COSTAR: Software Code Smell Detection through Tree-based Abstract Representation

Praveen Singh Thakur, Mahipal Jadeja *Member, IEEE*, Satyendra Singh Chouhan *Senior Member, IEEE*, and Santosh Singh Rathore

I. Computational Complexity Analysis

The overall complexity of COSTAR is dominated by AST path extraction and embedding generation. For a dataset with n code snippets, each having on average m AST nodes and $p = O(m)$ paths, the per-snippet complexity is:

$$O(m \cdot L^2),$$

where L is the average path length. Thus, the total complexity over all snippets is:

$$O(n \cdot m \cdot L^2).$$

where:

- n: Number of snippets.
- m: Average/max number of AST nodes per snippet.
- L: Average/max path length ($L = O(\log m)$ for balanced ASTs, $L = O(m)$ for degenerate ASTs).

Although COSTAR introduces a higher preprocessing cost due to AST path extraction, its scalability remains comparable to embedding-based approaches and practical for medium- and large-scale datasets.

II. Performance Measures

We evaluated the proposed COSTAR and other code smell detection models using various performance measures that include accuracy, precision, recall, f1-score, MCC, Cohen's Kappa, and AUCPR performance measures. High precision shows the model effectively reduces false positives, saving developer effort. High recall ensures minimal code smells are missed, preventing potential bugs. The F1-score combines these metrics, summarizing the model's accuracy in identifying true positives while avoiding false positives. Other performance measures further enhance the evaluation's soundness. The code smell detection models output a confusion matrix with parameters: T_S (True Smelly), T_{NS} (True Non-smelly), F_S (False Smelly), and F_{NS} (False Non-smelly). Performance measures are calculated using this matrix. The description of performance measures is given in the supplementary file uploaded with the paper[1].

Praveen Singh Thakur, Mahipal Jadeja, and Satyendra Singh Chouhan (Corresponding author) are with the Department of Computer Science and Engineering, Malaviya National Institute of Technology, Jaipur, 302017, INDIA E-mail: {2021rcp9036, mahipaljadeja.cse, sschouhan.cse}@mnit.ac.in

Santosh Singh Rathore is with the Department of Computer Science and Engineering, ABV-IIITM Gwalior, 474015 INDIA.
E-mail: santoshs@iiitm.ac.in

[1]https://github.com/psthakur14/COSTAR.git

To statistically compare different ML-based code smell detection models, we conducted a non-parametric analysis using the Wilcoxon signed-rank test with Bonferroni correction to control for Type I errors. The Wilcoxon signed-rank test is suitable for comparing two paired samples [1]. This test involves two hypotheses: the null hypothesis (H_0) and the alternative hypothesis (H_1). The null hypothesis typically states that there is no statistically significant difference between the two samples, while the alternative hypothesis claims that a significant difference exists.

III. Techniques to Build Code Smell Detection Mode

Logistic Regression models the relationship between dependent and independent variables. Based on this relationship, it predicts the probability of an instance belonging to a specific class [2]. Support Vector Machine is a versatile technique used for both classification and regression tasks. In classification, SVM creates a hyperplane that maximizes the margin between the closest data points of two classes [3]. Decision Tree is a tree-based classifier that builds a tree-like structure to represent and classify data [4]. Unlike DT, Random Forest constructs multiple decision trees from random samples of the data [2]. Naive Bayes utilizes Bayes' theorem to learn the data distribution. It assumes that the features of the instances are independent across all datasets [5]. K-Nearest Neighbors classifier employs the nearest neighbor concept to determine the class of the input instances. It classifies an instance based on the majority class of its k nearest neighbors [6]. Quadratic discriminant analysis follows the Gaussian mixture model concept to learn the distribution of data. It assumes each class forms a separate cluster with distinct covariance matrices. Therefore, it estimates the covariance matrix for each class individually [7]. AdaBoost builds the model by initially assigning equal weights to each instance. Based on the predictions, it identifies and increases the weights of misclassified instances. This process continues until the error reaches a minimum [8]. Gradient boosting combines multiple weak learner classifiers to make predictions [8]. On the other hand, extreme gradient boosting is an advanced gradient-boosting technique similar to gradient descent. It can handle missing data and prevent overfitting [9]. Techniques using neural networks, like MLP, contain one or more hidden layers. These hidden layers allow the MLP network to learn complex patterns from the training data. The MLP uses backpropagation to update the weights of the network, minimizing the predicted error [10]. Ensemble

TABLE I
DESCRIPTION OF THE MODEL'S PERFORMANCE MEASURES

Performance Measures	Description
Accuracy	It is used as a fundamental performance metric for evaluating the overall performance of the model. $accuracy = \frac{T_S+T_{NS}}{T_S+F_S+T_{NS}+F_{NS}}$
Precision	It represents the ratio of correctly predicted smelly items to the total number of predicted smelly items. $Precision = \frac{T_S}{T_S+F_S}$
Recall	It evaluates a model's ability to correctly identify actual smelly items out of all available smelly items. It is also known as sensitivity (true positive rate). $Recall = \frac{T_S}{T_S+F_{NS}}$
F1-score (F1-s)	It provides a balanced measure between precision and recall. The F1-score is calculated as the harmonic mean of precision and recall. $F1-score = \frac{2\times precision\times recall}{precision+recall}$
MCC	MCC gauges the correlation between actual and predicted values, offering a comprehensive performance evaluation, especially valuable for imbalanced datasets. $MCC = \frac{(T_S\times T_{NS})-(F_S\times F_{NS})}{\sqrt{(T_S+F_S)(T_{NS}+F_S)(T_S+F_{NS})(T_{NS}+F_{NS})}}$
Cohen's Kappa	The value for kappa can be less than 0 (-ve). However, instead of overall accuracy, Kappa takes imbalance class distribution into account. $cohen's\ kappa = \frac{ac_0-ac_e}{1-ac_e}$ Where ac_0 is the overall model's accuracy and ac_e is the sum of the probability that the prediction agrees with actual values by chance for the non-smelly class and for the smelly class, respectively.
AUCPR	AUCPR are computed by plotting a precision-recall curve for a binary classification model. This curve shows how precision and recall change as the decision threshold for classifying instances as positive or negative is adjusted.

Neural Network (ENN) combines multiple individual neural networks, each trained independently, to contribute to the final prediction. The final prediction is determined by techniques like majority voting [11]. Weighted Neural Network (WNN) is an enhanced neural network designed to prevent bias towards the majority class by increasing weights of minority class instances, allowing the classifier to learn more effectively from them [11].

IV. COMPARISION WITH STATE-OF-THE-ART

Table II compares COSTAR and previous research, regardless of the datasets. This implies that studies focusing on detecting code smells such as Data Class, God Class, Feature Envy, or Long Method are included in this comparison. The datasets included may vary across the respective studies, and the experimental environments can also differ. For the sake of comparison, we have included the results directly reported in the studies. In this context, we included the best results for each code smell as provided by the respective studies. Table II illustrates that the COSTAR provides improved results with respect to the three code smells out of the four. For the God Class code smell, COSTAR lags behind the technique presented by Rajwant et al. [12].

In [13], the authors investigated the performance of the Naïve Bayes (NB) classifier in code smell detection. They employed various imbalance learning techniques such as SMOTE, Class Balancer, and Cost-Sensitive Classifier to address data imbalance issues. In [14], the authors proposed a Convolutional Neural Network (CNN) model for detecting eight code smells, including the Long Method. They reported the highest F1-score (0.75) for the Long Method, which is significantly lower than the COSTAR technique. In [12], the authors investigated the performance of the ML technique on different severity levels. The final results were reported by calculating the average of all severity levels. [15] investigated the transfer learning approach in code smell detection, testing various deep-learning techniques. They found that the auto-encoder yielded more promising results than other methods. In [16], the author introduced a deep learning model, DeepSmells, by employing the LSTM immediately following the CNN network. In [17], the authors used a deep learning model to automatically extract the features from the source code and build the complex mapping between them.

TABLE II
THEORETICAL PERFORMANCE COMPARISON WITH OTHER WORKS

Related Work	Learning Models	Precision	Recall	F1-score
		God Class		
Rajwant et al. [12]	RF	0.85	0.85	0.85
Fabiano et al. [13]	NB + SMOTE	0.26	0.93	0.41
Hui et al. [17]	DNN	0.12	0.8	0.22
COSTAR	WNN	0.80	0.75	0.77
		Data Class		
Rajwant et al. [12]	DT	0.84	0.83	0.84
COSTAR	RF	0.91	0.81	0.85
		Feature Envy		
Tushar et al. [15]	Auto Encoder	0.18	0.24	0.21
Anh Ho et al. [16]	CNN + LSTM	0.34	0.25	0.29
Hui et al. [17]	DNN	0.36	0.88	0.51
COSTAR	RF	0.96	0.87	0.91
		Long Method		
Lin et al. [14]	CNN	0.53	0.67	0.75
Fabiano et al. [13]	LR	0.15	0.56	0.23
Hui et al. [17]	DNN	0.42	0.78	0.55
COSTAR	AdaBoost	0.93	0.82	0.86

A. Analysis of Performance Variability across Folds

We have analyzed the variability of used classifier performances across 10 folds. Figures 1, 2, 3, 4, 5, 6, 7 (boxplots) illustrate fold-to-fold fluctuations for all evaluation metrics (accuracy, precision, recall, F1, Cohen's Kappa, MCC, AUCPR). This analysis focuses on the degree of stability versus variability across classifiers and datasets rather than focusing on the absolute performance values. The analysis results are as follows.

1) **Data Class Dataset** For the Data Class dataset, most classifiers demonstrate low variability. LR, SVM, RF,

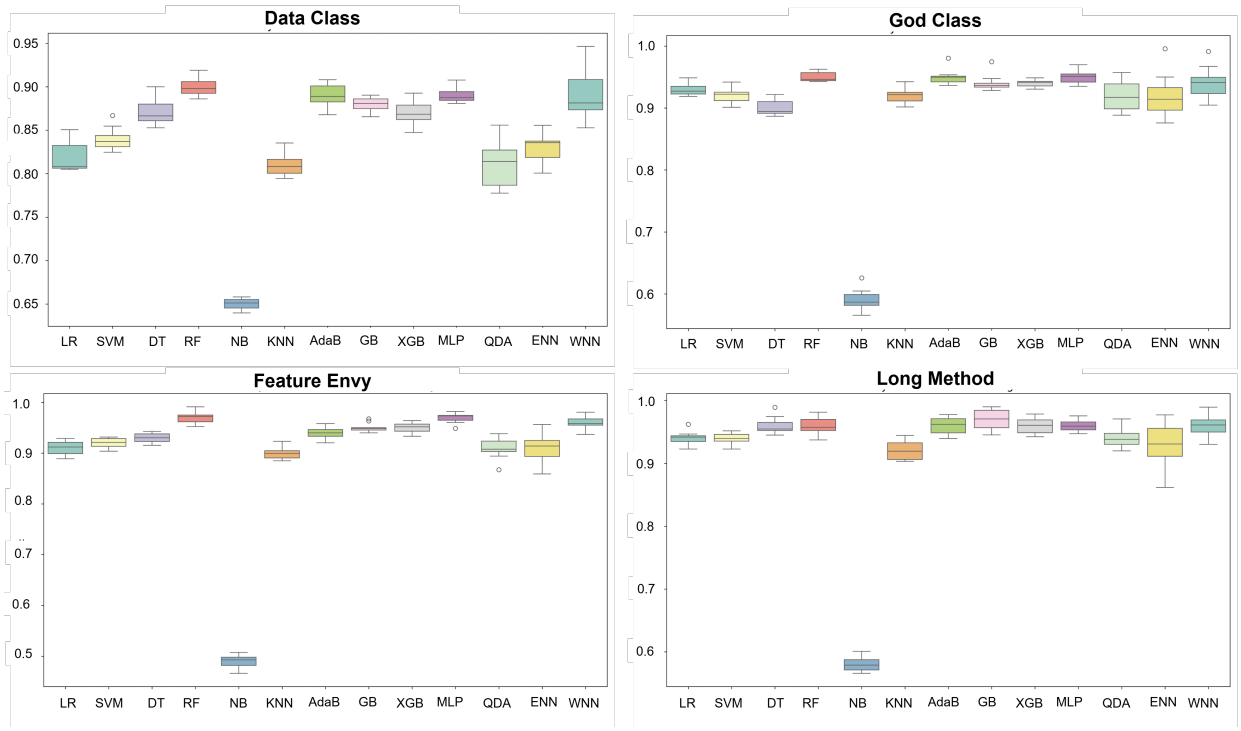

Fig. 1. Boxplots showing performance variability based on the Accuracy of COSTAR across 10 experimental repetitions for all four code smells

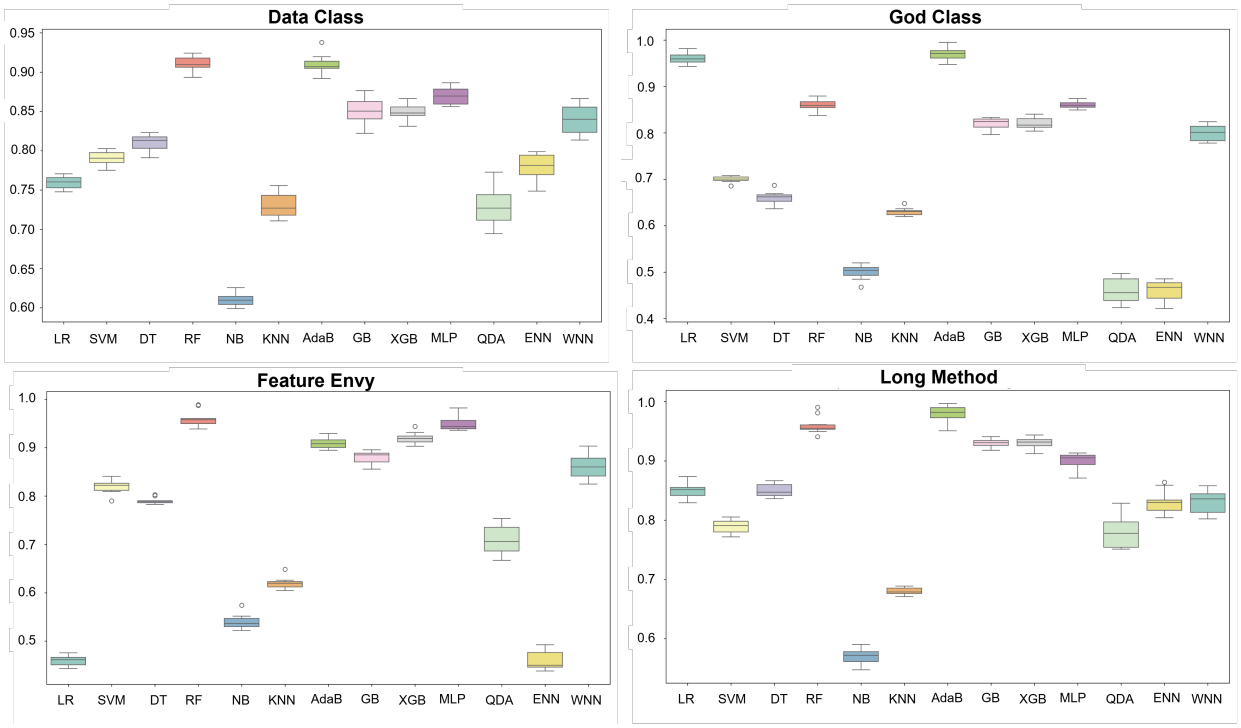

Fig. 2. Boxplots showing performance variability based on the Precision of COSTAR across 10 experimental repetitions for all four code smells

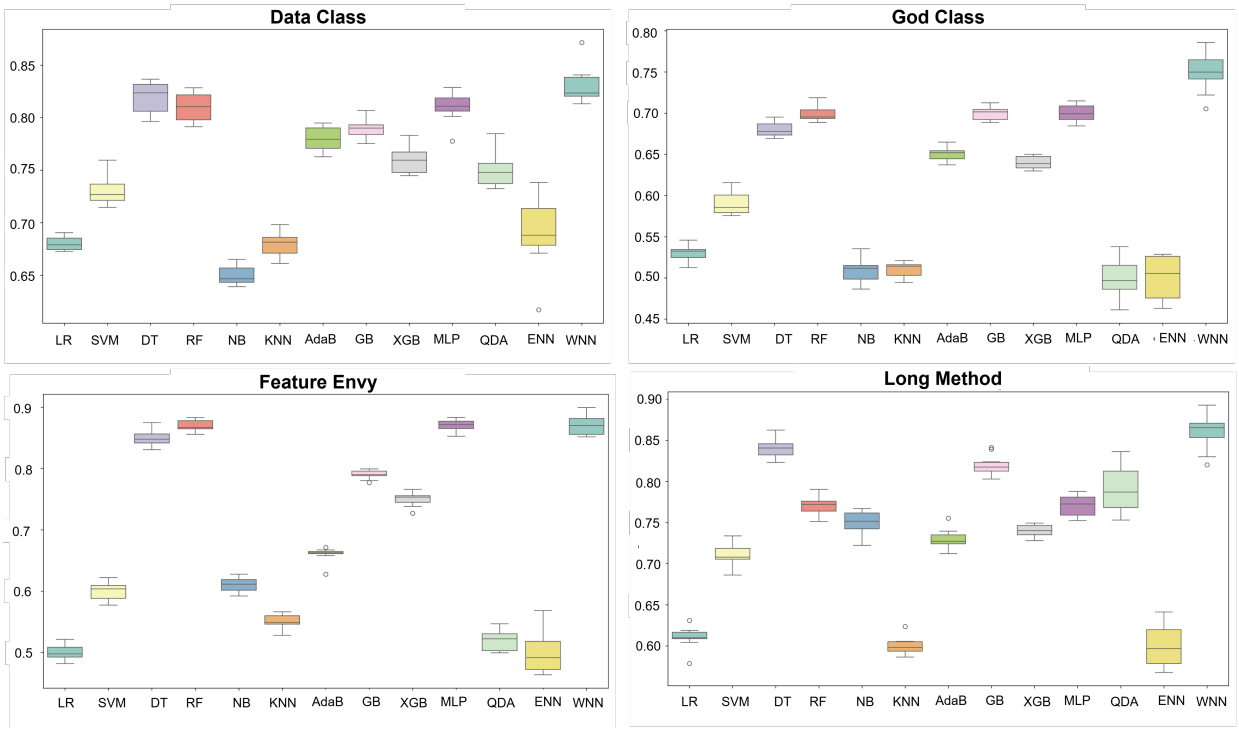

Fig. 3. Boxplots showing performance variability based on the Recall of COSTAR across 10 experimental repetitions for all four code smells

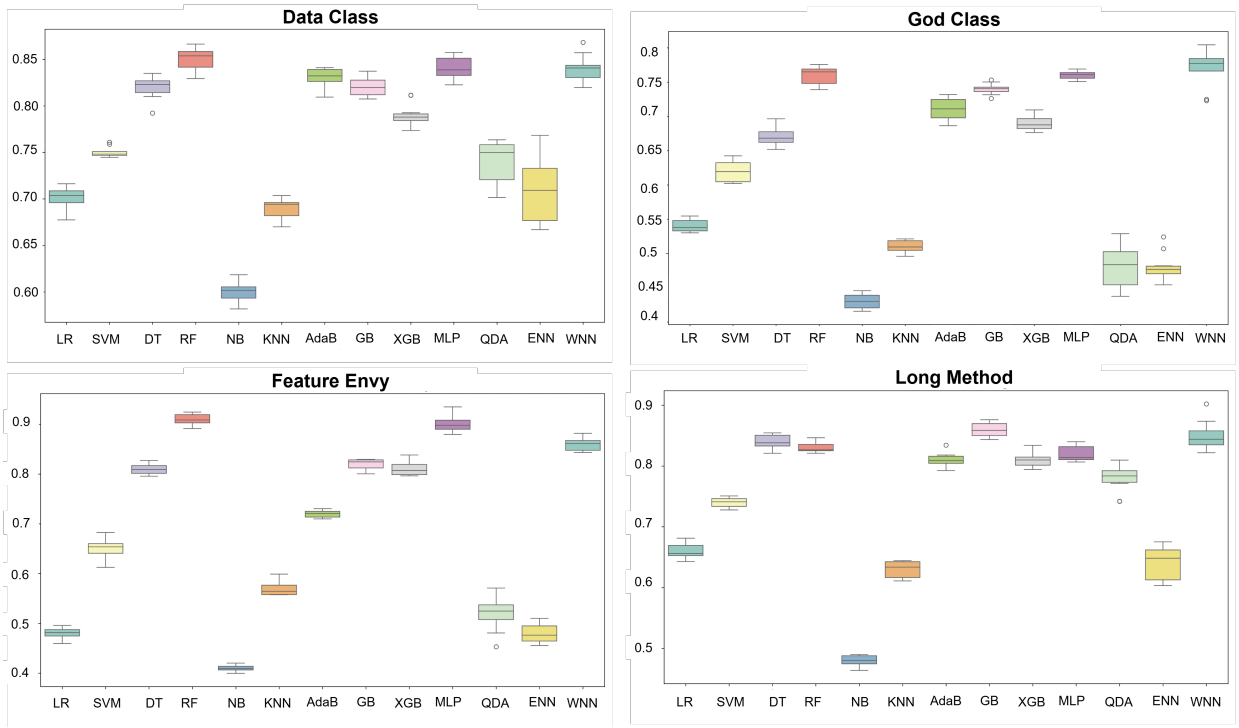

Fig. 4. Boxplots showing performance variability based on the F1-score of COSTAR across 10 experimental repetitions for all four code smells

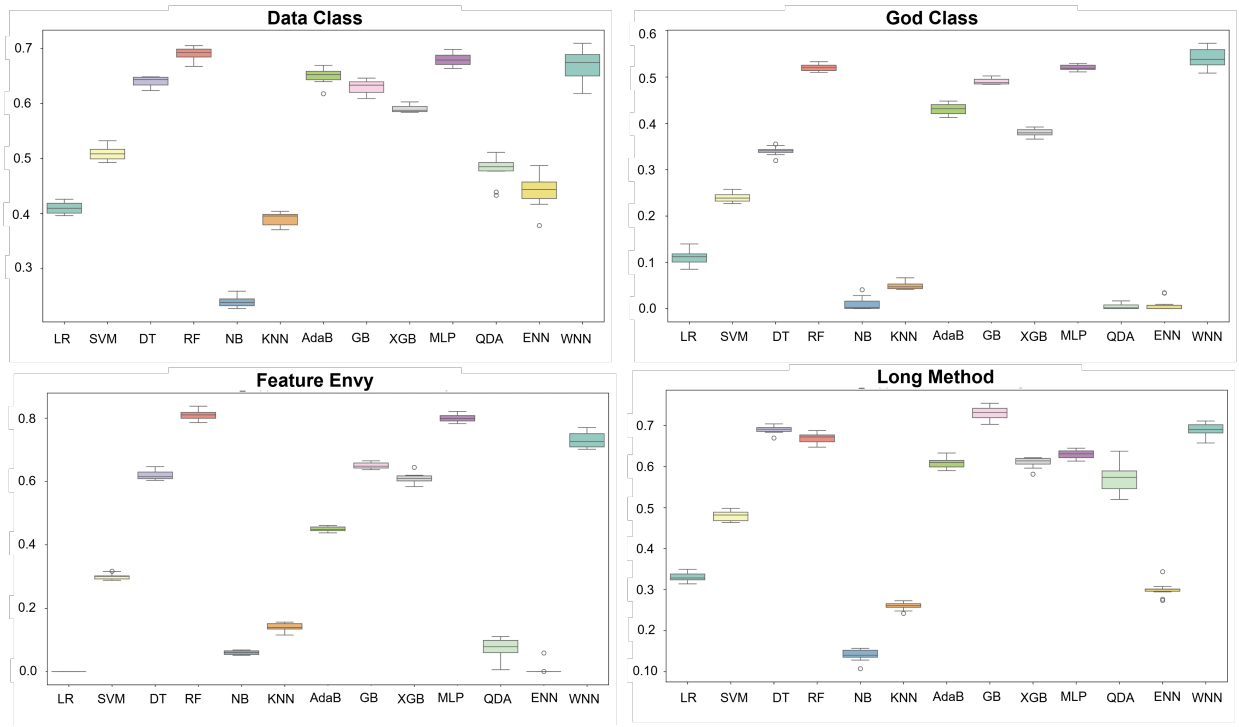

Fig. 5. Boxplots showing performance variability based on the Cohen Kappa of COSTAR across 10 experimental repetitions for all four code smells

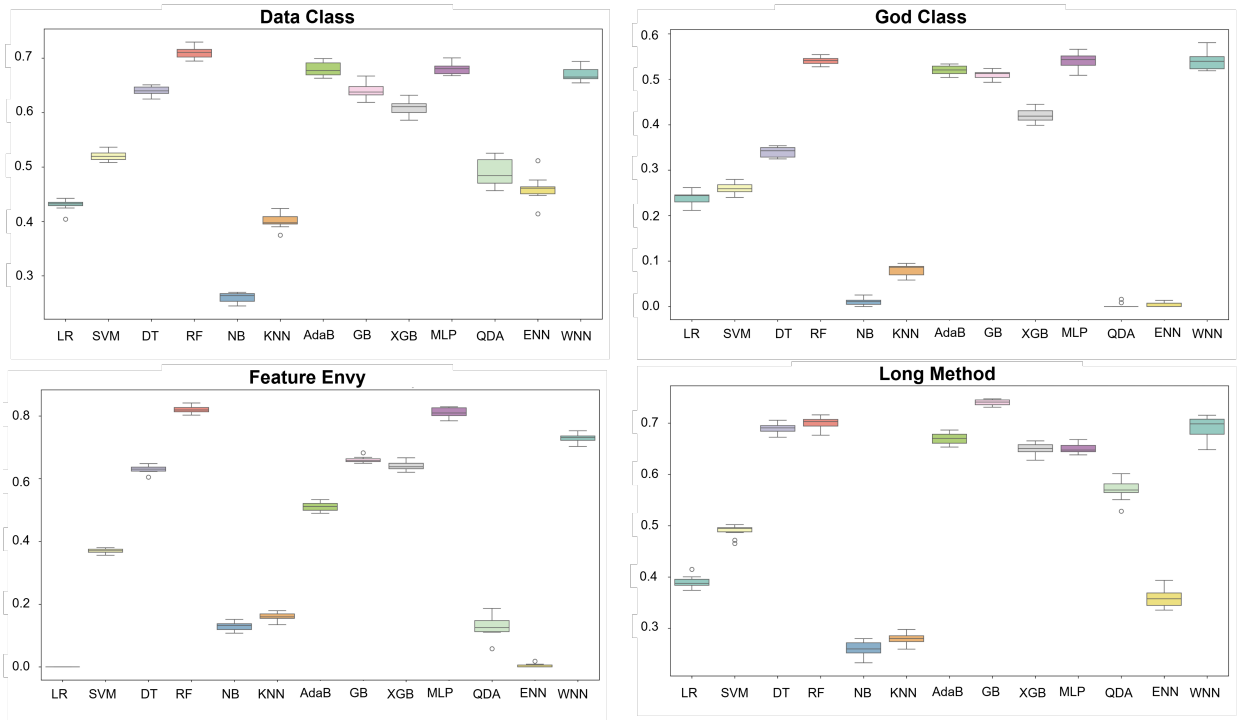

Fig. 6. Boxplots showing performance variability based on the MCC of COSTAR across 10 experimental repetitions for all four code smells

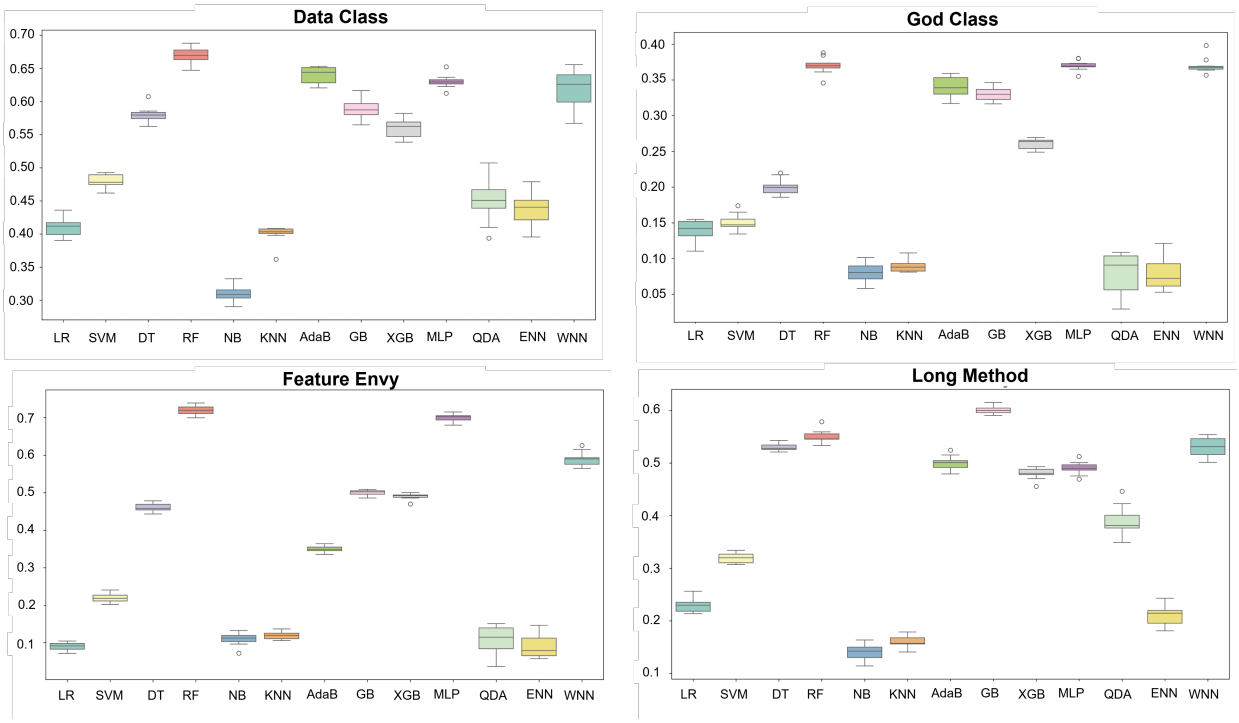

Fig. 7. Boxplots showing performance variability based on the AUCPR of COSTAR across 10 experimental repetitions for all four code smells

GB, and XGB exhibit narrow boxplots with IQRs below $\approx 0.03 - 0.04$ across most metrics, indicating that fold-to-fold variation is negligible. Their medians remain stable across accuracy, precision, recall, and F1-score, with no significant outliers. The MLP also shows narrow spreads, further confirming robustness.

By contrast, DT, AdaBoost, and KNN display moderate variability, with IQRs in the $\approx 0.06 - 0.08$ range. Their recall and MCC plots reveal folds that deviate noticeably, though overall distributions remain centered around competitive median values. WNN, ENN exhibit wider spreads ($\approx 0.07 - 0.10$), particularly in recall and AUCPR, highlighting reduced stability. NB and QDA show limited fold-to-fold variance, but their absolute scores cluster around poor baselines ($\approx 0.45 - 0.55$), making them consistently unreliable despite narrow IQRs.

2) **God Class dataset** For the God Class dataset, classifier variability is generally lower than in the Data Class, with several algorithms producing highly compact distributions. LR, SVM, Random Forest, GB, XGB, and MLP exhibit IQRs typically below ≈ 0.03, suggesting that fold-to-fold differences are minimal. Median values for accuracy, precision, and F1-score remain consistently high ($\approx 0.85 - 0.92$) with no extreme outliers, confirming stable predictive ability.

Decision Trees, AdaBoost, and KNN reveal moderate variability, with IQRs in the $\approx 0.05 - 0.07$ range, especially in recall and MCC. Despite slightly wider spreads, these classifiers maintain competitive median values. WNN and ENN show greater inconsistency, with IQRs approaching $\approx 0.08 - 0.10$ and outliers visible in recall and AUCPR. NB and QDA again remain weak; their distributions are narrow ($IQR < 0.04$) but centered at very low medians (≈ 0.50), making them consistently unreliable.

3) **Feature Envy dataset** For the Feature Envy dataset, variability is higher than in both the Data Class and the God Class. Random Forest, GB, and XGB still achieve stable performance, with IQRs around $\approx 0.04 - 0.05$ across accuracy, precision, and F1-score. Their medians remain among the top performers ($\approx 0.80 - 0.88$). SVM and LR also maintain relatively compact spreads, though with slightly wider boxes compared to tree-based ensembles.

However, DT, AdaB, and KNN exhibit moderate to high variability, with IQRs ranging from $\approx 0.07 - 0.10$ across multiple metrics. Their recall and AUCPR show notable fold-to-fold swings. WNN and ENN perform particularly inconsistently, with wide IQRs ($\approx 0.10 - 0.12$) and frequent outliers, suggesting sensitivity to data partitioning. NB and QDA again demonstrate consistently poor absolute performance, with accuracy and precision values clustering around $\approx 0.45 - 0.55$. Although their IQRs remain small (< 0.04), the consistently low medians indicate weak generalization.

4) **Long Method dataset** For the Long Method dataset, classifier variability is moderate, falling between God Class (stable) and Feature Envy (unstable). RF, GB, and XGB once again dominate with compact distributions, maintaining IQRs below $\approx 0.04 - 0.05$ across most metrics. Their medians remain consistently high ($\approx 0.82 - 0.90$), and very few outliers appear. SVM, LR, and MLP also demonstrate relatively narrow spreads,

confirming robust generalization.
DT, AdaB, and KNN exhibit moderate variability, with IQRs in the $\approx 0.06 - 0.08$ range, particularly affecting recall and MCC. WNN and ENN are less stable, with wider spreads ($\approx 0.08 - 0.10$) and visible fold-to-fold fluctuations in AUCPR. NB and QDA once again show narrow spreads ($IQR < 0.04$) but remain clustered at low medians ($\approx 0.45 - 0.55$), making them unreliable despite their apparent consistency.

REFERENCES

[1] R. F. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[2] K. Kirasich, T. Smith, and B. Sadler, "Random forest vs logistic regression: binary classification for heterogeneous datasets," *SMU Data Science Review*, vol. 1, no. 3, p. 9, 2018.

[3] A. Kaur, S. Jain, and S. Goel, "A support vector machine based approach for code smell detection," in *2017 International Conference on Machine Learning and Data Science (MLDS)*. IEEE, 2017, pp. 9–14.

[4] S. B. Kotsiantis, I. Zaharakis, P. Pintelas *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[5] D. Berrar, "Bayes' theorem and naive bayes classifier," *Encyclopedia of bioinformatics and computational biology: ABC of bioinformatics*, vol. 403, p. 412, 2018.

[6] H. A. Abu Alfeilat, A. B. Hassanat, O. Lasassmeh, A. S. Tarawneh, M. B. Alhasanat, H. S. Eyal Salman, and V. S. Prasath, "Effects of distance measure choice on k-nearest neighbor classifier performance: a review," *Big data*, vol. 7, no. 4, pp. 221–248, 2019.

[7] S. Bhattacharyya, A. Khasnobish, S. Chatterjee, A. Konar, and D. Tibarewala, "Performance analysis of lda, qda and knn algorithms in left-right limb movement classification from eeg data," in *2010 International conference on systems in medicine and biology*. IEEE, 2010, pp. 126–131.

[8] P. Bahad and P. Saxena, "Study of adaboost and gradient boosting algorithms for predictive analytics," in *International Conference on Intelligent Computing and Smart Communication*. Springer, 2020, pp. 235–244.

[9] Z. E. Aydin and Z. K. Ozturk, "Performance analysis of xgboost classifier with missing data," *Manchester Journal of Artificial Intelligence and Applied Sciences (MJAIAS)*, vol. 2, no. 02, p. 2021, 2021.

[10] S. Dewangan, R. S. Rao, A. Mishra, and M. Gupta, "A novel approach for code smell detection: an empirical study," *IEEE Access*, vol. 9, pp. 162 869–162 883, 2021.

[11] Z.-H. Zhou and S. Chen, "Neural network ensemble," *CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION-*, vol. 25, no. 1, pp. 1–8, 2002.

[12] R. S. Rao, S. Dewangan, A. Mishra, and M. Gupta, "A study of dealing class imbalance problem with machine learning methods for code smell severity detection using pca-based feature selection technique," *Scientific Reports*, vol. 13, no. 1, p. 16245, 2023.

[13] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, "On the role of data balancing for machine learning-based code smell detection," in *Proceedings of the 3rd ACM SIGSOFT international workshop on machine learning techniques for software quality evaluation*, 2019, pp. 19–24.

[14] T. Lin, X. Fu, F. Chen, and L. Li, "A novel approach for code smells detection based on deep leaning," in *Applied Cryptography in Computer and Communications: First EAI International Conference, AC3 2021, Proceedings 1*. Springer, 2021, pp. 171–174.

[15] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, "Code smell detection by deep direct-learning and transfer-learning," *Journal of Systems and Software*, vol. 176, p. 110936, 2021.

[16] A. Ho, A. M. Bui, P. T. Nguyen, and A. Di Salle, "Fusion of deep convolutional and lstm recurrent neural networks for automated detection of code smells," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 229–234.

[17] H. Liu, J. Jin, Z. Xu, Y. Zou, Y. Bu, and L. Zhang, "Deep learning based code smell detection," *IEEE transactions on Software Engineering*, vol. 47, no. 9, pp. 1811–1837, 2019.