An Empirical Framework for Code Smell Prediction using Extreme Learning Machine*

1st Himanshu Gupta
BITS Pilani Hyderabad Campus, India
f20150339@hyderabad.bits-pilani.ac.in

2nd Lov Kumar
BITS Pilani Hyderabad Campus, India
lovkumar@hyderabad.bits-pilani.ac.in

3rdLalita Bhanu Murthy Neti
BITS Pilani Hyderabad Campus, India
bhanu@hyderabad.bits-pilani.ac.in

***Abstract*—The software containing code smells indicates the violation of standard design and coding practices by developer during the development of the software system. Recent empirical studies observed that classes having code smells have higher probability of change proneness or fault proneness with respect to classes having no code smells [1]. The effort of removing bugs due to code smells increases exponentially if the smells are not identified during the earlier phases of software development. The code smell prediction using source code metrics can be used in starting phases of software development life cycle to reduce the maintenance and testing effort of software and also help in improving the quality of the software. The work in this paper empirically investigates and evaluates different classification techniques, feature selection techniques, and data sampling techniques to handle imbalance data in predicting 7 different types of code smell. The conclusion of this research is assessed over 629 application packages. The experimental finding confirms the estimating capability of different classifiers, feature selection, and data imbalance techniques for developing code smell prediction models. Our analysis also reveals that the models developed using one technique are superior than the models developed using other techniques.**

***Index Terms*—Code Smell, Software Engineering, Source Code Metrics, Machine Learning, Feature selection.**

I. Introduction

The smell present in source code of the software represent sign of poor design and coding environment. The testing team may not identify any bugs due to code smells in classes of current version of the software system but in future have a high probability to develop ones. Based on empirical study, researchers and experts in software quality have derived some methods that it is used to find the classes having code smells. The basis method used for removing these smell is code inspection i.e., manually supervising every line of code in the software package to find code smell [1]. The cost and effort required to remove code smell using code inspection method is very high due to manually finding. To automate this process, code smells prediction models are developed using source code metrics for predicting bad smell is software application. Further, the use of these developed prediction models reduces the maintenance and testing effort of software and also help in improving the quality of the software.

In this paper, a model for predicting different code smells such as Blob Class (BLOB), Complex Class (CC), Swiss Army Knife (SAK), Long Method (LM), Internal Getter/Setter (IGS), No Low Memory Resolver (NLMR), Member Ignoring Method (MIM), and Leaking Inner Class (LIC) [1] has been developed using different source code metrics. These all source code metrics are collected from java file of application packages. Since these metrics are used as input, so it is very important to remove irrelevant features and select right sets of relevant uncorrelated metrics out of these metrics. In order to achieve this objective, wilcoxon sign rank test and cross correlation analysis have been considered to select relevant uncorrelated metrics for code smells prediction. In this work, we have also analyzed the performance of extreme learning machine with different kernel functions and most frequently used classifiers such as linear regression, logistic regression, polynomial regression, and decision tree, different data imbalanced techniques using area under the curve (AUC) in order to predict software code smells. Three different Research Questions are addressed in this work:

- ***RQ1: what is the capability of various data sampling techniques to predict code smell?***
  In this work, we have considered three different data sampling techniques to handle imbalance class problem. The significance and reliability of these techniques are computed on different code smells using statistical test and AUC analysis.
- ***RQ2: what is the capability of various classifiers to predict code smells?***
  In this work, ELM with different kernels function have been considered to train the models for predicting code smells. The significance and reliability of these techniques are computed on different code smells using statistical test and AUC analysis.
- ***RQ3: what is the capability of selected features over original features to predict code smells?***
  In this work, significant and uncorrelated features are selected using wilcoxon sign rank test and cross correlation analysis. The significance and reliability of these techniques are computed on different code smells using statistical test and AUC analysis.

II. Related Work

Fabio Palomba used textual-based approach named as Textual Analysis for Code Smell Detection (TACO) for detecting

[1]https://github.com/sealuzh/user_quality/wiki/Code-Smells

978-1-5386-9325-4/19/$31.00 ©2019 IEEE

long Method smell [3]. It uses the elemental concept that long method have a set of separately unmanageable and unrelated code blocks. The comparison of TACO and DCOR (a structural based technique also used for detecting long method) using intersection and union revealed that found that TACO was able to achieve better precision and recall than DCOR. The author finally concluded that in order to improve the accuracy a combination of both approaches should be used since they are highly complementary. Palomba and his colleagues also investigates the impacts and their relating effect of co-occurring code smells on a similar code base [4].

Marco Zanoni and his colleagues used 16 different algorithms on 4 code smells using 74 software systems and obtained an accuracy of 95% with only 100 training examples [5]. The selection and labeling phase of example instances play an important role which ensures a homogeneous selection of instances on different projects and prioritizes the labeling of instances with a higher chance of being affected by a code smell and these instances are used in machine learning algorithms. The homogeneous selection of instances priorities the labeling with a higher chance of being affected by code smell and the selected instances are used to train machine learning algorithms.

III. RESEARCH BACKGROUND

We are using two steps process to select significant and uncorrelated features, ELM with three kernels and most frequently used classifiers to detect code smells and three different data sampling techniques to handle class imbalance problem.

A. Experimental Dataset

In this work, the empirical experiments are conducted on 629 open source projects data that are available on the GitHub. This dataset contains list of packages along with values of certain features and the code smells they exhibit. The characteristics of all code smells over 629 software application packages are given in Table I. From Table I, it has been observed that percentage of anti-pattern varies from 26 to 75%. Further I, it has been also observed that highest 75.04% application packages have No Low Memory Resolver (NLMR) code smell and the lowest 26.23% application packages have Swiss Army Knife code smell.

TABLE I: Number and Percentages of 5 Anti-Patterns in 226 Web-Services

	# No Smell	% No Smell	# Smelly	% Smelly
BLOB	237	37.68	392	62.32
LM	155	24.64	474	75.36
SAK	464	73.77	165	26.23
CC	187	29.73	442	70.27
IGS	278	44.20	351	55.80
MIM	260	41.34	369	58.66
NLMR	157	24.96	472	75.04
LIC	228	36.25	401	63.75

B. Software Metrics

In this experiment, software metrics are used as an input for developing a model for predicting code smell of software applications. The software metrics considered in this work are the ones, which are most frequently used by different researchers to measure internal structure of software system [6] [7] [8]. The software metrics used in this research are NOC, IPM, WMC, CC, NOCH, NBI, NOM, DIT, PPIV, LCOM, NOCH, LOCL, APD, XML, BSMC, NTO, WKL, GPS, BMAP, SQL, NET, and I/O. A complete description and abbreviation of these metrics are explained with practical applications in [9] [7] [10] [2].

C. Feature Selection Techniques

In this work, wilcoxon sign rank and cross correlation analysis have been considered to achieve above objective. Wilcoxon sign rank is used to test statistical significance between smelly applications and no-smelly applications. The null hypothesis for each metric i.e., this metric can not predict smell applications, which tested using Wilcoxon sign rank. In this work, the threshold p-value is considered as 0.05 i.e., the hypothesis is rejected if p-value is less than 0.05 else accepted. After finding significant features using above technique, cross correlation analysis has been considered to select the uncorrelated feature. The concept of this method is based on selection of independent features that have high correlation with output variables and low correlation with each other.

D. Classification Algorithms:

After selecting significant and uncorrelated features, ELM with three different kernel functions are used to train the code smell prediction model. The performance of these kernels are evaluated using three different performance parameters and compared with most frequently used classifiers.

E. Data Sampling Technique to handle Imbalanced Data:

The unequal representation of classes, leads to imbalanced data. It can be seen from Table I, SAK code smell is present in 165 out of 629 i.e., 26.23 % of applications have SAK code smell. From above analysis, we observed that our considered experimental data sets are not equally distributed and lead to class imbalance problem. To deal with above problem i.e., imbalanced class problem, three different techniques such as downscale sampling, random sampling, and upscale sampling techniques have been considered. The performance of these techniques are evaluated and compared with original data.

IV. RESEARCH FRAMEWORK AND EXPERIMENTAL RESULTS

Figure 1 provides a detailed framework for development of code smell prediction models using software metrics as input. The code smell and metrics data collected for 629 software application packages from open sources GitHub repository. The

[2]https://github.com/sealuzh/user_quality/wiki/Code-Quality-Metrics

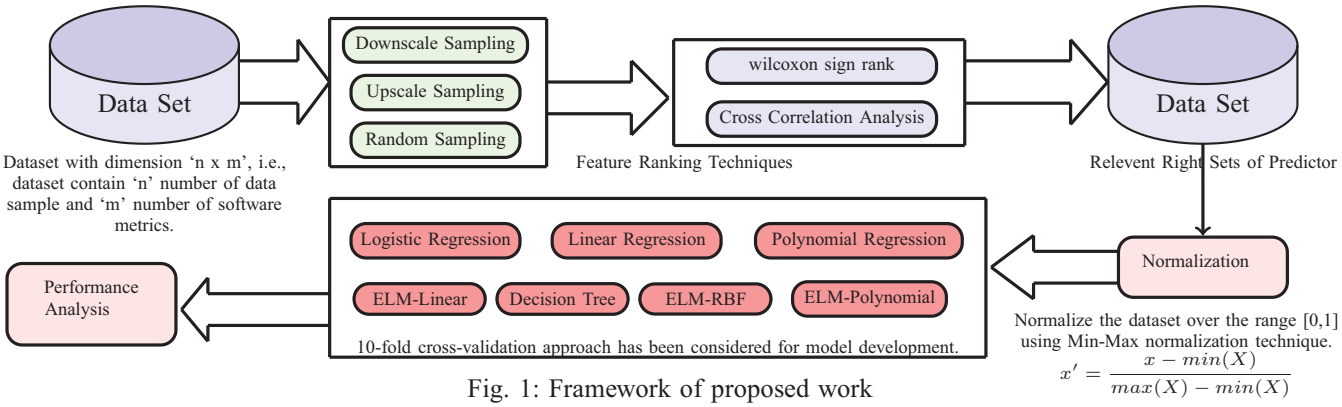
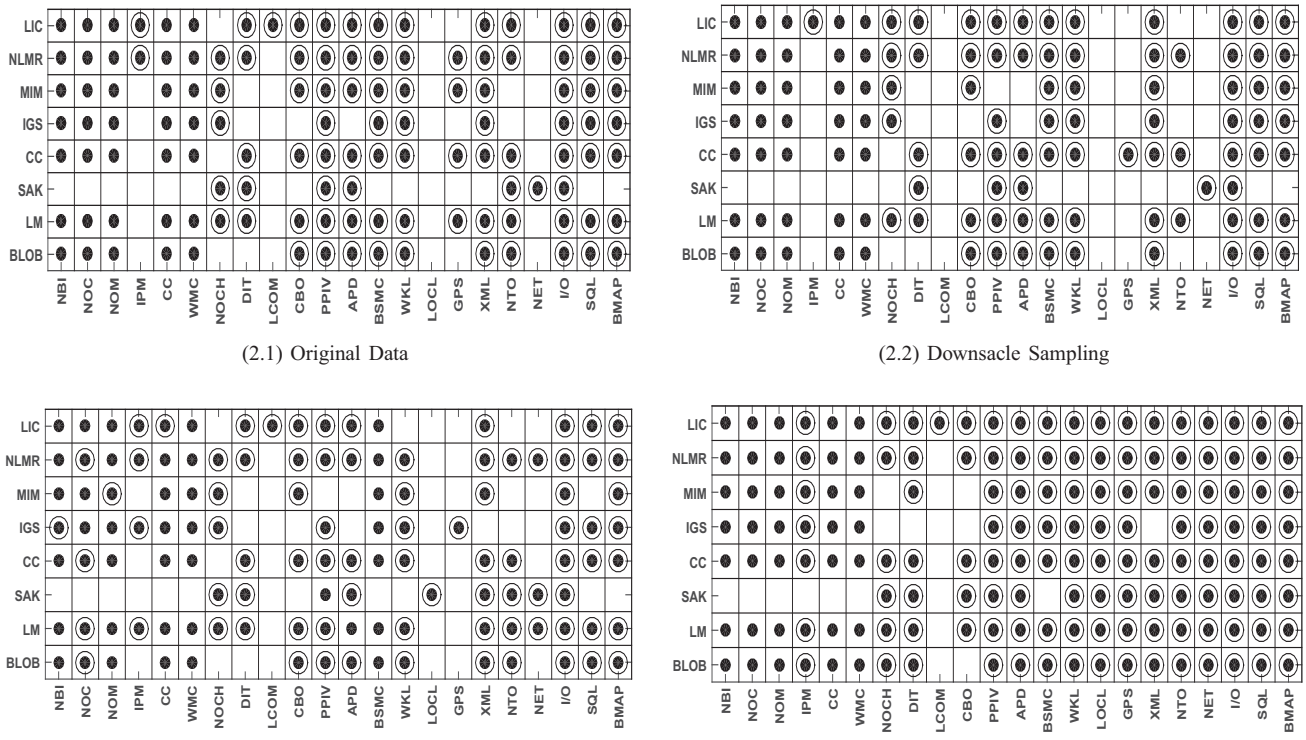


Fig. 1: Framework of proposed work

software metrics considered for this analysis are NOC, IPM, WMC, CC, NOCH, NBI, NOM, DIT, PPIV, LCOM, NOCH, LOCL, APD, XML, BSMC, NTO, WKL, GPS, BMAP, SQL, NET, and I/O. It can be seen from Table I that our considered datasets are suffering from class imbalance problem, in this work three different data sampling techniques are considered to handle data imbalance problem. The second step consists of applying the wilcoxon sign rank and cross correlation analysis on each balanced data to retrieve significant and uncorrelated features sets for code smell prediction. Then, we are using Min-Max Normalization to normalize the all selected features in the same range of 0 to 1. Further, ELM with different kernel functions and some most frequently used classifiers have been considered to train the models and validated these trained models using 10-fold cross-validation approach. Finally, we have considered three different performance parameters such as accuracy, F-Measure, and AUC to evaluate the significance and reliability and of these techniques.

A. Feature Selection Techniques:

In this section, results obtained for each code smells using wilcoxon sign rank and cross correlation analysis have been summarized. Figure 2 provides the selected features using wilcoxon sign rank and cross correlation analysis for each code smells of each data. The 1st sub-figure of Figure 2 provides the selected feature without using any data sampling



(2.1) Original Data

(2.2) Downsacle Sampling

(2.3) Random Sampling

(2.4) Upsacle Sampling

Fig. 2: Significant and Uncorrelated Metrics

techniques, similarly 2nd sub-figure of Figure 2 provides the results for upscale sampling, 3rd for random sampling, and last for upscale sampling technique. In Figure 2, two different symbols such as black circle denotes the features selected using wilcoxon sign rank and blank circle + black circle denotes the selected features using cross correlation analysis. It can be seen from Figure 2 that CBO, PPIV, BSMC, WKL, APD, XML, NTO, I/o, SQL, BMAP metrics are selected for code smell BLOB type in case of original data. Similarly, From Figure 2, we can also observed that I/O metrics are found to be significant across all code smells and all data sets.

B. Results and analysis

We used the selected relevant sets of metrics using above techniques as input to develop a model for predicting code smell. These models are trained using ELM with various kernel functions and most frequently used classifiers. The developed models are validated using 10-fold cross validation and performance of these models are computed in terms of three different performance parameters such as Accuracy, F-Measure, and AUC. Table II provides the AUC (same for other parameters) values obtained from individual model on BLOB code smell. It can be seen from Table II that:

- The AUC value of predictive models trained after applying upscale data sampling technique are relatively better than the other techniques.
- The AUC value of predictive models trained using ELM with polynomial kernel (ELM-Poly) and decision tree (DT) are relatively similar.
- The AUC value of predictive models trained by considering all features and selected significant features are relatively similar.

Figures 3 and 4 show the box-plot diagrams for AUC

TABLE II: AUC of BLOB code smell

	ORG							SRT							CCA						
	Lin	**Poly**	**Log**	**DT**	***ELM-LIN***	***ELM-Poly***	***ELM-RBF***	**Lin**	**Poly**	**Log**	**DT**	***ELM-LIN***	***ELM-Poly***	***ELM-RBF***	**Lin**	**Poly**	**Log**	**DT**	***ELM-LIN***	***ELM-Poly***	***ELM-RBF***
Original Data																					
Fold 1	0.66	0.70	0.69	0.78	0.69	0.73	0.69	0.70	0.73	0.70	0.79	0.66	0.67	0.70	0.61	0.62	0.68	0.83	0.63	0.68	0.67
Fold 2	0.74	0.79	0.79	0.84	0.76	0.77	0.77	0.77	0.76	0.78	0.84	0.73	0.75	0.74	0.68	0.60	0.61	0.71	0.54	0.66	0.65
Fold 3	0.69	0.70	0.69	0.87	0.70	0.75	0.68	0.74	0.69	0.73	0.82	0.73	0.75	0.74	0.68	0.72	0.68	0.76	0.54	0.69	0.62
Fold 4	0.71	0.73	0.72	0.81	0.71	0.76	0.71	0.65	0.63	0.67	0.74	0.64	0.64	0.65	0.61	0.57	0.62	0.76	0.60	0.63	0.60
Fold 5	0.72	0.67	0.69	0.75	0.70	0.71	0.70	0.63	0.72	0.69	0.72	0.70	0.70	0.70	0.62	0.62	0.61	0.74	0.57	0.62	0.60
Downscale Sampling																					
Fold 1	0.69	0.69	0.69	0.79	0.71	0.76	0.75	0.72	0.74	0.75	0.83	0.79	0.73	0.76	0.68	0.66	0.65	0.75	0.66	0.67	0.67
Fold 2	0.71	0.76	0.73	0.82	0.73	0.76	0.72	0.76	0.74	0.78	0.76	0.76	0.78	0.76	0.72	0.71	0.72	0.74	0.60	0.71	0.74
Fold 3	0.69	0.71	0.72	0.79	0.72	0.79	0.76	0.77	0.74	0.75	0.70	0.75	0.78	0.76	0.63	0.57	0.63	0.74	0.55	0.64	0.63
Fold 4	0.77	0.72	0.73	0.79	0.74	0.73	0.76	0.59	0.64	0.62	0.74	0.61	0.61	0.61	0.57	0.55	0.58	0.70	0.45	0.62	0.60
Fold 5	0.61	0.69	0.64	0.72	0.63	0.68	0.68	0.64	0.71	0.65	0.73	0.67	0.66	0.72	0.65	0.64	0.65	0.78	0.60	0.69	0.62
Random Sampling																					
Fold 1	0.73	0.76	0.72	0.81	0.71	0.74	0.74	0.72	0.69	0.74	0.77	0.70	0.72	0.68	0.67	0.69	0.74	0.74	0.65	0.72	0.72
Fold 2	0.65	0.68	0.66	0.74	0.69	0.71	0.69	0.70	0.61	0.68	0.78	0.65	0.69	0.66	0.62	0.65	0.62	0.73	0.66	0.68	0.72
Fold 3	0.71	0.70	0.74	0.80	0.74	0.76	0.71	0.74	0.77	0.76	0.84	0.78	0.78	0.78	0.71	0.70	0.73	0.80	0.71	0.74	0.74
Fold 4	0.72	0.70	0.75	0.75	0.74	0.76	0.75	0.70	0.68	0.73	0.78	0.72	0.73	0.76	0.74	0.74	0.72	0.76	0.69	0.77	0.74
Fold 5	0.74	0.77	0.74	0.75	0.73	0.75	0.77	0.74	0.74	0.77	0.73	0.79	0.76	0.75	0.72	0.67	0.75	0.84	0.73	0.73	0.71
Upscale Sampling																					
Fold 1	0.78	0.81	0.79	0.82	0.79	0.79	0.83	0.73	0.77	0.76	0.81	0.70	0.74	0.73	0.61	0.59	0.64	0.78	0.56	0.62	0.61
Fold 2	0.80	0.82	0.82	0.85	0.82	0.83	0.82	0.84	0.80	0.82	0.85	0.80	0.85	0.85	0.80	0.55	0.79	0.80	0.65	0.80	0.77
Fold 3	0.76	0.78	0.80	0.85	0.79	0.78	0.77	0.80	0.85	0.81	0.88	0.76	0.82	0.84	0.78	0.58	0.76	0.82	0.71	0.79	0.76
Fold 4	0.83	0.83	0.84	0.82	0.82	0.84	0.83	0.80	0.80	0.81	0.83	0.73	0.79	0.76	0.74	0.53	0.74	0.79	0.64	0.74	0.73
Fold 5	0.80	0.76	0.78	0.84	0.81	0.81	0.79	0.80	0.76	0.74	0.82	0.73	0.78	0.81	0.80	0.55	0.79	0.82	0.60	0.80	0.79

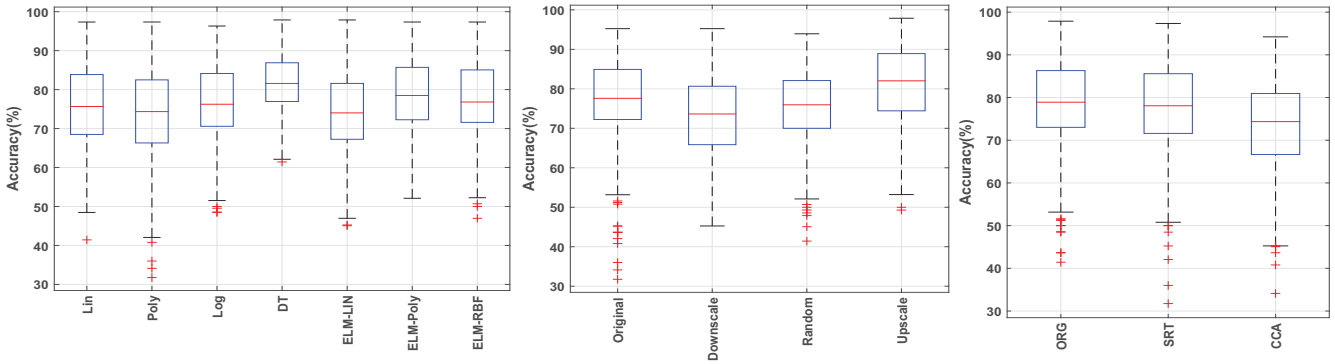


Fig. 3: Boxplots for Accuracy

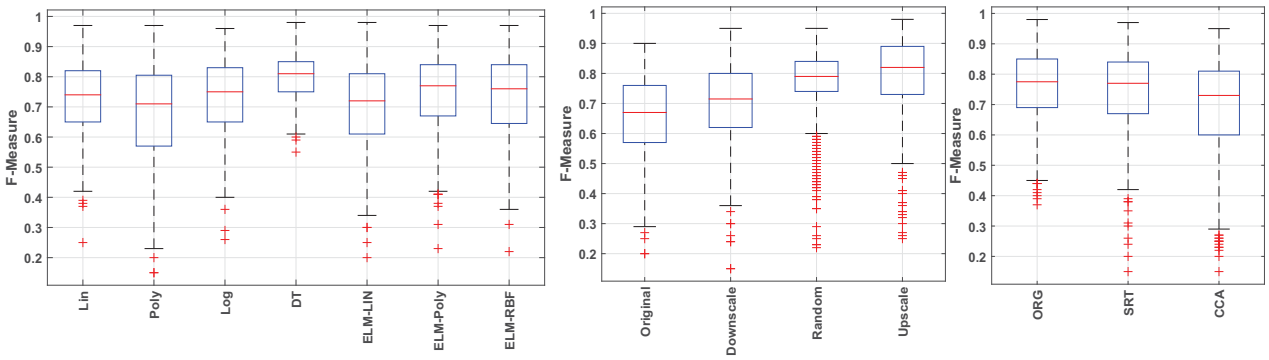


Fig. 4: Boxplots for F-Measure

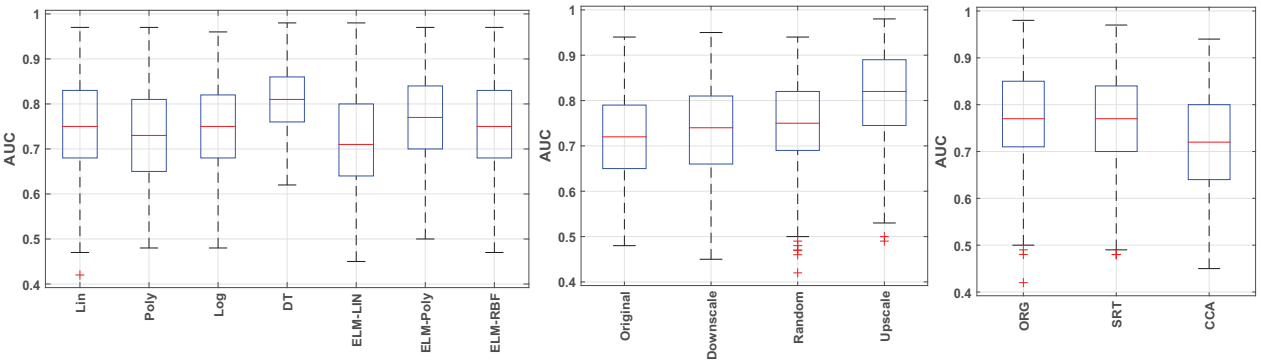


Fig. 5: Boxplots for AUC

on all 4 anti-patterns for feature selection techniques and classification techniques. Figure 3 reveals that the median or middle quartile of the AUC (marked with a red line) varies significantly across the box-plots. From Figure 3, we observe that the model built using the features selected by FS2 provides relatively better performance in detecting AP1 compared to the models built by the features selected by other feature selection techniques. Similarly the models built by the features selected by the feature selection techniques FS5, FS8, FS6 yield better performance in predicting anti-patterns AP2, AP3 and AP4 respectively. Figure 4 reveals that the median or middle quartile of the AUC (marked with a red line) varies significantly across the box-plots. From Figure 4 and Table IV, we observe that the model built using the SVML provides relatively better performance in detecting AP1 compared to the models built by other techniques. Similarly the models built by different classification techniques RF, RF, and SVMR yields better performance in predicting anti-patterns AP2, AP3 and AP4 respectively.

V. Comparison

RQ1: what is the capability of various data sampling techniques to predict code smell? In this study, three different types of data imbalance techniques such as downscale sampling, random sampling, and upscale sampling have been considered to handle class imbalance problem. The capability and significance of these techniques are evaluated and compared using Descriptive Statistics, Boxplots, and Statistical tests analysis.

Comparison of different sampling techniques using Boxplots and Descriptive Statistics:

Figures 3, 4, and 5 show the boxplot for different performance parameters such as accuracy, F-Measure, and AUC for each data sampling technique and also performance of the model developed using original data. The descriptive statistics of each box for different techniques are summarized in III. From Table III and Figures 3, 4, and 5, we can see that upscale technique achieved the best result in general, whereas the model developed without data sampling technique i.e., original data was the worst. Among the data sampling techniques, the downscale sampling technique provided worst results.

Comparison of different sampling techniques using Statistical tests:

After obtaining best sampling technique using boxplots and descriptive statistics of different performance parameters, we performed wilcoxon sign rank (WSR) test to evaluate the statistical significance different between the pairs of different data sampling techniques. The null hypothesis of the WSR is that *there is no significant difference between the performance of the model developed using different data sampling techniques*. Figure 6 shows the p-value using two different symbol i.e, circle: p-value$>$0.05 and cross: p-value$\leq$0.05. From

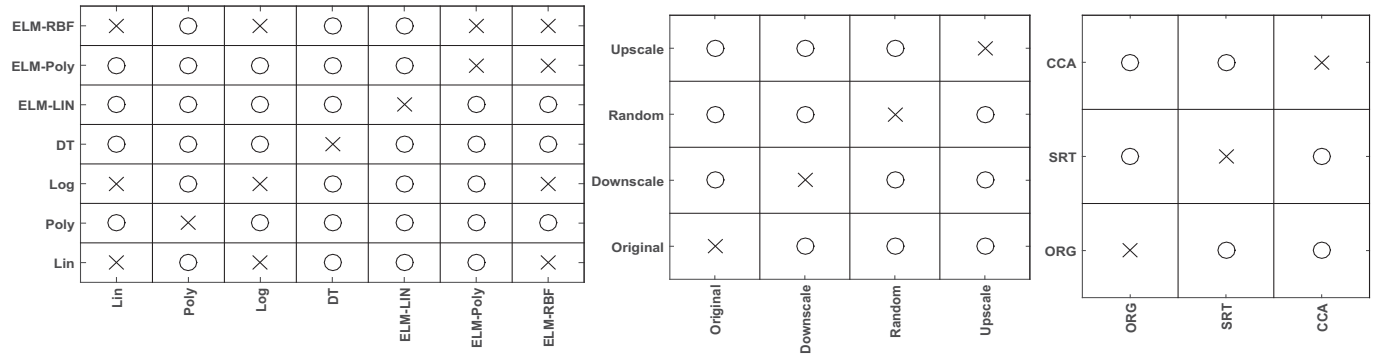


Fig. 6: wilcoxon sign rank

TABLE III: Statistical Measures on AUC values on prior and after SMOTE data

Accuracy						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
Original	31.75	95.24	77.57	77.60	72.22	84.92
Downscale	45.26	95.24	73.09	73.63	65.85	80.65
Random	41.43	93.94	75.73	75.95	70.00	82.09
Upscale	49.29	97.88	81.01	82.02	74.43	88.92
F-Measure						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
Original	0.20	0.90	0.66	0.67	0.57	0.76
Downscale	0.15	0.95	0.71	0.72	0.62	0.80
Random	0.22	0.95	0.77	0.79	0.74	0.84
Upscale	0.25	0.98	0.80	0.82	0.73	0.89
AUC						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
Original	0.48	0.94	0.72	0.72	0.65	0.79
Downscale	0.45	0.95	0.73	0.74	0.66	0.81
Random	0.42	0.94	0.75	0.75	0.69	0.82
Upscale	0.49	0.98	0.81	0.82	0.75	0.89

Figure 6, it can be seen that the considered null hypothesis is rejected at 0.05 level for all pairs i.e., the code smell prediction model developed using different data sampling techniques are significantly different.

RQ2: what is the capability of various classifiers to predict code smells?

In this paper, ELM with different kernels functions i.e., Linear, RBF, and Polynomial kernel function have been considered to train the models for predicting code smells. The capability and significance of these techniques are evaluated and compared using Descriptive Statistics, Boxplots, and Statistical tests analysis. Finally the performance of the code smell prediction models are compared using most frequently used classification techniques i.e., Logistic regression(LOG), Linear Regression (LIN), Decision Tree (DT), and Polynomial regression (POLY).

Comparison of Classification Techniques using Boxplots and Descriptive Statistics: The 1st sub-figure of Figures 3, 4, and 5 show the boxplot for different performance parameters such as accuracy, F-Measure, and AUC for each classifier. The descriptive statistics of each box for different classifiers are summarized in IV. From Table IV and Figures 3, 4, and 5, we can see that ELM with polynomial kernel achieved the best result among different kernel functions, where as linear kernel was the worst. Among all techniques, decision tree achieved good results.

TABLE IV: Statistical Measures on AUC values of Classification techniques.

Accuracy						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
Lin	41.43	97.35	75.90	75.66	68.50	83.86
Poly	31.75	97.35	73.65	74.36	66.31	82.50
Log	48.48	96.32	76.70	76.25	70.59	84.13
DT	61.43	97.88	81.70	81.58	76.95	86.89
ELM-LIN	45.07	97.88	74.05	74.02	67.25	81.59
ELM-Poly	52.11	97.35	78.41	78.48	72.26	85.71
ELM-RBF	46.97	97.35	77.54	76.82	71.59	85.06
F-Measure						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
Lin	0.25	0.97	0.73	0.74	0.65	0.82
Poly	0.15	0.97	0.68	0.71	0.57	0.81
Log	0.26	0.96	0.74	0.75	0.65	0.83
DT	0.55	0.98	0.81	0.81	0.75	0.85
ELM-LIN	0.20	0.98	0.70	0.72	0.61	0.81
ELM-Poly	0.23	0.97	0.75	0.77	0.67	0.84
ELM-RBF	0.22	0.97	0.73	0.76	0.65	0.84
AUC						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
Lin	0.42	0.97	0.75	0.75	0.68	0.83
Poly	0.48	0.97	0.73	0.73	0.65	0.81
Log	0.48	0.96	0.75	0.75	0.68	0.82
DT	0.62	0.98	0.81	0.81	0.76	0.86
ELM-LIN	0.45	0.98	0.72	0.71	0.64	0.80
ELM-Poly	0.50	0.97	0.76	0.77	0.70	0.84
ELM-RBF	0.47	0.97	0.75	0.75	0.68	0.83

Comparison of Classification Techniques using Statistical tests: In this work, we have also performed WSR test to evaluate the statistical significance difference between the pairs of different classifiers. The 1st sub-figure of Figure 6 shows the p-value using two different symbol i.e, circle: p-value$>$0.05 and cross: p-value$\leq$0.05. From Figure 6, it can be seen that the considered null hypothesis is rejected at 0.05 level for most pairs i.e., the code smell prediction model

TABLE V: Statistical Measures on AUC values of Feature Selection Techniques

Accuracy						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
ORG	41.43	97.88	79.01	78.91	73.02	86.31
SRT	31.75	97.35	77.80	78.08	71.60	85.58
CCA	34.13	94.21	73.74	74.36	66.67	80.95
F-Measure						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
ORG	0.37	0.98	0.76	0.78	0.69	0.85
SRT	0.15	0.97	0.75	0.77	0.67	0.84
CCA	0.15	0.95	0.69	0.73	0.60	0.81
AUC						
	Min	**Max**	**Mean**	**Median**	**25%**	**75%**
ORG	0.42	0.98	0.78	0.77	0.71	0.85
SRT	0.48	0.97	0.76	0.77	0.70	0.84
CCA	0.45	0.94	0.72	0.72	0.64	0.80

developed using different classifiers are significantly different.

RQ3: what is the capability of selected features over original features to predict code smells?

In this work, three different sets of features i.e., original features (ORG), relevant significant features computed using WSR test (SRT), relevant uncorrelated significant features computed using cross correlation analysis (CCA) have been used as input features to train the model for predicting different types of code smells. The capability and significance of these sets of features are evaluated and compared using Descriptive Statistics, Boxplots, and Statistical tests analysis.

Comparison of Feature Selection Techniques using Boxplots and Descriptive Statistics: The 2nd sub-figure of Figures 3, 4, and 5 show the boxplot for different performance parameters such as accuracy, F-Measure, and AUC for each set of features. The descriptive statistics of each box for different sets of features are summarized in V. From Table V and Figures 3, 4, and 5, we can see that small set of features using SRT have similar performance with original features, where as selected features after CCA was the worst. From these results, we can also observed that there exist small number of features which predict code smell better than all features.

Comparison of Feature Selection Techniques using Statistical tests:

From above analysis, we can observed that code smell prediction models developed using different sets of features have different value for performance parameters. In this work, we have also performed WSR test to evaluate the statistical significance difference between the pairs of different sets of features. The 3rd sub-figure of Figure 6 shows the p-value using two different symbol i.e, circle: p-value$>$0.05 and cross: p-value$\leq$0.05. From Figure 6, it can be seen that the considered null hypothesis is rejected at 0.05 level for all pairs i.e., the code smell prediction model developed using different sets of features are significantly different.

VI. Conclusion

The primary goal of this research is to find the effect of internal structure of software on different code smells i.e., source code metrics and code smells. In this study, we also empirically evaluated, analyzed, and compared the performance of ELM with different kernels, most frequently used classifiers, data sampling techniques for predicting different types of code smells using three different performance parameters. The major conclusions of this study are summarized as follows.

- The code smell prediction models developed using different sets of features, data sampling technique, and classifiers are significantly different.
- Selected relevant features provide relatively similar performance for predicting code smell in software to the models built using all features.
- Decision tree provide relatively better performance in detecting code smell compared to the models built by other classifiers.
- Upscale sampling technique provides relatively better performance in detecting code smells compared to the models built by original data, downscale sampling, random sampling.

References

[1] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1188–1221, 2018.

[2] F. A. Fontana, V. Ferme, M. Zanoni, and A. Yamashita, "Automatic metric thresholds derivation for code smell detection," in *Proceedings of the Sixth international workshop on emerging trends in software metrics*. IEEE Press, 2015, pp. 44–53.

[3] F. Palomba, "Textual analysis for code smell detection," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 769–771.

[4] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "A large-scale empirical study on the lifecycle of code smell co-occurrences," *Information and Software Technology*, vol. 99, pp. 1–10, 2018.

[5] 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.

[6] S. R. Chidamber and C. F. Kemerer, *Towards a metrics suite for Object-Oriented design*. ACM, vol. 26, no. 11.

[7] R. Malhotra and Y. Singh, "On the applicability of machine learning techniques for object oriented software fault prediction," *Software Engineering: An International Journal*.

[8] W. Li and S. Henry, "Maintenance metrics for the Object-Oriented paradigm," in *Proceedings of First International Software Metrics Symposium*, 1993, pp. 52–60.

[9] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," *Software process: Improvement and practice*.

[10] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for Object-Oriented software: An exploratory analysis," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629–639, 1998.