Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

# Comparative Study on Machine Learning Algorithms for Code Smell Detection

**Hayya U[1)], Theresia Ratih Dewi Saputri[2)] ***
[1)2)]Department of Informatics, Universitas Ciputra, Surabaya, Indonesia
[1)]hayyau01@student.ac.id, [2)]theresia.ratih@ciputra.ac.id

**Submitted** :Oct 16, 2025 | **Accepted** : Nov 18, 2025 | **Published** : Jan 02, 2026

**Abstract:** Detecting code smells is crucial for maintaining software quality, but rule-based methods are often not very adaptive. On the other side, existing machine learning studies often lack large-scale comparisons on modern datasets. The goal of this research is to comprehensively compare the performance of various machine learning algorithms for multi-label code smells classification in terms of effectiveness and efficiency. The dataset used in this research is SmellyCode++, containing more than 100,000 samples. Seven models: Logistic Regression, Linear SVM, Naive Bayes, Random Forest, Extra Trees, XGBoost, and LightGBM combined with Binary Relevance were trained on data balanced using random undersampling and multi-label synthetic minority over-sampling. The performance of each model was evaluated using the F1-Macro, Hamming Loss, and Jaccard Score metrics. A non-parametric statistical analysis was also conducted to validate the findings. The experiment found that ensemble-based models statically significantly outperformed the linear and probabilistic models. The performance among the top ensemble models was found to be statistically equivalent. With this statistical equivalence in accuracy, computational efficiency measured with training time became the critical tiebreaker. BR_RandomForest, BR_XGBoost, and BR_ExtraTrees proved highly efficient, while BR_LightGBM was significantly slower. This study concludes that BR_RandomForest offers the best overall trade-off in providing top tier accuracy combined with excellent computational efficiency, making it a robust choice for practical applications.

**Keywords:** Code Smell, Ensemble, Machine Learning, Multi-label, SmellyCode++

## INTRODUCTION

Software maintenance process frequently poses a challenge in the software development lifecycle, due to the complexity and lack of documentation in the existing source code (Lacerda, Petrillo, Pimenta, & Guéhéneuc, 2020). Yet, software maintenance is one of the most important aspects in making sure of software sustainability and quality. Because of that problem, software engineers began to look for ways to solve this issue, where one solution is to identify code smells. Code smells are basically code patterns that indicate problems in technical design. In this solution code smells serve as a benchmark for software implementation quality (Dewangan, Rao, Mishra, & Gupta, 2022; Fowler & Beck Boston, 2019).

Software engineers started to use rule-based static analysis tools to detect code smells, but this approach has some limitations caused by its rigid detection rules. When applied across projects these rigid rules reduce adaptability and consistency of detection (Alawadi et al., 2024). Machine learning approaches are started to be used as a promising alternative to solve this problem (Hilmi et al., 2023). Unlike rule-based approaches that rely solely on simple thresholds, machine learning approach can find combinations and patterns from existing code metrics, making it more flexible and reliable for code smell detection (Khleel & Nehéz, 2023).

Some studies in this field still rely on datasets that are outdated and relatively limited in size, making the generalizability of their findings questionable. To address this issue, the larger dataset called SmellyCode++, with approximately 100,000 samples, was developed to be a more representative and modern resource (Alomari, Alazba, Aljamaan, & Alshayeb, 2025). With this new large-scale dataset, a specific research gap emerged: the lack of baseline comparative studies evaluating the performance of machine learning models on the dataset.

To fill this gap, this study presents a systematic comparative analysis to evaluate the performance of seven popular machine learning models representing three main approaches: baseline models, bagging-based ensembles, and boosting on the SmellyCode++ dataset. This study provides the first comprehensive benchmark of baseline, bagging-based ensembles, and boosting models on the large-scale SmellyCode++ dataset, which analyzes the

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

trade-off between model accuracy and efficiency. This study also establishes a reproducible baseline for further research in the field of code smell detection on a large scale dataset. The insight from this research is important for practical adoption of the models tested in industrial settings, especially for the integration of the models in automated code review systems or continuous integration pipelines.

## LITERATURE REVIEW

In the development of machine learning applications for code smell detection, initially the single-label approach was the main approach used. But this became a problem because it turned out that code smells often appeared simultaneously in a sample (Palomba et al., 2018). Several recent studies have applied a multi-label approach to be able to provide a solution that is more appropriate to this nature of code smell. The study by Guggulothu & Moiz (2020) became a fundamental study because it demonstrates the application of multi-label approach on code smells by comparing multi-label classification strategies such as Binary Relevance, Classifier Chains, and Label Powerset.

From that study, further research has also been developed to deepen the understanding of this problem from different angles. Some researchers focused on comparing the methods; for instance, the study by Yadav, Rao, & Mishra (2024) evaluated three main transformation strategies using several decision tree models, while the study by Roy Chowdhuri & Gupta (2025) expanded the exploration by evaluating six multi-label classification methods and twelve base classifiers. Other researchers focused on how the quality of data affected model performance. The study by Khleel & Nehéz (2023), for example, showed that dataset imbalance can greatly affect model performance and highlighted the importance of using balancing techniques like oversampling to achieve better model accuracy. Also related to data quality, research by Nandini, Singh, & Rathee (2024) showed that feature selection can be used to reduce data dimensionality and improve model accuracy. Lastly, research by Hamouda, El-Korany, & Makady (2025) explored the types of code smell that have rarely been studied before.

In parallel with those Machine Learning studies, Deep Learning approaches have also gotten attention from researchers. For instance, a study by T. Sharma, Efstathiou, Louridas, & Spinellis (2021) focused on investigating the feasibility of applying deep learning models such as CNNs and RNNs directly to tokenized source code without extensive feature engineering. In a similar approach, other researchers explored more complex architectures. For example, the study by Ali, Rizvi, & Adil (2025) introduced RABERT, a transformer-based model designed to capture interdependencies between software metrics. This line of research also includes the use of models like CodeBERT and Graph Neural Networks (GNNs) to capture complex semantic code smell patterns.

Earlier paragraphs have illustrated two main learning-based research approaches. if we take a wider context, historically code smell detection began with traditional rule-based approach, which is often referred to as rigid and reliant on predefined threshold. As the evolution, feature-based traditional machine learning approaches offered more flexibility, although it is still heavily relied on extensive feature engineering process. Recently, Deep Learning Approaches emerged as a method that can automatically infer complex patterns directly from source code, often without any feature engineering.

However, the studies about traditional machine learning approaches mentioned earlier faced similar problems regarding their validity stemming from the usage of small datasets. Most published studies including those by Guggulothu & Moiz (2020), Yadav et al. (2024), and Roy Chowdhuri & Gupta (2025), still rely on dataset containing only 445 samples, recent study by Nandini et al. (2024) is also seen with similar size of dataset which used 450 samples. Other studies, such as those by Khleel & Nehéz (2023) and Hamouda et al. (2025), although using slightly larger datasets, are still relatively limited with approximately 1,700 and 8,500 samples, respectively. This dependence on small-scale datasets is the main justification for the need to reevaluate the performance of machine learning models using a more modern and representative dataset, such as SmellyCode++, which provides 107,554 samples (Alomari et al., 2025).

## METHOD

### Research Design

This research was designed in four main stages. The first stage involves understanding the dataset through Exploratory Data Analysis (EDA). The second stage is preprocessing for data preparation which includes data balancing steps. The third stage is the modelling stage using the Binary Relevance (BR) approach. The final stage is performance evaluation using multi-label metrics and non-parametric statistical tests. The systematic process flow is illustrated in Figure 1.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

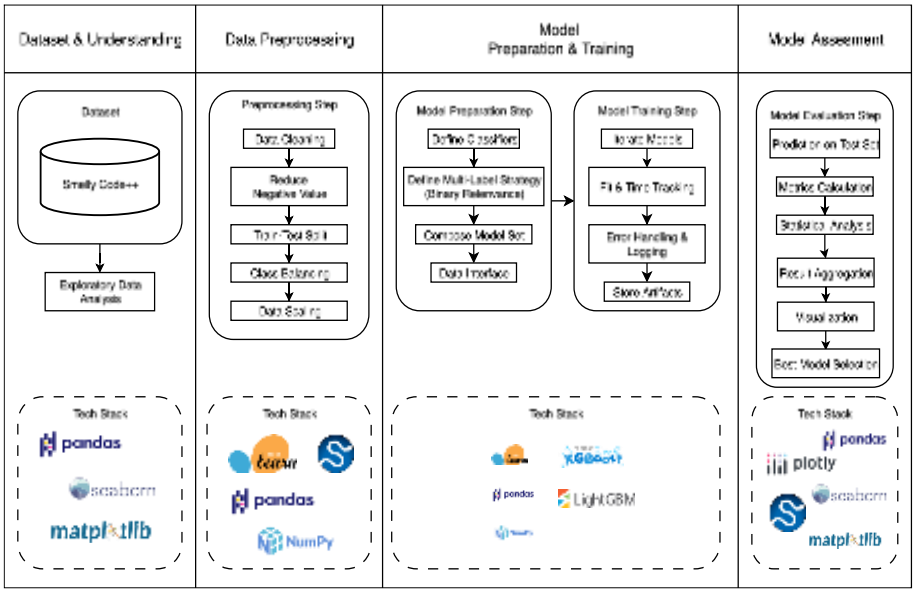

Fig. 1 Proposed Research Methodology

This research was conducted on a 2024 MacBook Pro, with an Apple M4 Pro 14-core CPU, 20-core GPU, and 24GB of LPDDR5 RAM. The environment used was Python version 3.11.11, running on MacOS version 26.0 (Tahoe) operating system. The primary libraries used in this implementation includes pandas (v2.2.3), numpy (v1.26.4), scipy (v1.13.1), matplotlib (v3.10.1), seaborn (v0.13.2), plotly (v6.3.0), scikit-learn (v1.6.1), scikit-multilearn (v0.2.0), XGBoost (v2.1.4), LightGBM (v4.6.0).

**Dataset and Understanding**

The dataset used in this study is SmellyCode++ by Alomari et al. (2025). The reason for choosing this dataset is that it is publicly available, recently released, and, most importantly, multi-labeled, which allows it to represent the co-occurrence of real-world code smells. The dataset contains 107,554 samples compiled from 103 open-source Java projects and covers four types of code smells which include Data Class, God Class, Feature Envy, and Long Method.

Exploratory Data Analysis stage is then conducted to understand the fundamental characteristics of the dataset, such as label distribution, class imbalance, and co-existence patterns between smells. The insights from this analysis will be used as the main basis for the preprocessing strategy, particularly data balancing.

**Data Preprocessing**

The preprocessing stage aims to prepare the raw data for modeling through a series of sequential steps so that it is ready for use by the model. The process begins with data cleaning to remove inconsistent or duplicate entries. After that, initial balancing between non-smelly (negative) and smelly (positive) samples is carried out by random undersampling in the negative class until a 1:1 ratio is achieved, in order to mitigate the bias of traditional machine learning models towards the majority class (Pecorelli, Di Nucci, De Roover, & De Lucia, 2020; Zhou, Liu, Yuan, & Jiang, 2024).

The dataset, which has been balanced between non-smelly and smelly samples, is then divided into 80% training and 20% testing for evaluation on data that the model has never seen before. This single train-test split strategy was chosen because the research focus was to establish an initial performance baseline on the new SmellyCode++ dataset, rather than measure generalization variance through k-fold cross-validation. Given the large scale of the dataset used, this single iterative split is considered sufficiently representative for this initial comparative evaluation. On the training data, multi-label over-sampling was performed using MLSMOTE to balance the distribution between code smell types, so that each positive class had adequate representation. This technique was chosen because it is effective in handling multi-label data imbalance (Alomari et al., 2025; Tarekegn, Giacobini, & Michalak, 2021).

Next, feature normalization is applied using StandardScaler to standardize feature magnitudes while preserving their original mean values (with_mean=False). This normalization method is chosen because it is widely used and effective for scale sensitive algorithms, such as Logistic Regression and Support Vector Machine, resulting in a more stable training process (Sharma, 2022).

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

**Model Preparation and Training**

At Model Preparation and Training Stage, algorithms are selected to compare three different computational approaches. These approaches include simple baseline models, bagging-based ensemble models, and modern boosting models. Seven algorithms in total were chosen to represent these approaches. Logistic Regression, Linear Support Vector Machine, and Naive Bayes represent the baseline models, Random Forest, and Extra Trees represent the bagging-ensemble models, and XGBoost and LightGBM represent the modern boosting models. Those models are trained using thir default parameters and combined with the multi-label classification strategy Binary Relevance. This way the evaluation only focuses on basic performance of each model before hyperparameter tuning (Bogatinovski, Todorovski, Džeroski, & Kocev, 2022). The application of default parameters on this experiment establishes a standard baseline and the Binary Relevance strategy was chosen for its simplicity and scalability for large datasets (Read, Pfahringer, Holmes, & Frank, 2011). All of the default parameters used are summarized in Table 1.

Table 1. The Default Parameters Used in Each Model

| Model | Parameter |
|---|---|
| BR_RandomForest | n_estimators=100, n_jobs=-1 |
| BR_LightGBM | n_estimators=100, n_jobs=-1 |
| BR_XGBoost | n_estimators=100 |
| BR_ExtraTrees | n_estimators=100 |
| BR_NaiveBayes | - |
| BR_LinearSVM | kernel=”linear” |
| BR_LogisticRegression | solver="lbfgs" |

The training process is then performed with automatic iteration for all model and binary relevance combination on the training dataset. During training, the computation time is recorded as part of the efficiency analysis. Additionally, error handling and logging mechanisms are implemented to ensure the experiment is reproducible. The successfully trained models are then stored together with their training time information so they can be used on the evaluation stage.

**Model Assessment**

The trained models are then tested using the test dataset that had been separated on preprocessing stage. Performance is then measured using multi-label metrics, such as Hamming Loss, Subset Accuracy, and F1-Score (micro and macro) (García-Pedrajas, Cuevas-Muñoz, Cerruela-García, & de Haro-García, 2024). Additionally, Precision, Recall, Jaccard Score, and training time were also recorded to provide a more comprehensive analysis (Rainio, Teuho, & Klén, 2024).

The results of each model combination tested were then compiled into several comparative tables for performance analysis. The evaluation focuses on comparing the F1 scores of each model and the relation between Hamming Loss and Jaccard Score, as well as the correlation between model performance and training time. Evaluation is also done per-label using Precision, Recall, and F1 scores as the metrics to evaluate the performance of each model combination on each type of code smell.

In addition to those aggregate metrics, a non-parametric statistical analysis is applied to rigorously validate the performance difference between models. The analysis is based on the per-sample Jaccard Score calculated for each data point in the test set, a standard example-based metric for multi-label evaluation (Madjarov, Kocev, Gjorgjevikj, & Džeroski, 2012). This metric was chosen because it correctly ignores the dominant true-negative matches, allowing the evaluation to focus only on the model’s ability to predict positive labels. This process will result in a matrix where each row represents a test sample and each column represents a model’s score, which allows for a paired comparative analysis on the performance of each model in individual sample.

Friedman Test is applied to determine whether there is a significant difference or not globally among the seven models used. When this test shows significance, then a post-hoc analysis using Wilcoxon signed-rank test corrected with the Bonferroni method is applied to conduct pairwise comparisons between the best performing model and every other model. This statistical pipeline follows the standard recommendations for classifier comparisons (Demšar, 2006).

The selection of the best overall model will be based on a comprehensive trade-off analysis between effectiveness and efficiency, which will be validated by statistical tests. Model effectiveness will be measured using F1-Macro as the main metric. Model efficiency will be measured using training time. Lastly, a statistical

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

significance analysis will be applied to validate the effectiveness findings. The best model will be chosen as one that offers the optimal balance between statistically validated performance and computational efficiency.

## RESULT

### Exploratory Data Analysis

On exploratory data analysis, it's found that the dataset used is very unbalanced. From the dataset, approximately 90.78% were non-smelly samples (did not contain code smell), while 9.22% were smelly samples that contained one or more types of code smell. It's also found that the distribution of smelly samples is also unbalanced. God Class is the most common type, with a percentage of 38.76%. The other types in order, are, Data Class at 29.38%, Feature Envy at 17.85%, and Long Method with the lowest percentage at 14.01%. A visualization of this distribution is presented in Figure 2.

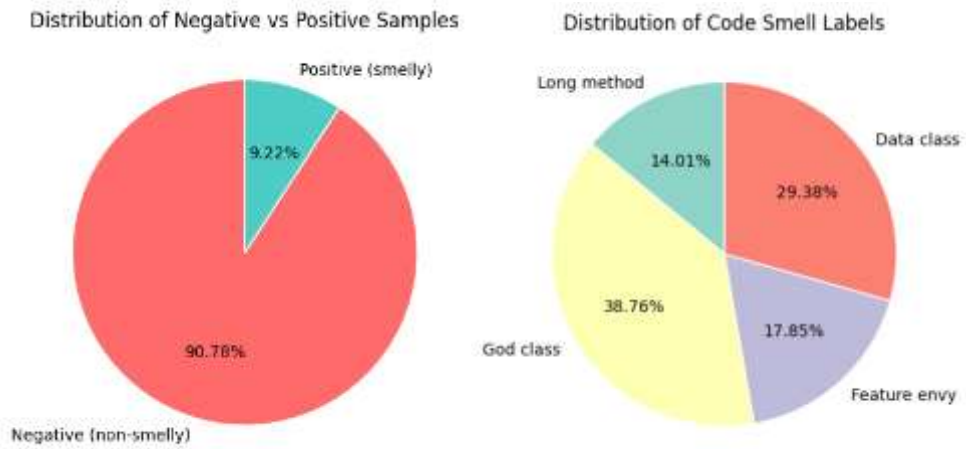


Fig. 2 Distribution of Non-Smelly and Smelly Samples, as well as Distribution Between Types of Code Smells.

Furthermore, the analysis also identified the coexistence of different types of code smells. A correlation was found between Feature Envy and Long Method with a correlation value of 0.65. Meanwhile, other pairs of code smell types showed weak or near-zero correlation values. The coexistence relationship between these labels is visualized in Figure 3.

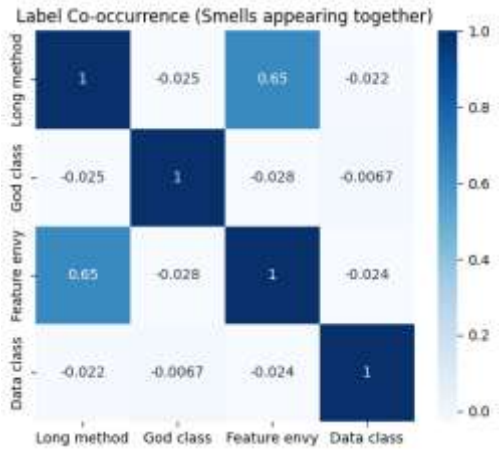


Fig. 3 Heatmap of Correlation Between Code Smells (Label Cooccurrence)

### Data Preprocessing Outcome

Data preprocessing started with checking for duplicates and missing values. The results of the check show that there are no duplicate entries or missing values, so the amount of data is unchanged. The next step is to balance the classes between non-smelly and smelly samples by applying random undersampling to the majority class samples. After this process, the dataset, which initially had a proportion of 90.78% non-smelly and 9.22% smelly, became balanced with a ratio of 1:1. A comparison of the class distribution before and after balancing is shown in Figure 4.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

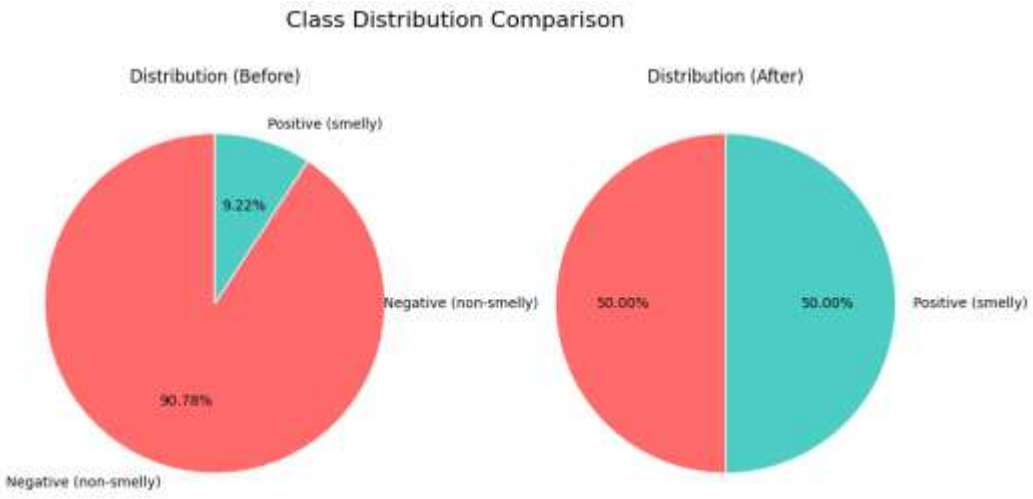


Fig. 4 Comparison of Class Distribution Before and After Balancing

The balanced dataset was then divided into a training set with the size of 80% and a test set with 20%. In the training data, oversampling was performed using the MLSMOTE to balance the distribution between code smells. Before MLSMOTE was applied, the label distribution in the training data was imbalanced with God Class at 38.76%, Data Class at 29.37%, Feature Envy at 17.86%, and Long Method at 14.01%. After applying MLSMOTE, the label distribution became more balanced with God Class at 28.74%, Feature Envy at 26.82%, Long Method at 22.65%, and Data Class at 21.78%. A comparison of the label distribution before and after the oversampling process can be seen in Figure 5.

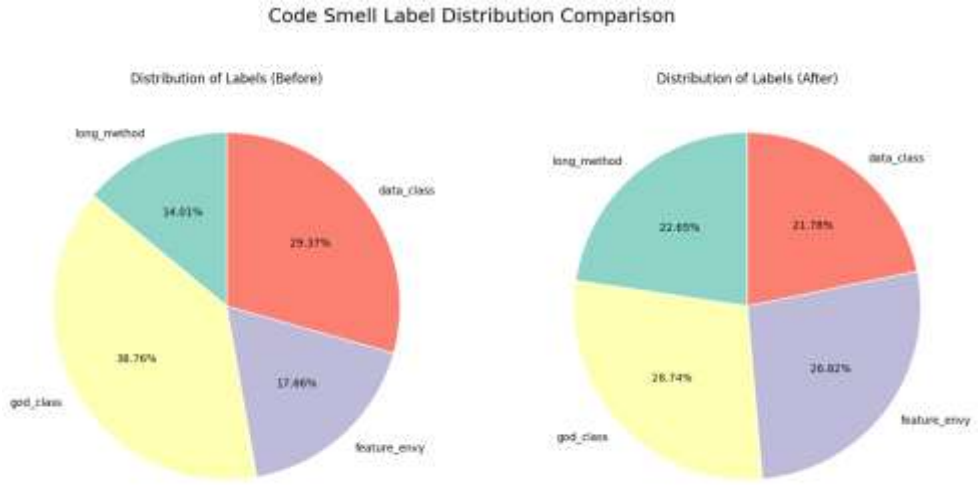


Fig. 5 Comparison of Smell Label Code Distribution

The next step is feature scaling using StandardScaler to equalize the scale between attributes without centering the data (with_mean=False).

**Model Performance Result**

The result of the experiment showed a significant difference in performance between the models tested. Ensemble-based models showed the highest performance with models such as BR_RandomForest, BR_LightGBM, BR_XGBoost, and BR_ExtraTrees obtaining F1-Macro of 62.16%, 61.36%, 61.26%, and 60.09%, respectively. On the other side, BR_NaiveBayes recorded an F1-Macro of 35.74%. Meanwhile, linear models such as BR_LinearSVM and BR_LogisticRegression recorded the lowest F1-Macro scores, namely 19.28% and 19.15%. To illustrate the performance gap visually, Figure 6 presents a bar chart comparing the F1-Macro scores for each model. A complete summary of the F1-Macro and F1-Micro scores for each model is presented in Table 2.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X


Fig. 6 Comparison of F1-Macro Scores for Each Model

Table 2. F1-Macro and F1-Micro Scores for Each Multi-Label Classification Model

| Model | F1-Macro (%) | F1-Micro (%) |
|---|---|---|
| BR_RandomForest | 62.16 | 67.06 |
| BR_LightGBM | 61.36 | 66.80 |
| BR_XGBoost | 61.26 | 66.30 |
| BR_ExtraTrees | 60.09 | 65.13 |
| BR_NaiveBayes | 35.74 | 30.60 |
| BR_LinearSVM | 19.28 | 41.75 |
| BR_LogisticRegression | 19.15 | 41.70 |

The next evaluation measures prediction quality using the Hamming Loss and Macro Jaccard Score metrics. Based on these metrics, ensemble models again show the best performance. Models such as BR_RandomForest, BR_LightGBM, BR_XGBoost, and BR_ExtraTrees showed competitive and similar scores with Hamming Loss of 8.40% to 8.80% and a Macro Jaccard Score of around 44.75% to 46.81%. On the other hand, BR_NaiveBayes recorded the highest Hamming Loss value of 55.12% and the lowest Macro Jaccard Score of 24.07%. Meanwhile, linear models such as BR_LinearSVM and BR_LogisticRegression recorded a Hamming Loss of around 11% and a Macro Jaccard Score of around 15%. A complete comparison of Hamming Loss and Macro Jaccard Score for all models is presented in Table 3.

Table 3. Comparison of Hamming Loss and Jaccard Score in Each Model

| Model | Hamming Loss (%) | Macro Jaccard Score (%) |
|---|---|---|
| BR_RandomForest | 8.40 | 46.81 |
| BR_LightGBM | 8.49 | 45.98 |
| BR_XGBoost | 8.83 | 45.82 |
| BR_ExtraTrees | 8.80 | 44.75 |
| BR_NaiveBayes | 55.12 | 24.07 |
| BR_LinearSVM | 11.06 | 15.59 |
| BR_LogisticRegression | 11.04 | 15.52 |

Computational efficiency evaluation shows that probabilistic and linear models have the fastest training times, with BR_NaiveBayes requiring only 0.01 seconds, followed by BR_LogisticRegression at 0.08 seconds and BR_LinearSVM at 0.13 seconds. In contrast, ensemble-based models require longer training times, starting from BR_ExtraTrees at 0.33 seconds, BR_RandomForest at 0.76 seconds, BR_XGBoost at 0.86 seconds, to BR_LightGBM, which recorded the longest time at 2.34 seconds. A comparison of the F1-Macro scores and training times for each model is summarized in Table 4.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Table 4. Comparison of Training Time and Model Performance

| Model | F1-Macro (%) | Training Time (second) |
|---|---|---|
| BR_RandomForest | 62.16 | 0.76 |
| BR_LightGBM | 61.36 | 2.34 |
| BR_XGBoost | 61.26 | 0.86 |
| BR_ExtraTrees | 60.09 | 0.33 |
| BR_NaiveBayes | 35.74 | 0.01 |
| BR_LinearSVM | 19.28 | 0.13 |
| BR_LogisticRegression | 19.15 | 0.08 |

**Statistical Significance Analysis**

The results of the Friedman Test, applied to the per-sample Jaccard score from each model, show that there is a statistically significant difference among models, having a Statistic of 3114.392 and a p-value approximately 0.0. The small p-value here, being less than our 0.05 significance level, confirms that the difference between model performance is statistically real and not caused by random chance, which allows for a post-hoc analysis.

A post-hoc analysis using the Wilcoxon signed-rank test with Bonferroni correction was then applied to compare the highest-scoring F1-Macro model, BR_RandomForest, with the other six models. The results of this test show that the superiority of ensemble models over probabilistic models such as BR_NaiveBayes, and linear models such as BR_LinearSVM, and BR_LogisticRegression, is confirmed to be statistically significant, with all corrected p-values being less than 0.001. The results also show an important finding that there is no statistically significant difference between BR_RandomForest and the other ensemble-based models such as BR_LightGBM, which had a corrected p-value of 1.000, BR_XGBoost with a corrected p-value of 1.000, and BR_ExtraTrees with a corrected p-value of 0.808. These high p-values indicate that the small differences on the average F1-Macro scores are likely caused by random chance. The pairwise comparison results are presented in Table 5.

Table 5. Post-Hoc Wilcoxon Test Result (Control: BR_RandomForest)

| Comparison (vs.) | p-value (Corrected with Bonferroni) | Significant (p<0.05)? |
|---|---|---|
| BR_LightGBM | 1.000 | no |
| BR_XGBoost | 1.000 | no |
| BR_ExtraTrees | 0.808 | no |
| BR_NaiveBayes | 0.000(<0.001) | yes |
| BR_LinearSVM | 2.08e−32 | yes |
| BR_LogisticRegression | 3.98e−33 | yes |

**Per-Label Model Performance Result**

Next, model performance was evaluated for detecting Data Class code smells. In this evaluation, ensemble-based models recorded the highest F1-Score. BR_XGBoost obtained an F1-Score of 57.37%, consisting of Precision 68.28% and Recall 49.47%. Other ensemble models such as BR_LightGBM and BR_RandomForest showed similar scores, with Precision values ranging from 68% to 72% and Recall below 50%. BR_NaiveBayes showed a different performance profile, producing a very high Recall of 93.91% but with a Precision of only 19.10%. On the other hand, the BR_LogisticRegression and BR_LinearSVM models did not produce any positive predictions for the Data Class label, so the Precision, Recall, and F1-Score values for these two models were 0.00%. Complete details of the performance of each model for the Data Class label are presented in Table 6.

Table 6. Model Performance on Data Class

| Model | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) | Support |
|---|---|---|---|---|---|
| BR_XGBoost | 68.28 | 49.47 | 57.37 | 87.83 | 657 |
| BR_LightGBM | 72.49 | 46.12 | 56.37 | 88.18 | 657 |
| BR_RandomForest | 71.71 | 44.75 | 55.11 | 87.93 | 657 |
| BR_ExtraTrees | 68.13 | 40.03 | 50.43 | 86.97 | 657 |
| BR_NaiveBayes | 19.10 | 93.91 | 31.75 | 33.14 | 657 |
| BR_LogisticRegression | 0.00 | 0.00 | 0.00 | 83.42 | 657 |
| BR_LinearSVM | 0.00 | 0.00 | 0.00 | 83.44 | 657 |

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Moving on to the evaluation of the Feature Envy label, it was found that most ensemble models recorded an F1-Score above 69%. BR_RandomForest obtained the highest F1-Score of 72.75%, with Precision of 73.59% and Recall of 71.93%. BR_NaiveBayes again showed a pattern with a very high Recall of 99.00% and low Precision of 12.79%. For linear models, BR_LinearSVM recorded an F1-Score of 0.50%, while BR_LogisticRegression obtained an F1-Score of 0.00%. The complete performance data for Feature Envy label detection is summarized in Table 7.

Table 7. Model Performance on Feature Envy

| Model | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) | Support |
|---|---|---|---|---|---|
| BR_RandomForest | 73.59 | 71.93 | 72.75 | 94.58 | 399 |
| BR_ExtraTrees | 72.99 | 70.43 | 71.68 | 94.41 | 399 |
| BR_XGBoost | 70.10 | 69.92 | 70.01 | 93.98 | 399 |
| BR_LightGBM | 70.90 | 67.17 | 68.98 | 93.93 | 399 |
| BR_NaiveBayes | 12.79 | 99.00 | 22.65 | 32.01 | 399 |
| BR_LinearSVM | 25.00 | 0.25 | 0.50 | 89.89 | 399 |
| BR_LogisticRegression | 0.00 | 0.00 | 0.00 | 89.94 | 399 |

The evaluation then continued on the God Class label. For this label, all models tested showed relatively high F1-Scores. BR_LightGBM recorded the highest F1-Score of 80.47%, consisting of Precision 82.11% and Recall 78.89%. Other ensemble models, including BR_RandomForest, BR_XGBoost, and BR_ExtraTrees, also obtained F1-Scores above 78%. For this label, linear models such as BR_LinearSVM and BR_LogisticRegression also recorded competitive F1-Scores of 76.63% and 76.60%, respectively. Meanwhile, BR_NaiveBayes obtained an F1-Score of 70.99%. Details of the performance comparison for the God Class label are presented in Table 8.

Table 8. Model Performance on God Class

| Model | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) | Support |
|---|---|---|---|---|---|
| BR_LightGBM | 82.11 | 78.89 | 80.47 | 91.63 | 867 |
| BR_RandomForest | 81.77 | 77.62 | 79.64 | 91.33 | 867 |
| BR_XGBoost | 80.21 | 77.62 | 78.90 | 90.93 | 867 |
| BR_ExtraTrees | 80.76 | 76.47 | 78.55 | 90.88 | 867 |
| BR_LinearSVM | 81.35 | 72.43 | 76.63 | 90.35 | 867 |
| BR_LogisticRegression | 81.43 | 72.32 | 76.60 | 90.35 | 867 |
| BR_NaiveBayes | 71.87 | 70.13 | 70.99 | 87.47 | 867 |

Finally, model performance evaluation was conducted for the Long Method label. For this label, all ensemble models recorded an F1-Score below 42%. BR_RandomForest showed the highest performance with an F1-Score of 41.12%, supported by Precision of 54.79% and Recall of 32.91%. BR_NaiveBayes again showed a pattern with a very high Recall of 98.72% and low Precision of 9.64%. Meanwhile, the linear models such as BR_LogisticRegression and BR_LinearSVM did not produce positive predictions, resulting in an F1-Score of 0.00%. Details of the evaluation results for the Long Method label are presented in Table 9.

Table 9. Model Performance on Long Method

| Model | Precision (%) | Recall (%) | F1-Score (%) | Accuracy (%) | Support |
|---|---|---|---|---|---|
| BR_RandomForest | 54.79 | 32.91 | 41.12 | 92.57 | 313 |
| BR_ExtraTrees | 55.11 | 30.99 | 39.67 | 92.57 | 313 |
| BR_LightGBM | 52.08 | 31.95 | 39.60 | 92.31 | 313 |
| BR_XGBoost | 48.56 | 32.27 | 38.77 | 91.96 | 313 |
| BR_NaiveBayes | 9.64 | 98.72 | 17.57 | 26.92 | 313 |
| BR_LogisticRegression | 0.00 | 0.00 | 0.00 | 92.11 | 313 |
| BR_LinearSVM | 0.00 | 0.00 | 0.00 | 92.09 | 313 |

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Issue 1, Januari 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15439
e-ISSN : 2541-2019
p-ISSN : 2541-044X

## DISCUSSIONS

The main finding of this study that has been validated statistically is that the ensemble-based models such as Random Forest, LightGBM, XGBoost and ExtraTrees significantly outperform probabilistic models such as NaiveBayes, and linear models such as LogisticRegression and Linear SVM.

This superiority can be explained theoretically by the fundamental ability of ensemble models in managing the bias-variance trade-off. Linear models such as BR_LogisticRegression have a high bias and fail to capture complex non-linear relationships in the code smell metrics. On the other side, ensemble models are specifically designed for this. Random Forest with BR_RandomForest, for example, applies bagging to aggerate the variance, creating hundreds of trees and aggregating their predictions through voting (Breiman, 2001). Boosting models such as BR_LightGBM and BR_XGBoost work on a similar principle, by combining multiple weak learners to create one strong learner. This ability to aggregate from many simple models allows the ensemble models to capture complex patterns stably, this explains why their performance statistically outperforms the probabilistic and linear models.

The variations on the ability to be detected between smell types as seen in table 6–9 provide some additional insights. The high performance of all models, including linear models, on the God Class with F1-Score ranging from 70.99% to 80.47% label indicates that its characteristics are relatively easy to separate linearly. On the other hand, the much lower performance on Data Class (F1-Score from 0% to 57.37%) and Long Method (F1-Score from 0% to 41.12%) shows that these smell patterns are more subtle and contextual, which pose a challenge even for more powerful models.

In terms of efficiency, the analysis in Table 4 becomes the very important tiebreaker for selecting the best overall model. As the statistical tests confirmed that the F1-Macro performance of the ensemble-models such as BR_RandomForest, BR_LightGBM, BR_XGBoost, and BR_ExtraTrees is statically equivalent, computational cost becomes the primary factor for practical implementation. The training time of BR_ExtraTrees is the fastest at 0.33 seconds, while BR_RandomForest and BR_XGBoost are near identical at 0.76 seconds and 0.86 seconds, respectively. BR_LightGBM at 2.34 seconds, is the slowest model to train, despite offering no statistical advantage in accuracy. Although BR_NaiveBayes has significantly faster training time at 0.01 seconds, its lower F1-Macro at 35.74% makes it a less viable option. This makes BR_RandomForest the best overall model. While BR_ExtraTrees is faster to train, BR_RandomForest provides the highest F1-Macro score of 62.16% within the statistically equivalent group, combined with excellent training efficiency of 0.76 seconds.

Theoretically, the contribution of this research by presenting a baseline performance benchmark of machine learning models on the SmellyCode++ dataset can serve as a foundation for further research in this field. On the practical side, the optimal trade-off between performance and efficiency found in models like BR_RandomForest (F1-Macro 62.16% in 0.76 seconds) offers practical value for the industry. These findings suggest that ensemble-based Machine Learning models can be used as a scalable alternative to rule-based detectors in CI/CD environments, where detection speed and reliability are important. The implementation of a two-stage data balancing strategy by combining undersampling and MLSMOTE in the methodology supports the findings of the study by Khleel & Nehéz (2023), and confirms that the treatment of data imbalance is a crucial step in obtaining reliable results.

This study has several limitations. First, the primary limitation of this study is the usage of single iterative train-test split with a ratio of 80 to 20, instead of using a more robust approach like k-fold cross-validation. Although the statistical test in this study confirms the validity of the findings on this specific test set, this design limits the ability to evaluate the generalizability of the models. Because of that, future research is highly recommended to replicate this study using strict multi-label stratified k-fold cross-validation to ensure that the findings can be generalized. Second, performance of the models tested are highly dependent on the SmellyCode++ dataset, this may cause the result to not generalize perfectly with projects that have different code smell definitions or labeling criteria. Third, this study is limited to feature-based Machine Learning models, which lack the ability to capture semantic-level features from the source code. This limitation is likely causing the lower performance on complex smells like Data Class or Long Method. This gives an opportunity for deep learning-based models in future research, which could improve detection accuracy by analyzing code context. Finally, the experiment conducted in this study only includes the use of the Binary Relevance multi-label strategy and applied default parameters to maintain fairness in model comparisons. Therefore, future research could also explore more complex multi-label classification methods such as Classifier Chains and Label Powersets, as well as to test the application of hyperparameter tuning to maximize the potential of the models.

## CONCLUSION

This study evaluates the performance of various machine learning algorithms in detecting code smells using a multi-label Binary Relevance approach with the SmellyCode++ dataset. The experiment confirmed two key findings. First, the statistical test results as seen in table 5, confirmed that the ensemble-based models such as BR_RandomForest, BR_LightBGM, BR_XGBoost, and BR_ExtraTrees, are statistically equivalent in terms of F1-Macro performance. Second, this ensemble group is confirmed to be statistically superior to probabilistic model

*name of corresponding author

e-ISSN : 2541-2019
p-ISSN : 2541-044X

such as BR_NaiveBayes, and linear models such as BR_LogisticRegression, and BR_LinearSVM. The equivalence in performance of those top models makes computational efficency a very important tiebreaker to determine which model is the best overall. BR_ExtraTrees is the fastest while BR_RandomForest and BR_XGBoost are highly efficent aswell. BR_LightGBM is significantly slower despite offering no statistical advantage. Therefore, this study concludes that BR_RandomForest offers the best overall trade-off, as it provides the highest average F1-Macro score within the equivalent top-tier group, combined with excellent training efficiency.

Per-label analysis confirms that the level of detection difficulty varies across categories. God Class is relatively easy to predict and can even be handled well by linear models, while Long Method is the most challenging as all models achieved a Macro F1-Score below 42%. These findings confirm that the effectivity of models is not uniform across all types of code smells and is greatly influenced by the feature distribution and attribute variation of each label.

The main limitations of this study lie in two aspects. First, the multi-label approach used is limited to the Binary Relevance strategy, which does not model the correlation between labels. Second, the scope of evaluation does not yet cover deep learning architectures that have the potential to capture more complex patterns. Therefore, further research can be directed along two main paths. It is recommended to explore more advanced multi-label strategies such as Classifier Chains or Label Powerset which able to capture label correlation and improve accuracy. In addition, the integration of deep learning-based code representation could be the next step to improve detection performance on code smells that are difficult to identify.

On a more explorative vision, future research could focus on hybrid learning models. This approach would integrate semantic code representations such as AST embeddings or GNNs that can deeply capture context and code structure, with powerful ensemble learning models like Random Forest. This vision not only aims to improve detection accuracy on complex code smells but also to enhance interpretability, which is a key factor for the adoption of these models in real-life industrial environments.

## ACKNOWLEDGEMENT

This research is funded by Universitas Ciputra Surabaya under the Implementation of the Internal Funding Program BIMA Research Scheme (DIP) Fiscal Year 2025/2026 under the agreement between the Research and Community Service Institute (LPPM) of Universitas Ciputra Surabaya with Contract Number 005/UC-LPPM/DIP-B/KP3/IX/2025, dated September 19, 2025

## REFERENCES

Alawadi, S., Alkharabsheh, K., Alkhabbas, F., Kebande, V. R., Awaysheh, F. M., Palomba, F., & Awad, M. (2024). FedCSD: A Federated Learning Based Approach for Code-Smell Detection. *IEEE Access*, *12*, 44888–44904. https://doi.org/10.1109/ACCESS.2024.3380167

Ali, I., Rizvi, S. S. H., & Adil, S. H. (2025). Enhancing Software Quality with AI: A Transformer-Based Approach for Code Smell Detection. *Applied Sciences (Switzerland)*, *15*(8). https://doi.org/10.3390/app15084559

Alomari, N., Alazba, A., Aljamaan, H., & Alshayeb, M. (2025). SmellyCode++: Multi-Label Dataset for Code Smell Detection. *Scientific Data* , *12*(1). https://doi.org/10.1038/s41597-025-05465-z

Bogatinovski, J., Todorovski, L., Džeroski, S., & Kocev, D. (2022). Comprehensive comparative study of multi-label classification methods. *Expert Systems with Applications*, *203*. https://doi.org/10.1016/j.eswa.2022.117215

Breiman, L. (2001). *Random Forests* (Vol. 45).

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. In *Journal of Machine Learning Research* (Vol. 7).

Dewangan, S., Rao, R. S., Mishra, A., & Gupta, M. (2022). Code Smell Detection Using Ensemble Machine Learning Algorithms. *Applied Sciences (Switzerland)*, *12*(20). https://doi.org/10.3390/app122010321

Fowler, M., & Beck Boston, K. (2019). *Refactoring Improving the Design of Existing Code Second Edition*. Retrieved from www.EBooksWorld.ir

García-Pedrajas, N. E., Cuevas-Muñoz, J. M., Cerruela-García, G., & de Haro-García, A. (2024, July 1). A thorough experimental comparison of multilabel methods for classification performance. *Pattern Recognition*, Vol. 151. Elsevier Ltd. https://doi.org/10.1016/j.patcog.2024.110342

Guggulothu, T., & Moiz, S. A. (2020). Code smell detection using multi-label classification approach. *Software Quality Journal*, *28*(3), 1063–1086. https://doi.org/10.1007/s11219-020-09498-y

Hamouda, E., El-Korany, A., & Makady, S. (2025). Smell-ML: A Machine Learning Framework for Detecting Rarely Studied Code Smells. *IEEE Access*, *13*, 12966–12980. https://doi.org/10.1109/ACCESS.2025.3530927

*name of corresponding author

Hilmi, M. A. Al, Puspaningrum, A., Darsih, Siahaan, D. O., Samosir, H. S., & Rahma, A. S. (2023). Research Trends, Detection Methods, Practices, and Challenges in Code Smell: SLR. *IEEE Access*, *11*, 129536–129551. https://doi.org/10.1109/ACCESS.2023.3334258

Khleel, N. A. A., & Nehéz, K. (2023). Detection of code smells using machine learning techniques combined with data-balancing methods. *International Journal of Advances in Intelligent Informatics*, *9*(3), 402–417. https://doi.org/10.26555/ijain.v9i3.981

Lacerda, G., Petrillo, F., Pimenta, M., & Guéhéneuc, Y. G. (2020). Code smells and refactoring: A tertiary systematic review of challenges and observations. *Journal of Systems and Software*, *167*. https://doi.org/10.1016/j.jss.2020.110610

Madjarov, G., Kocev, D., Gjorgjevikj, D., & Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, *45*(9), 3084–3104. https://doi.org/10.1016/j.patcog.2012.03.004

Nandini, A., Singh, R., & Rathee, A. (2024). Improving Code Smell Detection by Reducing Dimensionality Using Ensemble Feature Selection and Machine Learning. *SN Computer Science*, *5*(6). https://doi.org/10.1007/s42979-024-03013-x

Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., & De Lucia, A. (2018). A large-scale empirical study on the lifecycle of code smell co-occurrences. *Information and Software Technology*, *99*, 1–10. https://doi.org/10.1016/j.infsof.2018.02.004

Pecorelli, F., Di Nucci, D., De Roover, C., & De Lucia, A. (2020). A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. *Journal of Systems and Software*, *169*. https://doi.org/10.1016/j.jss.2020.110693

Rainio, O., Teuho, J., & Klén, R. (2024). Evaluation metrics and statistical tests for machine learning. *Scientific Reports*, *14*(1). https://doi.org/10.1038/s41598-024-56706-x

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, *85*(3), 333–359. https://doi.org/10.1007/s10994-011-5256-5

Roy Chowdhuri, S., & Gupta, M. (2025). Multilabel Classification in the Context of Code Smell Detection. *SN Computer Science*, *6*(5). https://doi.org/10.1007/s42979-025-03955-w

Sharma, T., Efstathiou, V., Louridas, P., & Spinellis, D. (2021). Code smell detection by deep direct-learning and transfer-learning. *Journal of Systems and Software*, *176*. https://doi.org/10.1016/j.jss.2021.110936

Sharma, V. (2022). A Study on Data Scaling Methods for Machine Learning. *International Journal for Global Academic & Scientific Research*, *1*(1). https://doi.org/10.55938/ijgasr.v1i1.4

Tarekegn, A. N., Giacobini, M., & Michalak, K. (2021, October 1). A review of methods for imbalanced multi-label classification. *Pattern Recognition*, Vol. 118. Elsevier Ltd. https://doi.org/10.1016/j.patcog.2021.107965

Yadav, P. S., Rao, R. S., & Mishra, A. (2024). An Evaluation of Multi-Label Classification Approaches for Method-Level Code Smells Detection. *IEEE Access*, *12*, 53664–53676. https://doi.org/10.1109/ACCESS.2024.3387856

Zhou, W., Liu, C., Yuan, P., & Jiang, L. (2024). An Undersampling Method Approaching the Ideal Classification Boundary for Imbalance Problems. *Applied Sciences (Switzerland)*, *14*(13). https://doi.org/10.3390/app14135421

*name of corresponding author