techniques are compared in Table 15. For this study, ten features are selected from each dataset shown in Table 3. This study observed that maximum algorithms show improved performance of SCA for each dataset by applying the feature selection technique, and only few of them did not perform well. In this study, we observed that the highest severity was detected by the gradient boosting model (*F*-measure—87.00% and accuracy—88.22%) from the DC dataset. The highest severity detected by the DT model (*F*-measure—90.00% and accuracy—86.00%) from GC dataset, and (*F*-measure—96.00% and accuracy—96.00%) from FE dataset. Similarly, the highest severity was detected by the XG Boost model (*F*-measure—100% and accuracy—99.12%) from the LM dataset.

Influence of Hyper-parameter Tuning on Algorithms

In response to research question 3 (RQ3), the influence of hyper-parameter tuning on the performance of all algorithms is studied. Table 16 represents the various parameter groups and obtained the highest SCA of for the DT algorithm. The DT algorithm obtained the highest SCA of 99.08% when the maximum level is 20, the numbers of trees are 20, and the number of the split is 15. Similarly, Tables 17 and 18 show the influence of parameter optimization on the SCA of all algorithms.

Table 16 Tuning parameters applied on DT algorithm

Parameters	Level	
Maximum level	20	
Criteria and their corresponding SCA (%)	Information gain	99.08%
	Accuracy	99.08%
	Gini Index	98.28%
	Entropy	98.34%
Minimal size of split	15	
Minimal gain	0.0	
BootStrap	False	
N_estimators	500	
Apply pre-pruning	True	
Apply pruning	False	

Table 17 POT’s effect on the LR model’s SCA

Penalty	Parameter ‘C’	SCA (%)
11	0.1	97.34
11	1.0	96.98
12	1.0	96.45
12	100	95.92

Discussion

In this paper, we have used four code smell severity dataset which includes god class, data class, feature envy and long method. The severity composition of each dataset is given in Table 2. In all the datasets, Severity 1 has the highest instances, and Severity 2 has the lowest instances. We have shown separate results for each severity of each dataset in “Results of Proposed Model”, and we observed a significant difference between the results of the four severities for each dataset because the combination of the number of severities in datasets is different. In “Results of Proposed Model”, we found the highest accuracy for severity 1 for each dataset because severity 1 has the highest instances and the lowest accuracy for severity 2 for each dataset because severity 2 has the lowest instances. In this way, we found a class

Table 18 POT’s effect on the RF, AdaBoost, XGBoost, and GB model’s SCA

Model name	Tree’s count	Maximum level	SCA (%)
RF	10	10	95.32
	15	20	96.98
	20	20	98.14
	30	25	97.12
	50	30	97.86
AdaBoost	10	3	95.12
	15	5	96.33
	20	10	97.85
	30	20	98.18
	50	30	95.34
XG Boost	10	5	97.54
	20	10	97.98
	30	15	99.12
	40	20	98.78
	50	25	97.00
GB	10	5	93.23
	20	10	93.65
	30	15	92.78
	40	20	92.14
	50	25	92.65

imbalance problem in all the datasets, which we will be considering in our future works.

In this experiment, we solved three research questions given in “Proposed Model and Dataset Description”. To answer research question 1, we have applied seven machine-learning algorithms described in “Logistic Regression” to “Gradient Boosting (GB Algorithm)”. We found that the DT algorithms give the best SCA of 86.00% and 96.00% for the GC and FE datasets (Table 14). Likewise, the gradient boost algorithm gives the best SCA of 88.22% for the DC dataset, and the XG Boost approach gives the maximum SCA of 99.12% for the LM dataset (Table 14). To answer research question two, we obtained the influence of the feature selection technique in Table 15, and we found that feature selection gives better performance in all the algorithms for each dataset. We applied the POT to each algorithm to answer research question three. The influence of the POT on each model performance is shown in Tables 16, 17 and 18.

Result Evaluation of Our Results with Other Related Works

In earlier literatures, many authors have applied different algorithms in different severity datasets, which we have seen in detail in “Related Work”. This section constructs a summarized evaluation of our work in comparison with Fontana et al. [4] and Abdou [27]. They both applied various MLAs and implemented multinomial classification to regression and binary classifiers for ordinal classification. A linear correlation-based filter method was also applied to select the best features by them. In addition, Abdou [27] applied projective adaptive resonance theory (PART) algorithm to learn the efficiency of software metrics to detect the code smells and a local interpretable model agnostic explanations (LIME) algorithm was applied to describe the ML models and interpretability.

In our approach, we applied four MLAs and three ensemble learning approaches to identify the severity of code smells. A Chi-square-based FST is applied to select the best metrics, and two-parameter optimization techniques (grid search and random search) are applied to optimize the best parameters from each model.

Table 19 compares our results with other works. In our approach, for the DC dataset GB algorithm achieved 88.22% SCA, while Fontana et al. [4] achieved 77.00% SCA using the O-RF algorithm, and Abdou [27] achieved 93.00% SCA using the O-R-SMO algorithm. Like this, the Abdou [27] model is best for severity detection in data class dataset.

To the GC dataset, our approach achieved 86.00% SCA using DT algorithm, while Fontana et al. [4] achieved 74.00% SCA using the O-DT algorithm, and Abdou [27] achieved 92.00% SCA using the R-B-RF algorithm. Like this, the Abdou [27] model is best for severity detection in God class dataset.

Table 19 Result evaluation of our method with other works

Year	Author name	Applied algorithms	Applied FSA and other techniques	Datasets			
				DC	GC	FE	
				LM		LM	
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
2017	Fontana et al. [4]	18 Machine learning methods for binary classification, and multinomial classification to regression	Linear co-relation filter method	O-Random Forest	77.00	O-DT	74.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-Random Forest	77.00	O-DT	74.00
2022	Abdou [27]	40 binary and multinomial classification	PART algorithm, LIME algorithm, ranking correlation algorithm	O-R-SMO	93.00	R-B-RF	92.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-RF	92.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)
				O-R-SMO	93.00	R-B-JRIP, O-R-SMO	97.00
				Best algorithm	SCA (%)	Best algorithm	SCA (%)

To the FE dataset, our approach achieved 96.00% SCA using DT algorithm, while Fontana et al. [4] achieved 93.00% SCA using the J48-Pruned algorithm, and Abdou [27] achieved 97.00% SCA using the R-B-JRIP and O-R-SMO algorithm. Like this, the Abdou [27] model is best for severity detection in FE dataset.

To the LM dataset, our approach achieved 99.12% SCA using XG Boost algorithm, while Fontana et al. [4] achieved 92.00% SCA using the B-Random Forest algorithm, and Abdou [27] achieved 97.00% SCA using the R-B-JRIP, O-B-RF, and O-R-JRip algorithm. Like this, our model is best for severity detection in LM dataset.

Statistical Analysis for Comparing Machine-Learning Models

From Table 15, we observed that there is not much difference in the results of two best models to the same dataset. In that case, we have used a statistical analysis method to choose the one of best model for each dataset. We applied a paired *T* test statistical analysis method on the two best models (according to Table 15) for each datasets. For this, we have selected two best models which are obtained best accuracy for each dataset and then applied paired *T* test.

A paired *T* test allows data analysis to consider different methods using the same dataset to see if the difference is minimal [44]. This method can be used to check that there is a statistically significant distinction between two models, allowing you to choose only the better one. We used mean accuracy as a measurement calculated across tenfold cross-validation in our studies, with set the significance level (i.e., $\alpha = 0.05$).

We developed the following hypotheses (H) values for each comparison:

Null hypothesis (H0): *X* and *Y*'s accuracy of the model are obtained from the same sample. As a result, the difference in levels of accuracy has a predicted value of 0 ($E[\text{diff}] = 0$). In essence, the two models are identical [45].

Alternative hypothesis (H1): The prediction accuracy is obtained from two separate models, $E[\text{diff}] \neq 0$. Essentially, the models are distinct, and one is superior to the other [45].

- *H0*: $ACCURACY_x = ACCURACY_y$ (the detection efficiency of the two models is identical).
- *H1*: $ACCURACY_x \neq ACCURACY_y$ (the detection efficiency of the two models differs significantly).

where *x* and *y* are the two models under consideration. If the function returns a *p* value less than alpha, we can correctly reject the null hypothesis.

```

if p > alpha:
    print("Fail to reject null hypothesis")
else:
    print("Reject null hypothesis")
    
```

In this study, we calculated two parameters: mean accuracy and *p* value for each dataset.

Mean accuracy: A model has higher mean accuracy that means the model is best for dataset as compared to less mean accuracy model. ***p* value:** If the *p* value is higher to alpha ($\alpha = 0.05$), the null hypothesis is assumed to be true. If the *p* value is below alpha, the null hypothesis is assumed to be incorrect. Then, we have selected to one of best model according to mean accuracy from given to two the model.

Table 20 displays the mean accuracy and *p* value of each classification model across each code smell dataset. From Table 20, we observed that the DC, GC, and LM dataset has *p* value less than 0.05. Therefore, the accuracy of both the models is different. In this way for DC dataset, the mean accuracy of XGBoost model is more than gradient boosting model so the XG Boost model is best for DC dataset. For the GC dataset, the mean accuracy of XG model is more than DT model so the XG Boost model is best for GC dataset. For the LM dataset, the mean accuracy of XG model is more than DT model so the XG Boost model is best for the LM dataset. The FE dataset has *p* value greater than 0.05. Therefore, the mean accuracy of both the models is same (Table 20).

Table 20 Statistical analysis results

Dataset name	Two best ML models	Mean accuracy (%)	<i>P</i> value	Results
DC	XG Boost	82.74	0.032	Reject null hypothesis (XG Boost model is best)
	Gradient boosting	80.23		
GC	DT	45.24	0.006	Reject null hypothesis (XG Boost model is best)
	XG Boost	74.60		
FE	DT	91.71	0.093	Fail to reject null hypothesis
	Gradient boosting	91.71		
LM	DT	88.89	0.005	Reject null hypothesis (XG Boost model is best)
	XG Boost	96.68		

Conclusion

In this research work, to analyze the code smell severities from software to decrease the maintenance work and enhance the software quality, and also find the best algorithms for detecting the code smell severities, we proposed the severity classification of code smell framework with multi-class classification approaches using four machine-learning and three ensemble learning algorithms. To select the significant features from each dataset, Chi-square feature selection algorithm is applied. Two-parameter optimization algorithms (grid search and random search) with fivefold cross-validation are used to enhance the SCA.

In this study, it is found that the GB method finds the maximum SCA of 88.22% using the feature selection algorithm, while the AdaBoost algorithm obtains the lowest result of 70.27% without using the feature selection algorithm for the DC dataset.

The DT method found the maximum SCA of 86.00% using the feature selection algorithm, while the KNN algorithm obtained the lowest result of 61.10% without using the feature selection algorithm for the GC dataset.

The DT algorithm found a maximum SCA of 96.00% using the feature selection algorithm, while the lowest result was 68.78% obtained by the KNN algorithm without using the feature selection algorithm for the FE dataset.

The XG Boost algorithm found a maximum SCA of 99.12% using the feature selection algorithm, while the lowest result was 91.64%, obtained by the KNN model by applying the feature selection approach for the LM dataset.

This study found that the GB method is best for the DC dataset, the XG Boost model is best for the LM dataset, and the DT model is best for the GC and FE datasets to detect the severity of code smells. Moreover, the Chi-square feature selection technique is always helpful for better detecting the severity of code smells.

The limitation of this study is that the code smell severity datasets have a class imbalance problem; therefore, in subsequent work, we intend to enhance outcomes by utilizing class balancing techniques to address the issue of class imbalance (present in the used dataset). In order to determine the most effective methods for code smell severity detection, other learning algorithms and feature selection strategies should be investigated.

Appendix

See Table 21.

Table 21 Description of all selected metrics [4]

Quality dimension	Selected metric	Metric name	Granularity
Size	LOC_project	Lines of code	Project, package, class
	LOC_package		
	LOC_type		
	NOMNAMM_project	Number of not accessor or mutator methods	Project, package, and class
	NOMNAMM_package		
	LOCNAMM_type	Lines of code without accessor or mutator methods	Class
	NOCS_project	Number of classes	Project, package
	NOCS_package		
–	Complexity	–	Class
Inheritance	NIM_type	Number of inherited methods	Class
	Method	–	–
Complexity	ATFD_type	Access to foreign data	Method
	Project	–	–
Size	AMW_type	Average methods weight	Class
	Package	–	–
Complexity	AMWNAMM_type	Average methods weight of not accessor or Mutator methods	Class
	Complex type	=	Method
Coupling	CBO_type	Coupling between objects classes	Class
Size	LOC_method	Lines of code	Method
	CYCLO_method	Cyclomatic complexity	
Complexity	NOAV_method	Number of accessed variables	
Coupling	CINT_method	Coupling intensity	

Author Contributions Conceptualization, MG, and RSR; data curation, SD; formal analysis, MG, RSR; investigation, SD, and RSR; methodology, MG, and RSR; supervision, MG, RSR; validation, RSR, SD, and MG; visualization, SD, and RSR; writing, SD, and RSR; review and editing, RSR, MG, and SRC.

Funding This study was not supported by any other sources.

Data availability The dataset can be found at <http://essere.disco.unimib.it/reverse/MLCSD.html>.

Declarations

Conflict of Interest No conflicts of interest exist, according to the authors, with the publishing of this work.

References

- Lehman MM. Programs, life cycles, and laws of software evolution. *Proc IEEE*. 1980;68(9):1060–76.
- Wieggers K, Beatty J. *Software Requirements*. London: Pearson Education; 2013.
- Chung L, Do PLJCS. On non-functional requirements in software engineering. In: Borgida AT, Chaudhri V, Giorgini P, Yue ES, editors. *conceptual modeling: foundations and applications (lecture notes in computer science)*. Cham: Springer; 2009. p. 363–79.
- Fontana FA, Zanoni M. Code smell severity classification using machine learning techniques. *Knowl Based Syst*. 2017. <https://doi.org/10.1016/j.knosys.2017.04.014>.
- Vidal SA, Marcos C, Diaz-Pace JA. An approach to prioritize code smells for refactoring. *Autom Softw Eng*. 2016;23(3):501–32. <https://doi.org/10.1007/s10515-014-0175-x>.
- Liu W, Wang S, Chen X, Jiang H. Predicting the severity of bug reports based on feature selection. *Int J Softw Eng Knowl Eng*. 2018;28(04):537–58. <https://doi.org/10.1142/S0218194018500158>.
- Tiwari O, Joshi R (2020) Functionality based code smell detection and severity classification. In: *ISEC 2020: 13th innovations in software engineering conference*, pp 1–5. <https://doi.org/10.1145/3385032.3385048>.
- Baarah A, Aloqaily A, Salah Z, Zamzeer M, Sallam M. Machine learning approaches for predicting the severity level of software bug reports in closed source projects. (*IJACSA*) *Int J Adv Comput Sci Appl*. 2019;10(8):285–94.
- Fontana FA, Mäntylä MV, Zanoni M, Marino A. Comparing and experimenting machine learning techniques for code smell detection. *Empir Softw Eng*. 2016;21(3):1143–91.
- Mhawish MY, Gupta M. Generating code-smell prediction rules using decision tree algorithm and software metrics. *Int J Comput Sci Eng (IJCSE)*. 2019;7(5):41–8.
- Mhawish MY, Gupta M. Predicting code smells and analysis of predictions: using machine learning techniques and software metrics. *J Comput Sci Technol*. 2020;35(6):1428–45. <https://doi.org/10.1007/s11390-020-0323-7>.
- Kaur I, Kaur A. A novel four-way approach designed with ensemble feature selection for code smell detection. *IEEE Access*. 2021;9:8695–707. <https://doi.org/10.1109/ACCESS.2021.3049823>.
- Pushpalatha MN, Mrunalini M. Predicting the severity of closed source bug reports using ensemble methods. In: Satapathy S, Bhatteja V, Das S, editors. *Smart intelligent computing and applications*. Smart innovation, systems and technologies, vol. 105. Singapore: Springer; 2019. https://doi.org/10.1007/978-981-13-1927-3_62.
- Alazba A, Aljamaan HI. Code smell detection using feature selection and stacking ensemble: an empirical investigation. *Inf Softw Technol*. 2021;138: 106648.
- Draz MM, Farhan MS, Abdulkader SN, Gafar MG. Code smell detection using whale optimization algorithm. *Comput Mater Continua*. 2021;68(2):1919–35.
- Dewangan S, Rao RS, Mishra A, Gupta M. A novel approach for code smell detection: an empirical study. *IEEE Access*. 2021;9:162869–83. <https://doi.org/10.1109/ACCESS.2021.3133810>.
- Reis JPD, Abreu FBE, Carneiro GDF. Crowd smelling: a preliminary study on using collective knowledge in code smells detection. *Empir Softw Eng*. 2022;27:69. <https://doi.org/10.1007/s10664-021-10110-5>.
- van Oort B, Cruz L, Aniche M, van Deursen A. (2021) The prevalence of code smells in machine learning projects, *IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*. Madrid, Spain. p. 1–8. <https://doi.org/10.1109/WAIN52551.2021.00011>.
- Fontana A, Mariani E, Morniroli A, Sormani R, Tonello A. (2011). An experience report on using code smells detection tools. In: *IEEE fourth international conference on software testing, verification and validation workshops, RefTest 2011*. Berlin: IEEE Computer Society; 2011. p. 450–7. <https://doi.org/10.1109/ICSTW.2011.12>.
- Boutaib S, Elarbi M, Bechikh S, Palomba F, Said LB. A bi-level evolutionary approach for the multi-label detection of smelly classes. In: *GECCO 22 companion*, July 9–13, 2022, Boston, MA, USA. ACM ISBN 978-1-4503-9268-6/22/07. (2022). <https://doi.org/10.1145/3520304.3528946>.
- Abdou AS, Darwish NR. Early prediction of software defect using ensemble learning: a comparative study. *Int J Comput Appl*. 2018;179(46):29–40. <https://doi.org/10.5120/ijca2018917185>.
- Dewangan S, Rao RS, Mishra A, Gupta M. Code smell detection using ensemble machine learning algorithms. *Appl Sci*. 2022;12(20):10321. <https://doi.org/10.3390/app122010321>.
- Dewangan S, Rao RS. Code smell detection using classification approaches. In: Udgata SK, Sethi S, Gao XZ, editors. *Intelligent systems; lecture notes in networks and systems*, vol. 431. Singapore: Springer; 2022. https://doi.org/10.1007/978-981-19-0901-6_25.
- Dewangan S, Rao RS, Yadav PS. Dimensionally reduction based machine learning approaches for code smells detection. In: *2022 international conference on intelligent controller and computing for smart power (ICICCCSP)*; 2022. p. 1–4. <https://doi.org/10.1109/ICICCCSP53532.2022.9862030>. Accessed 30 Jan 2023.
- Fowler M. *Refactoring: improving the design of existing code*. Boston: Addison-Wesley Longman Publishing Co. Inc. <http://www.refactoring.com/> (1999).
- Gupta A, Chauhan NK. A severity-based classification assessment of code smells in Kotlin and Java application. *Arab J Sci Eng*. 2022;47:1831–48. <https://doi.org/10.1007/s13369-021-06077-6>.
- Abdou A, Darwish N. Severity classification of software code smells using machine learning techniques: a comparative study. *J Softw Evol Proc*. 2022. <https://doi.org/10.1002/smr.2454>.
- Hejres S, Hammad M. Code smell severity detection using machine learning. In: *4th smart cities symposium (SCS 2021)*; 2021. p. 89–96. <https://doi.org/10.1049/icp.2022.0320>.
- Nanda J, Chhabra JK. SSHM: SMOTE-stacked hybrid model for improving severity classification of code smell. *Int J Inf Technol*. 2022. <https://doi.org/10.1007/s41870-022-00943-8>.
- Tempero E, Anslow C, Dietrich J, Han T, Li J, Lumpe M, Melton H, Noble J. The qualitas corpus: a curated collection of java code for empirical studies. In: *Proceedings of the 17th Asia Pacific*

- software engineering conference (APSEC 2010). IEEE Computer Society; 2010. p. 336–45. <https://doi.org/10.1109/APSEC.2010.46>.
31. Olbrich S, Cruzes D, Sjoberg DIK. Are all code smells harmful? A study of god classes and brain classes in the evolution of three open source systems. In: Proceedings of the IEEE international conference on software maintenance (ICSM 2010), Timisoara, Romania; 2010. p. 1–10. <https://doi.org/10.1109/ICSM.2010.5609564>.
 32. Marinescu C, Marinescu R, Mihancea P, Ratiu D, Wettel R. iPlasma: an integrated platform for quality assessment of object-oriented design. In: Proceedings of the 21st IEEE international conference on software maintenance (ICSM 2005) (industrial and tool Proceedings), tool demonstration track. Budapest, Hungary: IEEE; 2005. p. 77–80.
 33. Nongpong K. Integrating “code smell” detection with refactoring tool support. Ph.D. thesis, University of Wisconsin Milwaukee (2012).
 34. Marinescu R. Measurement and quality in object oriented design. Ph.D. thesis, Department of Computer Science, “Polytechnic” University of Timisoara (2002).
 35. Ali PJM, Faraj RH. Data normalization and standardization : a technical report. Mach Learn Tech Rep. 2014;1(1):1–6.
 36. Romero E, Sopena JM. Performing feature selection with multi-layer perceptrons. IEEE Trans Neural Netw. 2008;19(3):431–41. <https://www.geeksforgeeks.org/ml-chi-square-test-for-feature-selection>. Accessed 30 Jan 2023.
 37. Hsu CW, Chang CC, Lin CJ. A practical guide to support vector classification. Technical Report, Taiwan University; 2008. <https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf> (2020).
 38. Chaitra PC, Saravana Kumar R. A review of multi-class classification algorithm. Int J Pure Appl Math. 2018;118(14):17–26.
 39. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm. Accessed 30 Jan 2023.
 40. <https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/>. (Last Updated: 11 Oct 2021). Retrieved 26 Nov 2021.
 41. <https://analyticsindiamag.com/xgboost-internal-working-to-make-decision-trees-and-deduce-predictions/>. Last Updated 2 Nov 2020. Retrieved 26 Nov 2021.
 42. <https://www.analyticsvidhya.com/blog/2021/04/how-the-gradient-boosting-algorithm-works/>. (Last Updated 19 Apr 2021). Retrieved 26 Nov 2021.
 43. <https://www.geeksforgeeks.org/paired-t-test-a-detailed-overview/>. (Last Updated 28 Feb 2022). Retrieved 26 Jan 2023.
 44. <https://towardsdatascience.com/paired-t-test-to-evaluate-machine-learning-classifiers-1f395a6c93fa>. (Last Updated 6 July 2022). Retrieved 26 Jan 2023.
 - 45.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.