Dimensionally Reduction based Machine Learning Approaches for Code smells Detection

Seema Dewangan
Computer Science & Information Technology
Guru Ghasidas Vishwavidyalaya
Bilaspur (CG) India
sskd501@gmail.com

Rajwant Singh Rao
Computer Science & Information Technology
Guru Ghasidas Vishwavidyalaya
Bilaspur (CG) India
rajwantrao@gmail.com

Pravin Singh Yadav
Computer Science & Information Technology
Guru Ghasidas Vishwavidyalaya
Bilaspur (CG) India
pravinsingh1110@gmail.com

2022 International Conference on Intelligent Controller and Computing for Smart Power (ICICCSP) | 978-1-6654-7258-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICICCSP53532.2022.9862030

***Abstract*— Code smells refer to the lack of the software quality, such as difficulty in understandability and changeability. In this research work, we proposed a technique to detect the CSs using three machine learning algorithms. For this purpose, we used two class level and two method level datasets. In this research work, a principle component analysis (PCA) based dimensionality reduction technique is applied to select different components from each datasets. Four PCA based machine learning algorithms are applied, namely Principal component analysis based Logistic regression (PCA_LR), Principal component analysis based Random forest (PCA_RF), Principal component analysis based K-nearest neighbor (PCA_KNN), and Principal component analysis based Decision tree (PCA_DT). The 10-fold cross-validation is used to validate the model accuracy. In this research work, we found that the PCA_LR model gives the best accuracy, 99.97%, for the data class dataset, while the worst accuracy, 77.38%, was found by the PCA_KNN model for the long parameter list dataset.**

Keywords*— *Code smells detection, Dimensionally reduction, Machine learning algorithms.

I. INTRODUCTION

Code smells (CSs) define a source code or software variance that explains the abuse of essential design ethics such as module, modification, hierarchy, abstraction, and encapsulation [1]. CSs increase due to many reasons, e.g. inexperienced developers, target demands, competition etc. M. Fowler et al. [2] introduced 22 informal CSs. In the previous literature, various machine learning (ML) algorithms have been applied to detect CSs. Each algorithm produces different outcomes. There are three primary reasons for getting different outcomes: i) The developers can individually recognize the CSs and discover them in various methods. ii) Understanding among the detectors is low, i.e. various rules or tools identify an uncommon smell for different code components. iii) The threshold rate for recognizing the CSs may differ for the detectors.

While developing software, the software's functional and non-functional quality is essential for the developer to assure the software quality [3]. Most developers focus only on functional quality and ignore non-functional requirements, e.g. evolution process, maintenance work, comprehensibility, reusability etc.[4]. The need for a non-functional quality guide to refuse the quality of the software so that the difficulty and maintenance of software enhances.

We proposed four ML algorithms with an applied dimensionally reduction approach to detect the CSs from CS datasets. The summary of our work is given as follows: the second section explains the related literature review, and the third section presents the flow diagram of the research agenda. The fourth section describes the results and discussion, and the last section five discussed conclusion.

II. LITERATURE REVIEW

This part explains various authors' different ML approaches to detecting the CSs from CS datasets. F. A. Fontana et al.[5] introduced sixteen ML algorithms on the four CS datasets Data class, God class, Feature envy, and Long method. In our experiment work, we are using the Data class and God class. They obtained 99.02% accuracy for the Data class dataset using the B-J48 Pruned algorithm. D. Di Nucci et al. [6] have shown its results in graphical form, and they achieved around 84.00% accuracy used by Random forest and J48 approach for Feature envy dataset. Mhawish et al. [7] considered a CSs recognition framework used by the ML approach and software metrics. Furthermore, a genetically based metrics selection method is applied to enhance the CSs recognition results' efficiency. They utilized a parameter selection method with the grid search that increases the effectiveness of ML methods. In the RF approach to calculate the data class in the ORI_D dataset, they found the most significant result of 99.71%, and in the REFD_D dataset, they achieved the most critical result of 99.70%. S. Dewangan et al. [8] used six ML methods, grid search based parameter selection method, and Chi-square and Wrapper based metrics selection method to choose the most relevant metrics from all datasets and acquired 100% results by the LR approach for the long method dataset. Alazba et al. [9] utilized the gain metrics selection method and achieved the maximum result, 99.24%, using the Stack-SVM model. Yadav et al. [10] proposed a decision tree model with hyper parameter tuning to detect the CSs; they reached 97.62% in blob class and data class datasets. Dewangan et al. [11] proposed five ML classifiers to find the CSs from four CS datasets. Moreover, a metrics selection method extracts the most relevant metrics from all datasets. They found a 99.12% result using the RF model for the feature envy dataset. Reis et al.[12] used a Crowd-smelling method with applied collective knowledge in CSs detection for Long method, God class, and Feature envy datasets. They applied six ML models to find the CSs. They acquired 0.896% ROC in the Naive Bayes approach for God Class and 0.870% ROC in AdaBoostM1approach for the Long method dataset. The worst result acquired 0.570% in the RF approach for the Feature envy dataset. Khalid et al.[13] intended a comparative investigation for God Class of ML approaches on recognizing CSs with balanced and imbalanced datasets. They used twenty-eight ML approaches for identifying God Class. In this experiment, they used a dataset created from

978-1-6654-7258-6/22/$31.00 ©2022 IEEE

12,587 classes of 24 software systems in which 1958 classes were self certify. They used SMOTE system for data balancing. They found that most of the classifiers achieved the best performances. Using a ML classifier, Ashraf Abdou et al. [14] investigated the severity of CSs. They clarified the ML algorithms using a Local Interpretable Model Agnostic Explanations algorithm. Using Spearman's correlation metric, they were able to attain % to 97 %. Sofien Boutaib et al. [15] introduced a Bi-level Multi-Label Detection of Smells (BMLDS) method for identifying multi-label smells by reducing the population of classification series. They used a two-tiered approach, with the higher-level component determining the optimum categorization for each measured series and the lower-level part constructing the series.

III. FLOW DIAGRAM OF RESEARCH AGENDA

This section proposed an agenda for finding the CSs from CS datasets. In Fig 1, we have explained the proposed agenda step by step flow diagram.

Step1: In the first step, we are taken four CS datasets that is Data class (DC), God class (GC), Switch statements (SS), and a Long parameter list (LPL) from Fontana et al. [5]. These datasets are computed by 111 systems of various sizes and an extensive set of software metrics. Out of which 74 systems are used, 37 systems are not effective for detecting the CSs. They considered 61metrics for the class-level dataset (DC and GC) and 55 metrics for the method-level dataset (SS and LPL). Datasets are presented at http://essere.disco.unimib.it/reverse/MLCSD.html. The explanation of the dataset is given in table 1.

TABLE I. EXPLANATION OF DATASETS [5]

Datasets	Sample	Preferred Metrics
DC	420	61
GC	420	61
SS	420	55
LPL	420	55

Step2: In the second step, we used a min-max feature scaling technique to convert all dataset features from zero to one.

Step 3: In the third step, we used PCA based metrics selection method to extract the most influential metrics (components) from each dataset.

Step 4: In the fourth step, we consider four ML models, that is LR, RF, KNN, and DT models, with 10-fold cross-validation. Then found the model accuracy for each dataset.

IV. RESULTS AND DISCUSSION

In this research work, we took four CS datasets: Data class, God class, Switch statement, and Long parameter list, to find the code smells from a given dataset. We proposed four ML models with applied principle component analysis (PCA). The four proposed models are PCA_LR, PCA_RF, PCA_KNN, and PCA_DT. In each dataset, we selected the set of components (C-8, C-9, C-10, C-11, C-12, and All components) using PCA and then applied ML algorithms. The accuracy obtained for the Data class, God class, Switch statement, and Long parameter list dataset are given in table 2, table 3, table 4, and table 5, respectively.

The performance accuracy of the Data class dataset is 99.97% on the PCA_LR model with taking all components. At the same time, the weak performance was 88.09% on the PCA_KNN model with taking all components.

The performance accuracy of the God class dataset is 97.62% on the PCA_LR model with taking C-8, C-9, C-10, C-12, and PCA_RF model with taking all components. While the weak performance was 94.00% on the PCA_RF model with eight components.

The performance accuracy of the Switch statement dataset is 85.72% on the PCA_KNN model with taking nine components. While the weak performance was 73.80% on the PCA_RF model with ten components.

The performance accuracy of the Long parameter list dataset is 94.05% on the PCA_LR and PCA_RF model with taking all components. While the weak performance was 77.38% on the PCA_KNN model with eleven components.

TABLE II. RESULTS OF DATA CLASS DATASET

Models	C-8	C-9	C-10	C-11	C-12	All components
PCA_LR	98.81	97.62	96.43	98.80	99.96	**99.97**
PCA_RF	92.85	95.24	91.67	95.24	92.86	97.62
PCA_KNN	92.86	94.05	92.86	92.86	94.05	88.09
PCA_DT	98.96	98.96	98.56	98.96	99.24	99.38

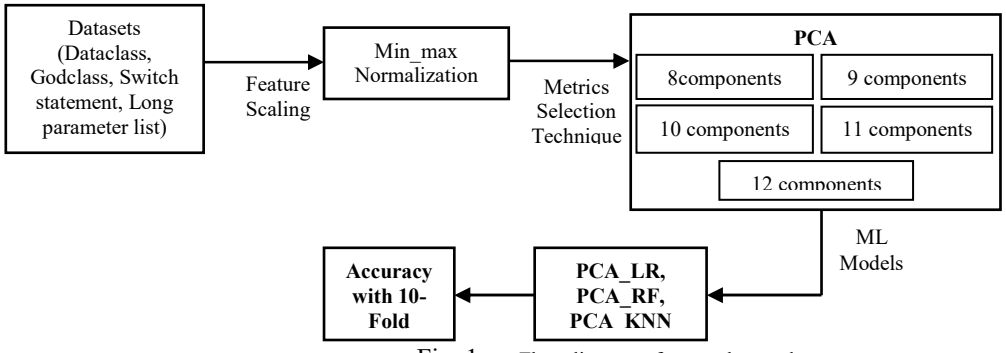

Fig. 1. Flow diagram of research agenda

TABLE III RESULTS OF GOD CLASS DATASET

Models	C-8	C-9	C-10	C-11	C-12	All components
PCA_LR	**97.62**	**97.62**	**97.62**	96.43	**97.62**	95.24
PCA_RF	94.00	95.24	95.24	94.05	96.43	**97.62**
PCA_KNN	95.24	95.24	94.05	94.05	94.05	94.05
PCA_DT	96.84	96.84	97.22	97.36	97.22	97.54

TABLE IV RESULTS OF SWITCH STATEMENT DATASET

Models	C-8	C-9	C-10	C-11	C-12	All components
PCA_LR	84.52	84.52	83.34	84.52	84.52	84.52
PCA_RF	76.20	75.00	73.80	77.38	84.54	83.34
PCA_KNN	83.34	**85.72**	84.54	83.34	78.57	80.95
PCA_DT	82.84	83.51	83.58	84.64	84.62	84.98

TABLE V RESULTS OF LONG PARAMETER LIST DATASET

Models	C-8	C-9	C-10	C-11	C-12	All components
PCA_LR	85.72	83.34	85.72	88.09	85.72	**94.05**
PCA_RF	90.48	86.90	89.29	88.09	84.52	**94.05**
PCA_KNN	85.72	83.34	82.14	77.38	80.95	84.34
PCA_DT	90.98	90.35	92.18	91.78	93.26	93.58

A. Comparative study of our results with the result of other authors:

In this sub-section, we compare our proposed model's result with the results of other literature. Table 6 shows the comparative study of our experiment effects with the development of other authors. In this observation, it is found that various authors applied different models in the earlier literature for the Data class dataset, and they got different results. Out of which, the best result was found by Dewangan et al.[8], but in this experiment, our approach PCA_LR has obtained the best accuracy of 99.97%, which is better than the earlier literature results. Our best result for the God class dataset is 97.62% using PCA_LR and PCA_RF, but Mahawish et al. [7] have improved performance at 98.48% using the GBT approach. The dataset Alazba et al. [9] had achieved 88.89% accuracy using the GP approach for the switch statement. Still, our approach has got 85.72% highest accuracy using the PCA_KNN method, which is not good compared to Alazba et al.[9] result. For the Long parameter list, dataset Alazba et al. [9] had achieved 92.50% accuracy using the GP approach. Still, our approach has 94.05% accuracy using PCA_LR and PCA_RF methods, which is better than Alazba et al.[9] result.

In this discussion, we observed that for the Data class and Long parameter list dataset, our approach got better results than previous literature. Still, for the God class and Switch statement dataset, our strategy has not yielded a good performance compared to previous literature.

TABLE VI. COMPARATIVE TABLE OF OUR RESULTS WITH THE RESULT OF OTHER AUTHORS

Publishing Date	Author's Name	Datasets							
		Data class		*God class*		*Switch Statement*		*Long parameter list*	
		Best Approach	*A (%)*	*Best Approach*	*A (%)*	*Best Approach*	*A (%)*	*Best Approach*	*A (%)*
2016	Fontana et al. [5]	B-J48 Pruned	99.02	Naive Bayes	97.55	-	-	-	-
2018	D. Nucci et al. [6]	RF and J48	~83.00	RF and J48	~83.00	-	-	-	-
2020	Mahawish et al. [7]	RF	99.70	GBT	98.48	-	-	-	-
2021	Dewangan et al. [8]	RF	99.74	RF	98.21	-	-	-	-
2021	Alazba et al. [9]	Stack-LR	98.92	Stack-SVM	97.00	GP	88.89	GP	92.50
2022	Our Approach	PCA_LR	**99.97**	PCA_LR, and PCA_RF	97.62	PCA_KNN	85.72	PCA_LR, and PCA_RF	**94.05**

V. CONCLUSION

This work examines the effectiveness of four different models named PCA_LR, PCA_RF, PCA_KNN, and PCA_DT to identify the CSs and improve performance accuracy. Our key contributions are given as follows: We used the pre-processing technique (min-max normalization) in the first step. We used the PCA analysis to select the best components from each dataset in the second step. In the third step, we used four ML models named LR, RF, KNN, and DT. Then in last, we found the performance accuracy of CSs. In this experiment, we investigate that 99.97% best accuracy obtained by the PCA_LR model for the Data class dataset, 97.62% best accuracy obtained by PCA_LR and PCA_RF model for the God class dataset, 85.72% best accuracy obtained by PCA_KNN model for Switch statement dataset, and 94.05% best accuracy obtained by PCA_LR and PCA_RF model for Long parameter list dataset.

REFERENCES

[1] G. Booch, "Object-oriented analysis and design", Addison-Wesley, 1980.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, "Refactoring: Improving the design of existing programs" ,1999.

[3] K. Wiegers and J. Beatty, Software Requirements. London, U.K.: Pearson Education, 2013.

[4] L. Chung and P. L. J. C. S. do, "On non-functional requirements in software engineering", in Conceptual Modeling: Foundations and Applications (Lecture Notes in Computer Science), A. T. Borgida, V. Chaudhri, P. Giorgini, and E. S. YuE, Eds. Cham, Switzerland: Springer, 2009, pp. 363_379.

[5] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection", Empirical Softw. Eng., vol. 21, no. 3, pp. 1143_1191, Jun. 2016.

[6] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: Are we there yet?" in Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER), Mar. 2018, pp. 612_621, doi: 10.1109/SANER.2018.8330266.

[7] Mhawish MY, Gupta M. Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(6): 1428–1445 Nov. 2020. DOI 10.1007/s11390-020-0323-7.

[8] S. Dewangan, R. S. Rao, A. Mishra and M. Gupta, "A Novel Approach for Code Smell Detection: An Empirical Study," in IEEE Access, vol. 9, pp. 162869-162883, 2021, doi: 10.1109/ACCESS.2021.3133810.

[9] Alazba, A., & Aljamaan, H.I., "Code smell detection using feature selection and stacking ensemble: An empirical investigation", Inf. Softw. Technol., 2021, 138.

[10] P. S. Yadav, S. Dewangan and R. S. Rao, "Extraction of Prediction Rules of Code Smell using Decision Tree Algorithm," 2021 10th International Conference on Internet of Everything, Microwave Engineering, Communication and Networks (IEMECON), 2021, pp. 1-5, doi: 10.1109/IEMECON53809.2021.9689174.

[11] Dewangan, S., Rao, R.S., "Code Smell Detection Using Classification Approaches", In: Udgata, S.K., Sethi, S., Gao, XZ. (eds) Intelligent Systems. Lecture Notes in Networks and Systems, vol 431. Springer, Singapore(2022). https://doi.org/10.1007/978-981-19-0901-6_25.

[12] Reis, J.P.d., Abreu, F.B.e. & Carneiro, G.d.F, "Crowdsmelling: A preliminary study on using collective knowledge in code smells detection", Empir Software Eng **27,** 69. https://doi.org/10.1007/s10664-021-10110-5.

[13] Khalid Alkharabsheh, Sadi Alawadi, Victor R. Kebande, Yania Crespo, Manuel Fernández-Delgado, José A. Taboada, "A comparison of machine learning algorithms on design smell detection using balanced and imbalanced dataset: A study of God class", Information and Software Technology, Volume 143, 2022, 106736, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2021.106736.

[14] Ashraf Abdou, Nagy Darwish, "Severity classification of software code smells using machine learning techniques: A comparative study", Journal of Software: Evolution and Process. (2022) 34. 37. 10.1002/smr.2454.

[15] S Boutaib, M Elarbi, S Bechikh, F Palomba, LB Said (2022). A Bi-level Evolutionary Approach for the Multi-label Detection of Smelly Classes. GECCO 22 Companion, July 9–13, 2022, Boston, MA, USA. ACM ISBN 978-1-4503-9268-6/22/07. https://doi.org/10.1145/3520304.3528946.