2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA) | 978-1-6654-4337-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICMLA52953.2021.00148

Voting Heterogeneous Ensemble for Code Smell Detection

Hamoud Aljamaan
Information and Computer Science Department
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
hjamaan@kfupm.edu.sa

***Abstract*—Code smells are poor design and implementation choices that hinders the overall software quality. Code smells detection using machine learning models has been an active research area to assist software engineers in identifying smelly code. In this paper, we empirically investigate the detection performance of Voting ensemble in detecting class-level and method-level code smells. We built our Voting ensemble in a heterogeneous manner using five different base models: Decision Trees, Logistic Regression, Support Vector Machines, Multi-Layer Perceptron, and Stochastic Gradient Descent models. Predictions output were aggregated using the Soft voting to form the final ensemble prediction output. Voting ensemble detection performance was evaluated against each base model and within the context of five code smells: God Class, Data Class, Long Method, Feature Envy, Long Parameter List, and Switch Statements smells. Statistical pairwise comparison results indicates the superior performance of Voting ensemble in detecting all code smells, while base models had varying detection performance across code smells.**

***Index Terms*—Code smells detection, Ensemble learning, Machine learning, Voting**

I. Introduction

Motivation. Software quality is an important success factor for any software product. There are many factors that might hinders the overall desired software quality. One of the main factors that negatively impacts the software quality and maintenance is poor design and code choices made by software engineers [1], formally known as code smells [2]. Machine learning models has been proposed for code smells detection allowing software quality analysts in identifying and prioritising the inspection of smelly code instances. Thus, assisting them in better allocation of quality assurance and refactoring activities and achieving the targeted software quality.

Objective. Code smells detection using machine learning models varies from one code smell to another. None of the proposed individual models in the literature proved to be stable in detecting different code smells types. Ensemble learning has been proven to provide a more stable performance in classification and regression problems in other domains [3]. However, the capabilities of ensemble learning has not been fully explored in code smells detection [4], [5]. In this paper, we aim to investigate the effectiveness of the Voting ensemble in detecting class-level and method-level code smells, and compare ensemble detection performance against the Voting ensemble constituent base models.

Contribution. The main contribution of this paper is to empirically investigate the effectiveness of Voting ensemble in detecting different code smells. Our proposed Voting ensemble was constructed in a heterogeneous manner using different base models types: Decision Trees, Logistic Regression, Support Vector Machines, Multi-Layer Perceptron, and Stochastic Gradient Descent models. Detection performance was statistically compared against all base models in detecting two class-level and four method-level code smells: God Class, Data Class, Long Method, Feature Envy, Long Parameter List, and Switch Statements code smells. Statistical pairwise comparisons proved the superiority and stability of our proposed Voting ensemble in detecting all code smells over all base models.

Structure. The rest of this paper is organized as follows: Section II summarizes related work on code smell detection using machine learning models. Section III describes the constructed Voting ensemble architecture. Section IV outlines the details of our conducted empirical study. Section V discusses the obtained results with models pairwise statistical analysis. Section VI concludes the paper with future work directions.

II. Literature Review

Code smell detection using machine learning models is an active research area. Many researchers employed several machine learning models to detect different code smells, and most of the existing research focuses on building individual stand-alone machine learning models, such as: Bayesian Belief Networks (BBNs) [6]–[9], Naive Bayes [10], SVM [9]–[13], Artificial Neural Network [14], Decision Trees [9], [10]. However, none of these individual machine learning models have been proven to be stable in detecting all code smells types.

Ensemble learning [15] is a promising area in machine learning due to its capabilities and stable performance in both classification and regression tasks. Nonetheless, relatively low number of studies were conducted to investigate the effectiveness of ensemble learning in code smell detection [4], [5]. Studies employing ensemble learning in code smells detection focused mainly on boosting ensemble learning (i.e. homogeneous ensembles).

Fontana et al. [16] conducted the largest experiment to examine the applicability of 16 machine learning models in detecting two class-level and two method-level code smells (Data Class, Large Class, Feature Envy, Long Method). Two boosting ensembles were used: AdaBoost and Random Forest. RF achieved one of highest performances in smells detection. In case of AdaBoost, application of the boosting technique on individual models did not always improve the detection performance, and in some cases, it made them worse.

In a recent study, Alazba and Aljamaan [17] conducted a study to examine the detection performance of 14 individual classifiers and three stacking heterogeneous ensemble models in detecting two class-level and four method-level code smells. Results indicates the stacking ensembles superiority over all individual classifiers. In summary, the effectiveness of the Voting ensemble was not previously investigated in the context of code smell detection. This paper aims to contribute in the area of code smells detection using ensemble learning.

III. Voting Ensemble

Voting ensemble [15] is an ensemble learning algorithm that combines the output of two or more base machine learning models. Voting ensemble can be either a heterogeneous ensemble or a homogeneous ensemble depending on the type of base models. If base models are from different types (e.g. Decision trees and Support Vector Machines), then the Voting ensemble will be categorized as a heterogeneous ensemble. Otherwise, if the base models is from the same type, then it will be categorized as a homogeneous ensemble.

Fig 1 presents the Voting ensemble architecture consisting mainly of two layers: base models layer and a voting layer. In the first layer, all base models are trained using the training data and then the models predictions is fed to the next layer to perform the voting. In the second layer, models predictions produced from the testing data is aggregated to form the ensemble prediction. In binary classification tasks, the Voting ensemble can aggregates models prediction in two modes: Hard vote or Soft vote. In Hard vote, the Voting ensemble output is the class prediction with the highest vote. While in Soft vote, base models prediction probabilities is averaged for each class, and the class with the highest average will be the ensemble output. Please, note that we assume equal weight to all base models predictions, however, Voting ensemble can assign different weights to base models predictions.

We selected a number of base models for our Voting ensemble: Decision Trees (DT), Logistic Regression (LR), Support Vector Machines (SVM), Multi-Layer Perceptron (MLP), and Stochastic Gradient Descent (SGD) models. Our selection criteria was targeting a heterogeneous ensemble built from different classification families. For instance, DT is a tree-based model, and MLP is an Artificial Neural Network model.

In our Voting ensemble, we built the previous base models using the default parameters set in Python scikit-learn library (v 0.24.2). Then, the base models predictions were aggregated using Soft voting mode, where the Voting ensemble output prediction is based on the calculated average probability.

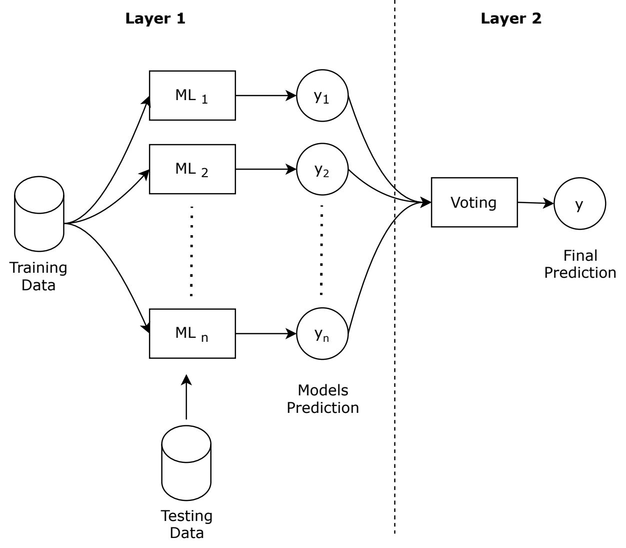

Fig. 1: Voting Ensemble Architecture

For illustration purposes, we will consider an example of the Voting ensemble output as presented in Table I. Table shows each class prediction probability outputted from all base models. e.g. MLP model predicts this test instance as smelly with prob = 0.7, while not smelly with a prob = 0.3. The Voting ensemble averages the prediction probabilities outputted from all base models, then, the Voting ensemble output prediction will be a smelly instance based on the higher class probability average (i.e. Smelly with prob = 0.6).

TABLE I: Voting ensemble sample prediction output

	p(Smelly)	p(NotSmelly)
DT	0.8	0.2
LR	0.3	0.7
SVM	0.6	0.4
MLP	0.7	0.3
SGD	0.6	0.4
Avg Prob	0.6	0.4

IV. Empirical Study

In this section, we will outline the details of our conducted empirical study. All steps were implemented using Python programming language.

A. Goal

The main goal of this empirical study is to investigate the detection performance of Voting ensemble in code smells detection. Goal can be formulated using the GQM [18] as follows: ***evaluate*** Voting ensemble for the ***purpose*** of code smells detection with ***respect*** to their detection performance measured in Accuracy, Brier, and AUC scores form the ***perspective*** of both software engineers and researchers in the ***context*** of 2 class-level and 4 method-level code smells datasets.

B. *Code Smell Datasets*

We used two class-level and four method-level code smells. Class-level smells datasets covers God Class and Data Class code smells, while method-level smells datasets covers: Long Method, Feature Envy, Long Parameter List, and Switch Statements code smells. Datasets were provided by F. Fontana et al. [16], one dataset per code smell type. Each dataset contains a total of 420 instances with a set of Object Oriented (OO) metrics as independent variables and one binary dependent variable to indicate whether the instance is smelly or not. In each dataset, the smelly instances forms 33% from the total 420 instances, while the remaining (i.e. 67%) are not smelly instances.

C. *Data Pre-Processing*

We created a pipeline to pre-process each code smell dataset consisting of three main steps: missing data imputation, feature scaling, and feature selection. In the first step, mean imputation was used to handle missing data in each dataset. Although, missing data represents less than 1% in each dataset, missing values were replaced by the mean of all values in the column (i.e. independent variable). Mean imputation will assist in building more accurate and robust detection models [19]. Next, we performed feature scaling using Min-Max normalization. For each feature, the minimum value is transformed into zero and the maximum value into one, while values in-between is transformed into a number between [0,1]. Feature scaling assist in building less biased and faster machine learning models [20]. The last step in our data pre-processing pipeline is feature selection. We used Gain ratio [21] as a feature selection technique, where each feature will receive a gain score between [0,1]. Any feature scoring less than the mean gain will be considered irrelevant and will be removed from the list of features. Removing irrelevant features will increase the reliability and effectiveness of the machine learning models [21], [22].

D. *Model Validation*

Machine learning models were validated using a stratified 10 folds cross-validation [23]. In 10 folds cross-validation, dataset is partitioned into 10 folds with 9 folds for training and the remaining fold for testing. This procedure is repeated 10 times, where in each iteration, the testing fold is swapped with one of the training folds making each testing fold used exactly once as a testing dataset. Model performance is averaged over all iterations to obtain a final performance. Stratified cross-validation ensures that each fold has the same instances proportion that belong to each class is approximately equal to the original dataset. We repeated the stratified 10 fold cross-validation ten times to produce low variance and more robust model performance estimates [24].

E. *Detection Performance Measures*

Code smell detection in this empirical study is a binary classification problem, where machine learning models will compete to correctly classify smelly and non-smelly code. Detection performance of our employed machine learning models were measured using three different evaluation measures: Accuracy, Brier, and Area Under Curve scores.

- **Accuracy:** Accuracy measure is one of the most commonly used evaluation measures in code smells detection. It is defined as the proportion of the correctly classified results (true negatives and true positives) in the population:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \times 100 \quad (1)$$

  Higher accuracy scores indicates the machine learning model capability to distinguish between smelly and non-smelly code.
- **Brier score:** Brier score measures the accuracy of probabilistic predictions by calculating the mean squared difference between the predicted probability (f_i) and the expected values (O_i). The score can be between zero and one, and models achieving lower Brier scores indicates better detection performance.

$$Brier\ score = \frac{1}{N}\sum_{i=1}^{N}(f_i - O_i)^2 \quad (2)$$

- **Area Under Curve (AUC):** AUC score represents the machine learning model skill at distinguishing between the positive and negative classes. AUC measures the area underneath the entire Receiver Operating Characteristics (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR). AUC can have a score between zero and one, with scores close to one indicating a skilful model. Figure 2 shows an example of a machine learning model ROC curve with AUC score of (0.79). The black diagonal line represents a model with (AUC = 0.5) similar to a randomly guessing model. AUC score of a zero shows a model predicting a negative class as a positive class and vice versa.

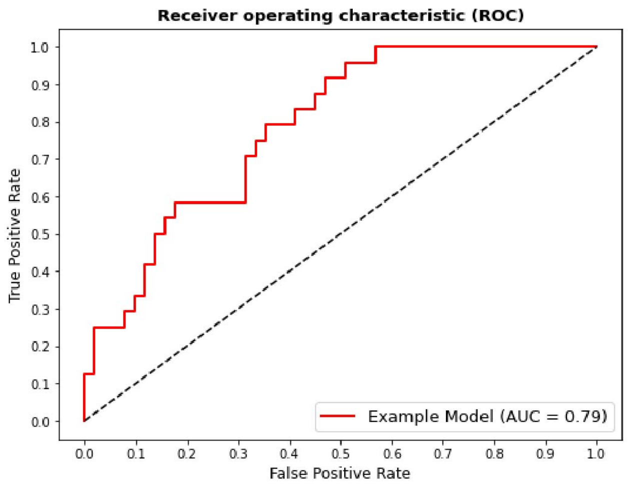

Fig. 2: ROC example

TABLE II: Code smells detection performance results

	God Class			Data Class			Feature Envy		
Classifiers	Accuracy	Brier	AUC	Accuracy	Brier	AUC	Accuracy	Brier	AUC
DT	95.43	0.05	0.94	98.64	0.01	0.99	94.05	0.06	0.93
LR	95.83	0.04	0.99	94.6	0.06	0.97	90.69	0.07	0.98
SVM	96.57	0.02	1	96.74	0.02	0.99	92.71	0.05	0.98
MLP	96.31	0.03	0.99	96.29	0.03	0.99	93.93	0.04	0.98
SGD	96.38	0.03	0.99	95.07	0.04	0.99	93.4	0.05	0.98
Vote	96.88	0.02	0.99	97.45	0.02	1	95.05	0.04	0.99

	Long Method			Long Parameter List			Switch Statements		
Classifiers	Accuracy	Brier	AUC	Accuracy	Brier	AUC	Accuracy	Brier	AUC
DT	99.07	0.01	0.99	90.14	0.1	0.89	85.83	0.14	0.83
LR	97.9	0.03	1	87.64	0.09	0.96	84.57	0.1	0.94
SVM	97.69	0.02	1	85.21	0.09	0.95	85.26	0.09	0.94
MLP	98.17	0.02	1	88.62	0.08	0.96	87.9	0.08	0.95
SGD	97.52	0.02	1	90.52	0.07	0.97	86.55	0.09	0.95
Vote	98.83	0.01	1	91.83	0.07	0.97	87.81	0.08	0.95

F. *Statistical Pairwise Comparison*

Statistical pairwise comparison was performed to examine whether the observed detection performance differences between models is significant or insignificant [25]. Our empirical study had a total of 90 pairwise comparisons (15 pairwise comparisons per code smell dataset x 6 code smell datasets). We used the Accuracy measure estimated over 100 iterations (10 stratified fold CV repeated 10 times) as the detection measure to perform the statistical test at a significance level (α = 0.05). The non-parametric Wilcoxon signed-rank test was performed, since the data was not normally distributed as indicated by the Kolmogorov-Smirnov test.

V. Results and Discussions

Table II present the detection performance results of our investigated machine learning models in terms of accuracy, Brier, and AUC scores. In class-level code smells, Voting ensemble competed for the highest detection performance in comparison to base models. In God Class smell detection, Voting ensemble achieved the highest detection performance, with marginal differences from the competing base models, except for the DT model where the difference was the largest. In Data Class smell detection, Voting ensemble was competing also for the highest detection performance and achieved the second best model after the DT model. It is noted that the DT model was the worst model to detect God Class smell, while being the best model in detecting the Data Class smell. In the context of method-level smells, Voting ensemble continued to compete for the highest detection performance in all types of method level smells. More specifically, it was notable that our base models struggled to achieve high detection performance in detecting Long Parameter List and Switch Statements smells. In the contrary, Voting ensemble achieved the highest detection performance in detecting these two challenging smells.

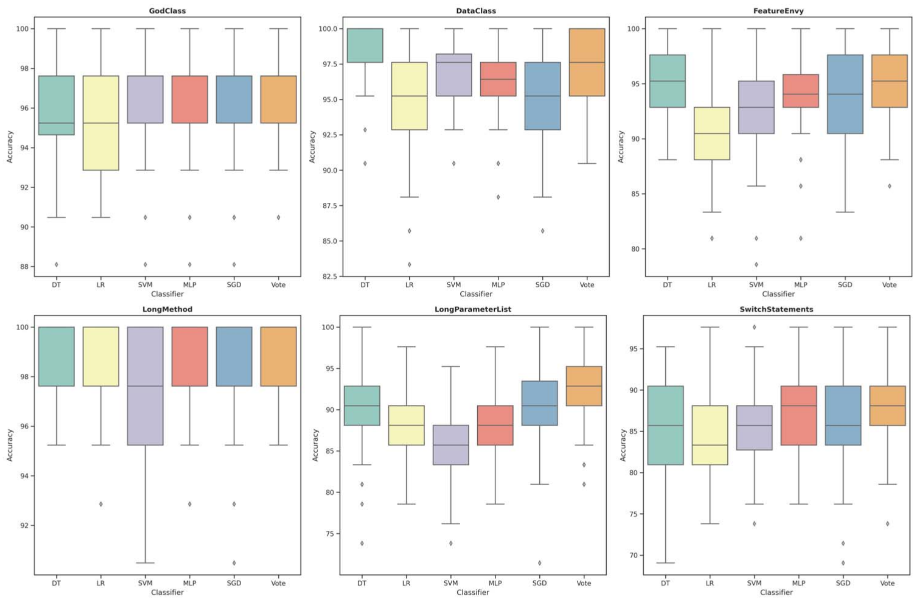

Fig. 3: Machine learning models detection accuracy boxplot over code smells

TABLE III: Statistical pairwise comparison

	DT	LR	MLP	SGD	SVM	Vote	#Wins	%Wins
DT		4	3	2	4	1	14	47%
LR					1		1	3%
MLP	2	5		3	4		14	47%
SGD	1	4	1		3		9	30%
SVM	1	3		1			5	17%
Vote	4	6	5	6	6		27	**90%**
#Losses	8	22	9	12	18	1		
%Losses	27%	73%	30%	40%	60%	**3%**		

To further examine the models accuracy scores distribution, we plotted the accuracy boxplots of the Voting ensemble against base models across all code smells, as shown in Fig 3. Base models showed varying detection accuracy in detecting different smell types. For instance, DT model was the best model in detecting Data Class smell, while being the worst in Switch Statements smell detection by showing the biggest box and longest whiskers. In the contrary, Voting ensemble was consist in showing a high and stable detection accuracy over base models. Voting ensemble had a small box and shorter whiskers in comparison to other base models, indicating less accuracy scores dispersion and more stable accuracy estimates.

Next in our analysis, we used the non-parametric Wilcoxon statistical test to examine whether the observed accuracy differences between machine learning models is statistically significant or not. Table III shows the Wilcoxon statistical output after performing a pairwise comparison between models across all code smells. Possible outcomes for each pairwise comparison can be: a win, a loss, or a tie (i.e. accuracy difference is insignificant), and each model will have a total of 30 pairwise comparisons (5 pairwise comparisons per dataset x 6 code smells). Numbers inside the table represents the total number of wins for a row model against a column model, and vice versa, the same number represents the total number of losses of a column model against the row model. Voting ensemble won 4 times against DT model, and it can be interpreted also as DT model lost 4 times against the Voting ensemble.

Pairwise comparisons results in Table III shows the superiority of the Voting ensemble over constituent base models, with the highest percentage of wins 90% (i.e. 27 wins out of the total 30 pairwise comparisons). DT and MLP models were the most winning base models with 47% wins for each model. LR model was the least winning base model with a single win (3%) against SVM. In addition, the Voting ensemble lost a single time against the DT model, and had the lowest percentage of losses (3%). DT model was the least losing model among individual models (27%), while the LR lost (73%) from the total pairwise comparisons.

We further expanded the previous table to detail the pairwise comparisons results within each code smell. Base models is showing inconsistent detection performance in detecting different code smells. For instance, the best base model (DT), is dominating all other models (including the Voting ensemble) in detecting the Data Class smell, while losing against them in detecting God Class smell. In another example, MLP model was winning over other base models in detecting Switch Statements smells, however, this high detection performance is not inline with the detection performance for other smells. On the contrary to all base models, the Voting ensemble is consistently superior in detecting all code smells, with only a single lost in detecting the Data class smell.

VI. Conclusion

This paper empirically investigated the detection performance of Voting ensemble in code smell detection over constituent base models. Detection performance was examined within the context of two class-level and four method-level code smells types. Code smells datasets were pre-processed in a three steps pipeline: missing data imputation, feature scaling, and feature selection. Then, the investigated machine learning models were built and their detection performance was examined. Furthermore, the non-parametric Wilcoxon statistical test was used to examine whether the observed detection performance difference between models was significant or not. Outcomes of our conducted empirical study can be summarized into two folds: First, the detection performance of base models varied between code smells, showing inconsistent code smell detection performance. Second, the Voting ensemble showed a consistent superior detection performance across all code smells indicating a more stable high detection performance.

TABLE IV: Statistical pairwise comparison within each code smell

	God Class		Data Class		Feature Envy		Long Method		Long Parameter List		Switch Statements	
	Win	Loss	Win	Loss	Win	Loss	Win	Loss	Win	Loss	Win	Loss
DT		4	5		2	1	4		3	1		2
LR		4		4		5		2	1	4		3
MLP	2	1	2	2	2	1	2	2	2	3	4	
SGD	2	1		4	2	1		3	3	1	2	2
SVM	2	1	2	2	1	4		3		5		3
Vote	5		4	1	5		4		5		4	
Total	11		13		12		10		14		10	

Work in this paper can be extended further to examine the capabilities of the Voting ensemble in code smells detection. Possible future directions can be listed as follows: First, Voting ensemble detection performance can be further examined with other code smell types (e.g. Swiss Army Knife). Second, hard voting mechanism can be used to build a new variant of the Voting ensemble, where we predict the class with the largest sum of votes from base models, and then empirically investigate its detection performance against our Voting ensemble (Soft voting). Third, Voting ensembles can be built in a homogeneous manner, where the base models are from the same model type (e.g. a homogeneous Voting ensemble to combine multiple fits of a base model with different hyperparameters).

Acknowledgment

The author would like to acknowledge the support of King Fahd University of Petroleum and Minerals (KFUPM) in the development of this work.

References

[1] A. Yamashita and L. Moonen, "Exploring the impact of inter-smell relations on software maintainability: An empirical study," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 682–691.

[2] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[3] A. Idri, M. Hosni, and A. Abran, "Systematic literature review of ensemble effort estimation," *Journal of Systems and Software*, vol. 118, pp. 151–175, 2016.

[4] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Information and Software Technology*, vol. 108, pp. 115–138, 2019.

[5] A. Al-Shaaby, H. Aljamaan, and M. Alshayeb, "Bad smell detection using machine learning techniques: a systematic literature review," *Arabian Journal for Science and Engineering*, vol. 45, no. 4, pp. 2341–2369, 2020.

[6] F. Khomh, S. Vaucher, Y.-G. Guéhéneuc, and H. Sahraoui, "A bayesian approach for the detection of code and design smells," in *2009 Ninth International Conference on Quality Software*. IEEE, 2009, pp. 305–314.

[7] F. Khomh, S. Vaucher, Y.-G. Gueheneuc, and H. Sahraoui, "Bdtex: A gqm-based bayesian approach for the detection of antipatterns," *Journal of Systems and Software*, vol. 84, no. 4, pp. 559–572, 2011.

[8] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, and H. Mei, "Can i clone this piece of code here?" in *Proceedings of the 27th IEEE/ACM international conference on automated software engineering*, 2012, pp. 170–179.

[9] L. Amorim, E. Costa, N. Antunes, B. Fonseca, and M. Ribeiro, "Experience report: Evaluating the effectiveness of decision trees for detecting code smells," in *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*. IEEE, 2015, pp. 261–269.

[10] F. A. Fontana, M. Zanoni, A. Marino, and M. V. Mäntylä, "Code smell detection: Towards a machine learning-based approach," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 396–399.

[11] A. Maiga, N. Ali, N. Bhattacharya, A. Sabane, Y.-G. Guéhéneuc, and E. Aimeur, "Smurf: A svm-based incremental anti-pattern detection approach," in *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012, pp. 466–475.

[12] A. Maiga, N. Ali, N. Bhattacharya, A. Sabané, Y.-G. Guéhéneuc, G. Antoniol, and E. Aimeur, "Support vector machines for anti-pattern detection," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2012, pp. 278–281.

[13] 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.

[14] D. K. Kim, "Finding bad code smells with neural network models," *International Journal of Electrical and Computer Engineering*, vol. 7, no. 6, p. 3613, 2017.

[15] L. Rokach, "Ensemble-based classifiers," *Artificial intelligence review*, vol. 33, no. 1, pp. 1–39, 2010.

[16] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1143–1191, 2016.

[17] A. Alazba and H. Aljamaan, "Code smell detection using feature selection and stacking ensemble: An empirical investigation," *Information and Software Technology*, p. 106648, 2021.

[18] V. R. Basili and H. D. Rombach, "The tame project: Towards improvement-oriented software environments," *IEEE Transactions on software engineering*, vol. 14, no. 6, pp. 758–773, 1988.

[19] E. Acuna and C. Rodriguez, "The treatment of missing values and its effect on classifier accuracy," in *Classification, clustering, and data mining applications*. Springer, 2004, pp. 639–647.

[20] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, p. 105524, 2020.

[21] A. G. Karegowda, A. Manjunath, and M. Jayaram, "Comparative study of attribute selection using gain ratio and correlation based feature selection," *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, pp. 271–277, 2010.

[22] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 science and information conference*. IEEE, 2014, pp. 372–378.

[23] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Montreal, Canada, 1995, pp. 1137–1145.

[24] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.

[25] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.