DUJASE Vol. 8 (1) 32-41, 2023 (January) *https://doi.org/10.3329/dujase.v8i1.72991*

Impact of Incorporating Community Smells with Object-Oriented Metrics to Predict Change-Prone Classes

Toukir Ahammed[1], Mahir Mahbub[2], Md. Hasan Tarek[3] and Ahmedul Kabir[1]

[1]*Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh*
[2]*Department of IoT and Robotics Engineering, Bangabandhu Sheikh Mujibur Rahman Digital University,* Gazipur, Bangladesh
[3]*Department of Computer Science and Engineering, Begum Rokeya University,* Rangpur, Bangladesh

**E-mail: kabir@iit.du.ac.bd*

Received on 14 September 2023, Accepted for Publication on 25 January 2024

ABSTRACT

Change-proneness is an important quality attribute that can help developers to maintain source code more effectively. Change-prone classes refer to those classes which are more likely to change across releases. Existing approaches have used object-oriented metrics to predict change-prone classes. However, community smells related information, which represents the communication issues among developers' community, has not yet been considered to predict change-prone classes. This study investigates the impact of including community smells-related information with existing Object Oriented (OO) metrics to predict a class to be change-prone in future software releases. It aims to understand the extent to which the addition of community smell-related information can contribute to the performance of change prediction models. To perform analysis, 317 releases of 14 open-source Java projects were selected from *GitHub*. The resulting dataset is used to predict change-prone classes with five different popular and widely used machine learning algorithms, namely K-nearest neighbor (KNN), Random Forest (RF), Naive Bayes (NB), Logistic Regression (LR) and Multilayer Perceptron (MLP). To evaluate the performance of the classifiers, accuracy, and F1-measure are used. The experimental results suggest that including community smell metrics along with the existing OO metrics gives better performance in predicting change-prone classes. Moreover, MLP and LR algorithms, which are both parametric models, are found to predict change-prone classes better when Community Smell(CMS) related features are incorporated, in comparison with other non-parametric models. By including CMS along with OO metrics, 1.51% increase in average accuracy and 3.52% increase in average F1 score for Logistic Regression, and 1.69% increase in average accuracy and 3.26% increase in average F1 score for Multi-layer Perceptron have been found.

Keywords: Change-proneness, Change prediction, Community smell, Object-oriented metrics

1. Introduction

Change-proneness refers to the extent of change carried out on a software artifact across different releases of the system. It is an important quality attribute of a class that defines how likely a class will change in the future. Prediction of such change-prone classes can help practitioners take preventive measures to reduce maintenance costs and to allocate resources more effectively [1]. Thus, the maintenance team can optimize and use their testing resources on the classes that have a higher likelihood of change.

Existing studies have investigated the ability of objectoriented metrics to predict change-prone classes [1], [2]. However, recent studies show that developers' interaction with source code is not only dependent on technical factors but also on community-related aspects [3]–[6]. Community-related circumstances can affect the way developers act in the source code [3]. Therefore, social aspects of the software development community need to be considered with technical factors while studying the underlying dynamics of the software system like change-proneness.

Under this context, this paper investigates the impact of including community smell, which denotes the organizational and social anti-patterns in the software development community that may lead to unforeseen project costs [7], to predict change-prone classes. In particular, this study aims to understand to what extent the addition of information related to community smells can contribute to the performance of change prediction models. Further, the usability and credibility of using community smell features to predict change-proneness are verified using multiple evaluation approaches. To predict change-prone class, this study considers three community smells, namely *Organizational Silo Effect*, *Lone Wolf Effect*, and *Radio Silence Effect* with existing object-oriented (OO) metrics. Five different popular and widely used machine learning algorithms, namely K-nearest neighbor (KNN), Random Forest (RF), Naive Bayes (NB), Logistic Regression (LR), and Multi-layer Perceptron (MLP), are used for prediction. For experimentation, a dataset is prepared from 317 releases of 14 open-source Java projects such as Apache Ant, Eclipse-CDT, OpenNLP, Tomcat, etc. To evaluate the models, performance metrics namely Accuracy and F1-measure are used in this study. The result of the study suggests that including community smells with existing OO metrics achieves better performance in predicting a change-prone class with LR and MLP classifiers.

The rest of the paper is organized as follows: Section II discusses the literature related to the prediction of changeprone classes while Section III describes the methodology of the study. Section IV presents and discusses the experimental results. Threats to the validity of the study are discussed in Section V. Finally, Section VI concludes the paper and outlines the future direction.

2. Related Work

Multiple factors can impact the change-proneness of the class. In [8], the authors empirically investigate the changeproneness of design patterns and the types of changes that occur to classes that play a role in some design patterns in contrast to the software evolution. It switches the focus from design patterns as a whole to design pattern roles at a more finer level to 12 different design patterns.

In [9], the anti-patterns are analyzed to find their impacts on the change-proneness of the classes. The authors answer several research questions related to the research topic by analyzing 54 releases of 4 different projects, i.e. ArgoUML, Eclipse, Mylyn, and Rhino which include 13 anti-patterns. They also show that class size alone cannot explain how classes with anti-patterns have a higher chance of undergoing a (fault-fixing) change than other classes. Finally, they show that structural changes have a greater impact on anti-pattern classes than on other classes. They concluded that when code smells are present, affected classes are more prone to change than non-smelly classes. This conclusion is also later confirmed by [10] [11]. So different software artifacts/metrics related to code smells can be useful to detect the change-proneness of classes.

Different source code metrics can impact the changeproneness [12] [13]. In [14], They present a technique for estimating change-prone classes that use both CK and QMOOD measures. Another research [15] proposed that process metrics can be useful in change predictions. In [16], The researchers examine the role of developer-related factors in change prediction using empirical evidence, such as, entropy of development process [17], number of developers working on a certain class [18], structural and semantic scattering of changes [13].

In [19], the authors argue that the code smells are affected by the community smells. To show the relation between the two types of smell, they surveyed 162 developers of nine opensource systems. They also make a deep analysis by examining an empirical study of 117 releases from these systems. According to their findings, most developers intuitively see community-elated aspects as sources of code smell persistence. They considered four different community smells to show this relation: Organisational Silo Effect, Black-cloud Effect, Lone-wolf Effect, and Bottleneck Effect [7]. In this paper, we argue that community smells also play a valid role in changeproneness detection similar to code smells.

3. Methodology

This study aims to understand the impact of including community smell-related information with object-oriented (OO) metrics to predict change-prone classes. For this purpose, a dataset is prepared from 14 open-source Java projects of *Github*[1]. First, community smells are detected to identify whether a class is affected by any community smell. Next, OO metrics are computed using a static code analysis tool. The change-proneness of classes is calculated by analyzing the change history from the repository. Fig. 1 describes the process of data collection.

Finally, the resulting dataset is used to predict change-prone classes with five popular and widely used machine learning algorithms. To evaluate the performance of the classifiers, accuracy, and F1-measure are considered. The process of model building and analysis is shown in Fig. 2. The following subsections describe the methodology in detail.

A. Project Selection

To perform the analysis, 317 releases of 14 open-source Java projects are selected from *Github*. The selection of projects is driven by the factors of considering projects of different codebase size, longevity, and activity. Projects are selected that have the number of classes greater than 500, have a change history of at least 5 years, and have the number of commits higher than 1000. This project selection process is similar to [19]. The details of the selected projects are provided in Table I with the project name, the link to the source code repository, the number of considered releases, and the number of analyzed classes. For example, 28 releases of *ActiveMQ*, a Java message broker system, are considered for analysis which covers about 14 years of lifetime and 9086 commits of this project.

[1] https://github.com

Table 1: List of Analyzed Projects

Project	Source Code	#Analysed Releases	#Analysed Commits	#Classes
ActiveMQ	github.com/apache/activemq	24	9086	4673
Ant	github.com/apache/ant	33	11797	1446
Cassandra	github.com/apache/cassandra	18	10102	2694
Cayenne	github.com/apache/cayenne	17	6569	5409
CXF	github.com/apache/cxf	16	14988	7495
Drill	github.com/apache/drill	24	3842	4267
Eclipse-CDT	github.com/eclipse-cdt/cdt	38	21818	9615
Jackrabbit	github.com/apache/jackrabbit	28	7060	3462
Jena	github.com/apache/jena	30	7911	9637
Mahout	github.com/apache/mahout	12	4026	1803
OpenNLP	github.com/apache/opennlp	8	1871	893
Pig	github.com/apache/pig	15	3476	2105
POI	github.com/apache/poi	19	8820	4406
Tomcat	github.com/apache/tomcat	35	19370	1883
Total		*317*	*130736*	*59788*

The source code of selected projects is available on *Github*. The selected projects have 317 releases and the change history of 130,736 commits in total.

B. *Community Smells Detection*

To detect community smells, the state-of-the-art tool *Codeface4Smells* [20] is used. This tool is publicly available and widely used by most of the studies related to community smells [3], [21]–[23]. Using this tool three community smells were detected which are *Organizational Silo*, *Lone Wolf*, and *Radio Silence*. Community smells are detected according to the identification pattern given by Tamburri et al. [21]. The description of three community smells, considered in this study, is given below.

1) *Organizational Silo (OS):* the presence of siloed areas of the software developer community that do not communicate within the defined communication channel such as mailing list [21].
2) *Lone Wolf (LW) or Missing Link:* refers to the situation where developers collaborate among themselves but do not communicate with each other [21].
3) *Bottleneck or Radio Silence (RS):* the presence of one member in every formal interaction across two or more sub-communities. Thus, there is little or no flexibility to introduce other parallel communication channels between the remaining members of the sub-communities [3].

A temporal window must be set to detect community smells because the software development community changes from time to time. In this study, release is chosen because the change-prone class will be identified at the release interval. After detecting community smells, developers in a release are divided into two categories - smelly developers and non-smelly developers. A developer involved in any kind of community smell is considered a smelly developer for that release. Otherwise the developer is considered as non-smelly.

Next, the involvement of classes in any kind of community smell is identified. A class is said to be affected by community smells if the class has been modified by a smelly developer. For example, if a class C_i is modified by a smelly developer in between two consecutive releases r_{j-1} and r_j, class C_i is considered smelly, otherwise it is non-smelly. Thus, three boolean values are computed representing the involvement of class C_i in any of the three community smells considered.

C. *Object-oriented Metrics Calculation*

The list of 15 object-oriented metrics [24] considered in this study is given below. The investigated metrics cover different object-oriented dimensions including size, complexity, coupling, cohesion, abstraction, encapsulation, and documentation. In this study, a static code analysis tool, *Understand*[2]

(version:5.1, Build:1029), is used to compute these metrics.

Thus, the values of these metrics are calculated for each class C_i in a release r_j.

- *LOC:* Class Lines of Code
- *NOM:* Number of local Methods [25]
- *NIM:* Number of Instance Methods
- *NIV:* Number of Instance Variables

[2] https://www.scitools.com/

- *WMC:* Weighted Methods per Class [26]
- *CBO:* Coupling Between Objects [26]
- *RFC:* Response For a Class [26]
- *LCOM:* Lack of Cohesion in Methods [26]
- *IFANIN:* Number of Immediate Base Classes
- *NOC:* Number of Immediate Sub-classes [26]
- *DIT:* Depth of Inheritance Tree [26]
- *RPM:* Ratio of Public Methods
- *RSM:* Ratio of Static Methods
- *CLOC:* Comment of Lines per Class
- *RCC:* Ratio Comments to Codes per Class

D. Change-prone Class Identification

In this study, a class is defined as change-prone class if it is changed in the subsequent release of the system, otherwise it is not change-prone [1]. To identify change-prone classes, two consecutive releases are compared using git command[3]to check whether a class is changed in between two releases. Then, if a class has at least one change, it is called a changeprone class. Change-proneness of a class C_i is computed as a binary value between two consecutive releases r_j and r_{j+1}.

After collecting all data, five different popular and widely used machine learning algorithms such as K-nearest neighbor (KNN), Random Forest (RF), Naive Bayes (NB), Logistic Regression (LR), and Multi-layer Perceptron (MLP) are used for the prediction of change-prone classes. To evaluate the performance of these models in predicting change-prone classes, Accuracy and F1-measure are used as performance metrics. For further explanation, the pairwise correlation and variance of variables are explored using Principal Component Analysis (PCA). Univariate Logistic Regression (ULR) is used to evaluate the relationship and individual effect of the metrics.

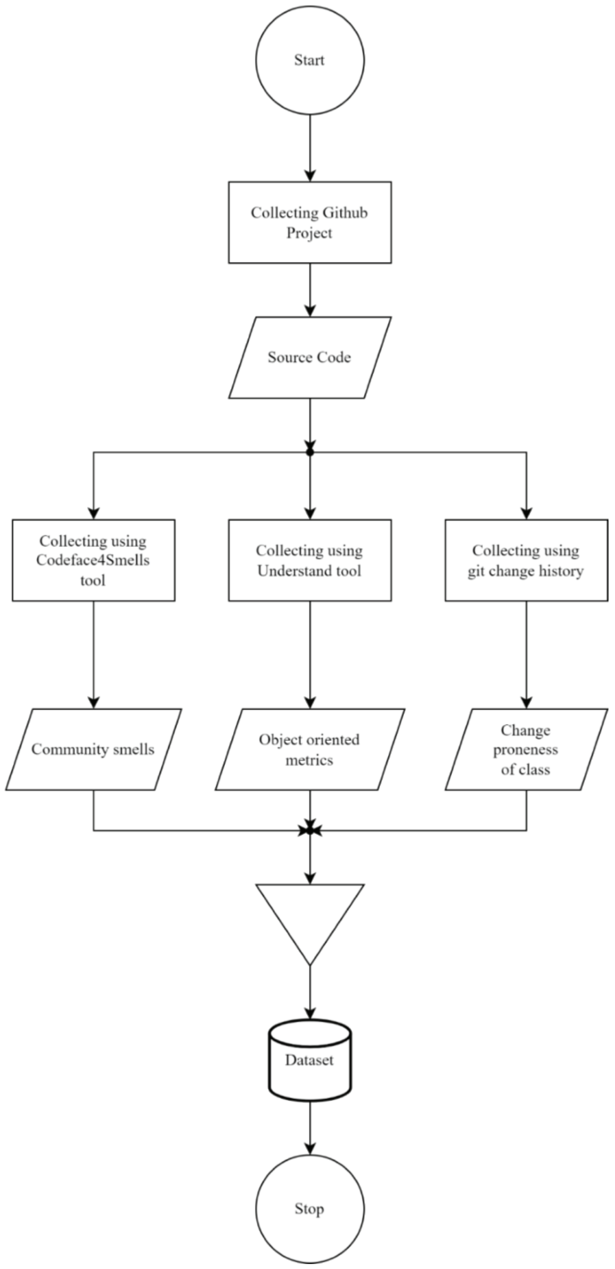
Start
Collecting Github Project
Source Code
Collecting using Codeface4Smells tool
Collecting using Understand tool
Collecting using git change history
Community smells
Object oriented metrics
Change proneness of class
Dataset
Stop

Fig. 1. Methodology of Data Collection.

In the following section (Section IV), the details of these approaches are discussed.

4. Experiment and Result Analysis

In this section, we have discussed the implementation details and the results of the selected software projects.

A. Implementation Details

In this experiment, we have analyzed five classifiers namely K-nearest neighbor (KNN) with $k = 3$, Random Forest (RF), Naive Bayes (NB), Logistic Regression (LR), and Multi-layer Perceptron (MLP) to understand the effect of

[3] git diff $r_j..r_{j+1}$–numstats

Community smell (CMS) metrics to predict change-proneness of a class

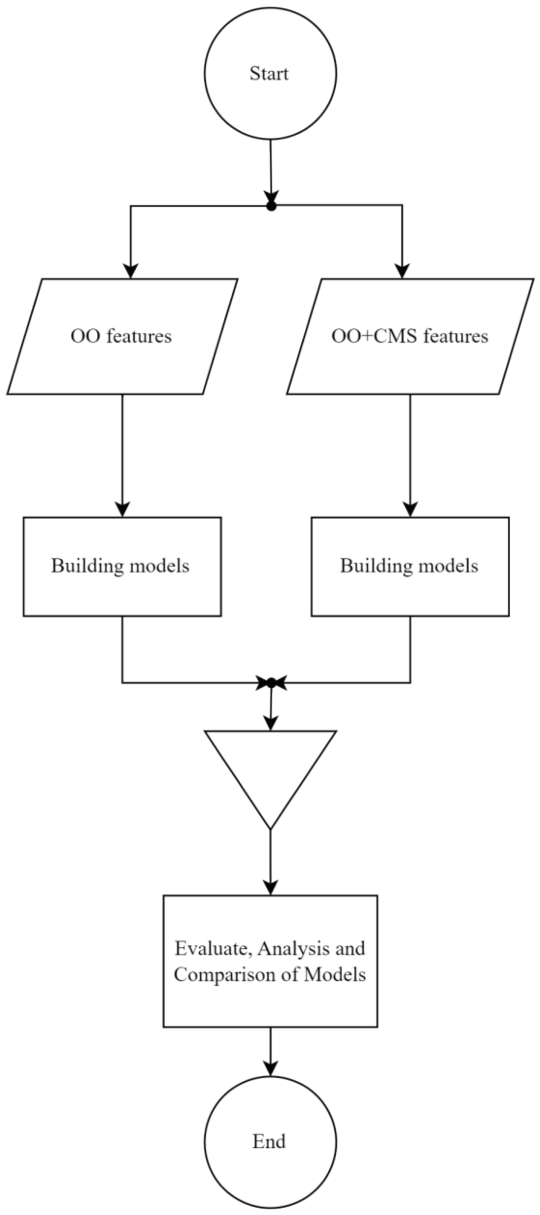

Fig. 2. Methodology of Model Building and Analysis.

incorporating with existing Object Oriented (OO) metrics. For a fair comparison of the results, we have performed a 10-fold cross-validation (10-CV) technique to evaluate the results. To measure the superiority of the classifiers, accuracy and F1measure metrics are used.

Before proceeding to the Accuracy and F1-measure [27] , we need to explain some complementary metrics such as True Positives(TP), True Negatives(TN), False Positives(FP), False Negative(FP), etc.

- *True Positives (TP):* It indicates correctly predicted positive values which means that both the value of the actual class and predicted class is positive.
- *True Negatives (TN):* It indicates correctly predicted negative values which means that both the value of the actual class and predicted class is negative.
- *False Positives (FP):* It indicates that the value of the actual class is negative and the predicted class is positive.
- *False Negatives (FN):* It indicates that the value of the actual class is positive and the predicted class is negative.

Here Accuracy and F1-measure are presented regarding explaining the model's evaluation.

- Accuracy: Accuracy is the proportion of the correctly predicted sample(CP) and total number of samples (TS).

$$\begin{aligned} Accuracy &= \frac{CP}{TS} \\ &= \frac{TP+TN}{TP+FP+FN+TN} \end{aligned} \tag{1}$$

- F1-measure: To explain uneven class distribution indicating the uneven cost of Precision and Recall, the F1measure is more suitable than the Accuracy. the Precision is the ratio of the correctly predicted positive sample and the total predicted positive sample. the Recall is the ratio of correctly predicted positive samples and the total observations in the actual positive class.

$$\begin{aligned} F1-measure &= \frac{2}{\frac{1}{Precision}+\frac{1}{Recall}} \\ &= \frac{TP}{TP+\frac{1}{2}\times(FP+FN)} \end{aligned} \tag{2}$$

B. Result and Discussion

Table 2 presents the *accuracy* results of the aforementioned five classifiers result with Object Oriented (*OO*) metrics and Object Oriented metrics along with Community Smell (*OO+CMS*) metrics. The last row of this table indicates the number of Win/Tie/Loss (W/T/L) comparing *OO+CMS* metrics with *OO* metrics results. This table shows that with *OO+CMS* metrics data LR and MLP classifiers perform better than other classification algorithms to predict the class most likely to be changed in future software releases. Overall, *OO+CMS* metrics tend to outperform *OO* metrics in terms of accuracy, with variations across datasets and algorithms. In Fig. 4, we can see the difference occurred in accuracy after incorporating the CMS metrics with OO metrics in Cayenne project.

Table 2: Accuracy Comparison Between *Oo+Cms* Metrics And *Oo* Metrics With Different Machine Learning Algorithms

Dataset	KNN		RF		NB		LR		MLP	
	OO+CMS	OO	OO+CMS	OO	OO+CMS	OO	OO+CMS	OO	OO+CMS	OO
ActiveMQ	84.86	85.15	84.58	84.50	81.34	82.40	85.99	85.84	86.08	85.94
Ant	72.54	72.80	72.75	74.08	70.91	72.15	75.83	75.64	75.68	75.63
Cassandra	77.71	79.15	78.77	79.82	74.44	69.79	77.01	71.17	77.90	72.75
Cayenne	73.17	67.59	73.76	68.57	74.48	69.55	78.81	72.53	79.27	72.63
CXF	70.56	71.56	73.08	73.44	72.56	73.14	75.27	74.99	75.57	75.28
Drill	71.08	70.98	72.12	73.07	72.35	71.63	74.05	73.07	74.24	73.51
Eclipse-CDT	86.56	85.66	85.52	85.65	83.23	84.47	87.76	87.75	*87.78*	*87.78*
Jackrabbit	87.76	88.41	87.89	88.41	83.05	83.49	88.37	87.82	88.64	88.05
Jena	85.54	85.67	87.08	86.96	82.89	84.02	88.31	87.88	88.43	87.92
Mahout	62.53	61.08	63.39	62.16	61.07	57.95	62.31	58.70	64.49	59.40
OpenNLP	76.67	78.16	74.81	79.23	74.48	73.44	79.18	76.78	79.49	76.63
Pig	82.65	81.85	82.14	81.84	78.55	81.16	83.62	83.58	83.97	83.83
POI	66.47	65.12	66.16	66.12	65.76	66.52	67.38	67.30	69.27	67.88
Tomcat	85.16	85.35	84.87	85.18	81.60	82.69	86.55	86.25	86.57	86.46
Win/Tie/Loss	6/0/8		6/0/8		5/0/9		14/0/0		13/1/0	

Table 3: F1-Measure Comparison Between *Oo+Cms* Metrics and *Oo* Metrics With Different Machine Learning Algorithms

Dataset	KNN		RF		NB		LR		MLP	
	OO+CMS	OO	OO+CMS	OO	OO+CMS	OO	OO+CMS	OO	OO+CMS	OO
ActiveMQ	82.29	82.36	82.06	81.74	80.45	80.55	80.59	80.16	81.33	80.56
Ant	70.24	71.00	69.94	71.75	68.93	69.18	67.21	66.15	67.57	66.14
Cassandra	77.02	78.09	78.26	78.94	73.74	64.62	75.45	64.03	76.49	68.66
Cayenne	72.01	65.15	72.13	65.59	73.20	65.61	75.66	62.15	76.55	64.46
CXF	68.92	69.49	70.45	70.51	71.01	70.27	69.08	68.24	70.86	69.81
Drill	69.50	69.37	70.38	70.90	68.04	65.75	69.37	66.88	70.73	68.46
Eclipse-CDT	82.55	82.17	82.00	81.96	81.78	82.28	82.33	82.31	82.32	82.25
Jackrabbit	86.17	86.84	86.30	86.81	83.81	82.33	84.84	82.53	85.65	83.07
Jena	83.46	83.00	84.31	83.51	82.57	82.14	84.16	82.45	84.23	82.43
Mahout	60.86	59.11	62.13	60.35	59.55	53.5	60.60	54.12	61.95	56.69
OpenNLP	74.31	74.46	72.71	76.80	74.46	70.92	75.55	68.24	76.17	68.57
Pig	80.20	79.87	79.94	79.58	78.24	79.48	78.33	78.16	79.30	79.00
POI	64.57	63.31	64.61	64.16	62.16	61.06	62.12	61.07	66.28	64.15
Tomcat	82.07	82.50	82.06	82.32	81.31	81.73	81.96	81.46	82.07	81.67
Win/Tie/Loss	7/0/7		7/0/7		9/0/ 5		14 /0/ 0		14/0/0	

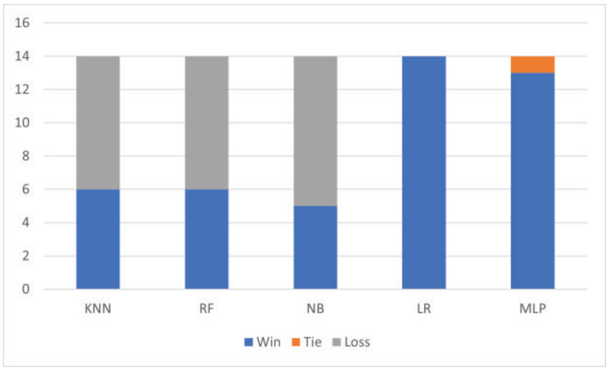

Fig. 3. Accuracy comparison between OO and OO+CMS features.

From Table 1, we can notice that the number of *classes*, *commits* and *releases* vary on different projects. Hence, we have calculated another evaluation metric *F1-measure* formulated as Eq. (2). Due to the imbalanced properties of the projects, *F1-measure* is more appropriate as an evaluation metric because it incorporates both *precision* and *recall* measures discussed in the above section. Table 3 presents the *F1-measure* results. From the last row of the table, we can see that LR and MLP classifiers outperform other classifiers' prediction performance. The reason behind this is due to the inclusion of *CMS* metrics to the existing *OO* metrics. Although NB performs well however LR and MLP perform better in predicting a class to be change-prone in future releases.

As shown in Fig. 3 and Fig. 6, non-parametric algorithms like KNN almost show balanced results (similar number of wins and losses) after inserting the community smells data.

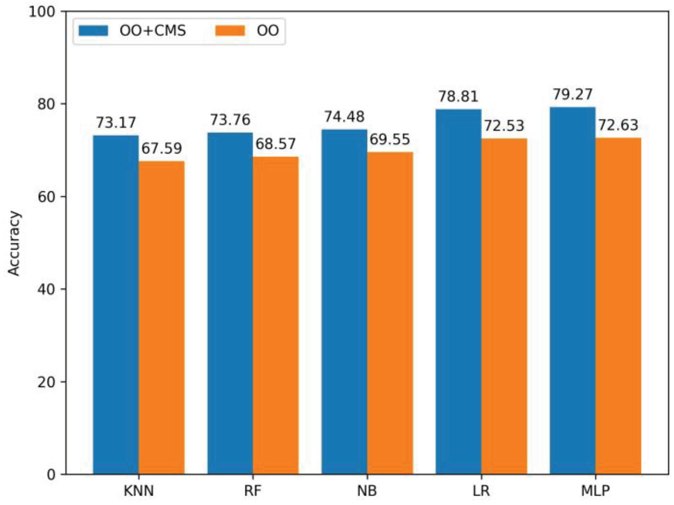

Fig. 4. Comparison of accuracy between OO and OO+CMS for different models in Cayenne project

This is likely due to the fact that non-parametric models do not consider feature importance in their models, and also because Boolean data have less significance in distance measurement. Parametric models, on the other hand, learn from the CMS features and show the significance of the CMS metrics as additional features to OO metrics. In Fig. 5, the difference is shown in F1 score after incorporating the CMS metrics with OO metrics in Cayenne project.

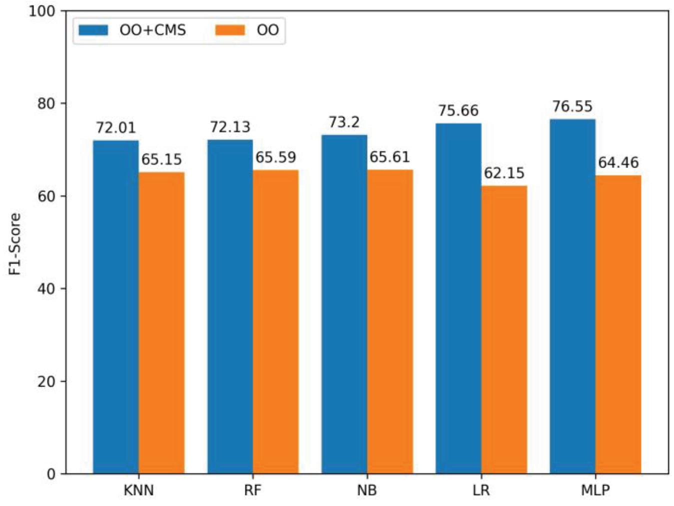

Fig. 5. Comparison of F1 score between OO and OO+CMS for different models in Cayenne project

To get better insight from the results, we have also presented the receiver operating characteristic curve (ROC) [28] shown in Fig. 7. The ROC curve is plotted by placing a false positive rate (1–specificity) on the x-axis and a true positive rate (sensitivity) on the y-axis. The area under the ROC curve (AUC) helps to visualize how well a machine learning classifier is performing. An AUC value of 0.50 indicates that the classifier has no distinguishing ability (i.e., no better than chance) and a value of 1.0 indicates perfect distinguishability. From Fig 7, we can observe that the performance of MLP and

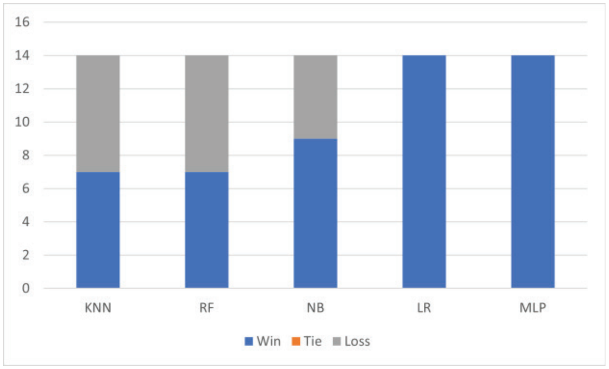

Fig. 6. F1 score comparison between OO and OO+CMS features.

LR classifiers is better than other classifiers which reflects the previously described *accuracy* and *F*1− *measures* results. Moreover, in terms of AUC score NB classifier also performs well with *CMS* metrics.

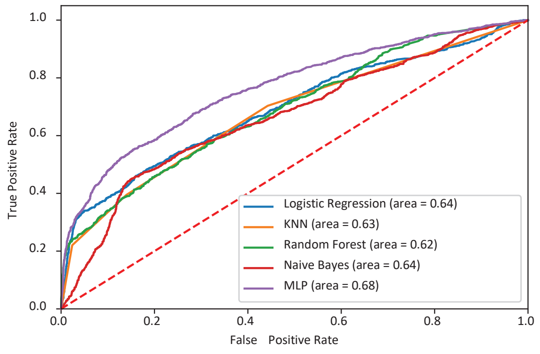

Fig. 7. AUC-ROC curve of Cayenne project

Table 4: Experimental Results of PCA For Cayenne Project

PC	Eigenvalue	Variance (%)	Cumulative	Correlated Metrics
PC-1	5.73	36.37	36.37	LOC, NOM, NIM, WMC,CBO
PC-2	1.75	11.13	47.5	OS, LW, RS
PC-3	1.59	10.11	57.61	DIT, RPM, RCC
PC-4	1.21	7.71	65.32	NIV, RFC, IFANIN
PC-5	1.09	6.91	72.23	DIT, RSM
PC-6	1.02	6.50	78.73	IFANIN, NOC, RSM

Therefore, LR and MLP classifiers achieve better performance including *CMS* metrics to the existing *OO* metrics to predict a change-prone class in future releases.

1) *Principal Component Analysis (PCA):* PCA is used for the purpose of dimensionality reduction based on variable pairwise correlation and variance for interpretation as well as further explanation [2]. PCA with 1 to 6 components is taken into consideration where the eigenvalue is larger than 1.0. Table 4 also shows the correlation between 6 principal components and the metrics. PCA provides guidance to the dimensionality reduction, metric selection, and detection of dependent variables for model building. The analysis for project *Cayenne* shows that PC-1, with the highest eigenvalue, explains 36.37% of the variance and is correlated with metrics such as LOC, NOM, NIM, WMC, and CBO. It also shows that DIT and RSM change together. Also, the community metrics OS, LW, RS are changing together. Therefore OS, LW, and RS are strongly related to PC-2. Other projects also show the same kind of properties. Therefore it can be said that the community smell metrics are strongly correlated.

2) *Univariate Logistic Regression (ULR):* The Univariate Logistic Regression analysis is conducted on our dataset to evaluate the relationship and individual effect of our metrics. The objective of this analysis is to explain the extent of variance in the dependent variable defined by the independent variables. In [2], this type of analysis proved to be significant in exploring the importance of the features. For project *Cayenne* as showed in Table 5, the value of NOM, NIV, WMC, IFANIN, NOC, and RSM do not pass the test with an alpha threshold of 0.05 which indicate their usefulness in the change-prone estimator. The community smell metrics all pass the test. Therefore NIM, CBO, CLOC, RCC, OS, LW, and

Table 5. Univariate Logistic Regression Experimental Results for Cayenne Project

Metric	Coefficient	p-value
LOC	0.0469	0.019
NOM	-0.089	0.867
NIM	0.1815	0.003
NIV	-0.0186	0.259
WMC	-0.089	0.892
CBO	0.1775	<0.01
RFC	0.0334	0.003
LCOM	-0.0456	0.005
IFANIN	0.0088	0.508
NOC	0.017	0.162
DIT	0.067	0.01
RPM	-0.0333	0.013
RSM	0.0133	0.282
CLOC	0.1135	<0.01
RCC	-0.1462	<0.01
OS	0.0952	<0.01
LW	-0.4754	<0.01
RS	0.8132	<0.01

RS show a good significance with relatively high regression coefficients which means that the mean change in the response variable is high for single unit change in the relevant dependent variable. Other projects also show the same kind of trends

5. Threats to Validity

This section discusses potential aspects that may threaten the validity of the study:

– *Threats to External Validity*: Threats to external validity deal with the generalization of the results. In this study, 317 releases of 14 open-source projects are analysed to understand the impact of including community smells related information on change-prone class prediction. To mitigate the threat of generalization, the projects having different codebase sizes (ranges from 3,476 to 21,818 commits), different age (ranges from 9 years to 18 years), and belonging to different application domains (DBMS, IDE, Web Framework, Data Analyzer etc.) are selected.

– *Threats to Internal Validity*: In this study, an opensource tool, *Codeface4Smells*, is used to detect community smells. The identified smells are directly included in the analysis of this study without further verification. However, this tool is commonly used to detect community smell in related studies [3], [22], [23], [29], [30]. This tool uses developer mailing list archives as the communication source and does not consider other communication channels, for example, Skype, Slack [31], etc. However, mailing list is the primary communication channel in the analysed communities according to contribution guidelines of evaluated projects. Moreover, object-oriented metrics are calculated using a static code analysis tool named *Understand* and the values are directly included in the dataset.

6. Conclusion

Community smells implies poor social and organizational phenomena that can lead to the emergence of social debt. Previous studies suggest that community related aspects should be considered while studying the evolution of source code. In this study, we have performed rigorous experiment on fourteen java projects with five different machine learning algorithms to evaluate the impact of community smells in predicting a class to be change-prone in future software release. The experimental results suggest that including community metrics to the the existing *OO* metrics give us better change-prone class prediction result. Moreover, MLP and LR machine learning algorithms have better prediction ability than others. Further experiment on other programming languages with more open source projects can be performed to understand the impact of community metrics to predict change-prone classes in future releases.

References

1. H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," *Empirical software engineering*, vol. 17, no. 3, pp. 200–242, 2012.

2. L. Kumar, S. K. Rath, and A. Sureka, "Empirical analysis on effectiveness of source code metrics for predicting change-proneness," in *Proceedings of the 10th Innovations in Software Engineering Conference*, pp. 4–14, 2017.

3. F. Palomba, D. Andrew Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?," *IEEE Transactions on Software Engineering*, vol. 47, no. 1, pp. 108–129, 2021.

4. B. Eken, F. Palma, B. Ays¸e, and T. Ays¸e, "An empirical study on the effect of community smells on bug prediction," *Software Quality Journal*, vol. 29, no. 1, pp. 159–194, 2021.

5. E. Caballero-Espinosa, J. C. Carver, and K. Stowers, "Community smells—the sources of social debt: A systematic literature review," *Information and Software Technology*, vol. 153, p. 107078, 2023.

6. Z. Huang, Z. Shao, G. Fan, J. Gao, Z. Zhou, K. Yang, and X. Yang, "Predicting community smells' occurrence on individual developers by sentiments," *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pp. 230–241, 2021.

7. D. A. Tamburri, P. Kruchten, P. Lago, and H. Van Vliet, "Social debt in software engineering: insights from industry," *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–17, 2015.

8. M. Di Penta, L. Cerulo, Y.-G. Gueh´ eneuc,´ and G. Antoniol, "An empirical study of the relationships between design pattern roles and class change proneness," in *2008 IEEE International Conference on Software Maintenance*, pp. 217–226, IEEE, 2008.

9. F. Khomh, M. Di Penta, Y.-G. Gueh´ eneuc,´ and G. Antoniol, "An exploratory study of the impact of antipatterns on class change-and faultproneness," *Empirical Software Engineering*, vol. 17, no. 3, pp. 243–275, 2012.

10. 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.

11. D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, "On the relation of test smells to software code quality," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12, IEEE, 2018.

12. L. Kumar, R. Jetley, and A. Sureka, "Source code metrics for programmable logic controller (plc) ladder diagram (ld) visual programming language," in *2016 IEEE/ACM 7th International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pp. 15–21, IEEE, 2016.

13. D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 5–24, 2017.

14. S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *2011 IEEE fourth international conference on software testing, verification and validation workshops*, pp. 566–571 , IEEE, 2011.

15. M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 407–437, 2013.

16. G. Catolino, F. Palomba, A. De Lucia, F. Ferrucci, and A. Zaidman, "Enhancing change prediction models using developer-related factors," *Journal of Systems and Software*, vol. 143, pp. 14–28, 2018.

17. A. E. Hassan, "Predicting faults using the complexity of code changes," in *2009 IEEE 31st international conference on software engineering*, pp. 78–88, IEEE, 2009.

18. R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "The limited impact of individual developer data on software defect prediction," *Empirical Software Engineering*, vol. 18, no. 3, pp. 478–505, 2013.

19. F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana, and R. Oliveto, "Poster: How do community smells influence code smells?," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pp. 240–241, IEEE, 2018.

20. S. Magnoni, "An approach to measure community smells in software development communities," Master's thesis, Politecnico di Milano, Italy, 2016.

21. D. A. Tamburri, F. Palomba, and R. Kazman, "Exploring community smells in open-source: An automated approach," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 630–652, 2021.

22. G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik, and F. Ferrucci, "Gender diversity and women in software teams: How do they affect community smells?," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pp. 11–20, IEEE, 2019.

23. G. Catolino, F. Palomba, D. A. Tamburri, and A. Serebrenik, "Understanding community smells variability: A statistical approach," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pp. 77–86, 2021.

24. F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan, "How does context affect the distribution of software maintainability metrics?," in *2013 IEEE International Conference on Software Maintenance*, pp. 350 – 359, IEEE, 2013.

25. B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

26. S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6 , pp. 476–493, 1994.

27. C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *European conference on information retrieval*, pp. 345–359, Springer, 2005.

28. Z. H. Hoo, J. Candlish, and D. Teare, "What is an roc curve?," *Emergency Medicine Journal*, vol. 34, no. 6, pp. 357–359, 2017.

29. F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach," *Journal of Systems and Software*, vol. 171, p. 110847, 2021.

30. G. Catolino, F. Palomba, D. A. Tamburri, and A. Serebrenik, "Understanding community smells variability: A statistical approach," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pp. 77–86, IEEE, 2021.

31. G. Voria, V. Pentangelo, A. Della Porta, S. Lambiase, G. Catolino, F. Palomba, and F. Ferrucci, "Community smell detection and refactoring in slack: The cadocs project," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 469–473, IEEE, 2022.