applied sciences

MDPI

Systematic Review

Machine Learning-Based Methods for Code Smell Detection: A Survey

Pravin Singh Yadav [1], Rajwant Singh Rao [1], Alok Mishra [2,3,*] and Manjari Gupta [4]

[1] Department of Computer Science and Information Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur 495009, Chhattisgarh, India; pravinsingh1110@gmail.com (P.S.Y.); rajwantrao@gmail.com (R.S.R.)
[2] Faculty of Engineering, Norwegian University of Science and Technology (NTNU), 7491 Trondheim, Norway
[3] Informatics and Digitalization Group, Molde University College, Specialized University in Logistics, 6402 Molde, Norway
[4] Computer Science, DST—Centre for Interdisciplinary Mathematical Sciences, Institute of Science, Banaras Hindu University, Varanasi 221005, Uttar Pradesh, India; manjari@bhu.ac.in
* Correspondence: alok.mishra@ntnu.no

Abstract: Code smells are early warning signs of potential issues in software quality. Various techniques are used in code smell detection, including the Bayesian approach, rule-based automatic antipattern detection, antipattern identification utilizing B-splines, Support Vector Machine direct, SMURF (Support Vector Machines for design smell detection using relevant feedback), and immune-based detection strategy. Machine learning (ML) has taken a great stride in this area. This study includes relevant studies applying ML algorithms from 2005 to 2024 in a comprehensive manner for the survey to provide insight regarding code smell, ML algorithms frequently applied, and software metrics. Forty-two pertinent studies allow us to assess the efficacy of ML algorithms on selected datasets. After evaluating various studies based on open-source and project datasets, this study evaluated additional threats and obstacles to code smell detection, such as the lack of standardized code smell definitions, the difficulty of feature selection, and the challenges of handling large-scale datasets. The current studies only considered a few factors in identifying code smells, while in this study, several potential contributing factors to code smells are included. Several ML algorithms are examined, and various approaches, datasets, dataset languages, and software metrics are presented. This study provides the potential of ML algorithms to produce better results and fills a gap in the body of knowledge by providing class-wise distributions of the ML algorithms. Support Vector Machine, J48, Naive Bayes, and Random Forest models are the most common for detecting code smells. Researchers can find this study helpful in better anticipating and taking care of software development design and implementation issues. The findings from this study, which highlight the practical implications of ML algorithms in software quality improvement, will help software engineers fix problems during software design and development to ensure software quality.

Keywords: antipattern; code smell; code smell detection; identification; machine learning algorithms

check for updates

Citation: Yadav, P.S.; Rao, R.S.; Mishra, A.; Gupta, M. Machine Learning-Based Methods for Code Smell Detection: A Survey. *Appl. Sci.* **2024**, *14*, 6149. https://doi.org/10.3390/app14146149

Academic Editors: Andrea Prati, Linda Vickovic and Maja Braović

Received: 20 April 2024
Revised: 20 June 2024
Accepted: 11 July 2024
Published: 15 July 2024

1. Introduction

Code smells can indicate deeper problems in the software, adversely affecting the software quality [1–5]. The code smell detection model identifies software issues that may cause severe problems in the future. Identifying code smells is always recommended, which helps to reduce the software maintenance cost and increases the code reusability [3,6]. Software metrics, detection models, pattern-matching approaches, etc., are effective methods for detecting code smells. A variety of software metrics can help to detect code smells. The proper detection of code smells also depends upon how well the model performs and detects various code smells. The domain of code smell detection has witnessed the development of various approaches, from rule-based, metric-based, to machine learning (ML)-based. ML is expanding quickly and finding applications in many different areas.

Solutions based on ML have proven very useful for complicated issues that challenge conventional approaches; because of that, this study is mainly focused on ML-based approaches. In the literature, many ML algorithms are used, such as Decision Tree, Decision Tree (J48), Decision Tree (c4.5), Naive Bayes, Support Vector Machines (SVMs), Random Forest, etc.

Nucci et al. [7] applied thirty-two ML algorithms like AdaBoost, J48, Random Forest, Naive Bayes, JRip, LIBSVM, Sequential Minimal Optimization, etc. They found 96% accuracy and 90% F-measure, the best-known results compared to the other works in this area. Guggulothu et al. [8] applied some ML algorithms with a 10-fold cross-validation method, including the J48 (C4.5) algorithm, Random Forest, JRip, Naive Bayes, Sequential Minimal Optimization, and K-nearest neighbors (where k = 2), and found that tree-based classifiers provide better results among all applied algorithms. Iqbal et al. [9] anticipated two skeletons, one with feature selection and the other without feature selection. They compared their results with techniques such as Multilayer Perceptron, SVM, Rule-Based Learning, Naive Bayes, K-nearest neighbors, Random Forest, etc. They observed that their method performed better than these classification methods. Pecorelli et al. [10] compared ML algorithms with DECOR (DEtection and CORrection), a heuristic-based approach on thirteen open-source software systems, and noticed that DECOR generally provides a better result than ML algorithms.

This study presents a systematic literature review (SLR) to find the applications of ML-based methods in code smell detection. The investigation examined various factors that influence the performance of code smell detection, including the ML algorithm employed, feature selection, evaluation metrics for quality assessment, software metrics, and datasets. This review process involved scrutinizing primary research references from diverse datasets. The code smell detection model is constructed based on an analysis of existing datasets, while the necessary information for the detection model is derived from software metrics. Software practitioners need a quality product that is easily maintainable and error-free. This study will help software professionals to find efficient methods and techniques to eliminate code smells at the early stages of development and ultimately build high-quality products in time. The following are some benefits of code smell detection:

- The better the code smell detection process, the better the software quality.
- Software metrics improve the detection process by focusing on different aspects of the software.
- The possibility of refactoring can be easily identified using this process.

Motivation and Contribution

The existing SLR discussed various ML algorithms, but class-wise categorizations need to be included [11–16]. This SLR illustrates multiple studies of several classes of ML algorithms, the distribution of their subclasses, and the appropriate percentage for each study. This SLR also examined various sources to evaluate the effectiveness of various ML algorithms. Researchers presented and discussed many SLRs, datasets, and performance analysis tools but could not discuss software metrics that are found to be more helpful and the dataset language used in various studies [11,14,16].

The contributions of this SLR are listed below:

i. The survey presents a unique approach to code smell detection, examining the SLR of forty-two papers (ARTN1 to ARTN42) from 2005 to 2024.
ii. This study not only identifies various factors that can impact software quality, such as code smells, software metrics, and datasets, but also provides practical guidelines for addressing these issues. These guidelines include best practices for testing procedures, popular software metrics, and methods for detecting code smells, offering valuable insights for software practitioners. We investigate various factors that may distress any software. The survey also provides some guidelines for handling such problems. These guidelines are theoretical concepts and practical tools that empower professionals to address software quality issues effectively.

iii. We analyzed different issues in the code smell detection space and examined the current solution to those problems. Identifying and analyzing issues involves systematically searching and gathering relevant research articles and popular tools used in code smell detection. Examining the solution involves critically analyzing the methodologies, algorithms, and frameworks developed to address these issues. Based on our rigorous analysis, we identified best practices. This comprehensive approach reassures the audience about the validity and reliability of our findings, instilling confidence in the thoroughness of our research.

iv. Different ML algorithms were analyzed in this field, and it was found that the ML algorithms performed better in most cases. In this SLR, different classes of ML algorithms used for code smell detection (Decision Trees, Ensemble Learners, Bayesian Learners, Rule-Based Learning, Support Vector Machines, neural networks, and miscellaneous) are analyzed with their different algorithms.

The primary goal of conducting this study is to examine, assess, and review the code smell detection methods from the following perspectives:

i. The ML algorithms are used in this field.
ii. Code smells that are being focused on in the literature.
iii. Identification of frequently used metrics.
iv. Identification of the metrics that appeared to be more effective.
v. Identification of various datasets.
vi. Identification of datasets of different languages frequently used in this area.
vii. Outlining the other performance measures used with various ML algorithms.

Figure 1 represents the different SLR states, such as the planning, conducting, and reporting states. The planning stage contains two steps: Step 1: Originate the problem/need to conduct the SLR, define the specific requirement for the survey, and identify the object the SLR seeks to address. Step 2 is to expand the review protocol, which identifies research questions that need to be addressed in the SLR. These research questions are extracted from previously conducted SLRs and motivated by previous studies in this field. The related work is included in Section 2. Section 3 regards conducting the SLR process. In this stage, search questions are framed first (mentioned in Section 3.2); in the next step, these questions are searched in the literature from a list of resources (mentioned in Section 3.2). In the next step, the relevant articles from the previous step are selected. After this step, irrelevant research articles are eliminated by considering a selected research article's abstract, title, and conclusion. The next step is quality evaluations, which check the quality of selected research articles using assessment questions (mentioned in Section 3.3). The evaluation score (mentioned in Table A7) is evaluated, and based on selection criteria (mentioned in Section 4.1.2), the poor-quality papers are eliminated in the next step. After this step, relevant research articles are analyzed, as mentioned in Section 3.4. The primary goal of this analysis is to present facts and figures and answer each research question. Section 4 reports findings and discussions. Section 5 reported possible validity threats. Appendix A illustrates the selected research articles and their scores on different assessment criteria.

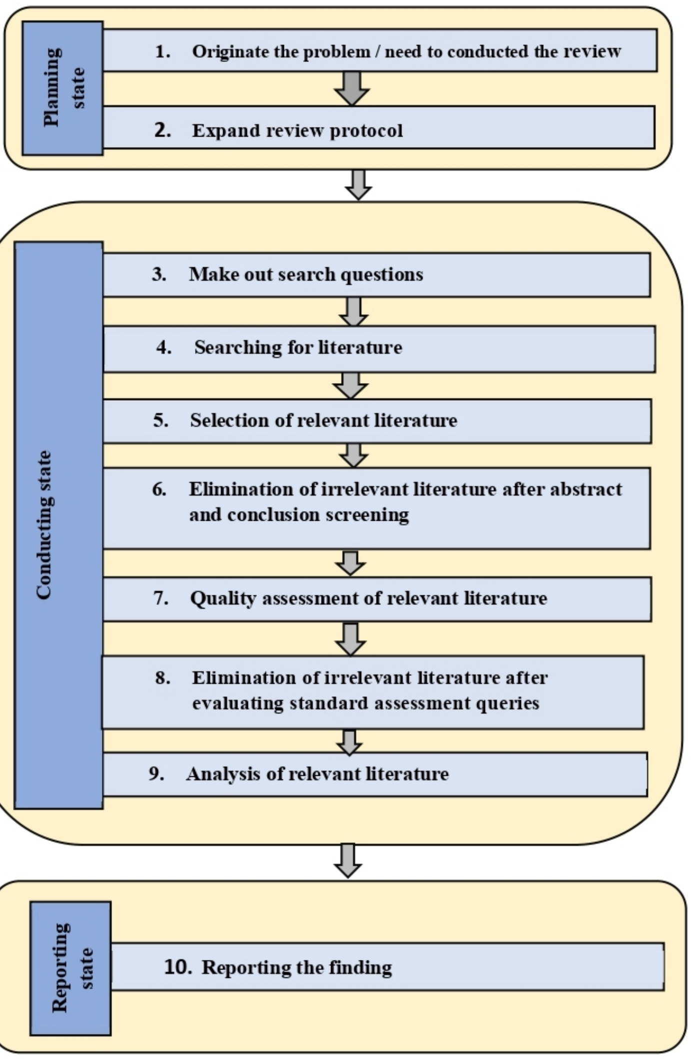

Planning state
1. Originate the problem / need to conducted the review
2. Expand review protocol
Conducting state
3. Make out search questions
4. Searching for literature
5. Selection of relevant literature
6. Elimination of irrelevant literature after abstract and conclusion screening
7. Quality assessment of relevant literature
8. Elimination of irrelevant literature after evaluating standard assessment queries
9. Analysis of relevant literature
Reporting state
10. Reporting the finding

Figure 1. Systematic literature review steps.

2. Related Works

Table 1 shows a comparative analysis of our study with existing literature review studies. Singh et al. [17] analyzed 238 research papers between 1990 and 2015 and discussed the detection of code smells for refactoring. Sobrinho et al. [18] discussed which types of code smells are more extensively researched, reviewed 531 research articles between 1990 and 2017, and connected the researchers studying bad smells other than Duplicate Code using methodological aspects.

Table 1. Comparison with existing studies.

Sr. No.	Author	Year	No. of Research Papers Studied.	Period	Type	Considered Aspects
1	Zhang et al. [19]	2011	39	2000–2009	SLR	Code smell identification, tools, and methods to detect smells. Taxonomy of classifying code smell.
2	Rasool et al. [20]	2015	109	1999–2015	SLR	Code smell detection, classification, comparison, and evaluation detection tools.

Table 1. *Cont.*

Sr. No.	Author	Year	No. of Research Papers Studied.	Period	Type	Considered Aspects
3	Fernandes et al. [21]	2016	107	2005–2014	SLR	Detection tools for code smell, feature evaluation of detection tools, detection techniques, discuss quantitative and qualitative data about the tools.
4	Gupta et al. [22]	2017	60	1996–2016	SLR	The impact of the presence of code smell on the software, as well as the detection method and their correlation.
5	Sharma et al. [23]	2017	445	1999–2016	SLR	Characteristics of code smell, detection methods, tools, the impact of smells on productivity.
6	Sobrinho et al. [18]	2017	351	1990–2017	SLR	Connecting the researchers studying bad smells other than Duplicate Code, methodological aspects, the effect of the combination of bad smells' presence, and improvement in code smell detection.
7	Haque et al. [24]	2018	NA	NA	Survey	Detection approach-based, described its impacts on software, discussed methods to identify the smell.
8	Singh et al. [17]	2018	238	1990–2015	SLR	Detection of code smell for refactoring with different detection approaches.
9	Caram et al. [11]	2019	26	1999–2016	Mapping Study	Used search-based techniques and applied ML algorithms for code smell identification.
10	Kaur et al. [12]	2019	NA	NA	Review	Comparative study on code smell detection, search-based technique.
11	Azeem et al. [13]	2019	15	2000–2017	SLR	ML algorithms are applied for detection and considered independent variables, dependent variables, evaluation metrics, and performance meta-analysis (impact of the independent variable, ML algorithm performance, training strategies).
12	Kaur et al. [12]	2020	20	2005–2020	Review	Frequently used ML algorithms, detected bad smells, and code smell detection using ML and hybrid approaches.
13	Reis et al. [15]	2021	83	2000–2019	SLR	ML algorithms, visualization-based methods, 3D visualization methods, interactive ambient visualization, city metaphors, polymetric views, or graph models.
14	Shaaby et al. [16]	2020	17	2005–2018	SLR	ML algorithm to detect code smells, 27 code smells used for study, 16 ML algorithms, and ensemble techniques.
15	Zhang et al. [25]	2024	86	2010–2023	SLR	Analyzed supervised learning-based code smell detection.
16	Our Work		42	2005–2024	SLR	ML algorithms are used to detect code smells, class-wise categorizations, code smells considered for the study, metrics, datasets, dataset language, and performance measurements of ML algorithms.

Note: NA means 'Not Available'.

Sharma et al. [23] reviewed 445 papers that reported code smell identification tools, techniques, characteristics, and the impact of smells on productivity between 1999 and

2016. Azeem et al. [13] conducted an SLR with the help of ML algorithms and emphasized commonly used methods of independent and dependent variables for code smells. Kaur et al. [14] performed an SLR using ML algorithms and emphasized improvement in software quality. Reis et al. [15] discussed ML algorithms, visualization-based approaches, 3D techniques, and interactive ambient visualization and reviewed 83 relevant papers between 2000 and 2019. Shaaby et al. [16] reviewed 17 articles between 2005 and 2018, studied 27 code smells, and discussed 16 ML and ensemble techniques. The earlier SLRs covered different ML algorithms; however, no class-wise classifications exist. This SLR provides a unique perspective by showing the distribution of subclasses of various ML algorithms, the number of studies in each class, and the correct proportion for each research. The researchers provided and discussed many SLRs, datasets, and performance analysis tools. However, they did not address the language used by the datasets (mentioned in Section 4.2.6).

3. Procedures of Code Smell Detection Approaches

Figure 1 presents a comprehensive methodology for planning, accessing, and analyzing the process. In the planning stage, the problem is organized, and the review protocol, a detailed plan for conducting the literature review, is expanded. This stage involves identifying the problem and steps to find it in the literature, ensuring a thorough understanding of the issue at hand. Furthermore, the conducting stage undertakes the following series of tasks carefully designed to contribute to the overall research process.

Designing search questions, searching for digital libraries, and selecting the relevant literature are based on rigorous and thorough search strategies (discussed below in Section 3.2). The elimination process involves a careful review of the abstract and conclusion of each study. This process ensures that only studies that meet the research objectives and quality standards are included in the review. The process is further refined through the use of assessment queries (discussed below in Section 3.3, Table 2), which are specific questions designed to assess the relevance and quality of each study.

Table 2. Research questions.

RQ-No.	Research Questions	Motivation
RQ-1	Which ML algorithms are used to detect code smell?	Study various sources to evaluate the performance of different ML algorithms [11–16].
RQ-2	Which code smells are measured?	Detection of popular code smells with ML algorithms [11–16].
RQ-3	What software metrics are used for code smell detection?	Identification of metrics frequently used to detect code smells [13,16].
RQ-3.1	Which metrics were found more useful?	Identify the most critical metrics for detecting code smells [13,16].
RQ-4	What are the various datasets used for code smell detection?	Find out which datasets are used to detect code smells [12,14,16].
RQ-4.1	What are the various dataset languages used for code smell detection?	List the different dataset languages that are frequently used [12,14].
RQ-5	What are the performance measurements of the different ML algorithms used?	Find different evaluation techniques to measure the results of various ML algorithms [11,13,14,16].

3.1. Research Questions

After studying many research articles, the questions have been listed in a simplified manner. Table 2 shows the list of research questions (RQ); some questions have sub-questions. RQ-1, RQ-2, and RQ-5 incorporated complete research questions, and RQ-3 and RQ-4 incorporated sub-questions. RQ-1 studies various sources to evaluate the performance of different ML algorithms. This question explores different ML algorithms to detect smells and determine which methods perform better. RQ-2 discovers popular code smells with the ML algorithms. This question aims to collect information about code smells in studies nowadays. RQ-3 identifies critical metrics used for the detection of code smells. RQ-3.1 identifies frequently used software metrics. RQ-4 aims to know the different datasets used for identifying code smells. RQ-4.1 collects information on datasets and the used dataset languages. RQ-5 discusses different evaluation techniques to evaluate the results of various ML algorithms.

In Appendix A, the most relevant 42 research articles are tabulated. These articles are not just a collection of information, but they are the foundation of our study, providing practical insights and guiding our research. Each article has been assigned a unique attribute research article number (ARTN) for this SLR, ensuring their uniqueness, efficiency, and accessibility. Tables A2–A5 include the ARTN, year, tool/technique, smells, algorithms, dataset, performance metrics, and software metrics attributes, all crucial for our research. Tables A2–A5 provide detailed insights into each aspect of the reviewed research articles, facilitating easier navigation and interpretation of the data, thereby enhancing the overall understanding of the research.

3.2. Search Strategy

A robust search strategy is essential for a comprehensive SLR. The search strategy functions as a detailed map, facilitating the identification and selection of relevant study articles. Our search is not only thorough but also comprehensive and accurate, using PICO (Population, Intervention, Comparison, and Outcome). PICO serves as a comprehensive framework for creating research queries. We also employed Boolean expressions 'AND' and 'OR' to expand the scope of our search. These expressions allow for the inclusion of multiple search terms and variations, increasing the likelihood of finding relevant articles. For example, a research query might include questions such as the following:

What are the prevalent code smells [Outcomes/Result] detectable [Population/Space] utilizing ML algorithms? [Intervention]

The alternate string/words of the search question are as follows.

- The following possible words are included in search questions to find articles belonging to 'code smell'.

Code Smells—('code smell' OR 'code smells' OR 'code bad smells' OR 'bad smells' OR 'bad code smells' OR 'anomalies' OR 'antipatterns' OR 'antipattern' OR 'design defect' OR 'design fault' OR 'design-smells' OR 'design flaw').

- The following possible words are included in search questions to find articles belonging to 'machine learning'.

Machine Learning—('machine learning' OR 'supervised learning' OR 'classification' OR 'Machine Learning-based' OR 'regression' OR 'unsupervised learning').

- The following possible words are included in search questions to find articles belonging to 'prediction'.

Prediction—('prediction' OR 'detection' OR 'identification' OR 'prediction model' OR 'detection model' OR 'model').

- All search strings are combined using Boolean operators to make the final search string:

(('code smell' OR 'code smells' OR 'code bad smells' OR 'bad smells' OR 'bad code smells' OR 'anomalies' OR 'antipatterns' OR 'antipattern' OR 'design defect' OR 'design fault' OR 'design-smells' OR 'design flaw') AND/OR ('machine learning' OR 'supervised learning' OR 'classification' OR 'Machine Learning-based' OR 'regression' OR 'unsupervised learning') AND/OR ('prediction' OR 'detection' OR 'identification' OR 'prediction model' OR 'detection model' OR 'model'))

The final search string is searched in the following list of resources to select the research article for the literature review:

i. Springer

ii. ACM Digital Library
iii. IEEE Xplore
iv. Wiley Online Library
v. Google Scholar
vi. Scopus
vii. Science Direct
viii. Web of Science

The inclusion and exclusion of relevant articles are based on several selection and elimination rules. These are the following:

Selection rules:

i. Experimental results are based on ML algorithms.
ii. Results of studies based on non-ML algorithms.
iii. Experimental results are based on comparing statistical techniques and are ML-based.

Elimination rules:

i. Research articles on the ML algorithms without investigational study.
ii. Research articles on ML algorithms other than code smell detection.
iii. The research article results of the same author in a conference or journal are similar.

The research article is placed if any of the conditions in the research article selection rule are met. If any conditions in the research article elimination rule are not met, the research article is not included. The complete selection or elimination rules required are shown in Table 3. This table has three possible values, 'No', 'Partly', and 'Yes', to examine the research papers and assign the score. This is further discussed in Section 3.3.

Table 3. Assessment questions.

Que-No.	Standard Assessment Questions
Que-1	Are the objectives of the research clearly stated?
Que-2	Is the size of the dataset sufficient?
Que-3	Are the results and different evaluation techniques clearly explained?
Que-4	ML algorithms used are sufficiently defined and justified.
Que-5	Different metrics for various code smells are selected.
Que-6	Are there threats to the validity of the research mentioned?
Que-7	Are all independent variables clearly defined?
Que-8	What is the contribution to future research?
Que-9	Is a comparative analysis of different techniques available?

Figure 2 shows the procedure for selecting the research articles from the listed resources. Initially, 311 articles are selected (2005 to 2024) through listed resources with the help of search strategies. In the next step, 265 articles are excluded by considering the articles' titles, abstracts, and conclusions. The remaining 46 articles (2005 to 2022) are focused on for full-text evaluation. A total of 4 articles were eliminated in the quality assessment process because their research score was below 2, and finally, 42 articles were included in the review process. The selection of research articles in this review process depends on the availability of articles in selected databases.

Figure 3 presents the number of research articles reviewed in this SLR and their year of publication. This figure also highlights the impact of the research. In 2019, the graph showed the highest peak, indicating a significant year in which numerous ML algorithms were proposed in research studies. This underscores the importance and relevance of the research findings in the field of machine learning and code smell detection.

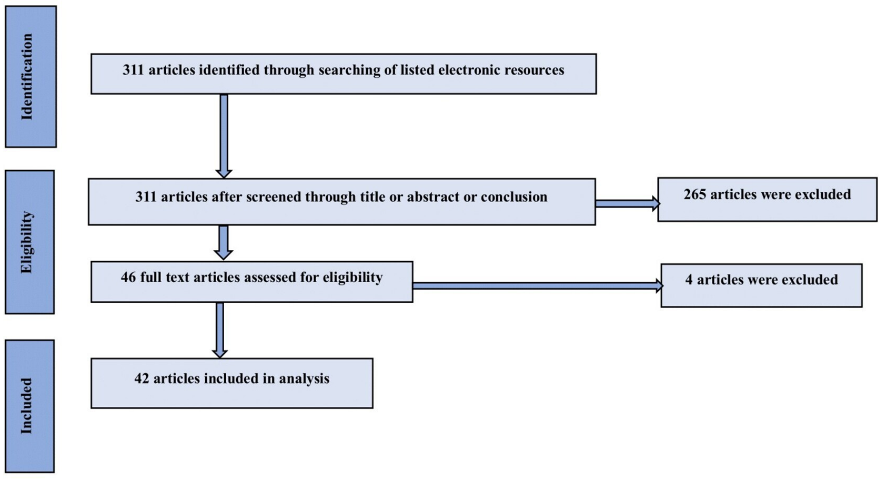

Identification
Eligibility
Included
311 articles identified through searching of listed electronic resources
311 articles after screened through title or abstract or conclusion
265 articles were excluded
46 full text articles assessed for eligibility
4 articles were excluded
42 articles included in analysis

Figure 2. A systematic literature review flow diagram of article inclusion.

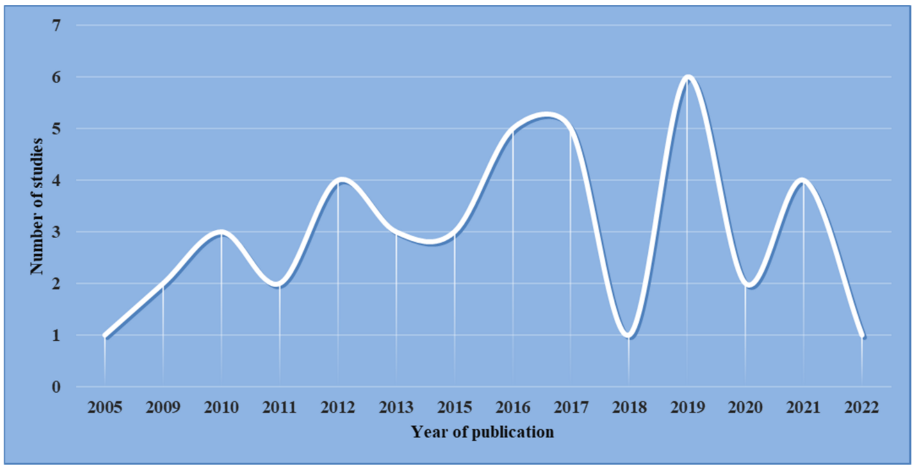


Figure 3. Distribution of studies during different years.

3.3. Evaluation of Research Article

Our evaluation process is meticulously designed, with standard assessment questions (Que) formulated (Table 3) to evaluate the correctness and weakness of research articles. This provides a clear and objective basis for assessing the articles' quality. The questions are assigned the research scores 0, 0.5, and 1, where 0, 0.5, and 1 represent the objective of the standard assessment question being not defined, partially defined, and well defined, respectively. The evaluated research score is secured, ranging from 0 to 9. The evaluation for each article is shown in Table A7. After a detailed analysis, selection, and elimination of articles, as shown in Table A1, this study found that 36 belong to above-average and 6 research articles belong to below-average categories.

3.4. Analysis of the Research Article

Each research paper undergoes a comprehensive analysis to determine whether the research questions were answered by the initial studies (Table A6 shows the information with assessment query score). The overview of the title, author's name, publication details, dataset information, machine learning algorithm, and metrics utilized are outlined. The details of each research question analysis taken for the SLR have been recorded. The main objective of the analysis is to show facts and figures and address each research question. The performance is evaluated using measures like area under the ROC Curve (AUC ROC), accuracy, etc. This evaluation helps to know the strengths and weaknesses of

ML algorithms, datasets, and feature selection techniques in code smell detection. Column charts and bar charts are used to explain the research.

4. Findings and Discussion

The SLR division conducted a comprehensive discussion of various research articles, recognizing their significant contributions to the field. It provided a summary of the articles, with each section providing a detailed analysis of the research articles. Finally, it concluded with the valuable information and insights gleaned from this study, underscoring the importance of these research articles.

4.1. Description of Articles

This section meticulously explains the most relevant research articles. Forty-two research papers were carefully selected based on their quality. This study exclusively focused on research papers from various journals and conferences where ML algorithms were applied. Each research article was scrutinized to extract critical information and find answers to all research questions.

4.1.1. Publication Sources

The list of publications included in Table 4 is of particular significance from the perspective of code smell detection. The research articles are organized by publication name, type, ISSN number, number of studies, and impact factor. For instance, there are seventeen research articles from IEEE indicating its prominence in this field, and one research paper is from Electronic Notes in Theoretical Computer Science, Software Quality Journal, and Journal of Computer Science and Technology, showcasing their contributions. Four articles from Empirical Software Engineering, etc., further highlight the sources' diversity.

Table 4. Publication summary.

Name of Publication	Type	ISSN Number	Impact Factor	No of Studies
Electronic Notes in Theoretical Computer Science	Journal	15710661	0.357	1
Software Quality Journal	Journal	09639314, 15731367	2.1	1
Journal of Computer Science and Technology	Journal	18604749, 10009000	1.9	1
International Journal of Rough Sets and Data Analysis	Journal	2334-4598	NA	1
International Journal of Electrical and Computer Engineering (IJECE)	Journal	2088-8708	0.376	1
International Journal of Computer Sciences and Engineering	Journal	2347-2693	3.802	1
Elsevier The Journal of Systems and Software	Journal	1641212	2.829	1
IEEE Access	Journal	2169-3536	3.476	2
Empirical Software Engineering	Journal	1382-3256, 1573-7616	3.762	4
Tech Science Press Computers, Materials and Continua Tech Science Press	Conference	NA	NA	1
Scopus, IEEE, and Science Direct	Conference	NA	NA	1
Scopus and IEEE	Conference	NA	NA	2
International Joint Conference on Computer Science and Software Engineering (JCSSE)	Conference	NA	NA	1
International Conference on Quality Software	Conference	NA	NA	1
International Conference on Machine Learning and Data Science	Conference	NA	NA	1
International Conference on Intelligent Computing and Control Systems (ICICCS 2019) IEEE Xplore	Conference	NA	NA	1

Table 4. *Cont.*

Name of Publication	Type	ISSN Number	Impact Factor	No of Studies
International Conference on Enterprise Information Systems (ICEIS 2017)	Conference	NA	NA	1
International Conference on Computer Science and Information Technology (CSIT)	Conference	NA	NA	1
International Conference on Advanced Information Networking and Applications	Conference	NA	NA	1
IEEE Transactions on Software Engineering	Conference	NA	NA	1
IEEE International Conference on Software Maintenance	Conference	NA	NA	1
IEEE	Conference	NA	NA	6
Conference on Software Maintenance and Reengineering	Conference	NA	NA	1
Conference on Reverse Engineering	Conference	NA	NA	1
ASE’16, 3–7 September 2016, Singapore, Singapore	Conference	NA	NA	2
ASE’12, 3–7 September 2012, Essen, Germany	Conference	NA	NA	1
2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)	Conference	NA	NA	1
IEEE 24th International Conference on Program Comprehension (ICPC)	Conference	NA	NA	1
International Conference on the Quality of Information and Communications Technology	Conference	NA	NA	1
IEEE ASE 2013, Palo Alto, USA	Conference	NA	NA	1
Conference on Reverse Engineering	Conference	NA	NA	1
Grand Total				42

Note: NA means ‘Not Available’.

4.1.2. Quality Evaluation

The assessment queries are used to evaluate the values of the research article. Based on the research score (RS), the article is divided into three categories:

(i) Above average: RS >5;
(ii) Below average: $2 \leq$ RS ≤ 5;
(iii) Elimination: RS < 2.

4.2. Rationalize the Research Questions

4.2.1. RQ1: Which ML Algorithms Are Used to Detect Code Smells?

The different ML algorithms used are shown below.

i. Decision Trees;
ii. Ensemble Learners;
iii. Bayesian Learners;
iv. Rule-Based Learning;
v. Support Vector Machines;
vi. Neural networks.

Figure 4 illustrates different ML algorithms used to detect code smells. The ML algorithms are divided into subcategories.

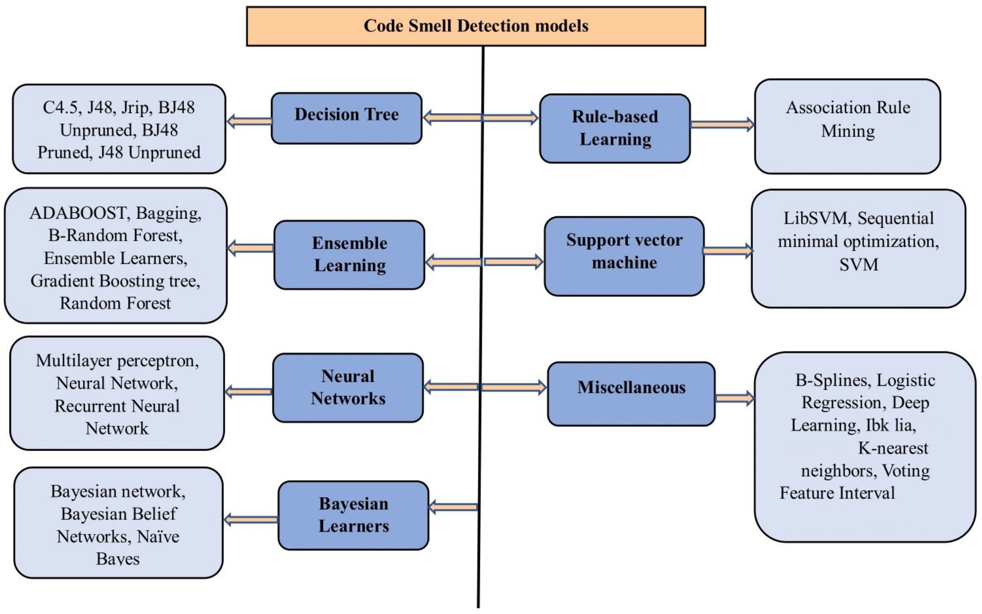

Code Smell Detection models
Decision Tree
C4.5, J48, Jrip, BJ48 Unpruned, BJ48 Pruned, J48 Unpruned
Ensemble Learning
ADABOOST, Bagging, B-Random Forest, Ensemble Learners, Gradient Boosting tree, Random Forest
Neural Networks
Multilayer perceptron, Neural Network, Recurrent Neural Network
Bayesian Learners
Bayesian network, Bayesian Belief Networks, Naïve Bayes
Rule-based Learning
Association Rule Mining
Support vector machine
LibSVM, Sequential minimal optimization, SVM
Miscellaneous
B-Splines, Logistic Regression, Deep Learning, Ibk lia, K-nearest neighbors, Voting Feature Interval

Figure 4. Different ML algorithms used for detecting code smells.

Table 5 shows the different categories of ML algorithms, the number of studies, and their percentage out of 42 (total number of selected articles for this SLR). For example, ensemble learning is used in 21 research articles out of the selected research articles, and their percentage out of 42 is 50%. It also indicates that Decision Tree, SVM, and ensemble learning techniques are often used in code smell detection. Some methods are more often used, so the total percentage is more than 100 in the table, and the number of studies in the table is more than the total number of research articles considered for this study.

Table 5. Machine learning-based code smell detection techniques.

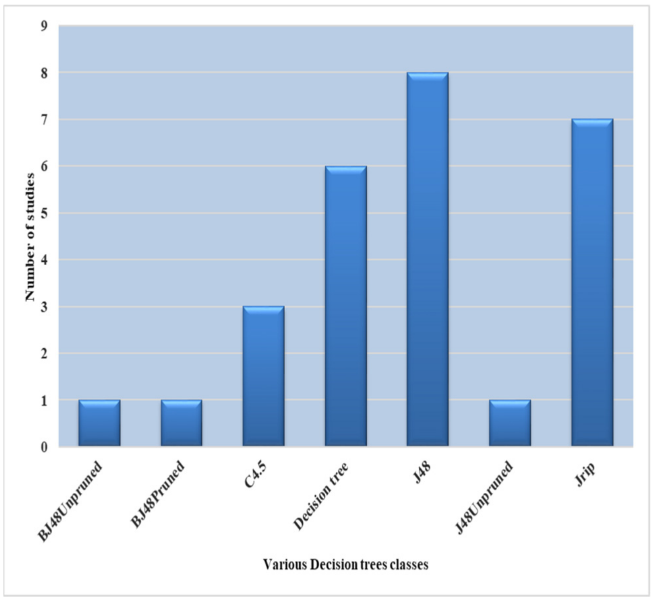
ML Algorithms	Number of Research Studies (Out of 42)	% of Research Studies (Out of 42)
Bayesian learning	18	42.85714286
Decision Tree	27	64.28571429
Ensemble learning	21	50
Miscellaneous	12	28.57142857
Neural network	7	16.66666667
Rule-Based Learning	3	7.142857143
SVM	22	52.38095238

Nucci et al. [7] applied J48, Random Forest, Naive Bayes, LIBSVM, JRip, ADABOOST, and Sequential Minimal Optimization over 74 software systems. Pecorelli et al. [10] used Naive Bayes, J48, Random Forest, JRip, and SVM over JAVA projects and found optimal results. Figures 5 and 6 illustrate the ML algorithms used in code smell detection. Figure 5 indicates the different methods applied in research studies: Decision Tree, Ensemble Learners, Bayesian Learners, Rule-Based Learning, Support Vector Machines, neural networks, and miscellaneous.

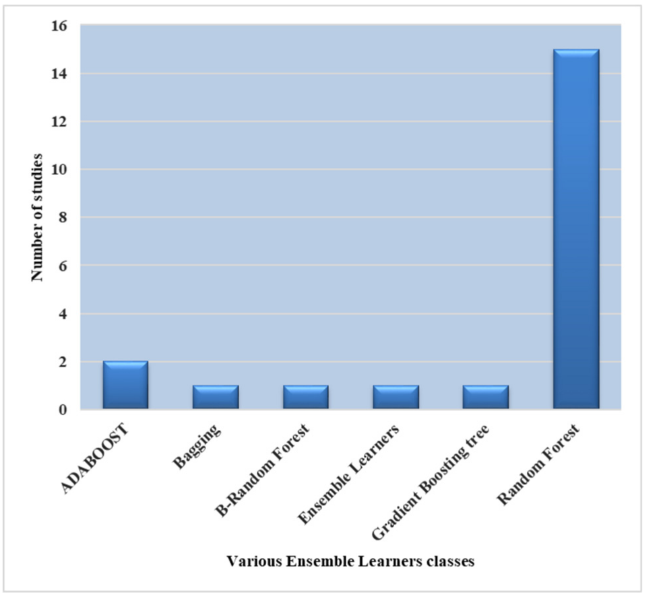
The number of studies on Decision Trees, ensemble learning, Bayesian learning, Rule-Based Learning, SVM, neural networks, and miscellaneous based on code smell detection methods are shown in Figure 5a–g.

Figure 6 shows the ML algorithms used in code smell detection. The research studies that make use of subclass techniques of the Decision Tree, ensemble learning, Bayesian learning, SVM, and neural network approaches are J48 (29%), Random Forest (71%), Naive Bayes (72%), SVM (54%), and Multilayer Perceptron (72%). These findings indicate the prevalence of these ML algorithms in code smell detection, providing valuable insights for software engineers and researchers in the field.

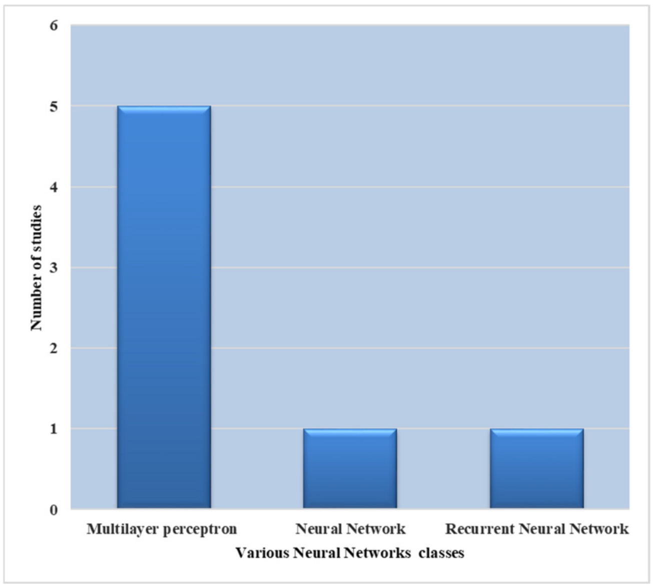
The ML algorithms, deep learning, J48, Naive Bayes, SVM, and Random Forest algorithms are employed. The number of research studies performed using these techniques is 15, 13, 12, 2, and 8, respectively.



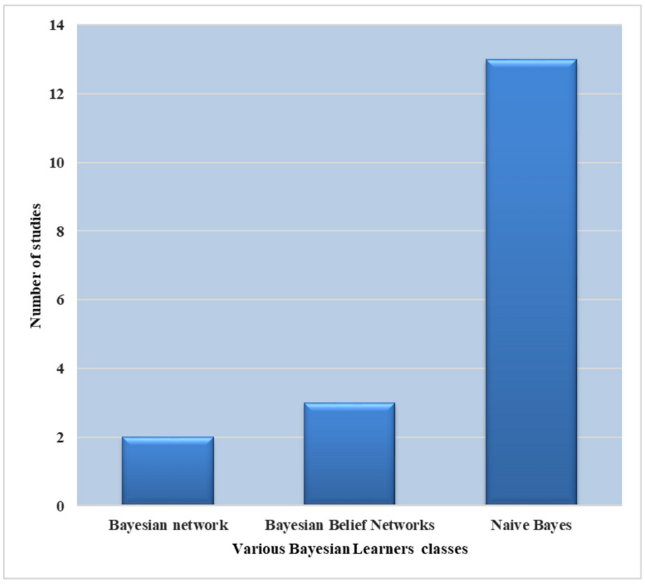
(**a**)



(**b**)

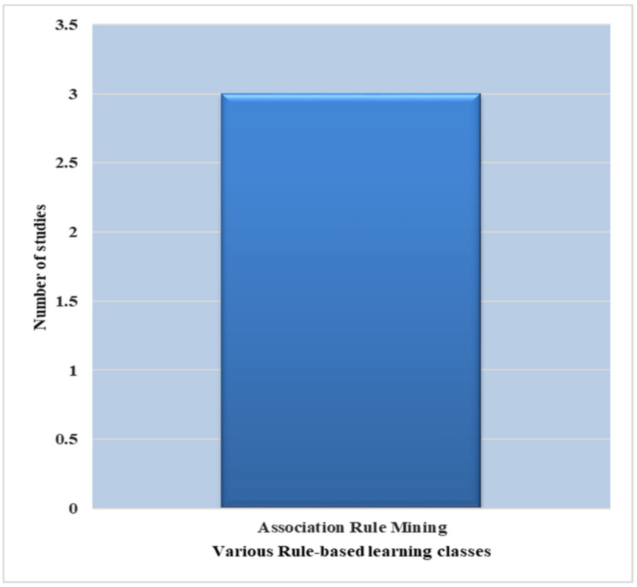


(**c**)

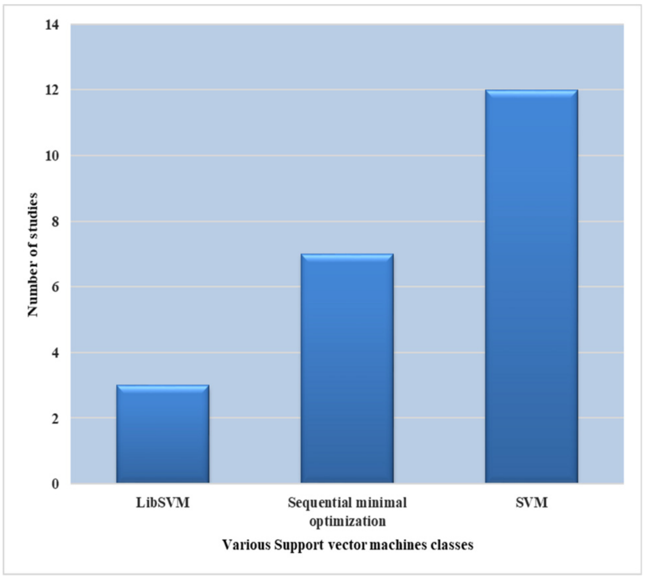


(**d**)

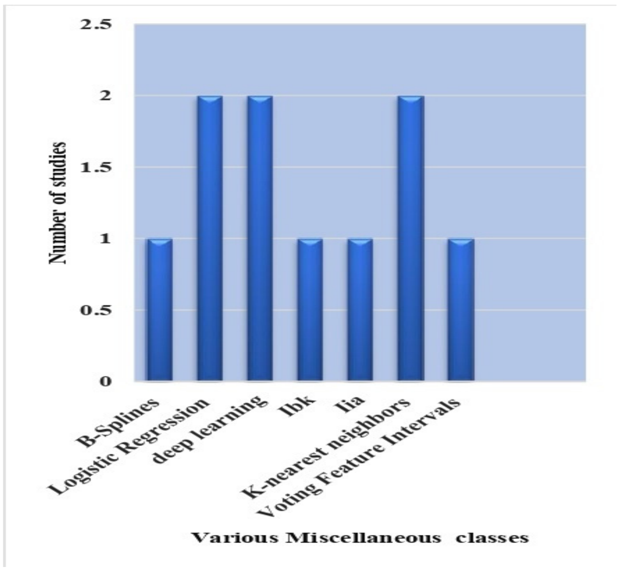
Figure 5. *Cont.*



(**e**)



(**f**)



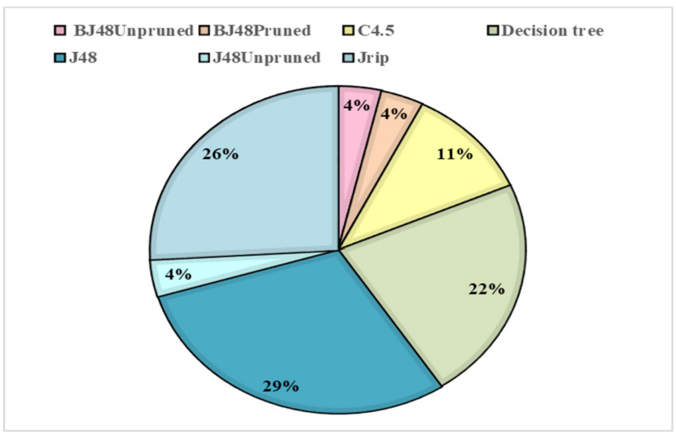
(**g**)

Figure 5. Number of research studies using distinct classes of ML algorithms: (**a**) bar graph of Decision Trees; (**b**) bar graph of Ensemble Learners; (**c**) bar graph of neural networks; (**d**) bar graph of Bayesian Learners; (**e**) bar graph of Rule-Based Learning; (**f**) bar graph of Support Vector Machines; (**g**) bar graph of miscellaneous classes.

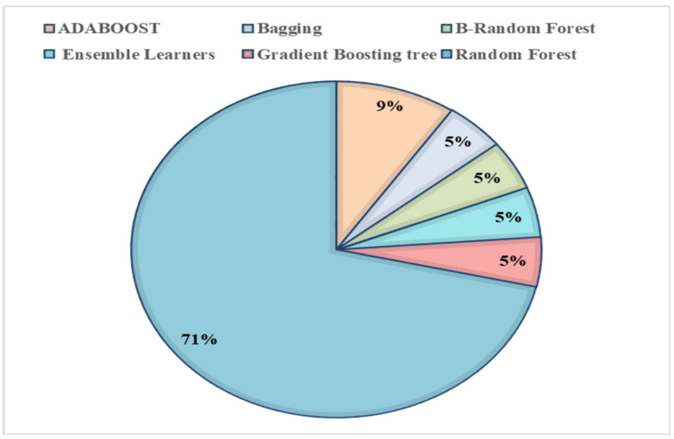
Mhawish et al. [3] applied Random Forest, Gradient Boosting Tree, Decision Tree, deep learning, SVM, Multilayer Perceptron, GA (Genetic Algorithm), Naive Bayes, and GA CFS (Correlation-based Feature Selection) over Fontana et al. [26]. Furthermore, 74 open-source systems were used; the highest and precision values are 96.43% and 98.18%, respectively, for the Random Forest algorithm. Guggulothu et al. [8] used the B-Random Forest, Random Forest, B-J48 UnPruned, B-J48 Pruned, and J-48 Unpruned approaches and found 95–98% accuracy. Fontana et al. [26] utilized J48, C4.5 Decision Tree, JRip, Naive Bayes, Random Forest, Sequential Minimal Optimization, LibSVM over Antipattern Scanner, iPlasma, Fluid Tool, Marinescu detection rule, PMD, and the Fluid Tool dataset and found 96–99% accuracy. Kaur et al. [27] proposed an SVM approach over ArgoUML v0.19.8 and Xerces v 2.7.0 datasets and found better results. Nizam et al. [28] developed a deep learning system for code smell detection. They also used K-nearest neighbors and cosine similarity machine learning algorithms for detailed comparison. Shah et al. [29] proposed a CloudScent model, an open-source methodology for detecting code smells. The results demonstrate that the model can accurately detect eight code smells in the cloud.

4.2.2. RQ2: Which Code Smells Are Measured?

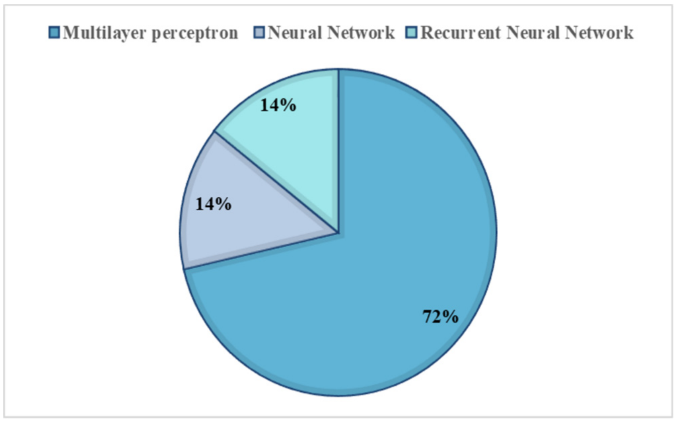
This section discusses code smells detected in various research articles. Figure 7 shows a variety of code smells observed in research articles considered in the study. It was found that researchers focused on more than thirty code smells. Data Class, Long Method, Feature Envy, and God Class are the most frequently analyzed code smells. Articles ARTN1, ARTN7, ARTN9, ARTN17, ARTN27, ARTN31, and ARTN39 include a study about the Lazy class. Articles ARTN10, ARTN16, ARTN18, ARTN21, and ARTN31 concentrated on the analysis of Duplicate Code. Spaghetti Code smells are discussed in the research articles ARTN5, ARTN8, ARTN11, ARTN12, ARTN30, and ARTN39. Research article ARTN17 discusses 11 code smells. The code smells are Lazy Class, God Class, Swiss Army Knife, Long Parameter List, etc. In this SLR, God Class code smell detection is found in a maximum number of research articles. Research article ARTN39 discusses nine code smells: Long Method, Spaghetti Code, Feature Envy, Parallel Inheritance, Large Class, Data Class, Lazy Class, Functional Decomposition, and Long Parameter List.



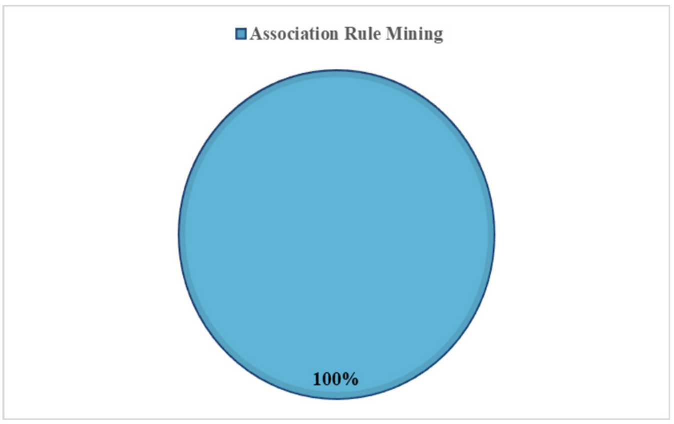
(**a**)



(**b**)

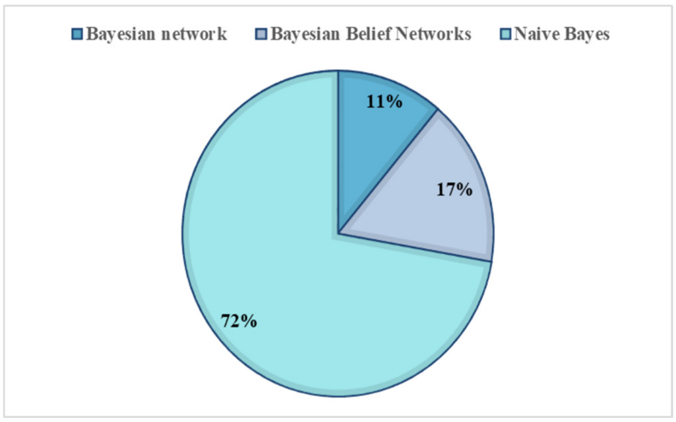


(**c**)

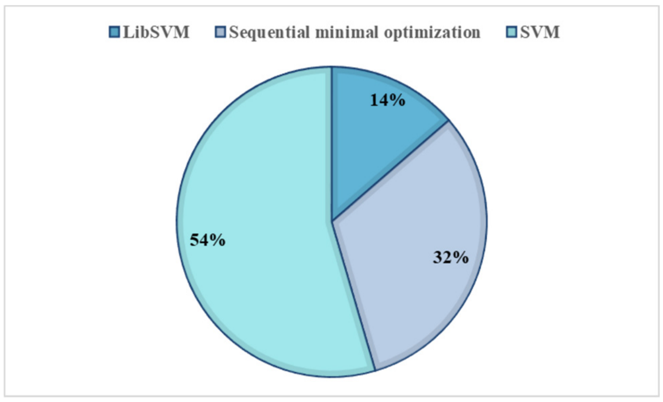


(**d**)

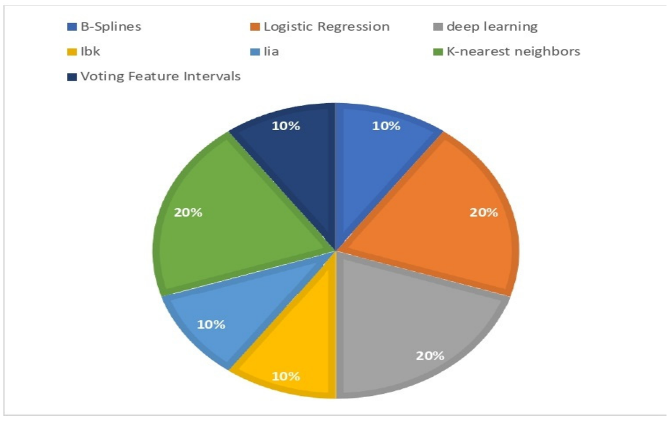
Figure 6. *Cont.*



(**e**)



(**f**)



(**g**)

Figure 6. Subclasses' distribution of the Decision Tree, Bayesian learning, ensemble learning, SVM, Rule-Based Learning, neural network, and miscellaneous classification algorithms with appropriate study percentages: (**a**) Decision Tree-based code smell detection techniques; (**b**) Ensemble Learner-based code smell detection; (**c**) neural network-based code smell detection techniques; (**d**) Rule-Based Learning-based code smell detection techniques; (**e**) Bayesian Learner-based code smell detection techniques; (**f**) Support Vector Machine-based code smell detection approaches; (**g**) miscellaneous-based code smell detection techniques.

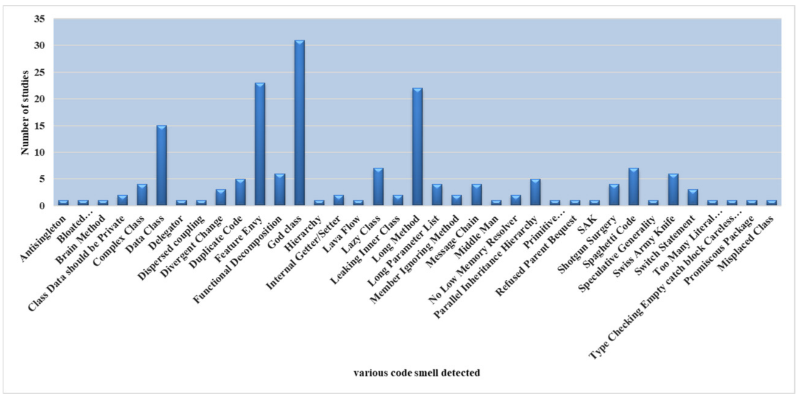


Figure 7. Number of studies of different code smells.

4.2.3. RQ3: What Software Metrics Are Used for Code Smell Detection?

Research articles employ a variety of software metrics, and in this study, more than 290 metrics are used. Figure 8 shows these metrics, arranged from most to least frequently used. This figure includes only the most and least used metrics for clarity. The following section will delve into the significance and application of these software metrics, providing a comprehensive understanding of their use in code smell detection.

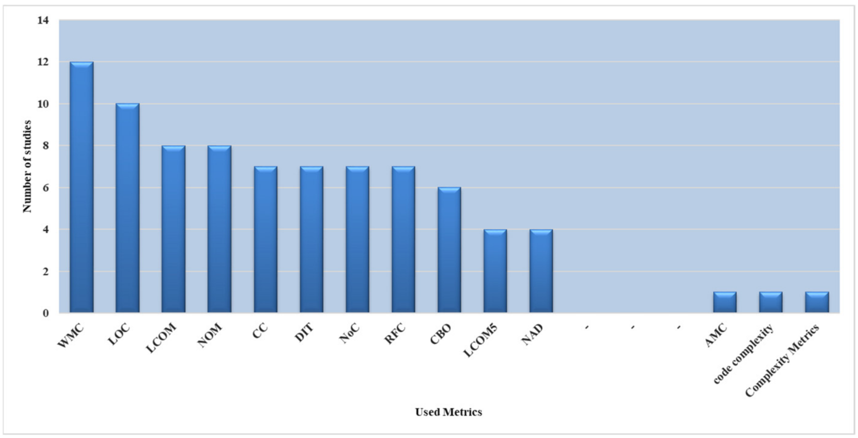


Figure 8. List of frequently used metrics for code smell detection. '-' represents some software metrics between the most and least frequently used metrics for clear visibility.

Object-oriented metrics study various features of object-oriented software like coupling, cohesion, and reusability in articles ARTN1, ARTN4, ARTN6, ARTN7, ARTN9, ARTN11, ARTN12, ARTN13, ARTN15, ARTN16, ARTN23, and ARTN38. The line of code (LOC), a traditional and frequently used software metric, is also considered in this research. Effective line of code (ELOC) is another metric that has been applied in several articles (ARTN13, ARTN17, ARTN21, ARTN27, ARTN30, ARTN31, ARTN32, ARTN34, ARTN38, and ARTN39). LCOM (Lack of COhesion in Methods), a metric that counts the number of method pairs where their class properties are not shared, is used in articles ARTN2, ARTN8, ARTN22, ARTN23, ARTN27, ARTN30, ARTN32, ARTN34, ARTN38, and ARTN39. ARTN34, for instance, considered LOCL with a sampling technique over 629 source project datasets.

Other metrics are applied in article ARTN20, such as the Number of Methods Overridden (NMO), Number Of Public Methods (NOPLM), Number Of Non-Accessor Methods (NONAM), Number Of Protected Attributes (NOPRA), NUMBER OF DEFAULT ATTRIBUTES (NODA), Number Of Static Methods (NOSM), Number Of Static Attributes (NOSA), NUMBER OF NON—CONSTRUCTOR METHODS (NONCM), Number Of Constructor Methods (NOCM), and the NUMBER OF FINAL AND STATIC METHODS (NoFSM). Article ARTN8 proposed the BDTEX (Bayesian Detection EXpert) approach with other metrics such as Coupling Between Objects (CBO), a Goal Question Metric (GQM), and the Number of Attributes Declared (NAD) metrics. Draz et al. [30] proposed a search-based system and employed metrics such as LOC, NAD, McCabe (used for software quality and security such as cyclomatic complexity (CC), actual cyclomatic complexity, etc.), Weighted Methods for Class (WMC), number of children (NOC), CBO, LCOM, CC, Response For A-Class (RFC), Number of Parameters (NOPARAM), and the Depth of Inheritance Tree (DIT). Their method had an average precision and recall of 94.24% and 93.4%, respectively, demonstrating the high effectiveness of these metrics in code smell detection.

The frequently used metrics in articles ARTN3, ARTN8, ARTN13, ARTN16, ARTN17, ARTN20, ARTN21, ARTN22, ARTN23, ARTN27, ARTN30, ARTN31, ARTN32, ARTN34, ARTN36, ARTN38, ARTN39, etc., are LOC, NoC, LCOM, WMC, and DIT. These metrics are widely used because they provide valuable insights into the structure and complexity of the code, which are key factors in code smell detection. The article used coupling and cohesion metrics CBO, SQL software metrics for code smell detection in articles ARTN1, ARTN4, ARTN7, ARTN9, ARTN13, ARTN15, ARTN16, ARTN20, ARTN23, ARTN27, ARTN32, ARTN34, ARTN38, and ARTN39, and also applied data sampling techniques to handle imbalanced data in prediction. LOC, WMC, DIT, LCOM, NoC, CC, NOM (Number of

Methods), etc., are the most commonly employed metrics. So, these metrics belong to the category of thorough utility. It also illustrates that AMC, code complexity, complexity metrics, etc., are rarely applied in identifying code smells and fall under the category of partial utility as a result.

4.2.4. RQ3.1: Which Metrics Are Found More Useful in Code Smell Detection?

Our research delved into the use of various software metrics in code smell detection, a crucial aspect of software quality analysis. These metrics, including object-oriented, coupling, complexity, size, cohesion, quality, GQM, LCOM5, NMD, and NAD metrics, have been extensively used in research articles. Among them, LOC, NOM, LCOM, WMC, CC, DIT, etc., have emerged as the most frequently applied metrics, proving their effectiveness in code smell detection. This comprehensive understanding of software metrics in code smell detection significantly contributes to the field.

4.2.5. RQ4: What Are the Various Datasets Used for Code Smell Detection?

Our study has comprehensively analyzed code smell detection by encompassing many datasets. These datasets, such as Xerces, ArgoUML, Gantt Project, Azureus, Eclipse, and object-oriented metrics datasets, are among the most commonly used ones. These are followed by Apache Ant, Fluid Tool, iPlasma, PMD, etc. This diverse range of datasets, each with unique characteristics and challenges, allowed us to gain a more thorough understanding of code smell detection, making our research findings more robust and insightful. Figure 9 shows a few commonly used datasets.

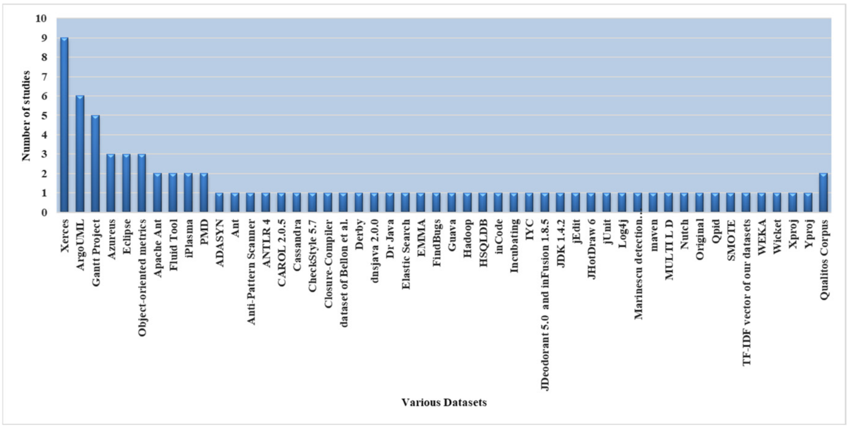


Figure 9. Research conducted on the various datasets used in code smell detection.

4.2.6. RQ4.1: What Are the Various Dataset Languages Used for Code Smell Detection?

A wide variety of dataset languages have been applied in code smell detection. Notably, JAVA emerges as the primary dataset language for code smell detection. The dataset is further categorized into open-source, projects written in JAVA, and any language and varies in size: large, small, and medium. ARTN10 employed C# as the dataset language. ARTN3 used XML with the JAVA dataset language. This diversity in dataset languages underscores the multifaceted nature of code smell detection, as different programming languages may have different code smells and quality issues. Understanding the usage and characteristics of these dataset languages can provide valuable insights for developing language-specific code smell detection techniques. Figure 10 shows the applied dataset languages.

The following are some brief descriptions of a variety of datasets:

- Open-source dataset: It includes freely available software projects such as WEKA, Xerces v2.7.0, IYC, Gantt Project v1.10.2, Eclipse, Guava, Closure Compiler, and jUnit datasets.

- Student dataset: It includes datasets developed by students, such as Fontana et al., who designed datasets using four open-source JAVA projects with the help of three master's degree students [26].
- The language dataset includes a dataset selected from different language projects, such as JAVA, XML, C#, etc. JAVA is the most frequently used language in research articles.
- Others: This study has observed that the main datasets used are open-source datasets belonging to the JAVA language. Researchers in ARTN1, ARTN7, ARTN9, and ARTN10 have also used some private industrial datasets.

This study has observed that the main datasets used are open-source datasets belonging to the JAVA language.

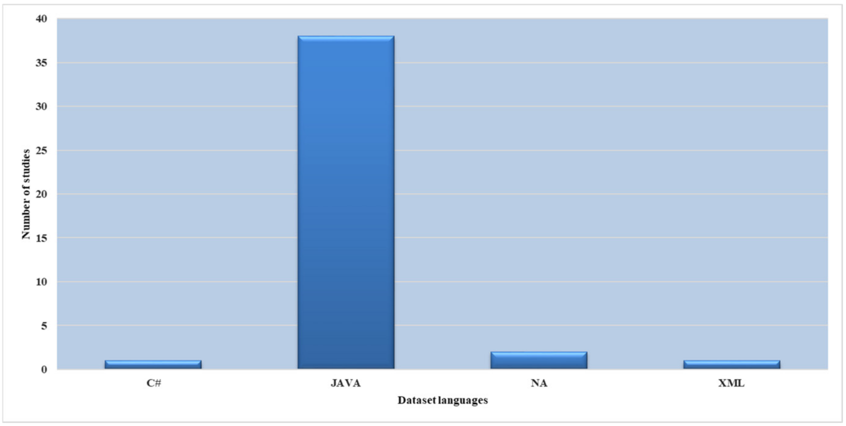


Figure 10. List of dataset languages used for code smell detection.

4.2.7. RQ5: What Are the Performance Measurements of the Different ML Algorithms Used?

Various performance measures are applied in code smell detection, as shown in Figure 11. Some performance measures are effectively used, such as precision (21 times), recall (21 times), accuracy (22 times), and F-measure (18 times). Some rarely used techniques are kept in miscellaneous. Table 6 shows that the performance measures precision, recall, accuracy, F-measure, root mean square error (RMSE), Matthew correlation coefficient (MCC), hamming score, mean absolute error (MAE), etc., are rarely used. ARTN15, ARTN29, ARTN32, ARTN35, ARTN37, and ARTN42 used AUC ROC for performance measurement. ARTN15 and ARTN37 used GMean for performance analysis. These performance measures are crucial in evaluating the effectiveness and efficiency of different ML algorithms in code smell detection.

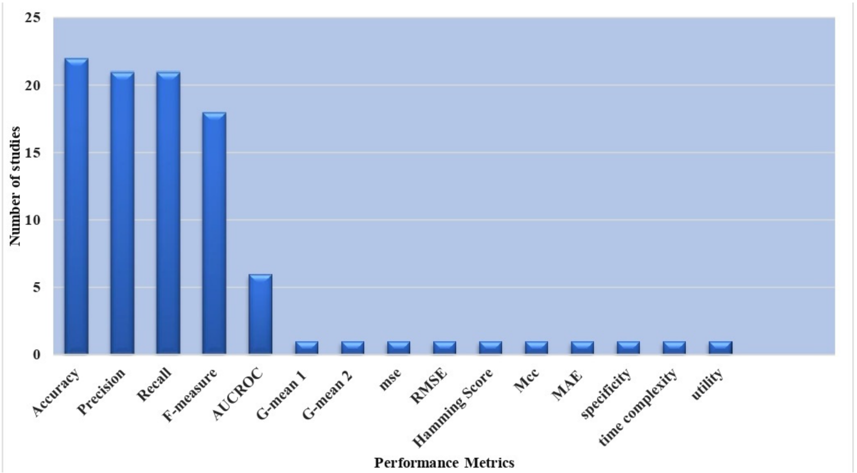


Figure 11. Performance measures used in various types of research.

Table 6. Analyzing performance indicators across studies.

Performance Measures	Description	Research Articles
Accuracy	Out of all projections, how many have been accurate overall?	ARTN1, ARTN2, ARTN6, ARTN7, ARTN9, ARTN15, ARTN18, ARTN20, ARTN22, ARTN24, ARTN27, ARTN29, ARTN32, ARTN33, ARTN34, ARTN35, ARTN36, ARTN37, ARTN38, ARTN40, ARTN41, ARTN42
Precision	It provides data on how well the classifier performs in terms of false positives.	ARTN2, ARTN3, ARTN4, ARTN5, ARTN8, ARTN10, ARTN11, ARTN12, ARTN13, ARTN14, ARTN16, ARTN17, ARTN18, ARTN19, ARTN21, ARTN25, ARTN28, ARTN30, ARTN39, ARTN41, ARTN42
Recall	It provides data on how well the classifier performs in avoiding false negatives.	ARTN2, ARTN3, ARTN4, ARTN5, ARTN8, ARTN10, ARTN11, ARTN12, ARTN13, ARTN14, ARTN16, ARTN17, ARTN18, ARTN19, ARTN21, ARTN25, ARTN28, ARTN30, ARTN39, ARTN41, ARTN42
F-measure	A single score that combines recall and precision.	ARTN13, ARTN14, ARTN15, ARTN16, ARTN17, ARTN19, ARTN29, ARTN30, ARTN32, ARTN33, ARTN34, ARTN35, ARTN36, ARTN37, ARTN38, ARTN40, ARTN41, ARTN42
AUC ROC	The curve between specificity and recall. It calculates the effectiveness.	ARTN15, ARTN29, ARTN32, ARTN35, ARTN37, ARTN42
G-mean (G-mean 1, G-mean 2)	It combines a true positive rate and a true negative rate.	ARTN38
MSE	An average of the square of the differences between the predicted and actual value.	ARTN26
RMSE	It is the MSE value's square root.	ARTN23
Miscellaneous	It includes Hamming Score, Exact Match Ratio, time complexity, standard deviations, effectiveness, and utility.	ARTN7, ARTN8, ARTN10, ARTN23, ARTN24, ARTN26, ARTN30, ARTN31, ARTN35, ARTN37

Our research demonstrated the effectiveness of various ML algorithms in code smell detection. The SVM, Random Forest, J48, and Naive Bayes models have emerged as the most popular and accurate methods, as confirmed by their frequent use in research articles (Table 6). The accuracy, AUC RoC, precision, and recall are manually analyzed from different research articles. The accuracy score, in particular, is often used to measure the model's performance, providing a reliable benchmark for future ML algorithms in code smell detection.

4.3. Opportunities Trends

Software projects are growing more popular due to the digital era's expansion, which has increased the number of modules and the size of these projects. The increased modules can increase the chances of errors in the future and the possibility of severe problems in the software. Code smell detection removes the cause that may create poor performance and future issues in software. It is challenging to find detection rules for code smells; however, some metric-based methods have also been developed. Most of the datasets are from the project/industrial datasets. If various datasets are easily accessible, their detection will be more accurate.

4.4. Insightful Discussion

This section discusses various SLR study findings, which will help software developers to find key points to improve software quality. New researchers can enhance their knowledge for their upcoming research studies. The insights gained from this research include the most effective metrics and dataset languages, the most commonly used datasets, and the performance measurements of different ML algorithms, all of which can guide future research and software development practices.

RQ-1 categorizes ML algorithms into Decision Tree, ensemble learning, Bayesian learning, Rule-Based Learning, SVM, neural network, and miscellaneous. Decision Trees, SVM, ensemble learning, and their subclasses Random Forest, J48, SVM, Naive Bayes, and Multilayer Perceptron are frequently used to identify code smells.

This study found that 27 of 42 research articles used Decision Tree algorithms to identify code smells, as shown in Table 5. Most research studies employ this method because of its excellent performance and widespread availability of relevant software. This study reveals a need for Rule-Based Learning applications to deal with software smells, as only 3 of 42 research articles used this algorithm. More research studies are needed in this area, and future studies may need to use and explore the capability of this algorithm to detect code smells.

RQ-2 discovered that the various researchers employed over 30 code smells more than 140 times in their investigation. Data Class, Feature Envy, God Class, and Long Method are the most frequently used code smells and are given more consideration in recent research, as shown in Figure 7. These code smells may be chosen for the following reasons:

- Many different design concerns are covered.
- They severely compromise the integrity of the program.
- Code smells of these types are the most common type; these code smells are easily recognizable and explained.

Most existing research in ML algorithms has concentrated on a small subset of possible code smells. These studies did not examine all the recommended code smells. Thus, other code smells that significantly impact software quality may need to be noticed. Therefore, there is an opportunity for further research to investigate these unexplored code smells and their effects on software quality [3]. This research gap presents a potential avenue for future improvements in code smell detection and software quality assurance.

RQ-3 discusses various software metrics used in research articles. AMC, code complexity, complexity metrics, etc., are rarely used. In contrast, LOC, WMC, DIT, LCOM, CC, and NoC are frequently used metrics because they provide valuable insights into software code's structural complexity and design issues. These metrics correlate with code smells, which indicate poor code quality and potential maintenance problems.

RQ-4 shows the utilization of datasets in our research. Due to their size and diversity, code smells are often detected using the Xerces, ArgoUML, Gantt Project, Azureus, Eclipse, and object-oriented metrics datasets, which enable the detection of a wide variety of code smells. These datasets are widely used in the software development community and are well documented and simple to utilize.

Based on our findings, earlier studies tended to place more emphasis on the Java language. The majority of these studies used Java-based platforms to test their methodologies. Therefore, further study is required to uncover the efficacy of ML algorithms in detecting code smells in languages other than Java.

RQ-5 discusses the various performance measures. In this study, the most frequently used performance measures are precision (21 times), recall (21 times), accuracy (22 times), and F-measure (18 times).

These performance measures can measure the efficiency of deployed ML algorithms for code smell detection. However, evaluating the model's efficacy by considering recall and precision together is essential.

Furthermore, when dealing with unbalanced datasets in which negative and positive classes are unequal, it is not desirable to rely only on accuracy.

Few research studies have used correlation analysis to calculate the associations between smells. Therefore, more studies are needed to address this issue and create more effective models for detecting code smells.

5. Threats to Validity

This study has not considered all aspects of code smell detection and follows a search strategy to retrieve research articles. The search strategy involved a comprehensive search string encompassing eight digital libraries, including original words, different synonyms, and spellings of the search word. The inclusion/exclusion of relevant articles was carefully considered, depending on their availability and relevance. The selection and elimination rules are discussed in Section 3.2. After analyzing several pertinent articles, it was observed that the recent work is more appropriate. However, it is important to note that the search strategy, while thorough, may have some limitations, such as the potential for missing articles due to inconsistent keywords in different articles or the exclusion of some conference and journal research papers. These limitations should be considered when interpreting this study's results.

This study includes the maximum number of relevant articles on detecting code smell using ML algorithms; so it may be less likely to miss any related articles. This SLR excluded some conference and journal research papers using selection and elimination criteria, as discussed in Section 3.2. The criteria for inclusion were based on the paper's relevance to the research questions and the quality of the research methodology. If a research paper met these criteria, it was included in the study. A few good-quality papers may not meet our selection/inclusion criteria.

In this SLR, during the searching process, it has been observed that various digital libraries are not efficient because of inconsistent keywords in different articles. So, few papers or studies may have been missed. Various researchers followed Kitchenham et al.'s [31] research evaluation and validation method. According to Kitchenham et al., one researcher assesses the quality, and the second performs the validation. In our study, one researcher extracted and evaluated data from randomly chosen articles. The value of research articles is verified from various articles. The parameters set for quality evaluation are not inferior to quality assessment. It may increase the skewness toward quality evaluation because it might require more work to answer the research questions. Despite these challenges, the study's methodology was thorough and rigorous, ensuring the reliability of the results.

6. Conclusions

This study conducted an SLR to analyze and show different aspects of code smell detection with ML algorithms. The SLR covers the class-wise distribution of ML algorithms, a unique aspect not yet present in the literature, providing up-to-date knowledge to researchers for future studies. The class-wise distribution of ML algorithms is significant as it provides insights into the popularity and effectiveness of different ML algorithms in code smell detection. This study offers detailed insights into code smells, primarily focusing on ML algorithms and software metrics applied in studies, which have been found to be beneficial. The review procedure is framed with five research questions. Table A7 shows the evaluation of research articles and their score. Section 3.3 reports various categories of research articles; 36 are above average, and 6 are below average. Tables A2–A5 presents the attributes used to identify the articles. This study aims to facilitate a better understanding of code smell detection with ML algorithms and help researchers identify the popular methods and the methods that perform better.

This SLR summarizes different ML algorithms, performance measures, datasets, and software metrics for detecting code smells. Some main conclusions are as follows:

- Decision Trees, Ensemble Learners, Bayesian Learners, Rule-Based Learning, Support Vector Machines, neural networks, and other miscellaneous ML algorithms are used to detect code smells. The most frequently used ML algorithms from other classes are Naive Bayes, JRip, J48, Random Forest, and SVM.
- This study examined many datasets, some of which are open-source and a few of which are industrial datasets. The datasets are written in different languages, such as C#, JAVA, and XML; of these, JAVA language datasets are more popular.

- A massive range of datasets has been utilized. Xerces, ArgoUML, Gantt Project, Azureus, Eclipse, and object-oriented metrics datasets are frequently used. Apache Ant, Fluid Tool, iPlasma, PMD, etc., are also used in this area.
- Popularly used performance measures are precision, recall, accuracy, and F-measure. MCC, MAE, Hamming score, etc., are also used for performance assessment but are used less frequently than the rest.

This study is a valuable resource for software developers, providing a comprehensive discussion of code smell detection. It enhances developers' understanding of code smells, equipping them with the knowledge to identify and eliminate them. This understanding can be applied to improve software quality and resolve issues. The practical implications of this study's findings are significant, as they can directly impact the quality and maintainability of software, a key concern for developers.

Better code can be written with an understanding of code smells. Over 30 code smells are discussed in this study. The findings of this research study provide insight into how to enhance its quality and maintainability. Software engineers can better apply the most efficient methods of code smell detection by keeping up with the latest developments in this area. This study covers a survey utilizing ML algorithms from 2005 to 2024. Well-known ML algorithms and practical software metrics are included in this research. LOC, WMC, DIT, LCOM, CC, and NoC are the most used metrics for spotting code smells, and 27 articles applied Decision Tree techniques. In this study, the definition, causes, and effects of code smells are provided for the reader. Methods for the detection of code smells are also covered. This knowledge will empower software developers to create more stable and maintainable programs and allow them to identify and fix code smells before releasing them to the public.

The following are future recommendations for software professionals and researchers working in code smell detection. These recommendations are not just suggestions but are crucial for the advancement of code smell detection. These can inspire further research and innovation in this field, providing a roadmap for researchers to contribute to the evolution of code smell detection. By following these recommendations, researchers can feel inspired and motivated to contribute significantly to this field.

(i) Few research studies have investigated how effectively neural networks, Rule-Based Learning, etc., predict outcomes. Some ML algorithms for detecting code smells have never been applied.
(ii) Few datasets are publicly available. Researchers should have unrestricted access to these datasets to conduct more research studies.
(iii) Only a few researchers have conducted generalized analyses; thus, the results of various ML-based algorithms should be more widely applicable. If one adopts this perspective, it will be simple to compare efficiency and performance.
(iv) Overfitting and class imbalance are two major problems in this field. Addressing these is necessary because most of the datasets have a skewed distribution.
(v) The efficacy of ML algorithms should be evaluated with other statistical approaches.

Author Contributions: P.S.Y.: Conceptualization, critical analysis, validation, figure and table preparation, writing review and editing. R.S.R.: Conceptualization, critical analysis, validation, supervision, writing review and editing. A.M.: Supervision, critical analysis, writing review and editing. M.G.: Supervision, writing review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

- **Table A1** lists the various types of research articles.
- **Table A2** includes details such as the unique identifier for each article (Article No.), the serial number as listed in the reference section (Ref. no), the year of publication, the name of the journal or conference in which the research article is published (Publication), and whether the research article is a journal or conference paper (Journal/Conference).
- **Table A3** focuses on the dataset used in the reviewed research articles. It contains the unique identifier for each article (Article no.), the name of the dataset (Dataset), the language of the dataset (Dataset Language), and the type of dataset, such as open-source software (OSS) or industrial (Dataset Type).
- **Table A4** addresses the methodologies and tools used in the research articles. It includes the unique identifier for each article (Article no.), the tools or technologies used (Tools/Tech.), the specific code smells covered (Smells), the algorithms used to classify the code smells (Algorithm), and the performance metrics employed to evaluate the methodologies (Performance Metrics).
- **Table A5** details the software metrics utilized in the research articles. It lists the unique identifier for each article (Article no.) and the specific software metrics used in the research (Software Metrics Name).
- **Table A6** checks for the answers to research questions.
- **Table A7**, a comprehensive evaluation, thoroughly examines the research article's performance. This meticulous evaluation instils confidence in the reliability and validity of the research article's findings.

Table A1. Classification and ranking of studies on ML algorithms for code smell detection.

Categories	Research Papers	Score
Above average	ARTN1, ARTN2, ARTN3, ARTN4, ARTN5, ARTN7, ARTN8, ARTN10, ARTN11, ARTN12, ARTN13, ARTN14, ARTN15, ARTN16, ARTN17, ARTN18, ARTN19, ARTN20, ARTN21, ARTN22, ARTN23, ARTN24, ARTN26, ARTN27, ARTN28, ARTN29, ARTN30, ARTN32, ARTN33, ARTN36, ARTN37, ARTN38, ARTN39, ARTN40, ARTN41, ARTN42	>5
Below average	ARTN6, ARTN9, ARTN25, ARTN31, ARTN34, ARTN35	$\geq$2

Table A2. The study's analysis—general information.

Article No.	Ref. no	Year	Publication	Journal/Conference
ARTN1	[32]	2005	Elsevier Electronic notes in Theoretical Computer Science, Science Direct	Journal
ARTN2	[33]	2009	2009 Ninth International Conference on Quality Software	Conference
ARTN3	[34]	2009	2009 16th Working Conference on Reverse Engineering	Conference
ARTN4	[35]	2010	14th European Conference on Software Maintenance and Reengineering	Conference
ARTN5	[36]	2010	2010 Seventh International Conference on the Quality of Information and Communications Technology	Conference
ARTN6	[37]	2010	Scopus and IEEE	Conference

Table A2. *Cont.*

Article No.	Ref. no	Year	Publication	Journal/Conference
ARTN7	[38]	2011	2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)	Conference
ARTN8	[39]	2011	Elsevier The Journal of Systems and Software	Journal
ARTN9	[40]	2012	IEEE	Conference
ARTN10	[41]	2012	ASE'12, 3–7 September 2012, Essen, Germany	Conference
ARTN11	[42]	2012	ASE'12, 3–7 September 2012, Essen, Germany	Conference
ARTN12	[43]	2012	19th Working Conference on Reverse Engineering	Conference
ARTN13	[44]	2013	IEEE ASE 2013, Palo Alto, USA	Conference
ARTN14	[45]	2013	IEEE Transactions on Software Engineering	Conference
ARTN15	[46]	2013	IEEE International Conference on Software Maintenance	Conference
ARTN16	[47]	2015	IEEE	Conference
ARTN17	[48]	2015	IEEE	Conference
ARTN18	[49]	2015	Empirical Software Eng	Journal
ARTN19	[50]	2016	IEEE 24th International Conference on Program Comprehension (ICPC)	Conference
ARTN20	[26]	2016	Empirical Software Eng, springer	Journal
ARTN21	[51]	2016	ASE'16, 3–7 September 2016, Singapore, Singapore	Conference
ARTN22	[52]	2016	7th International Conference on Computer Science and Information Technology	Conference
ARTN23	[53]	2016	Scopus, IEEE, and Science Direct	Conference
ARTN24	[54]	2017	19th International Conference on Enterprise Information Systems (ICEIS 2017)	Conference
ARTN25	[27]	2017	International Conference on Machine learning and Data Science	Conference
ARTN26	[55]	2017	Knowledge-Based Systems	Journal
ARTN27	[56]	2017	Scopus International Journal of Electrical and Computer Engineering (IJECE)	Journal
ARTN28	[57]	2017	Scopus and IEEE	Conference
ARTN29	[7]	2018	IEEE	Conference
ARTN30	[10]	2019	2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)	Conference
ARTN31	[58]	2019	International Conference on Intelligent Computing and Control Systems (ICICCS 2019) IEEE Xplore	Conference
ARTN32	[8]	2019	International Journal of Rough Sets and Data Analysis April–June 2019	Journal
ARTN33	[59]	2019	International Journal of Computer Sciences and Engineering	Journal
ARTN34	[60]	2019	IEEE	Conference
ARTN35	[61]	2019	IEEE	Conference
ARTN36	[3]	2020	Journal of computer science and technology	Journal
ARTN37	[62]	2020	Springer Software Quality Journal	Journal
ARTN38	[63]	2021	IEEE Access	Conference

Table A2. *Cont.*

Article No.	Ref. no	Year	Publication	Journal/Conference
ARTN39	[30]	2021	Tech Science Press Computers, Materials and Continua Tech Science Press	Conference
ARTN40	[64]	2021	International Conference on Advanced Information Networking and Applications, Springer, Cham.	Conference
ARTN41	[65]	2021	IEEE	Journal
ARTN42	[66]	2022	Empirical Software Engineering	Journal

Table A3. The study's analysis—dataset information.

Article No.	Dataset	Dataset Language	Dataset Type
ARTN1	WEKA and IYC	JAVA	OSS, Industrial
ARTN2	Gantt Project v1.10.2 and Xerces v2.7.0	JAVA	OSS
ARTN3	Eclipse JDT and Xerces v2.7.0	XML, JAVA	OSS
ARTN4	NA	JAVA	OSS
ARTN5	Gantt Project v1.10.2 and Xerces v2.7.0	JAVA	OSS
ARTN6	Expert's knowledge and object-oriented metrics	JAVA	OSS
ARTN7	Seven datasets from the previous literature	NA	Industrial
ARTN8	Gantt Project v1.10.2 and Xerces v2.7.0	JAVA	OSS
ARTN9	Electricity calculating program and movie rental program	JAVA	Industrial
ARTN10	Xproj and Yproj	C#	Microsoft Project (Industrial)
ARTN11	Xerces v2.7.1 and Azureus v2.3.0.6, ArgoUML v0.19.8	JAVA	OSS
ARTN12	Xerces v2.7.0 and Azureus v2.3.0.6, ArgoUML v0.19.8	JAVA	OSS
ARTN13	jEdit, Apache Ant, five projects of Android APIs and Apache Tomcat	JAVA	OSS
ARTN14	Twenty open-source software programs	JAVA	OSS
ARTN15	Seventy-six open-source software programs of Qualitas Corpus (two datasets, one for method-level smells, and another one for class-level smells)	JAVA	OSS
ARTN16	Closure Compiler, maven, Guava, Eclipse, and jUnit	JAVA	OSS
ARTN17	CheckStyle 5.7, JDeodorant 5.0, PMD 5.1.1, and inFusion 1.8.5, iPlasma, and inCode	JAVA	OSS
ARTN18	Dataset of Bellon et al., TF-IDF vector of our datasets	JAVA	OSS
ARTN19	Ten open-source projects	JAVA	OSS
ARTN20	Four datasets were acquired (one for each code smell)	JAVA	OSS
ARTN21	ANTLR to tokenize the source code, the RNNLM Toolkit, Apache Ant, Hibernate, JDK, ArgoUML, CAROL, and JHotDraw	JAVA	OSS
ARTN22	BCEL, Maven Core, and Commons 10. Another three projects belong to the biggest airline companies	JAVA	OSS
ARTN23	Object-oriented metrics	JAVA	OSS
ARTN24	Gantt Project	JAVA	OSS

Table A3. *Cont.*

Article No.	Dataset	Dataset Language	Dataset Type
ARTN25	Argo UML v0.19.8 and Xerces v 2.7.0	JAVA	OSS
ARTN26	iPlasma, Antipattern Scanner, Fluid Tool, PMD, and Marinescu detection rule	JAVA	OSS
ARTN27	Object-oriented metrics	JAVA	OSS
ARTN28	Object-oriented metrics	JAVA	OSS
ARTN29	74 software systems	JAVA	OSS
ARTN30	Ant, Cassandra, Derby, ArgoUML, Eclipse, Hadoop, Elastic Search, HSQLDB, Incubating, Qpid, Wicket, Xerces, and Nutch	JAVA	OSS
ARTN31	Implemented in JAVA using Net Beans IDE	JAVA	NA
ARTN32	The dataset objects consist of methods from 74 different JAVA systems	JAVA	NA
ARTN33	Fontana et al. [26]	NA	OSS
ARTN34	Data from 629 open-source projects are accessible on GitHub.	JAVA	OSS
ARTN35	Long Method, Data Class, God Class, and Feature Envy from Fontana et al. [26]	JAVA	OSS
ARTN36	ORI D, REFD D, and MULTI L D	NA	OSS
ARTN37	Four open-source JAVA projects and two method-level datasets from Fontana et al. [26]	JAVA	OSS
ARTN38	Dr JAVA, EMMA, and Find Bugs	JAVA	OSS
ARTN39	Argo UML, Azure, Gantt Project, Log4j, and Xerces-J	JAVA	OSS
ARTN40	Original, SMOTE, and ADASYN datasets	NA	Open-source projects
ARTN41	Qualitas Corpus	JAVA	OSS
ARTN42	18 datasets of different years	JAVA	OSS

Note: NA means ‘Not Available’.

Table A4. The study’s analysis—smells, tools, algorithm, and evaluation metrics information.

Article No.	Tools/ Tech.	Smells	Algorithm	Performance Metrics
ARTN1	IYC	Delegator, Lazy Class, Long Method, Feature Envy, God Class	C4.5, Decision Tree	Accuracy
ARTN2	Bayesian approach	Blob	Bayesian Belief Networks	Precision, Recall, Accuracy
ARTN3	NA	God Class	Naive Bayes	Precision, Recall
ARTN4	ABS (antipattern identification using B-Splines)	Blob	Bspl	Precision, Recall
ARTN5	IDS (Immune-based Detection Strategy), based on Artificial Immune Systems	Functional Decomposition, Blob, and Spaghetti Code	Immune-inspired Approach	Precision, Recall

Table A4. *Cont.*

Article No.	Tools/ Tech.	Smells	Algorithm	Performance Metrics
ARTN6	Metrics collection tool	Long Method	Eclipse Plugin Binary Logistic Regression (BLR)	Accuracy
ARTN7	A methodology for predicting bad smells from the software design model	Feature Envy, Message Chain, Long Method, Switch Statement, Middle Man, Long Parameter List, and Lazy Class	Random Forest, J48, Naive Bayes, Logistic, IBI, Voting Feature Intervals, and IBk	Hypothesis test, the predictive value of tests, prediction accuracy, and sensitivity and specificity
ARTN8	A GQM-based approach, BDTEX	Blob, Functional Decomposition, and Spaghetti Code	Bayesian Belief Networks	Precision, Recall, and utility
ARTN9	BSDT (BAD SMELL DETECTING TOOL)	Long Method, Switch Statement, Long Parameter List, Parallel Inheritance Hierarchy, Lazy Class, Large Class, and Data Class	A Switch Statement Rules, Naive Bayes	Accuracy
ARTN10	Code clone detection tools	Duplicate Code	Bayesian Network	Precision, Recall, Effectiveness
ARTN11	Support Vector Machine direct	Functional Decomposition, Spaghetti Code, and SAK	SVM	Precision, Recall
ARTN12	SMURF	Blob, Spaghetti Code, Functional Decomposition, and Swiss Army Knife	SVM	Precision, Recall
ARTN13	HIST	Feature Envy, Shotgun Surgery, Blob, Parallel Inheritance Hierarchy, Divergent Change	Association Rule Mining	Precision, Recall, F-measure
ARTN14	HIST	Blob, Divergent Change, Parallel Inheritance Hierarchy, Shotgun Surgery, and Feature Envy	Association Rule Mining	Precision, Recall, F-measure
ARTN15	Code smell detection tool, Weka tool	Data Class, Feature Envy, God Class, and Long Method	LibSVM, J48, Sequential minimal optimization, JRip, Random Forest, Naive Bayes	Accuracy, F-measure, ROC
ARTN16	NA	Duplicate Code, Switch Statement, and Divergent Change	Association Rule Mining	Precision, Recall, and F-measure
ARTN17	Decision Tree (C5.0)	Refused Parent Bequest, Speculative Generality, Complex Class, Antisingleton, Message Chain, Long Parameter List, Class Data should be Private, Lazy Class, Swiss Army Knife, God Class	SVM, Bayesian Belief Networks	Precision, Recall, F-measure
ARTN18	Clone detection tools (CDTs)	Duplicated Code	Bayesian Network	Precision, Recall, Accuracy

Table A4. *Cont.*

Article No.	Tools/ Tech.	Smells	Algorithm	Performance Metrics
ARTN19	TACO (Textual Analysis for Code Smell Detection)	Feature Envy, Blob, Long Method, Misplaced Class, Promiscuous Package	Structural techniques, TACO	Precision, Recall, F-measure
ARTN20	iPlasma, PMD, Fluid Tool, Antipattern Scanner, Marinescu	Long Method, God Class, Data Class, Feature Envy	J48, Naïve Bayes, Sequential minimal optimization, JRip, Random Forest, SVM	The accuracy obtained by the ML-algorithms
ARTN21	RNNLM Toolkit	Duplicated Code	Deep Learning	Precision, Recall
ARTN22	Metric and a rule-based automated antipattern detection	Swiss Army Knife, Blob, and Lava Flow	Three Mechanism Filtering Mechanism, Static Code Analyzer, and Metric Analyzer	Accuracy
ARTN23	DTReg tool	God Class, Feature Envy, Long Method, Empty catch block, Careless cleanup, Type Checking, Nested try statement, Exception thrown in finally block, Unprotected main, Dummy handler, Over logging	SVM, Multilayer Perceptron, Radial Basis Function Neural Networks Linear Regression (LR), Decision Tree Forest (DFT), k-fold cross validation	Root mean square error and mean absolute error
ARTN24	The Weka tool to implement the algorithms analyzed	Data Class, Long Method, Feature Envy, and God Class	J48, JRip, Naive Bayes, Sequential Minimal Optimization, SVM, Random Forest	Accuracy, Efficiency
ARTN25	SVMCSD	Large Class, Feature Envy, Data Class, and Long Method	SVM	Precision, Recall
ARTN26	An approach based on machine learning	Long Method, Data Class, Feature Envy	C4.5, J48, Naive Bayes, JRip, Random Forest, Decision Tree, Sequential Minimal Optimization, LibSVM	Standard deviations, descending values of Spearman’s, performance indicators (except Tb), mad, mse, acc
ARTN27	Tensor flow	God Class, Lazy Class, Data Class, Feature Envy, Large Class, Parallel Inheritance Hierarchies	SciTools Understand, JUnit-4.10 Multilayer Perceptron	Accuracy
ARTN28	Weka	God Class, Feature Envy, Long Method, Data Class	Eclipse Plugin, Analyst 4J, Xercesv, ArgoUML SVM, random under sampling	Recall, Precision
ARTN29	NA	God Class, Feature Envy, Data Class, and Long Method	Random Forest, JRip, J48, LIBSVM, ADABOOST, Naive Bayes, Sequential Minimal Optimization	Mean Accuracy, F-Measure, and AUC ROC
ARTN30	ML algorithm with a heuristic	Spaghetti Code, Long Method, Class Data should be private, Complex Class, and God Class	J48, Random Forest, Naive Bayes, SVM, JRip.	Precision, Recall, F-measure, Mcc (Matthews Correlation Coefficient)

Table A4. *Cont.*

Article No.	Tools/ Tech.	Smells	Algorithm	Performance Metrics
ARTN31	Code detection techniques	Lazy Class Detector, Primitive Obsession Detector, Too Many Literal Detectors, Bloated Code Detector, Feature Envy Detector, and Duplicated Code Detector	SVM, Random Forest	Time complexity
ARTN32	Datasets have been prepared based on the tools, manual labeling process.	Shotgun Surgery, Message Chaining	J48, C4.5, Random Forest, JRip, Naive Bayes, Sequential Minimal Optimization, K-nearest neighbors	Area under the ROC, Accuracy, and F-measure
ARTN33	Using a Decision Tree technique and software metrics, detect code smells	Long Method, Data Class, Feature Envy, and God Class	Decision Tree	Accuracy, F-measure
ARTN34	Empirical framework empirically investigates and evaluates different classification techniques, feature selection techniques, and data sampling techniques	Blob Class, Resolver, Member Ignoring Method, Complex Class, Internal Getter/Setter, Long Method, No Low Memory, and Leaking Inner Class	Classification techniques, feature selection techniques, and data sampling techniques to handle imbalanced data in prediction	Accuracy, F-measure, ACC
ARTN35	Especially multi-label classification methods, ML algorithms	Long Method, God Class, Feature Envy, Data Class	Decision Tree, Naive Bayes, Random Forest, Neural Network, SVM	Exact Match Ratio, Hamming Score, AUC ROC, Accuracy, F-measure
ARTN36	A strategy using machine learning and software metrics	Data Class, Feature Envy, Long Method, and God Class	Random Forest, Gradient Boosting Tree, Decision Tree, Deep Learning, SVM, Multilayer Perceptron	Accuracy, F-measure
ARTN37	Design Features and Metrics for JAVA	Long Method, Feature Envy	B-Random Forest, Random Forest, BJ48U, B-J48 Pruned, J48U	Accuracy, F-measure, ROC area
ARTN38	CKJM, JCODODOR, and WEKA	God Class, Brain Method, Shotgun Surgery, Message Chains, Data Class, Dispersed Coupling	Ensemble Learning, Bagging, Random Forest	Accuracy (P1), G-meam2 (P3), G-mean 1(P2), and F-measure (P4)
ARTN39	Search-based approach	Large Class, Long Method, Feature Envy, Spaghetti Code, Data Class, Lazy Class, Functional Decomposition, Parallel Inheritance, and Long Parameter List	Search-based approach	Precision, Recall
ARTN40	Code smell prediction models, feature extraction	Blob Class, Complex Class, Internal Getter/Setter, Leaking Inner Class, Long Method, No Low Memory Resolver, Member Ignoring Method, and Swiss Army Knife	SMOTE, ADASYN sampling methods	Accuracy AUC F-measure

Table A4. *Cont.*

Article No.	Tools/ Tech.	Smells	Algorithm	Performance Metrics
ARTN41	A Novel Approach for the Detection of Code Smells	God Class, Long Method, Data Class, Feature Envy	Naive Bayes, Multilayer Perceptron, KNN, Logistic Regression, Decision Tree, and Random Forest, validation technique: 10-fold cross-validation	Accuracy, Recall, F-measure, and Precision
ARTN42	Crowdsmelling approach	God Class, Feature Envy, Long Method	J48, Random Forest, ADABOOST, Sequential Minimal Optimization, Multilayer Perceptron, Naive Bayes	Accuracy, Precision, Recall, ROC, and F-Measure

Note: NA means 'Not Available'.

Table A5. The study's analysis—S/W metrics.

Article No.	S/W Metrics
ARTN1	Object-oriented metrics IYC, coupling and cohesion, complexity metrics, the size measure, the complexity measure
ARTN2	Goal Question Metric (GQM), LCOM5 metric, and NMD and NAD metrics
ARTN3	NOC, DTT, NMD, and NAD
ARTN4	Quality metrics cohesion, coupling, and complexity
ARTN5	Goal Question Metric (GQM), metric-based heuristics
ARTN6	NBD (Nested Block Depth), VG (cyclomatic complexity), and code complexity metrics- MLOC (Method Lines Of Code) and PAR (Number of Parameters)
ARTN7	Design model metrics: number of members, number of attributes, number of operations, Number of Parameters, response for a class, number of classes, weighted attributes per class, weighted methods per class, appearance in diagrams, Depth of Inheritance Tree, number of children, number of inherited attributes, number of inherited operations, attribute hiding factor, attribute inheritance factor, coupling factor, method hiding factor, method inheritance factor, polymorphism factor, number of actors, number of components, number of name spaces, Coupling Between Objects, number of abstractions, number of times class is used as employment parameter type, and number of dependencies
ARTN8	A Goal Question Metric (GQM) NMD and NAD, LCOM5, and NoDC
ARTN9	Lines of code, Number of Methods, Tree, Depth of Inheritance, Number of Parameters, number of attributes, Method Lines of Code, weighted methods per class, McCabe cyclomatic complexity, Lack of Cohesion of Methods, and number of children
ARTN10	Number of Invocations, Number of Library Invocations, Number of Lines, Number of Local Invocations, Number of Other Invocations, Number of Parameter Accesses, whether it is Test Code4, Number of Field Accesses
ARTN11	Object-oriented metrics, more than 60 metrics
ARTN12	Object-oriented metrics
ARTN13	LOC, weight methods per class (WMC), Coupling Between Objects (CBO), and Number of Methods (NOM)
ARTN14	NA
ARTN15	Complexity, cohesion, size, coupling, and a large set of object-oriented metrics
ARTN16	NOM (Number of Methods), WMC (weight methods per class), and CBO (Coupling Between Objects)

Table A5. *Cont.*

Article No.	S/W Metrics
ARTN17	Number of Parameters (NOParam), the number of lines of code (LOC), Depth of Inheritance Tree (DIT)
ARTN18	Metric-based approach
ARTN19	Structural metrics, like size and complexity metrics.
ARTN20	LOC, NOM, NOPK, LOCNAMM, NOCS, NOA, and WMC, NOMNAMM CYCLO, WMCNAMM, MAXNESTING, AMW, WOC, NOP, NOAV, CLNAMM, ATLD, NOLV, AMWNAMM LCOM5, TCC FANOUT, FDP, RFC, CBO, ATFD, CINT, CDISP, CFNAMM, CC, CM LAA, NOAM, NOPA DIT, NOI, NOC, NMO, NIM, LAA, MaMCL, MeMCL, NODA, NOPVA, NOII, NOPRA, NOFA, NOFSA, NOSA, NONFSA, NOABM, NoNFNSA, NOCM, NOFM, NOFNSM, NoFSM, NONCM, NONFNABM, NONFNSM, NODM, NOPM, NOPRM, NOPLM, NONAM, and NOSM
ARTN21	LOC, AST-based
ARTN22	LCOM RFC NAM NADCs, OPT, and TSCs
ARTN23	LOC, CBO, DIT, RFC, WMC, NOC, LCOM, CC, Dependencies (Dcy and Dcy), Javadoc function (jf), MHF, cyclomatic complexity (Ocavg), Javadoc Javadoc methods (jm), line of code (jLOC), AHF
ARTN24	NA
ARTN25	No. of object-oriented metrics
ARTN26	Measured through Spearman's
ARTN27	LOC, CC, NOC, RFC, DIT, WMC, CBO, and LCOM
ARTN28	Static code metrics-based approach
ARTN29	61 source code metrics were computed at class level and 82—at method level
ARTN30	LCOM, NOPA, NMNOPARAM, ELOC, LOC_METHOD, NOA, NOM, NP, and WMC
ARTN31	NA
ARTN32	LOC, NOM, NOCS, NOPK, LOCNAMM, NOA, NOMNAMM CYCLO, WMCNAMM, WMC, MAXNESTING, WOC, CLNAMM, NOP, NOAV, ATLD, NOLV, AMWNAMM, LCOM5, TCC, FANOUT, AMW, ATFD, RFC, CBO, CFNAMM, FDP, CINT, CDISP, CM, CC, NOAM, NOPA DIT, LAA, NOI, NMO, NIM, NOC, and NOII
ARTN33	82 software metrics and 61 software metrics in the class-level code smells
ARTN34	NOC, NOCH, WMC, IPM, CC, NBI, NOM, NOCH, DIT, LCOM, PPIV, LOCL, APD, XML, BSMC, NTO, WKL, GPS, BMAP, SQL, NET, and I/O
ARTN35	82 features (software metrics), MLC evaluation metrics
ARTN36	In the method-level code smells, 82 software metrics are computed; in the class-level code smells, 61 software metrics are computed
ARTN37	There are 82 measures in the created MLD training dataset, 25 are class metrics, 5 are package metrics, and 6 are project-level metrics
ARTN38	Weighted methods per class, Number of Methods, Depth of the Inheritance Tree, and response for a class. Coupling between object classes, lack of cohesion in methods, the normalized version of LCOM inheritance coupling, coupling between methods, average method complexity, number of public methods for a class, also known as CIS (class interface size), data access metric, measure of aggregation, measure of functional abstraction, cohesion among methods of a class, cyclomatic complexity, lines of code, afferent coupling, efferent coupling, maximum cyclomatic complexity, average cyclomatic complexity
ARTN39	LOC, NAD, McCabe, NOC, CBO, LCOM, CC, LCOM, WMC, NOPARAM, NOM, RFC, DIT, and NMO
ARTN40	Android-oriented metrics, complexity metrics, object-oriented metrics, and dimensional metrics
ARTN41	61 software metrics for the class-level and 82 software metrics for the method-level code smells
ARTN42	For God Class, a set of 61 metrics was used, and for the other two code smells

Note: NA means 'Not Available'.

Table A6. Mapping of research question with the score.

Article No.	RQ1	RQ2	RQ3	RQ4	RQ5	Score
ARTN30	√	√	√	√	√	8
ARTN29	√	√	√	√	√	8
ARTN12	√	√	√	√	√	8
ARTN42	√	√	√	√	√	8
ARTN37	√	√	√	√	√	7.5
ARTN32	√	√	√	√	√	7.5
ARTN10	√	√	√	√	√	7.5
ARTN11	√	√	√	√	√	7.5
ARTN41	√	√	√	√	√	7.5
ARTN38	√	√	√	√	√	7
ARTN39	√	√	√	√	√	7
ARTN36	√	√	√	√	√	7
ARTN26	√	√	√	√	√	7
ARTN23	√	√	√	√	√	7
ARTN16	√	√	√	√	√	7
ARTN14	√	√	×	√	√	7
ARTN8	√	√	√	√	√	7
ARTN3	√	√	√	√	√	7
ARTN28	√	√	√	√	√	6.5
ARTN20	√	√	√	√	×	6.5
ARTN21	√	√	√	√	√	6.5
ARTN17	√	√	√	√	√	6.5
ARTN18	√	√	√	√	√	6.5
ARTN13	√	√	√	√	√	6.5
ARTN15	√	√	√	√	√	6.5
ARTN7	√	√	√	√	√	6.5
ARTN5	√	√	√	√	√	6.5
ARTN33	√	√	√	√	√	6
ARTN19	√	√	√	√	√	6
ARTN22	√	√	√	√	√	6
ARTN2	√	√	√	√	√	6
ARTN1	√	√	√	√	√	6
ARTN40	√	√	√	√	√	5.5
ARTN24	√	√	×	√	√	5.5
ARTN27	√	√	√	√	√	5.5
ARTN4	√	√	√	×	√	5.5
ARTN34	√	√	√	√	√	5
ARTN35	√	√	√	√	√	5
ARTN25	√	√	√	√	√	5
ARTN9	√	√	√	√	√	5
ARTN6	√	√	√	√	√	5
ARTN31	√	√	×	√	√	4

Table A7. Assessment query score and total score.

Article No.	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q1	Total Score
ARTN30	0.5	1	1	1	1	1	1	0.5	1	0.5	8
ARTN29	1	1	1	0.5	0.5	1	1	1	1	1	8
ARTN12	1	1	1	1	0.5	1	1	0.5	1	1	8
ARTN42	1	1	1	1	0	1	1	1	1	1	8
ARTN37	1	0.5	1	1	0.5	1	0.5	1	1	1	7.5
ARTN32	1	0.5	1	1	1	0.5	1	1	0.5	1	7.5
ARTN10	1	0.5	1	1	1	0.5	1	0.5	1	1	7.5
ARTN11	1	1	0.5	1	0.5	1	0.5	1	1	1	7.5

Table A7. *Cont.*

Article No.	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q1	Total Score
ARTN41	1	1	1	1	0	1	0.5	1	1	1	7.5
ARTN38	1	0.5	0.5	1	1	0	1	1	1	1	7
ARTN39	1	0.5	1	1	1	0.5	0	1	1	1	7
ARTN36	1	0.5	1	1	0.5	1	1	0	1	1	7
ARTN26	1	1	1	1	0	1	1	0	1	1	7
ARTN23	1	0.5	1	1	1	0	1	0.5	1	1	7
ARTN16	1	1	0.5	1	1	0	0.5	1	1	1	7
ARTN14	1	1	1	1	0	1	0.5	1	0.5	1	7
ARTN8	1	1	1	1	1	0.5	0.5	0.5	0.5	1	7
ARTN3	1	0.5	1	1	1	0	1	0.5	1	1	7
ARTN28	1	0.5	1	1	0.5	0.5	0.5	0.5	1	1	6.5
ARTN20	1	1	1	0.5	1	0	1	0.5	0.5	1	6.5
ARTN21	1	1	0	1	0.5	1	0.5	1	0.5	1	6.5
ARTN17	1	0.5	1	1	1	0	1	0.5	0.5	1	6.5
ARTN18	1	0.5	1	1	0.5	0.5	0.5	0.5	1	1	6.5
ARTN13	1	0	1	0.5	1	1	0.5	1	0.5	1	6.5
ARTN15	1	0.5	1	1	0.5	0	1	0.5	1	1	6.5
ARTN7	1	1	1	1	1	0	0.5	0	1	1	6.5
ARTN5	1	0.5	0.5	1	1	0.5	0.5	0.5	1	1	6.5
ARTN33	1	0	1	1	0.5	0.5	1	0	1	1	6
ARTN19	1	0.5	0.5	1	0	0.5	0.5	1	1	1	6
ARTN22	1	0.5	1	0.5	1	0	0	1	1	1	6
ARTN2	1	0.5	1	0.5	1	0	0.5	0.5	1	1	6
ARTN1	1	0.5	1	0.5	1	0	0.5	0.5	1	1	6
ARTN40	1	0.5	1	1	0.5	0.5	0	0	1	1	5.5
ARTN24	1	0.5	1	1	0	1	0	0	1	1	5.5
ARTN27	1	0.5	1	1	1	0	0	1	0	1	5.5
ARTN4	1	0.5	0.5	1	1	0	0.5	0.5	0.5	1	5.5
ARTN34	1	0.5	1	1	1	0	0	0	0.5	1	5
ARTN35	1	0.5	0	1	1	0.5	0	1	0	1	5
ARTN25	1	0.5	0.5	1	0	0	0.5	0.5	1	1	5
ARTN9	1	0.5	0.5	1	1	0	0.5	0	0.5	1	5
ARTN6	1	0	1	0.5	1	0.5	0	0.5	0.5	1	5
ARTN31	1	0.5	0	1	0	0	0.5	0	1	1	4

References

1. Dewangan, S.; Rao, R.S.; Yadav, P.S. Dimensionally Reduction based Machine Learning Approaches for Code smells Detection. In Proceedings of the 2022 International Conference on Intelligent Controller and Computing for Smart Power (ICICCSP), Hyderabad, India, 21–23 July 2022; pp. 1–4. [CrossRef]
2. Yadav, P.S.; Dewangan, S.; Rao, R.S. Extraction of Prediction Rules of Code Smell using Decision Tree Algorithm. In Proceedings of the International Conference on Internet of Everything, Microwave Engineering, Communication and Networks (IEMECON), Jaipur, India, 1–2 December 2021; pp. 1–5. [CrossRef]

3. Mhawish, M.Y.; Gupta, M. Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics. *J. Comput. Sci. Technol.* **2020**, *35*, 1428–1445. [CrossRef]
4. Dewangan, S.; Rao, R.S. Method-Level Code Smells Detection Using Machine Learning Models. *Lect. Notes Netw. Syst.* **2023**, *725*, 77–86. [CrossRef]
5. Yadav, P.S.; Rao, R.S. Feature reduction techniques based code smell prediction. *I-Manag. J. Softw. Eng.* **2022**, *17*, 6–11. [CrossRef]
6. Dewangan, S.; Rao, R.S. Code Smell Detection Using Classification Approaches. *Lect. Notes Netw. Syst.* **2022**, *431*, 257–266. [CrossRef]
7. Di Nucci, D.; Palomba, F.; Tamburri, D.A.; Serebrenik, A.; De Lucia, A. Detecting code smells using machine learning techniques: Are we there yet? In Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 20–23 March 2018; pp. 612–621. [CrossRef]
8. Guggulothu, T.; Moiz, S.A. Detection of Shotgun Surgery and Message Chain Code Smells using Machine Learning Techniques. *Int. J. Rough Sets Data Anal.* **2019**, *6*, 34–50. [CrossRef]
9. Iqbal, A.; Aftab, S.; Ullah, I.; Bashir, M.S.; Saeed, M.A. A Feature Selection based Ensemble Classification Framework for Software Defect Prediction. *Int. J. Mod. Educ. Comput. Sci.* **2019**, *11*, 54–64. [CrossRef]
10. Pecorelli, F.; Palomba, F.; Di Nucci, D.; De Lucia, A. Comparing heuristic and machine learning approaches for metric-based code smell detection. In Proceedings of the International Conference on Program Comprehension, Montreal, QC, Canada, 25–26 May 2019; pp. 93–104. [CrossRef]
11. Caram, F.L.; Rodrigues, B.R.D.O.; Campanelli, A.S.; Parreiras, F.S. Machine Learning Techniques for Code Smells Detection: A Systematic Mapping Study. *Int. J. Softw. Eng. Knowl. Eng.* **2019**, *29*, 285–316. [CrossRef]
12. Kaur, A.; Dhiman, G. A review on search-based tools and techniques to identify bad code smells in object-oriented systems. *Adv. Intell. Syst. Comput.* **2019**, *741*, 909–921. [CrossRef]
13. Azeem, M.I.; Palomba, F.; Shi, L.; Wang, Q. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Inf. Softw. Technol.* **2019**, *108*, 115–138. [CrossRef]
14. Kaur, A.; Jain, S.; Goel, S.; Dhiman, G. A Review on Machine-learning Based Code Smell Detection Techniques in Object-oriented Software System(s). *Recent Adv. Electr. Electron. Eng.* **2020**, *14*, 290–303. [CrossRef]
15. dos Reis, J.P.; Abreu, F.B.E.; Carneiro, G.D.F.; Anslow, C. Code Smells Detection and Visualization: A Systematic Literature Review. *Arch. Comput. Methods Eng.* **2022**, *29*, 47–94. [CrossRef]
16. Al-Shaaby, A.; Aljamaan, H.; Alshayeb, M. Bad Smell Detection Using Machine Learning Techniques: A Systematic Literature Review. *Arab. J. Sci. Eng.* **2020**, *45*, 2341–2369. [CrossRef]
17. Singh, S.; Kaur, S. A systematic literature review: Refactoring for disclosing code smells in object oriented software. *Ain Shams Eng. J.* **2018**, *9*, 2129–2151. [CrossRef]
18. Sobrinho, E.V.D.P.; De Lucia, A.; Maia, M.D.A. A Systematic Literature Review on Bad Smells-5 W's: Which, When, What, Who, Where. *IEEE Trans. Softw. Eng.* **2021**, *47*, 17–66. [CrossRef]
19. Zhang, M.; Hall, T.; Baddoo, N. Code Bad Smells: A review of current knowledge. *J. Softw. Maint. Evol. Res. Pract.* **2011**, *23*, 179–202. [CrossRef]
20. Rasool, G.; Arshad, Z. A review of code smell mining techniques. *J. Softw. Evol. Process* **2015**, *27*, 867–895. [CrossRef]
21. Fernandes, E.; Oliveira, J.; Vale, G.; Paiva, T.; Figueiredo, E. A Review-based Comparative Study of Bad Smell Detection Tools. In Proceedings of the EASE '16: 20th International Conference on Evaluation and Assessment in Software Engineering, Limerick, Ireland, 1–3 June 2016; pp. 1–12. [CrossRef]
22. Gupta, A.; Suri, B.; Misra, S. A systematic literature review: Code bad smells in java source code. *Lect. Notes Comput. Sci.* **2017**, *10408*, 665–682. [CrossRef]
23. Sharma, T.; Spinellis, D. A survey on software smells. *J. Syst. Softw.* **2018**, *138*, 158–173. [CrossRef]
24. Haque, M.S.; Carver, J.; Atkison, T. Causes, impacts, and detection approaches of code smell: A survey. In Proceedings of the ACMSE 2018 Conference, Richmond, KY, USA, 29–31 March 2018; pp. 1–8. [CrossRef]
25. Zhang, Y.; Ge, C.; Liu, H.; Zheng, K. Code smell detection based on supervised learning models: A survey. *Neurocomputing* **2024**, *565*, 127014. [CrossRef]
26. Fontana, F.A.; Mantylä, M.; Zanoni, M.; Marino, A. Comparing and experimenting machine learning techniques for code smell detection. *Empir. Softw. Eng.* **2016**, *21*, 1143–1191. [CrossRef]
27. Kaur, A.; Jain, S.; Goel, S. A Support Vector Machine Based Approach for Code Smell Detection. In Proceedings of the International Conference on Machine Learning and Data Science, Noida, India, 14–15 December 2017; pp. 9–14. [CrossRef]
28. Nizam, A.; Avar, M.Y.; Adaş, Ö.K.; Yanık, A. Detecting Code Smell with a Deep Learning System. In Proceedings of the Innovations in Intelligent Systems and Applications Conference, Sivas, Turkiye, 11–13 October 2023; pp. 1–5. [CrossRef]
29. Shah, R.N.; Mohamed, S.A.; Imran, A.; Kosar, T. CloudScent: A Model for Code Smell Analysis in Open-Source Cloud. In Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Naples, Italy, 4–6 December 2023; pp. 69–75. [CrossRef]
30. Draz, M.M.; Farhan, M.S.; Abdulkader, S.N.; Gafar, M.G. Code Smell Detection Using Whale Optimization Algorithm. *Comput. Mater. Contin.* **2021**, *68*, 1919–1935. [CrossRef]
31. Kitchenham, B.; Charters, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*; Keele University: Keele, UK, 2007; Volume 2.

32. Kreimer, J. Adaptive Detection of Design Flaws. *Electron. Notes Theor. Comput. Sci.* **2005**, *141*, 117–136. [CrossRef]
33. Khomh, F.; Vaucher, S.; Guéehéeneuc, Y.G.; Sahraoui, H. A bayesian approach for the detection of code and design smells. In Proceedings of the International Conference on Quality Software, Jeju, Republic of Korea, 24–25 August 2009; pp. 305–314. [CrossRef]
34. Vaucher, S.; Khomh, F.; Moha, N.; Guéhéneuc, Y.G. Tracking design smells: Lessons from a study of God classes. In Proceedings of the Working Conference on Reverse Engineering, Lille, France, 13–16 October 2009; pp. 145–154. [CrossRef]
35. Oliveto, R.; Khomh, F.; Antoniol, G.; Guéhéneuc, Y.G. Numerical signatures of antipatterns: An approach based on B-Splines. In Proceedings of the European Conference on Software Maintenance and Reengineering, Madrid, Spain, 15–18 March 2010; pp. 248–251. [CrossRef]
36. Hassaine, S.; Khomh, F.; Guéhéneucy, Y.G.; Hamel, S. IDS: An immune-inspired approach for the detection of software design smells. In Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology, Porto, Portugal, 29 September–2 October 2010; pp. 343–348. [CrossRef]
37. Bryton, S.; Abreu, F.B.E.; Monteiro, M. Reducing subjectivity in code smells detection: Experimenting with the Long Method. In Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology, Porto, Portugal, 29 September–2 October 2010; pp. 337–342. [CrossRef]
38. Maneerat, N.; Muenchaisri, P. Bad-smell prediction from software design model using machine learning techniques. In Proceedings of the International Joint Conference on Computer Science and Software Engineering, Nakhonpathom, Thailand, 11–13 May 2011; pp. 331–336. [CrossRef]
39. Khomh, F.; Vaucher, S.; Guéhéneuc, Y.G.; Sahraoui, H. BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. *J. Syst. Softw.* **2010**, *84*, 559–572. [CrossRef]
40. Danphitsanuphan, P.; Suwantada, T. Code smell detecting tool and code smell-structure bug relationship. In Proceedings of the Spring World Congress on Engineering and Technology, Xi'an, China, 27–30 May 2012; pp. 1–5. [CrossRef]
41. Wang, X.; Dang, Y.; Zhang, L.; Zhang, D.; Lan, E.; Mei, H. Can I clone this piece of code here? In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Essen, Germany, 3–7 September 2012; pp. 170–179. [CrossRef]
42. Maiga, A.; Ali, N.; Bhattacharya, N.; Sabané, A.; Guéhéneuc, Y.G.; Antoniol, G.; Aimeur, E. Support vector machines for anti-pattern detection. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Essen, Germany, 3–7 September 2012; pp. 278–281. [CrossRef]
43. Maiga, A.; Ali, N.; Bhattacharya, N.; Sabané, A.; Guéhéneuc, Y.G.; Aimeur, E. SMURF: A SVM-based incremental anti-pattern detection approach. In Proceedings of the Working Conference on Reverse Engineering, Kingston, ON, Canada, 15–18 October 2012; pp. 466–475. [CrossRef]
44. Palomba, F.; Bavota, G.; Di Penta, M.; Oliveto, R.; De Lucia, A.; Poshyvanyk, D. Detecting bad smells in source code using change history information. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Silicon Valley, CA, USA, 11–15 November 2013; pp. 268–278. [CrossRef]
45. Palomba, F.; Bavota, G.; Di Penta, M.; Oliveto, R.; Poshyvanyk, D.; De Lucia, A. Mining version histories for detecting code smells. *IEEE Trans. Softw. Eng.* **2015**, *41*, 462–489. [CrossRef]
46. Fontana, F.A.; Zanoni, M.; Marino, A.; Mäntylä, M.V. Code smell detection: Towards a machine learning-based approach. In Proceedings of the IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, 22–28 September 2013; pp. 396–399. [CrossRef]
47. Fu, S.; Shen, B. Code Bad Smell Detection through Evolutionary Data Mining. In Proceedings of the International Symposium on Empirical Software Engineering and Measurement, Beijing, China, 22–23 October 2015; pp. 1–9. [CrossRef]
48. Amorim, L.; Costa, E.; Antunes, N.; Fonseca, B.; Ribeiro, M. Experience report: Evaluating the effectiveness of decision trees for detecting code smells. In Proceedings of the International Symposium on Software Reliability Engineering, Gaithersbury, MD, USA, 2–5 November 2015; pp. 261–269. [CrossRef]
49. Yang, J.; Hotta, K.; Higo, Y.; Igaki, H.; Kusumoto, S. Classification model for code clones based on machine learning. *Empir. Softw. Eng.* **2015**, *20*, 1095–1125. [CrossRef]
50. Palomba, F.; Panichella, A.; De Lucia, A.; Oliveto, R.; Zaidman, A. A textual-based technique for Smell Detection. In Proceedings of the IEEE International Conference on Program Comprehension, Austin, TX, USA, 16–17 May 2016; pp. 1–10. [CrossRef]
51. White, M.; Tufano, M.; Vendome, C.; Poshyvanyk, D. Deep learning code fragments for code clone detection. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 3–7 September 2016; pp. 87–98.
52. Aras, M.T.; Selcuk, Y.E. Metric and rule based automated detection of antipatterns in object-oriented software systems. In Proceedings of the International Conference on Computer Science and Information Technology, Amman, Jordan, 13–14 July 2016; pp. 1–6. [CrossRef]
53. Tarwani, S.; Chug, A. Predicting maintainability of open source software using Gene Expression Programming and bad smells. In Proceedings of the International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), Noida, India, 7–9 September 2016; pp. 452–459. [CrossRef]
54. Hozano, M.; Antunes, N.; Fonseca, B.; Costa, E. Evaluating the accuracy of machine learning algorithms on detecting code smells for different developers. In Proceedings of the International Conference on Enterprise Information Systems, Porto, Portugal, 26–29 April 2017; Volume 2, pp. 474–482. [CrossRef]

55. Fontana, F.A.; Zanoni, M. Code smell severity classification using machine learning techniques. *Knowl. Based Syst.* **2017**, *128*, 43–58. [CrossRef]
56. Kim, D.K. Finding Bad Code Smells with Neural Network Models. *Int. J. Electr. Comput. Eng. (IJECE)* **2017**, *7*, 3613–3621. [CrossRef]
57. Kaur, K.; Jain, S. Evaluation of machine learning approaches for change-proneness prediction using code smells. *Adv. Intell. Syst. Comput.* **2017**, *515*, 561–572. [CrossRef]
58. Jesudoss, A.; Maneesha, S.; Durga, T.L.N. Identification of code smell using machine learning. In Proceedings of the International Conference on Intelligent Computing and Control Systems, Madurai, India, 15–17 May 2019; pp. 54–58. [CrossRef]
59. Mhawish, M.Y.; Gupta, M. Generating Code-Smell Prediction Rules Using Decision Tree Algorithm and Software Metrics. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 41–48. [CrossRef]
60. Gupta, H.; Kumar, L.; Neti, L.B.M. An empirical framework for code smell prediction using extreme learning machine. In Proceedings of the Annual Information Technology, Electromechanical Engineering and Microelectronics Conference, Jaipur, India, 13–15 March 2019; pp. 189–195. [CrossRef]
61. Kiyak, E.O.; Birant, D.; Birant, K.U. Comparison of Multi-Label Classification Algorithms for Code Smell Detection. In Proceedings of the International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 11–13 October 2019; pp. 1–6. [CrossRef]
62. Guggulothu, T.; Moiz, S.A. Code smell detection using multi-label classification approach. *Softw. Qual. J.* **2020**, *28*, 1063–1086. [CrossRef]
63. Kaur, I.; Kaur, A. A Novel Four-Way Approach Designed with Ensemble Feature Selection for Code Smell Detection. *IEEE Access* **2021**, *9*, 8695–8707. [CrossRef]
64. Gupta, H.; Kulkarni, T.G.; Kumar, L.; Neti, L.B.M.; Krishna, A. An Empirical Study on Predictability of Software Code Smell Using Deep Learning Models. *Lect. Notes Netw. Syst.* **2021**, *226*, 120–132. [CrossRef]
65. Dewangan, S.; Rao, R.S.; Mishra, A.; Gupta, M. A novel approach for code smell detection: An empirical study. *IEEE Access* **2021**, *9*, 162869–162883. [CrossRef]
66. dos Reis, J.P.; Abreu, F.B.E.; Carneiro, G.D.F. Crowdsmelling: A preliminary study on using collective knowledge in code smells detection. *Empir. Softw. Eng.* **2022**, *27*, 69. [CrossRef]