

Empirical Analysis on Effectiveness of NLP Methods for Predicting Code Smell

Himanshu Gupta(✉), Abhiram Anand Gulanikar(✉), Lov Kumar(✉), and Lalita Bhanu Murthy Neti(✉)

BITS Pilani, Hyderabad Campus, Hyderabad, India
{f20150339h,f20150105h}@alumni.bits-pilani.ac.in,
{lovkumar,bhanu}@hyderabad.bits-pilani.ac.in

Abstract. A code smell is a surface indicator of an inherent problem in the system, most often due to deviation from standard coding practices on the developer's part during the development phase. Studies observe that code smells made the code more susceptible to call for modifications and corrections than code that did not contain code smells. Restructuring the code at the early stage of development saves the exponentially increasing amount of effort it would require to address the issues stemming from the presence of these code smells. Instead of using traditional features to detect code smells, we use user comments (given on the packages' repositories) to manually construct features to predict code smells. We use three Extreme learning machine kernels over 629 packages to identify eight code smells by leveraging feature engineering aspects and using sampling techniques. Our findings indicate that the radial basis functional kernel performs best out of the three kernel methods with a mean accuracy of 98.52.

Keywords: Extreme learning machine · Natural language processing · Radial basis kernel

1 Introduction

The existence of code smells in source code points towards poor design and violations in standard coding practices [1]. The code smells may not necessarily be identified as defects in the software in the current phase, but these code classes have a high likelihood of developing bugs in the future. Since these code smells do not cause defects, the only way to identify them is based on inspection, i.e., manually combing through thousands of lines of code to find code smells. This method is highly disorganized and costly and becomes more inefficient along with scaling of code package size. In our work, we are automating this process of identifying code smells. We are using the input source code packages to build our set of source

H. Gupta and A. A. Gulanikar—The research associated to this paper was completed during author's undergraduate study at BITS Pilani, Hyderabad Campus.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12957, pp. 43–53, 2021.
https://doi.org/10.1007/978-3-030-87013-3_4

code metrics to develop a model to locate and predict code smells in the application packages. These models will reduce the cost and efficiency of maintaining software while enforcing standard coding practices and improving its quality.

In this paper, we used three kernels: Linear kernel, radial basis function kernel, and polynomial kernel to develop models to predict the following eight code smells, namely, Swiss Army Knife (SAK), Long Method (LM), Member Ignoring Method (MIM), No Low Memory Resolver (NLMR), Blob Class (BLOB), Internal Getter/Setter (IGS), Leaking Inner Class (LIC) and Complex Class (CC). These source code metrics are from the application's source code packages and are used to engineer relevant features and select relevant metrics. We used the Wilcoxon Rank Sum Test to achieve the second of these objectives. In this work, we have analyzed the performance with various kernel functions using accuracy, area under the curve (AUC), and F-measure to predict code smells. We have attempted to answer three research questions in this paper:

- **RQ1: Discuss the ability of different NLP Methods to generate features that help detect code smells.** In traditional code smell detection techniques, code smell metrics are present, which help detect code smells. We have manually constructed 100 features derived from reviews of peer developers' software about the software's source code in this problem. We have a Continuous Bag of words and the Skip-gram method to construct features for detection. We will use accuracy, Area under the curve, and F1 Score to compare each technique's performance.
- **RQ2: Explore the potential of Data Sampling Techniques to discover code smells** Instead of using just the original data, we have used three sampling techniques to generate datasets. SMOTE [2] (Synthetic Minority Over-sampling), borderline SMOTE [3], and SVM SMOTE (Support Vector Machine SMOTE) [4], along with original data, gives us four sets of data. We compare the performance of these datasets using Area under curve and statistical significance tests.
- **RQ3: Study the capacity of various ELM Kernels to predict code smells.** We have used three Extreme Learning Machine kernels to detect code smells from the various features and datasets. Linear Kernel (LINK), Radial Basis Function kernel (RBF), and Polynomial kernel have been used for classification. Their performance has been compared using statistical significance tests and Area Under the Curve Analysis.

Organization: The paper is prepared as follows: The 2nd section summarizes the associated work. The 3rd section offers an in-depth review of all the components used in the experiment. The 4th section describes the study's framework pipeline and how the components described in Sect. 3 interact with each other. The 5th section provides the experimental outcome and the 6th section answers the questions raised in the introduction. In the 7th section we conclude our research.

2 Related Work

Evgeniy et al. used contextual analysis of document data to generate features making use of word sense clarification. Long Ma et al. used Word2Vec to output word

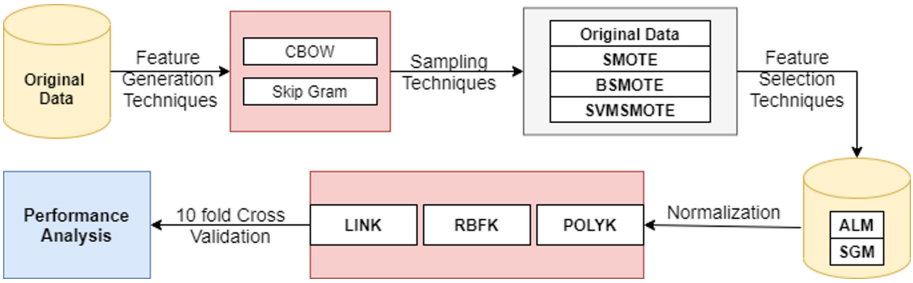

Fig. 1. Flowchart of the research framework

vectors to represent large pieces of texts or entire documents. He used CBOW and skip-grams as component models of Word2Vec to create word vectors and then evaluate word similarity [5]. Hui Han et al. introduced over-sampling techniques of Borderline-SMOTE as a variant of SMOTE where only minority examples near borderline are over-sampled [3]. Josey Mathew et al. proposed a kernel-based SMOTE (SVM SMOTE) algorithm which directly generates the minority data points. His proposed SMOTE technique performs better than other SMOTE techniques in 51 benchmark datasets [4]. Guang-Bin Huang et al. proposed Extreme Learning Machine (ELM), which randomly chooses hidden nodes and determines the Single-hidden Layer Feed forward Neural Networks (SLFN) weights [6]. Francisco Fernández-Navarro et al. proposed a modified version of ELM, which uses Gaussian distribution to parameterize the distribution called the radial basis function. She used ELM is used to optimize the parameters of the model [7].

3 Research Methodology

A detailed description of the Dataset, Data Sampling Techniques, Feature Generation Techniques, Feature Selection Techniques, and Classification Algorithms is given below.

3.1 Experimental Data Set

In this research, our main database comprised of 629 freely available software packages. Our dataset consisted of a list of packages, and the code smells present

Table 1. Statistics on code smell distribution by type

Code smell names	Repository without any code smell	Repository with code smell	Percent of classes without code smell	Percent of classes with code smell
Complex Class (CC)	230	399	63.4%	36.5%
Leaking Internal Class (LIC)	160	469	74.5%	25.4%
Blob Class	460	169	26.8%	73.1%
No Low Memory Resolver (NLMR)	190	439	69.7%	30.2%
Internal Getter Setter (IGS)	264	365	58.02%	41.9%
Member Ignoring Method (MIM)	265	364	57.8%	42.1%
Swiss Army Knife (SAK)	155	474	75.3%	24.6%
Long method (LM)	225	404	64.2%	35.7%

in them. The characteristics and patterns exhibited by all of the code smells are presented in Table 1. Table 1 shows that the code smells are present in a range from 25.4% to 73.1%. We also observed that the lowest presence of any code smell we found was 25.4% for the BLOB Class code smell, while the highest presence observed at 75.04% was for Swiss Army Knife (SAK) code smell.

3.2 Data Sampling Techniques

We use three data sampling techniques to generate additional datasets to mitigate the bias in the dataset:

- **SMOTE** [2] randomly chooses samples from K nearest neighbors from minority class. The synthetic data would be made between the randomly selected sample and the k nearest neighbor.
- **Borderline SMOTE** [3] works on a similar principle but creates data only along the classes' boundary line, not to introduce external bias.
- **SVM-SMOTE** [4,8] uses Support Vector Machine instead of K nearest neighbor to generate samples between a chosen sample and the decision boundary.

3.3 Feature Generation Techniques

We use two architectures from the word2vec [5] techniques, namely the Continuous Bag of Words (CBOW) and Skip-gram. Continuous Bag of Words method [9] uses the surrounding words to speculate the present word. Since it derives from the bag of words model, words present in the window (surrounding the current word) are not differentiated based on the current word's distance. The skip-gram model [10] makes use of the current word to predict the context. Here words nearer are more heavily weighted than words farther away from the present word. Comparing the two models, the CBOW model is faster than skip-grams, but skip-grams perform better when uncommon words are involved.

3.4 Feature Selection Techniques

We have generated 100 feature metrics, but they might not be relevant to the code smells we have considered. We use the Wilcoxon signed-rank test to get the statistical relation between the smelly and clean applications. We have set 0.05 as the outset for the p-value, and we reject the hypothesis if the value is lower. We employ cross-correlation analysis to select uncorrelated features. Our selected variables share a high correlation to the output variables and have a low correlation between themselves.

3.5 Classification Algorithms

This paper uses three ELM kernel functions [6,11] to train models to predict code smells, namely the Linear Kernel function, Radial basis kernel function, and

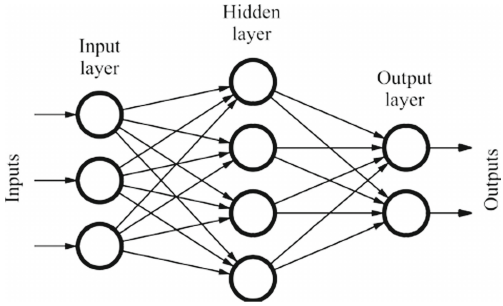

Fig. 2. Architecture diagram of extreme machine learning kernel

polynomial kernel function [12]. As shown in Fig. 2, Extreme Learning Machines (ELM) can be simply defined as feed-forward neural networks, and they can be used for clustering, classification, regressing, among other things. These three kernel methods work best for different data types based on whether it is linearly separable and the problem is linear or nonlinear. Kernel functions are mathematical functions used to transform training data into higher dimensions. The linear kernel is generally chosen when dealing with linearly separable data; it is also most commonly used when many features are in a dataset. The Radial basis function kernel is a non-linear kernel used for training SVMS when solving nonlinear problems. The polynomial kernel function is also used to train nonlinear models. It is faster and requires fewer resources to train the linear or polynomial kernel functions than radial basis functions. Still, they are less accurate in comparison to the RBF kernel. We also use ten-fold cross-validation to overcome overfitting and selection-bias issues and obtain insights on our model's performance on an independent dataset. We use the area under the curve (AUC) and F-measure, among other tests, to compare their performance.

4 Research Framework

We make use of the code data from 629 open-source software packages on GitHub. To eliminate the class imbalance problem in the data, we use SMOTE, Borderline SMOTE, and SVM SMOTE [2–4]to get four datasets: the Original Dataset (ORD) SMOTE Dataset, Borderline SMOTE Dataset, and SVM SMOTE Dataset. We use three kernel functions, the linear kernel function, the radial basis kernel function, and the polynomial kernel function. To compare the accuracy over all the four datasets, we have created and used the area under the curve (AUC) and F-measure, among other tests, to compare their performance. Figure 1 provides a clear representation of the same.

5 Experimental Results

Tables 2A and 2B give accuracy and AUC values for all the ELM methods, using feature engineering methods and all feature selection techniques. Table 3

Table 2. Area Under Curve and Accuracy figures for ELM models trained on the original dataset

(a) Accuracy values

Original Data

	ALM						SGM					
	CBOW			SKG			CBOW			SKG		
	LINK	RBFK	POLYK	LINK	RBFK	POLYK	LINK	RBFK	POLYK	LINK	RBFK	POLYK
BLOB	62.80	100.00	100.00	73.77	100.00	100.00	62.64	100.00	72.02	63.91	68.04	76.63
LM	75.36	100.00	100.00	75.68	99.84	100.00	75.36	100.00	100.00	75.36	100.00	100.00
SAK	75.20	100.00	100.00	74.72	100.00	100.00	75.83	75.83	100.00	73.77	100.00	94.28
CC	71.70	100.00	100.00	71.54	100.00	100.00	70.27	100.00	100.00	70.75	100.00	93.80
IGS	67.73	99.68	100.00	68.04	63.43	100.00	61.84	100.00	65.66	57.71	74.56	100.00
MIM	58.98	100.00	100.00	59.46	82.35	100.00	59.62	100.00	94.59	59.62	88.71	100.00
NLMR	81.08	100.00	100.00	81.56	100.00	100.00	75.04	100.00	99.84	75.04	94.75	93.16
LIC	70.11	100.00	100.00	65.18	85.06	100.00	64.86	100.00	97.46	66.45	78.54	100.00

(b) AUC Values

Original Data

	ALM						SGM					
	CBOW			SKG			CBOW			SKG		
	LINK	RBFK	POLYK	LINK	RBFK	POLYK	LINK	RBFK	POLYK	LINK	RBFK	POLYK
BLOB	0.64	1.00	1.00	0.78	1.00	1.00	0.60	1.00	0.78	0.62	0.86	0.84
LM	0.72	1.00	1.00	0.75	1.00	1.00	0.72	1.00	1.00	0.67	1.00	1.00
SAK	0.71	1.00	1.00	0.68	1.00	1.00	0.69	0.72	1.00	0.65	1.00	0.99
CC	0.75	1.00	1.00	0.70	1.00	1.00	0.65	1.00	1.00	0.69	1.00	0.98
IGS	0.74	1.00	1.00	0.74	0.74	1.00	0.65	1.00	0.73	0.61	0.84	1.00
MIM	0.63	1.00	1.00	0.61	0.91	1.00	0.62	1.00	0.99	0.61	0.96	1.00
NLMR	0.84	1.00	1.00	0.85	1.00	1.00	0.71	1.00	1.00	0.67	0.99	0.98
LIC	0.76	1.00	1.00	0.67	0.98	1.00	0.67	1.00	1.00	0.68	0.94	1.00

and Table 4 summarize the various statistical measures of different metrics used in our research. It is pretty evident from Tables 2A and 2B that Radial Basis Function and polynomial perform much better for most samples than Linear kernel. Table 3B Fig. 3 shows that models trained using all metrics perform better than those using significant metrics. The high values of the performance indicators encourage the use of the code smell prediction model. The following observations can be made from the Results obtained:

- The performance of all models varies greatly with the minimum accuracy being 56.41% and the maximum accuracy being 100% AUC follows similar trend to accuracy but f-measure varies the most from minimum value being 0.02 and maximum being 1.
- It is observed that radial basis kernel and polynomial kernel perform much better than linear kernel across all three statistical measures that is AUC, accuracy and f-measure and they also indicate the high efficiency of the models which are developed.
- It is observed that Linear kernel performs the best with Class NLMR (77.5%) and the worst with MIM class (61.95%).

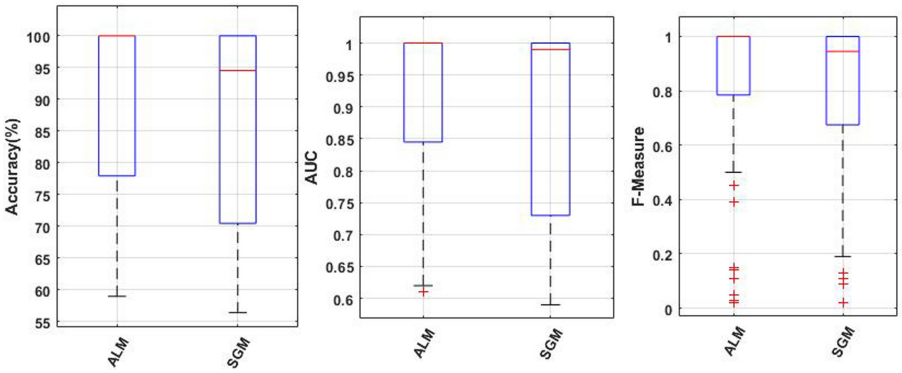

Fig. 3. Box plot comparison between All Metrics

Table 3. Statistical measure

(a) Feature Generation Techniques

	Min	Max	Mean	Median	25th	75th
Accuracy						
CBOW	57.98	100.00	88.63	100.00	74.61	100.00
SKM	56.41	100.00	87.82	99.85	73.98	100.00
AUC						
CBOW	0.59	1.00	0.90	1.00	0.78	1.00
SKM	0.59	1.00	0.90	1.00	0.80	1.00
F Measure						
CBOW	0.02	1.00	0.86	1.00	0.74	1.00
SKM	0.05	1.00	0.84	1.00	0.72	1.00

(b) Features Selection Metrics

	Min	Max	Mean	Median	25th	75th
Accuracy						
ALM	58.98	100.00	90.72	100.00	77.94	100.00
SGM	56.41	100.00	85.73	94.55	70.46	100.00
AUC						
ALM	0.61	1.00	0.93	1.00	0.84	1.00
SGM	0.59	1.00	0.88	0.99	0.73	1.00
F Measure						
ALM	0.02	1.00	0.88	1.00	0.78	1.00
SGM	0.02	1.00	0.83	0.95	0.68	1.00

6 Comparison

RQ1: Discuss the ability of different NLP Methods to generate features that help detect code smells. Table 3A and Fig. 4 shows that CBOW performs slightly better than skip-grams across accuracy and F-measure metric. It is a known fact that CBOW is many times faster to train compared to skip-grams. CBOW performs marginally better when common words are considered, while skip-gram performs better on rare words or phrases [8]. Our model performs better on CBOW, indicating that user comments from which the feature vectors are generated have a higher occurrence of common words over rare words. Table 6B shows us the result of the Ranksum test of vectors generated using these two methods, and we can conclude that the vectors generated are highly uncorrelated.

RQ2: Explore the potential of Data Sampling Techniques to discover code smells. Table 4B and Fig. 5 shows that the data sampling techniques perform better than the original data in AUC, accuracy, and F-measure. Although all three SMOTE techniques, BorderlineSMOTE and SVM-SMOTE, perform nearly the same, SVM-SMOTE performs the best. SVM-SMOTE performs better than others because they use KNN. SVM can employ kernels to lead to a better hyperplane in higher dimensions. KNN uses euclidean distance, on the other hand, which may not work well in the same case. Also, KNN computes the nearest neighbors' distance, leading to more unsatisfactory performance when working on a large dataset.

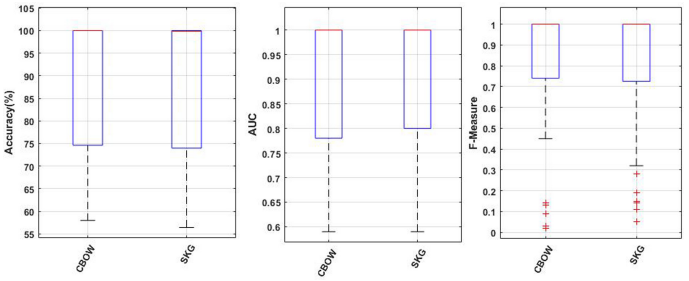

Fig. 4. Accuracy, AUC and F-measure box-plot of different feature generation techniques

Table 4. Statistics about different datasets and machine learning models

(a) Performance on various ELM Kernels

Accuracy						
	Min	Max	Mean	Median	25th	75th
LINK	56.41	82.59	69.84	70.57	64.39	75.11
RBFK	63.43	100.00	98.52	100.00	100.00	100.00
POLYK	64.92	100.00	96.32	100.00	95.57	100.00
AUC						
	Min	Max	Mean	Median	25th	75th
LINK	0.59	0.90	0.74	0.74	0.66	0.80
RBFK	0.72	1.00	0.99	1.00	1.00	1.00
POLYK	0.71	1.00	0.98	1.00	0.99	1.00
F Measure						
	Min	Max	Mean	Median	25th	75th
LINK	0.02	0.86	0.62	0.69	0.59	0.75
RBFK	0.32	1.00	0.98	1.00	1.00	1.00
POLYK	0.45	1.00	0.96	1.00	0.96	1.00

(b) Performance on different Datasets

	Min	Max	Mean	Median	25th	75th
Accuracy						
ORD	57.71	100.00	86.66	96.10	73.77	100.00
SMOTE	57.02	100.00	88.78	100.00	74.46	100.00
BSMOTE	56.41	100.00	88.50	100.00	73.14	100.00
SVMSMOTE	58.69	100.00	88.96	99.95	74.61	100.00
AUC						
ORD	0.60	1.00	0.88	0.99	0.72	1.00
SMOTE	0.61	1.00	0.91	1.00	0.81	1.00
BSMOTE	0.59	1.00	0.91	1.00	0.80	1.00
SVMSMOTE	0.61	1.00	0.92	1.00	0.82	1.00
F Measure						
ORD	0.02	1.00	0.74	0.96	0.50	1.00
SMOTE	0.58	1.00	0.89	1.00	0.75	1.00
BSMOTE	0.57	1.00	0.89	1.00	0.74	1.00
SVMSMOTE	0.59	1.00	0.89	1.00	0.75	1.00

Table 5A gives us the result of the Ranksum test of the datasets generated using these methods. We observe that all the datasets generated from smoothing techniques vary a lot from the original dataset, and we can conclude that the datasets are highly uncorrelated. We also observe that SMOTE,

Borderline-SMOTE, and SVM-SMOTE are very similar to each other, and hence the performance of the models trained over them also show similar trends.

RQ3: Study the capacity of various ELM Kernels to predict code smells. Table 4A and Fig. 6 shows the three kernel methods' performance in terms of accuracy, AUC, and F-measure. Since our data does not have a linear distribution, we observe that the linear kernel method's performance is relatively lackluster. Polynomial and RBF both perform significantly better than the linear kernel due to a fixed small number of features. It is observed that the RBF kernel shows the best performance of the three. Table 5B shows the result of the Ranksum tests on models generated using the different ELM kernels. We can observe that the prediction models developed using the various methods are significantly different from each other, and the models are highly unrelated.

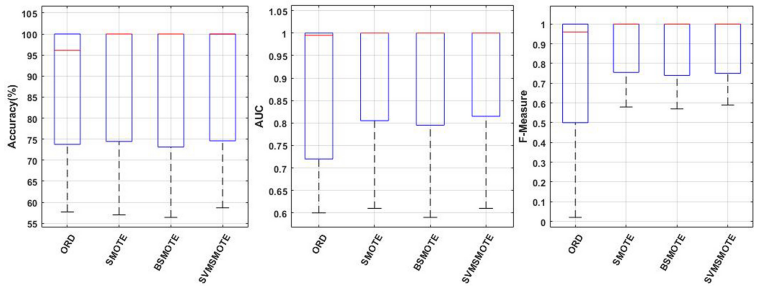

Fig. 5. Comparison between different data sampling techniques.

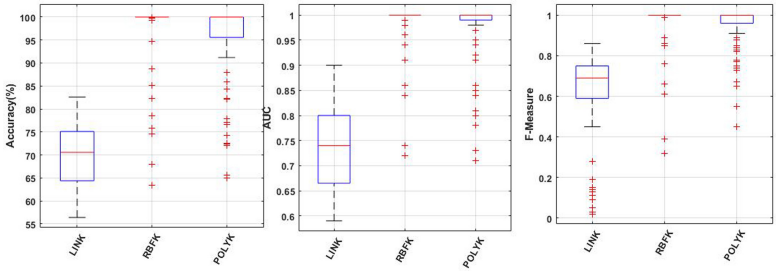

Fig. 6. Box-plot comparison between different ELM Kernel methods

Table 5. Ranksum Test

(a) Different sampling methods

	ORD	SMOTE	BSMOTE	SVMSMOTE
ORD	1.00	0.12	0.19	0.12
SMOTE	0.12	1.00	0.76	0.89
BSMOTE	0.19	0.76	1.00	0.83
SVMSMOTE	0.12	0.89	0.83	1.00

(b) Model similarity

	LINK	RBFK	POLYK
LINK	1.00	0.00	0.00
RBFK	0.00	1.00	0.00
POLYK	0.00	0.00	1.00

Table 6. Ranksum Test

(a) Feature Combination		
	ALM	SGM
ALM	1.00	0.00
SGM	0.00	1.00

(b) Feature Generation Methods		
	CBOW	SKM
CBOW	1.00	0.43
SKM	0.43	1.00

7 Conclusion

This paper provides the empirical evaluation of code smell prediction utilizing various ELM methods, feature generation methods using NLP techniques, feature selection, and data sampling techniques. The models are evaluated using ten-fold cross-validation, and their prediction abilities are compared using accuracy, AUC, and F-measure. We draw the following conclusions from our research study:

- CBOW performs better than skip-grams in feature generation.
- SVM-SMOTE performs best among the data sampling techniques.
- Models based on all metrics perform better than models based on significant metrics created using the Wilcoxon signed-rank test.
- RBF kernel performs best among the EML methods in predicting code smells.

References

1. Van Emden, E., Moonen, L.: Java quality assurance by detecting code smells. In: 2002 Proceedings of the Ninth Working Conference on Reverse Engineering, pp. 97–106. IEEE (2002)
2. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
3. Han, H., Wang, W.-Y., Mao, B.-H.: Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In: Huang, D.-S., Zhang, X.-P., Huang, G.-B. (eds.) ICIC 2005. LNCS, vol. 3644, pp. 878–887. Springer, Heidelberg (2005). https://doi.org/10.1007/11538059_91
4. Mathew, J., Luo, M., Pang, C.K., Chan, H.L.: Kernel-based smote for SVM classification of imbalanced datasets. In: IECON 2015–41st Annual Conference of the IEEE Industrial Electronics Society, pp. 001127–001132. IEEE (2015)
5. Ma, L., Zhang, Y.: Using Word2Vec to process big text data. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 2895–2897. IEEE (2015)
6. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1–3), 489–501 (2006)
7. Fernández-Navarro, F., Hervás-Martínez, C., Sanchez-Monedero, J., Gutiérrez, P.A.: MELM-GRBF: a modified version of the extreme learning machine for generalized radial basis function neural networks. Neurocomputing **74**(16), 2502–2510 (2011)
8. Wang, Q., Luo, Z., Huang, J., Feng, Y., Liu, Z.: A novel ensemble method for imbalanced data learning: bagging of extrapolation-SMOTE SVM. Comput. Intell. Neurosci. **2017** (2017)

9. Wang, Q., Xu, J., Chen, H., He, B.: Two improved continuous bag-of-word models. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 2851–2856. IEEE (2017)
10. Guthrie, D., Allison, B., Liu, W., Guthrie, L., Wilks, Y.: A closer look at skip-gram modelling. In: LREC, vol. 6, pp. 1222–1225. Citeseer (2006)
11. Micchelli, C.A., Pontil, M., Bartlett, P.: Learning the kernel function via regularization. J. Mach. Learn. Res. **6**(7) (2005)
12. Prajapati, G.L., Patle, A.: On performing classification using SVM with radial basis and polynomial kernel functions. In: 2010 3rd International Conference on Emerging Trends in Engineering and Technology, pp. 512–515. IEEE (2010)