Journal of Information & Knowledge Management
Vol. 22, No. 4 (2023) 2350047 (18 pages)

DOI: 10.1142/S0219649223500478

World Scientific
www.worldscientific.com

SMOTE-Based Homogeneous Prediction for Aging-Related Bugs in Cloud-Oriented Software

Harguneet Kaur* and Arvinder Kaur
University School of Information Communication and Technology, GGSIPU
Dwarka, New Delhi, India
**harguneet.13316490018@ipu.ac.in*

Published 25 July 2023

Abstract. Software aging is the process caused by Aging-Related Bugs (ARBs) which leads to the depletion of resources and degradation of performance in the long run. ARBs are difficult to find and replicate in future studies as they are less in number, thus prediction of ARB is necessary to save cost and time in the testing phase. ARBs are present in low proportion as compared to non-ARBs known as the class Imbalance problem resulting in insufficient training dataset for prediction models. In this study, Synthetic Minority Oversampling Technique (SMOTE) is applied along with homogeneous cross-project ARB prediction to reduce the effect of imbalance problem in software. SMOTE is oversampling of the minority instances synthetically to balance the dataset and improve the capability of defect prediction models. Homogeneous cross-project prediction is implemented where the datasets are different but the distribution of metric sets of both training and testing datasets is similar. The experiment is conducted on five cloud-oriented software such as Cassandra, Hive, Storm, Hadoop HDFS and Hadoop Mapreduce. The novelty of this study is the combination of SMOTE and homogeneous cross-project defect prediction for ARBs in cloud-oriented software. The comparative analysis is also conducted to understand the difference between SMOTE and non-SMOTE results with the help of machine learning classifiers. The result conveys that SMOTE is an efficient method to address class imbalance problem in ARB prediction.

Keywords: SMOTE; imbalance; cross-project; aging related bug; prediction; software aging.

1. Introduction

In software aging, software tends to fail in continuous runtime due to internal changes in the system. This causes performance degradation and resource consumption which may result in severe failure or system crash. It is observed in the operating system, web server, cloud computing infrastructure, virtual machine, etc. The cause for the software aging may be memory leakage, unterminated threads, unreleased files and lock, disk fragmentation, overflow error, dereference of NULLL pointer and uninitialised memory read. The aging-related bug (ARB) occurs when the garbage collection fails to release its resources after the complete execution of the

* Corresponding author.

program which causes a hindrance in the performance of the system. In the emerging trend, cloud computing software systems are more deployed in the cloud where different applications share the platform resources.

In most of the projects, there are many non-defective classes (majority class) than defective classes (minority class) known as the class imbalance problem. Usually commonly used prediction models assume that number of minority classes are equal to majority classes which may lead to ignore minority defective instances in an imbalanced dataset. This produces partial results and it is not recommended to use the results in practice. To solve the class imbalance problem, oversampling and undersampling techniques are applied where new instances are added to the minority class in oversampling and existing instances are removed from the majority class in undersampling. In both techniques, the distribution of data is balanced to improve the performance of prediction models. According to our best knowledge, oversampling is better than undersampling because the existing samples are not removed from the majority class as it may possess useful information for prediction models. Synthetic Minority Oversampling Technique (SMOTE) is the common synthetic oversampling technique that uses K-nearest neighbour (KNN) algorithm to select proper instances to create minor defective classes synthetically. SMOTE is based on the hypothesis that classes close in distance are more similar than classes far apart. In software defect prediction (SDP), machine learning (ML) algorithms are applied to historical data for training the prediction models to predict defective instances. Historical data is prepared from software projects using the source code metrics and the label indicating whether the instance is defective or not.

In this study, ARBs are predicted in cloud-oriented software systems using a homogeneous Cross-Project Defect Prediction (CPDP) (Herbold, 2013, 2017) approach after the application of oversampling technique. ARBs are very few as compared to non-ARBs which gives rise to class imbalance problem. To remove the class imbalance problem, oversampling SMOTE technique is applied to cloud-oriented datasets for predicting ARBs. The CPDP approach is implemented because of the non-availability of suitable training datasets with very few ARBs. Cross-project defect prediction is defined as predicting the bugs with different training and testing datasets whereas both the datasets are same in Within Project Defect Prediction (WPDP) approach. Homogeneous CPDP is defined as the datasets used for prediction being different but the metrics used for it are the same. The two steps are applied in this study, first oversampling technique SMOTE on the minority instances to balance the non-aging (majority) and aging (minority)-related bugs and second, ARBs are predicted in the testing dataset with the help of different training datasets using fault prediction models. In previous studies (Malhotra and Jain, 2020; Malhotra and Lata, 2020), SMOTE is applied using with-in-project defect prediction on open-source datasets. According to our knowledge, this is the first time that SMOTE is applied to cloud-oriented dataset using the homogeneous CPDP approach to predict ARBs. The experiment is conducted on five cloud-oriented software for comparing the performance of predicting ARBs on the application with

SMOTE and without SMOTE using ML classifiers. It has been observed that on applying SMOTE to the training dataset, the results are improved when compared with non-SMOTE results.

The rest of the paper is structured as follows. Section 2 discusses the related work on ARB prediction and Sec. 3 describes the framework proposed for predicting ARBs in cloud-oriented software. Section 4 explains the research methodology including aging-related keywords, software metrics, datasets and ML algorithms used in this study. This section also discussed performance measures and experimental setup. Section 5 presents the results and analysis thoroughly. Conclusion, threats to validity and future directions are reported in Sec. 6.

2. Related Work

Software aging is the process of accumulating run-time errors known as ARBs in the software resulting in performance degradation and resource consumption and can lead to system crash. ARBs are difficult to reproduce in the dataset due to their low proportion as compared to non-ARBs. Therefore, ARBs are said to be imbalanced in nature. This imbalance problem has to be eliminated with the help of imbalance mitigation procedures before predicting the ARBs. Feng *et al.* (2021) proposed a novel technique called Complexity-based OverSampling Technique (COSTE) to remove the class imbalance problem in SDP. The synthetic instances are created based on complexity for oversampling. Malhotra and Jain (2020) worked on oversampling techniques of ensemble methods to predict software defects using an imbalanced dataset on open-source java projects. Xu *et al.* (2020) proposed cross-project ARB prediction to reduce the marginal and conditional distribution difference jointly between the source and target project by eliminating the class imbalance problem. Chouhan and Rathore (2021) presented the oversampling technique which uses generative adversarial networks to create synthetic samples for minority class ARB patterns and performed software aging bug prediction. Qin *et al.* (2018) proposed a transfer learning-based aging-related bug prediction (TLAP) approach to reduce class imbalance problem and performed cross-project prediction to get a sufficient training dataset. In this study, cross-project ARB prediction is implemented on cloud-oriented software after using an imbalance mitigation procedure-SMOTE.

In a nutshell, the novelty of the work is divided into two sections: (1) Application of oversampling technique-SMOTE on cloud-oriented dataset to balance the ARBs and non-ARBs. (2) The CPDP approach is executed and the comparison is performed between SMOTE and non-SMOTE results using different classifiers of prediction. Wu *et al.* (2020) and Kumar and Sureka (2018) sorted the ARBs manually for prediction but in this study automatically extracted ARBs are used with reference to our previous study (Kaur and Kaur, 2022). Kumar and Sureka (2018) stated that ARBs can be predicted with the help of source code metrics therefore we have

used this application in our study and collected static code metrics for cloud-oriented dataset. Software aging was introduced systematically by Huang in 1995.

Cotroneo *et al.* (2010, 2013) empirically predicted aging-related metrics with the help of code metrics but did not balance the dataset using oversampling technique-SMOTE and the recent research is mostly on static open-source available dataset-LINUX and MySQL. The novelty of this study is that it performed ARB prediction on automatically extracted dataset from cloud-oriented software as Ficco *et al.* (2018) also predicted ARBs in real time data applications. Qin *et al.* (2015, 2018, 2020) experimented for the prediction of ARB using transfer learning approach on Linux and MySQL but not on cloud-oriented software (Andrade *et al.*, 2021). According to recent studies (Xu *et al.*, 2020), the dataset collection for the prediction is evaluated manually (Parri *et al.*, 2021) but in this study, automatic extraction of the dataset is performed with the oversampling technique i.e. SMOTE and then the CPDP approach is followed for prediction of ARBs. Earlier the research has been performed on open-source available datasets but this study collected cloud-oriented software and predicted ARBs using CPDP approach after the application of SMOTE oversampling technique.

3. Imbalance Mitigation Framework for ARB

ARB prediction of cloud-oriented software undergoes three phases: Automatic extraction of ARB reports using the keywords; applying SMOTE as an imbalance mitigation procedure to remove the class imbalance problem and comparing CPDP and WPDP approach for ARB prediction in open-source cloud-oriented software. It is aimed to design the Framework shown in Fig. 1 which has been divided into four modules for ARB prediction: ARB report collection through automata (M1), Applying SMOTE on cloud-oriented software (M2), performing CPDP of cloud-oriented software and WPDP in cloud-oriented dataset (M3), comparing SMOTE with non-SMOTE results for ARBs (M4). The bug reports collected in this study are CLOSED, RESOLVED and unique (not duplicates of other reports). These are collected from the Jira repository through REST API. Then ARB reports are refined

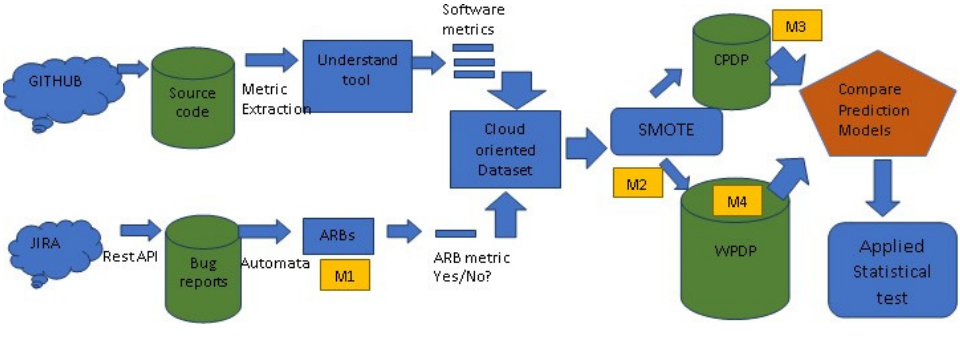

Fig. 1. Imbalance mitigation framework for ARB prediction.

from other reports by reading the description of bug reports and whether aging-related keywords are present or not. These ARB bugs are mapped to the corresponding affected java files which are labelled as ARB-prone files and non-affected java files are marked as non-ARB files. The source code is extracted from GitHub repository and then the metric extraction is performed through Understand tool (Cotroneo *et al.*, 2013). Now the dataset is formed for the prediction purpose on summing up the source code metrics and the labelled ARB metric Yes/No for each java source file. It has been observed that ARB-prone files are found to be very less in number as compared to non-ARB files which give rise to a class imbalance problem. The next step is applying the imbalance mitigation procedure as SMOTE on the dataset which reduces the class imbalance problem so that prediction can be performed effectively. CPDP and WPDP are implemented using ML algorithms for predicting ARBs on cloud-oriented datasets. The prediction models are compared based on performance measures to identify which classifier performed better for predicting ARBs in cloud-oriented datasets. The comparative study is done for cross-project defect prediction on the basis of SMOTE and non-SMOTE applications of ARBs. To perform ARB prediction, the following research questions are formulated.

RQ1. *Does SMOTE contribute to the reduction of the class imbalance problem*?

SMOTE Chawla *et al.* (2002) is the commonly used synthetic oversampling technique in SDP. It works based on KNN algorithm where new synthetic instances are generated by combining the minority class instance at its KNN. The basic aim of using KNN algorithm in SMOTE is that it guarantees the location of newly generated instances in the area of minority class. However, to evaluate whether SMOTE improves the class imbalance problem in a cloud-oriented dataset, experiments are conducted to compare the results before and after applying SMOTE.

RQ2. *Which ML algorithm suits best with SMOTE to predict ARBs*?

Software aging is the class imbalance problem where predicted class i.e. aging-prone files are less in number as compared to aging-free files. When data is imbalanced, it becomes difficult to predict bugs due to insufficient training dataset. Therefore in this study, SMOTE is first applied to the training dataset and then ML algorithms are applied on the dataset. The evaluation is done with the help of performance measures that which ML classifier performs better with the application of SMOTE. The novelty in this study is that SMOTE is applied with the CPDP approach in cloud-oriented software to predict ARBs.

4. Research Methodology

4.1. *Aging-related keywords*

Software aging is the phenomenon where the performance of software decreases due to the accumulation of run-time errors in the long run. These errors may be caused

due to the overconsumption of resources like file descriptors, database connections, unreleased files and locks. When the limit of the resources extends, it causes memory leakage or deadlock state where software can even crash. Thus it has been observed that software aging is caused due to the memory leakage, deadlock state, overflow, null pointer exception, divide by zero error, etc. which are classified as aging-related keywords mentioned in Table 1. The bug reports which are collected from the REST API of all the cloud-oriented datasets are analysed through the SEARCH KEYWORD algorithm implemented in our previous work (Kaur and Kaur, 2022) for the collection of ARB reports. For the reference it is shown in Algorithm 1.

4.2. *Software metrics*

SDP usually predicts the defects in the dataset with the help of software metrics. These metrics (Cotroneo *et al.*, 2010) are the measure of software performance, productivity, planning and many other software applications. This study aims to

Table 1. Keywords.

Keywords
"'race' 'leak' memory aging overflow deplet Overflow NPE 'null pointer' 'Buffer exhausted' 'deadlock' 'flush' 'Leak' Memory LEAK 'MEMORY' 'OVERFLOW'

Algorithm 1. Search_Keyword

Input: The Excel File is imported in which the Bug Id, Bug description and Bug key of all the bug reports of a Dataset

1. Excel File is converted into list 'AL_BUG' holding the information of Bug Description, Bug Id and Bug Key for all the bugs
2. Create New List of Aging Related Keywords 'ARK_LST'-'null pointer' 'race' 'memory' 'deadlock' 'leak' 'Overflow' 'Buffer exhausted' 'NPE' with all the possibilities of capitalisation of the word.
3. **while** *Check if mentioned keywords 'ARK_LST' are present in the description of each bug report in AL_BUG list* **do**
   A new list 'FIN_AR' is created of only aging related bug reports;
4. Duplicates are removed in FIN_AR if any aging related bug is repeated using Set conversion.
5. Convert FIN_AR into dataframe and then export it to excel file AR_FILE.
6. Repeat this process for each dataset.

Output: AR_FILE is exported retaining aging related bug reports

predict aging faults in cloud-oriented software with the help of static code metrics. The metrics are extracted from the datasets using the Understand tool (Cotroneo *et al.*, 2013). It has been proved that ARBs are directly related to software metrics (Cotroneo *et al.*, 2010). The two categories of metrics are extracted in this paper tabulated in Table 2: Program size and McCabe's cyclomatic complexity. Program size measures the size of the program and McCabe's cyclomatic complexity measures the complexity of the program which controls the number of paths a program can go through.

4.3. *Datasets*

Cloud-oriented software are open for the new opportunities to access the different tools and services over internet. They are majorly applicable for data storage and data access for executing various applications using shared resources. These applications run for a long time therefore aging is observed in these software. Aging is caused due to memory leak, deadlock, null pointer exception, etc., in nutshell if the resources get exhausted or misused then there can be deadlock and memory leak. For our implementation, we have collected five available open-source cloud-oriented

Table 2. Software metrics.

S. no.	Metrics	Type
1	AvgCyclomatic, AvgCyclomaticModified, AvgCyclomaticStrict, AvgEssential, Cyclomatic, CyclomaticModified, CyclomaticStrict, Essential, MaxCyclomatic, MaxCyclomaticModified, MaxCyclomaticStrict, MaxEssential, SumCyclomatic, SumCyclomaticModified, SumCyclomaticStrict, SumEssential	McCabe's cyclomatic complexity
2	AvgLine, AvgLineBlank, AvgLineCode, AvgLineComment, CountClassBase, CountClassDerived, CountDeclClass, CountDeclClassMethod, CountDeclClassVariable, CountDeclExecutableUnit, CountDeclFile, CountDeclFunction, CountDeclInstanceMethod, CountDeclInstanceVariable, CountDeclInstanceVariablePrivate, CountDeclInstanceVariableProtected, CountDeclInstanceVariablePublic, CountDeclMethod, CountDeclMethodAll, CountDeclMethodDefault, CountDeclMethodPrivate, CountDeclMethodProtected, CountDeclMethodPublic, CountInput, CountLine, CountLine Html, CountLine Javascript, CountLine Php, CountLineBlank, CountLineBlank Html, CountLineBlank Javascript, CountLineBlank Php, CountLineCode, CountLineCode Javascript, CountLineCode Php, CountLineCodeDecl, CountLineCodeExe, CountLineComment, CountLineComment Html, CountLineComment Javascript, CountLineComment Php, CountOutput, CountPath, CountPathLog, CountSemicolon, CountStmt, CountStmtDecl, CountStmtDecl Javascript, CountStmtDecl Php, CountStmtExe, CountStmtExe JavascriptCountStmtExe Php, MaxNesting, RatioCommentToCode	Program Size

applications: Hadoop Mapreduce, Hadoop HDFS, Cassandra, Hive and Storm. These software contain thousands of java files as the cloud-oriented applications are large in size as compared to standard available datasets like: MYSQL and LINUX. The experiments are conducted on these datasets after extracting the same from GitHub — www.github.com. The java files are extracted in the zip folder and then metrics are drawn from it using Understand tool. Using the aging-related keywords, bug reports are categorised into two types: ARB reports and non-ARB reports. The version, number of files, ARB-prone files and percentage of ARB-prone files for each dataset are given in Table 3.

4.4. *Machine learning classifiers*

SMOTE — The class imbalance problem is generally faced by every software engineering practitioner nowadays which greatly reduces the performance of fault prediction models in SDP. An imbalanced dataset indicates when non-predicting class instances are more than predicting class instances which create bias result that tends to ignore buggy or defective classes. To remove this problem in SDP, many class imbalance techniques are being proposed out of which oversampling and undersampling are two methods usually practiced. Oversampling is preferred over undersampling as undersampling removes some instances which may contain important information. SMOTE is the oversampling technique in which synthetic minority instances are generated on the basis of feature space i.e. KNN algorithm. KNN algorithm is based on supervised learning which considers the similarity between new data and available data and keep the new data into the similar available data. Here, "new data" refers to synthetically created minority instances where as "available data" refers to already available minority instances. KNN is a non-parametric which means it does not make any assumptions on underlying data. It has been seen that SMOTE is applied on training dataset so that ML classifier fits into the data. The testing dataset remains unchanged so that it correctly represents the original data. The ML classifiers (Wan *et al.*, 2019) given below are considered in this study Xu *et al.* (2020) as these classifiers predict the minority class (buggy class) effectively in bug prediction. There are more efficient ML classifiers but for the

Table 3. Dataset description.

Dataset	Version	ARBs	Files	ARB-prone files	% ARB-prone files
Cassandra	3	601	2764	385	13.93
Hadoop Mapreduce	0.23.0	192	1412	161	11.4
Hadoop HDFS	0.20.0	377	521	170	32.63
Hive	3.1.0	517	7112	367	5.16
Storm	2.3	82	2371	131	5.53

prediction of ARBs, the following are the effective classifiers referred to the recent research (Wan *et al.*, 2019).

4.4.1. *Bagging*

Bagging is the application of decision tree ML algorithm where ensemble learning is employed for more robust prediction results. Ensemble learning is a ML algorithm where different models are trained to solve the problem and combined to implement prediction. It is a meta estimator that reduces the variance of decision tree by initiating randomization in its process and develops an ensemble of it. Each classifier is trained on row sampling with replacement technique such that size of each sample equals or is less than the size of actual training dataset. Then the voting algorithm is employed where the output of all the models are combined.

4.4.2. *Adaboost*

Adaboost is an adaptive boosting technique where each row is assigned sampled weight and the base learners are assigned sequentially. The base learners are decision trees with depth equal to 1 i.e. the stumps are used instead of trees. It functions by assigning more weight on the wrongly classified instances and less weight on correctly classified instances. Combination of all the weak classifiers i.e. stumps make the strong classifier which is the idea behind the Adaboost algorithm.

4.4.3. *Logistic regression*

Logistic regression is a supervised classification algorithm where y variable (output) takes discrete values for a given set of features x (input). It is similar to linear regression, the only difference is that it uses sigmoid function. It is an S-shaped curve that takes values between 0 and 1 and growth is exponential in number. Logistic regression helps in predicting the probability of aging-prone files.

4.4.4. *Naive Bayes*

Naive Bayes is a supervised learning algorithm based on Bayes theorem which belongs to the family of probabilistic classifier. It works on Bayesian network model which achieves high accuracy and is highly scalable which requires parameters to be linear. It basically uses the approach of "maximum likelihood" and the conditional probability method. The main advantage of Naive Bayes is it can work on small data also for prediction model.

4.4.5. *LogitBoost*

Logitboost is also the boosting algorithm similar to Adaboost algorithm which performs adaptive logistic regression. The difference lies in the loss i.e. Adaboost minimises the exponential loss whereas Logitboost minimises the logistic loss.

4.4.6. *Random forest*

It is also an ensemble technique where multiple decision trees are implemented for modelling. Each decision tree considers row sampling and feature sampling where the sample data is always less than the original data. Then every decision tree is trained on sampled data and gives the output as a prediction. It uses bootstrap aggregator algorithm where the prediction performance of testing data is dependent on maximum voting. Random forest works very well among other ML classifiers as it creates low variance i.e. less output errors that can make high accuracy results. If a single decision tree is taken then it creates low bias and high variance but if the data is trained on multiple decision trees then the output produces high bias and low variance.

4.4.7. *J48*

It is the widely used ML algorithm which examines the data continuously and categorically. J48 is the decision tree algorithm based on Iterative Dichotomiser 3. It is also known as C4.5 algorithm which generates decision tree and includes other features also such as derivation of rules, account for missing values and decision tree pruning.

4.4.8. *Decision table*

Decision table is similar to decision tree classification model used for the prediction of bugs in the dataset. It is a hierarchical table which is similar to dimensional stacking.

4.5. *Performance measures*

Software aging is proved to be an imbalance problem where ARB files are less in comparison to non-ARB files. As per the recent research, accuracy, F-measure and precision are not used in imbalanced datasets therefore Recall, False Positive Rate (FPR) and Balance (Bal) are evaluated and compared in our study (Qin *et al.*, 2020). To have a clear vision of performance measures it is necessary to understand the confusion matrix first shown in Table 4. True Positive (TP) represents if the predicted label and actual label both are positive whereas True Negative (TN) represents if predicted and actual both are negative. False Negative (FN) defines when the predicted label is negative and actual label is positive i.e. if the instance is

Table 4. Confusion matrix.

		Prediction class	
		ARB-free	ARB-prone
Actual Class	ARB-free	TN	FP
	ARB-prone	FN	TP

not correctly predicted. False Positive (FP) is if the predicted label is positive and actual label is negative.

Recall is generally used when it is required to measure the false negative. It is the proportion of actual positive which is predicted correctly as positive.

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})}. \tag{1}$$

FPR measures the negative labels which are wrongly identified as positive labels from negative classes.

$$\text{FPR} = \frac{\text{FP}}{(\text{FP} + \text{TN})}. \tag{2}$$

Balance (Bal) — Bal is a tradeoff between recall and FPR as when recall increases FPR decreases therefore a measure is required which balances both Recall and FPR.

$$\text{Bal} = 100 - \frac{\sqrt{(0 - \text{FPR})^2 + (100 - \text{Recall})^2}}{\sqrt{2}}. \tag{3}$$

4.6. *Experimental setup*

In this study, five cloud-oriented datasets are taken from GitHub. The experiment is performed on five large cloud-oriented software which are available open-source at GitHub. The source code metrics are then extracted from the zip folder of these datasets using Understand tool (Cotroneo *et al.*, 2013). We have used java projects as our tool extracts the metrics from java files. Then these metrics are stored in an excel file for each dataset. The metrics for each instance/java class file for a dataset are used for the prediction of ARBs. The next step is the collection of bug data from the bug repository. Each dataset contains thousands of bug reports which are not possible to be extracted on a click. Therefore, REST API is used from JIRA repository where 1,000 bug reports are collected at a time. Then changing the configuration of the API, we can download as many thousand bug reports from the repository. Now the bug reports are also available in an excel file. In earlier research, bug reports are manually studied and found out which are aging-related by reading its description. But in our earlier research (Kaur and Kaur, 2022) we have implemented the algorithm to sort ARB reports from the bug data in a fraction of seconds. This algorithm transfers the manual task to the automata process for the collection of dataset. After the collection of ARB reports, it is observed that these bug reports are less in number as compared to other bug reports. Then it can be declared that software aging is the class imbalance problem. If the prediction is performed on an imbalanced dataset then it may produce unreliable results which cannot be further used. Therefore, we have applied SMOTE oversampling technique which oversamples the minority instances and balances the minority with the majority instances. The aging bug reports are mapped with the effected java class files in the

metrics data i.e. now each instance/java file has one metric more — ARB metric. If the java file is affected with aging then its ARB metric is marked with "1" otherwise "0". Now the prediction is performed using CPDP and WPDP approach with the help of WEKA tool (Markov and Russell, 2006). In CPDP, training and testing datasets are different whereas in WPDP both the datasets are same. We have implemented eight ML classifiers to build fault prediction models. Three performance measures are used to evaluate the results of prediction for ARBs in cloud-oriented software. Thus there are $5 * 5 = 25$ pairwise cases altogether for within and cross-project prediction approach. We have applied 8 classifiers, calculated three performance measures on five large datasets thus this produces $8 * 3 * 25 = 600$ results which are fair enough to predict ARB. SMOTE is applied on training dataset to reduce class imbalance problem and thus SMOTE and non-SMOTE results are compared on the basis of different performance measures to build fault prediction models for cloud-oriented software.

For the complete understanding of SMOTE on cloud-oriented software for ARB prediction, several experiments are conducted. First, we investigated whether SMOTE reduces the class imbalance problem or not on the basis of Recall value to answer the research question RQ1.The results of SMOTE and non-SMOTE are compared by applying ML algorithms. For each experiment, Recall value and other performance measures are calculated which is considered as a comparable measure. In this process, 25 experiments are conducted for cross-project ARB prediction on cloud-oriented datasets. In addition, other performance measures are also evaluated like FPR and Balance to gain the confidence in the results of ARB prediction. Recall should be high and FPR must be low for building accurate ARB prediction models with the oversampled datasets. Before applying ML classifier, SMOTE is applied on training datatset to train the learners and then other testing datasets are supplied for predicting ARBs. Thus in this study we have achieved cross-project bug prediction on cloud-oriented software. In CPDP, different datasets are applied for training and testing to build prediction models as to validate the model's performance.

5. Results and Analysis

In this section, experimental results are discussed to answer the research questions by comparing the performance of SMOTE with non-SMOTE technique across each performance measure by applying ML classifiers using CPDP and WPDP approach.

RQ1. *Does SMOTE contribute in the reduction of class imbalance problem*?

To answer RQ1, the results of SMOTE are compared with non-SMOTE on the basis of Recall performance measure. SMOTE is the oversampling technique which generates synthetic instances on the basis of K-NN algorithm. SMOTE reduces the class imbalance problem where instances of predicting class are minor than instances of non-predicting class. Table 5 records the recall performance measure for combination

Table 5. Comparison of SMOTE with non-SMOTE for ARB prediction on basis of Recall value.

Training-testing dataset	S-Naïve bayes	Naïve bayes	S-Logistic regression	Logistic regression	SMO	S-Decision table	Decision table	S-Random forest	Random forest	S-J48	J48	S-Bagging	Bagging	S-Logit boost	S-Ada boost
Cassandra–Cassandra	0.38	0.36	0.28	0.14	0.00	0.46	0.11	0.58	0.14	0.58	0.25	0.54	0.16	0.43	0.45
Cassandra–Hadoop Mapreduce	0.28	0.28	0.20	0.08	0.00	0.09	0.11	0.19	0.12	0.26	0.24	0.27	0.16	0.34	0.35
Cassandra–Hadoop HDFS	0.27	0.26	0.18	0.08	0.00	0.09	0.09	0.16	0.11	0.34	0.29	0.33	0.13	0.37	0.43
Cassandra–Hive	0.41	0.42	0.28	0.16	0.00	0.14	0.15	0.23	0.18	0.35	0.26	0.32	0.23	0.40	0.44
Cassandra–Storm	0.31	0.28	0.20	0.10	0.00	0.08	0.18	0.22	0.11	0.38	0.20	0.29	0.15	0.34	0.39
Hadoop Mapreduce Cassandra	0.38	0.36	0.26	0.19	0.00	0.05	0.10	0.10	0.07	0.21	0.17	0.12	0.06	0.15	0.04
Hadoop Mapreduce–Hadoop Mapreduce	0.29	0.27	0.23	0.12	0.01	0.26	0.08	0.99	0.21	0.72	0.24	0.53	0.21	0.36	0.26
Hadoop Mapreduce Hadoop HDFS	0.28	0.26	0.15	0.10	0.01	0.13	0.06	0.12	0.08	0.31	0.17	0.15	0.08	0.19	0.11
Hadoop Mapreduce Hive	0.43	0.42	0.23	0.16	0.03	0.08	0.13	0.14	0.11	0.24	0.16	0.18	0.10	0.20	0.08
Hadoop Mapreduce Storm	0.30	0.27	0.12	0.07	0.02	0.05	0.09	0.08	0.05	0.21	0.13	0.14	0.05	0.12	0.03
Hadoop HDFSCassandra	0.68	0.47	0.74	0.58	0.08	0.34	0.43	0.48	0.40	0.37	0.47	0.48	0.45	0.41	0.56
Hadoop HDFS-Hadoop Mapreduce	0.75	0.42	0.71	0.31	0.12	0.49	0.31	0.46	0.36	0.42	0.42	0.51	0.41	0.69	0.71
Hadoop HDFS-Hadoop HDFS	0.82	0.56	0.85	0.39	0.14	0.70	0.42	1.00	0.51	0.91	0.52	0.91	0.50	0.80	0.79
Hadoop HDFS-Hive	0.70	0.48	0.74	0.44	0.19	0.65	0.46	0.51	0.40	0.46	0.41	0.56	0.46	0.65	0.63
Hadoop HDFS-Storm	0.75	0.43	0.74	0.36	0.10	0.57	0.32	0.42	0.32	0.28	0.31	0.47	0.37	0.60	0.57
Hive-Cassandra	0.14	0.10	0.19	0.01	0.00	0.00	0.01	0.04	0.01	0.09	0.02	0.08	0.00	0.12	0.26
Hive-Hadoop Mapreduce	0.11	0.11	0.18	0.03	0.00	0.01	0.02	0.03	0.00	0.10	0.05	0.01	0.01	0.13	0.28
Hive-Hadoop HDFS	0.14	0.16	0.04	0.01	0.00	0.00	0.03	0.06	0.00	0.12	0.04	0.08	0.01	0.10	0.27
Hive–Hive	0.23	0.21	0.17	0.05	0.00	0.01	0.02	0.99	0.06	0.64	0.10	0.39	0.03	0.15	0.12
Hive-Storm	0.12	0.11	0.11	0.01	0.00	0.00	0.02	0.02	0.01	0.11	0.03	0.04	0.01	0.06	0.07
Storm-Cassandra	0.21	0.17	0.29	0.32	0.00	0.05	0.16	0.16	0.08	0.27	0.19	0.22	0.09	0.28	0.30
Storm-Hadoop Mapreduce	0.19	0.16	0.25	0.17	0.00	0.06	0.14	0.19	0.10	0.22	0.18	0.21	0.14	0.26	0.38
Storm-Hadoop HDFS	0.18	0.17	0.34	0.20	0.00	0.05	0.14	0.20	0.11	0.23	0.20	0.25	0.15	0.32	0.39
Storm-Hive	0.26	0.22	0.28	0.20	0.00	0.06	0.18	0.20	0.14	0.22	0.20	0.24	0.18	0.27	0.28
Storm–Storm	0.19	0.18	0.37	0.19	0.00	0.11	0.15	1.00	0.12	0.62	0.14	0.45	0.09	0.32	0.29

of all the datasets using CPDP and WPDP approach where S denotes the Smote. For example "S-Naive Bayes" means Naive Bayes applied with Smote and "Naive Bayes" denotes Naive Bayes applied without Smote. It can be clearly seen in all the experiments, for example: SMOTE J48 achieves the recall value (0.72), while non-SMOTE J48 covers 0.24 in Hadoop Mapreduce dataset. We have compared the recall value for each dataset by applying different ML algorithms. Higher the recall value, better will be the results and also means more accurately the ARBs are predicted. To conclude the above observation, the diversity of SMOTE is better than non-SMOTE based upon recall values. This has confirmed the expectation and the answer to RQ1 is "Yes" that SMOTE reduces the class imbalance problem by oversampling the instances of minor classes.

RQ2. *Which ML algorithm suits best with SMOTE to predict ARBs*?

Table 6. Results of SMOTE on the basis of Recall performance measure.

Training-testing dataset	SMOTE-bagging	SMOTE-adaboost	SMOTE-logistic	SMOTE-Naïve bayes	SMOTE-logit boost	SMOTE-random forest	SMOTE-J48	SMOTE-decision table
Cassandra–Cassandra	0.54	0.45	0.28	0.38	0.43	0.58	0.58	0.46
Cassandra-Hadoop Mapreduce	0.27	0.35	0.20	0.28	0.34	0.19	0.26	0.09
Cassandra-Hadoop HDFS	0.33	0.43	0.18	0.27	0.37	0.16	0.34	0.09
Cassandra-Hive	0.32	0.44	0.28	0.41	0.40	0.23	0.35	0.14
Cassandra-Storm	0.29	0.39	0.20	0.31	0.34	0.22	0.38	0.08
Hadoop Mapreduce Cassandra	0.12	0.04	0.26	0.38	0.15	0.10	0.21	0.05
Hadoop Mapreduce–Hadoop Mapreduce	0.53	0.26	0.23	0.29	0.36	0.99	0.72	0.26
Hadoop Mapreduce Hadoop HDFS	0.15	0.11	0.15	0.28	0.19	0.12	0.31	0.13
Hadoop Mapreduce Hive	0.18	0.08	0.23	0.43	0.20	0.14	0.24	0.08
Hadoop Mapreduce Storm	0.14	0.03	0.12	0.30	0.12	0.08	0.21	0.05
Hadoop HDFS-Cassandra	0.48	0.56	0.74	0.68	0.41	0.48	0.37	0.34
Hadoop HDFS-Hadoop Mapreduce	0.51	0.71	0.71	0.75	0.69	0.46	0.42	0.49
Hadoop HDFS-Hadoop HDFS	0.91	0.79	0.85	0.82	0.80	1.00	0.91	0.70
Hadoop HDFS-Hive	0.56	0.63	0.74	0.70	0.65	0.51	0.46	0.65
Hadoop HDFS-Storm	0.47	0.57	0.74	0.75	0.60	0.42	0.28	0.57
Hive-Cassandra	0.08	0.26	0.19	0.14	0.12	0.04	0.09	0.00
Hive-Hadoop Mapreduce	0.01	0.28	0.18	0.11	0.13	0.03	0.10	0.01
Hive-Hadoop HDFS	0.08	0.27	0.04	0.14	0.10	0.06	0.12	0.00
Hive–Hive	0.39	0.12	0.17	0.23	0.15	0.99	0.64	0.01
Hive-Storm	0.04	0.07	0.11	0.12	0.06	0.02	0.11	0.00
Storm-Cassandra	0.22	0.30	0.29	0.21	0.28	0.16	0.27	0.05
Storm-Hadoop Mapreduce	0.21	0.38	0.25	0.19	0.26	0.19	0.22	0.06
Storm-Hadoop HDFS	0.25	0.39	0.34	0.18	0.32	0.20	0.23	0.05
Storm-Hive	0.24	0.28	0.28	0.26	0.27	0.20	0.22	0.06
Storm–Storm	0.45	0.29	0.37	0.19	0.32	1.00	0.62	0.11

Table 7. Results of SMOTE on the basis of FPR performance measure.

Training-testing dataset	SMOTE-bagging	SMOTE-adaboost	SMOTE-logistic	SMOTE-Naïve bayes	SMOTE-logit boost	SMOTE-random forest	SMOTE-J48	SMOTE-decision table
Cassandra–Cassandra	0.05	0.08	0.05	0.09	0.07	0.04	0.11	0.02
Cassandra-Hadoop Mapreduce	0.08	0.13	0.04	0.09	0.11	0.05	0.15	0.03
Cassandra-Hadoop HDFS	0.07	0.09	0.02	0.05	0.07	0.03	0.12	0.02
Cassandra-Hive	0.10	0.15	0.05	0.15	0.14	0.07	0.15	0.04
Cassandra-Storm	0.04	0.06	0.02	0.07	0.05	0.03	0.09	0.02
Hadoop Mapreduce Cassandra	0.02	0.01	0.07	0.10	0.03	0.02	0.07	0.01
Hadoop Mapreduce–Hadoop Mapreduce	0.01	0.02	0.03	0.09	0.04	0.00	0.01	0.01
Hadoop Mapreduce Hadoop HDFS	0.02	0.01	0.03	0.05	0.03	0.02	0.09	0.01
Hadoop Mapreduce Hive	0.03	0.01	0.05	0.15	0.04	0.02	0.09	0.01
Hadoop Mapreduce Storm	0.01	0.00	0.02	0.08	0.01	0.01	0.07	0.00
Hadoop HDFS-Cassandra	0.17	0.19	0.37	0.40	0.15	0.16	0.24	0.13
Hadoop HDFS-Hadoop Mapreduce	0.28	0.37	0.48	0.59	0.39	0.21	0.29	0.37
Hadoop HDFS-Hadoop HDFS	0.11	0.33	0.45	0.63	0.36	0.00	0.05	0.21
Hadoop HDFS-Hive	0.26	0.33	0.47	0.57	0.37	0.22	0.27	0.38
Hadoop HDFS-Storm	0.16	0.19	0.40	0.44	0.22	0.12	0.17	0.26
Hive-Cassandra	0.01	0.07	0.03	0.02	0.03	0.01	0.03	0.00
Hive-Hadoop Mapreduce	0.00	0.14	0.04	0.02	0.06	0.01	0.04	0.00
Hive-Hadoop HDFS	0.01	0.11	0.00	0.03	0.02	0.01	0.03	0.00
Hive–Hive	0.00	0.02	0.02	0.06	0.02	0.00	0.00	0.00
Hive-Storm	0.00	0.01	0.00	0.02	0.01	0.00	0.02	0.00
Storm-Hadoop Mapreduce	0.06	0.19	0.11	0.04	0.11	0.05	0.10	0.03
Storm-Hadoop HDFS	0.05	0.16	0.10	0.03	0.12	0.03	0.10	0.01
Storm-Hive	0.05	0.11	0.08	0.06	0.08	0.03	0.07	0.02
Storm–Storm	0.01	0.06	0.03	0.03	0.03	0.00	0.00	0.01

To answer RQ2, different performance measures like Recall and FPR are used to compare the overall performance of SMOTE and non-SMOTE on applying ML classifiers in WEKA tool. In Feng *et al.* (2021), high Recall and low FPR are preferred as the more stable performance measures for oversampling techniques to predict ARBs. Tables 6 and 7 show the recall and FPR values, respectively, of all eight ML classifiers. Table 6 shows that Naïve Bayes and Adaboost gives good results with SMOTE for predicting ARBs using cross-project approach on the basis of Recall performance measure. On the same hand, FPR performance measure is also observed in Table 7 where classifier Decision Table brings out better results when used with SMOTE. As per study (Xu *et al.*, 2020), these ML classifiers are used when the instances of predicting class are less than non-predicting class. Thus SMOTE with Adaboost achieves higher Recall value and is considered an accurate classifier.

These experimental results show that SMOTE achieves the best overall performance for predicting ARBs of cloud computing datasets.

6. Conclusion

In SDP, majority datasets are imbalanced, therefore oversampling techniques are the solution to avoid this problem. SMOTE technique is used to score high recall value and low FPR in predicting ARBs in cloud-oriented datasets. SMOTE generates

synthetic instances on the basis of distance between them to remove class imbalance problem in datasets. In this study ARB prediction is performed on cloud-oriented software by applying machine learning algorithms. SMOTE is applied to remove the class imbalance problem and prediction is performed using both CPDP and WPDP approach.

Empirically the effect of SMOTE is evaluated and compared with non-SMOTE results using ML classifiers (Bagging, Adaboost, Logistic, Naïve Bayes, LogitBoost, Random Forest, J48, Decision Table) on five cloud-oriented datasets using CPDP approach. The experimental results show that SMOTE worked significantly better than non-SMOTE in terms of Recall, FPR, Balance performance measures at the confidence level of 95%. When SMOTE is used on cloud-oriented datasets to predict ARBs and CPDP approach is used, it has been observed that classifier Decision Table (average-0.06) brings better results with SMOTE on the basis of FPR value. Adaboost (average-0.34) and Random Forest (average-0.34) performed well with SMOTE on the basis of Recall value on cloud-oriented datasets for predicting ARBs. Due to the better performance of SMOTE on cloud-oriented datatsets, it is recommended to use as an efficient alternative to address class imbalance problem for predicting ARBs.

6.1. *Threats to validity*

In this section, threats to external, internal and construct validity are discussed for our study.

External Validity — In this study, five cloud-oriented datasets are used and cross-project approach is applied which resulted in 25 ($5 * 5$) experimental datasets. Source code metrics are used in the experiments, therefore the results cannot be claimed to other type of metrics such as aging-related metrics and process metrics. Datasets used in the study are easily available, thus it would be possible to replicate the experiment using different types of metrics.

Industrial datasets are not used due to the license limitation although industrial datasets provide more reliable results. More ML classifiers could be used but with limited time and cost few classifiers are used. Internal Validity — The experiments performed in this study used cross-project approach, therefore SMOTE is applied on training dataset and not on testing dataset to balance the training dataset and to predict the ARB on testing dataset.

Construct Validity — Threshold-dependent performance measures (Recall, FPR, Balance) are used to evaluate the performance of oversampling technique. These selected performance measures are generally used Qin *et al.* (2018) in imbalanced datasets to predict software defects. If other performance measures are used, different results may be obtained.

To reach a more general conclusion, it is aimed in future to use more datasets with more type of metrics and classifiers using various performance measures in predicting ARBs.

6.2. *Future direction*

In our future work, it is intended to use more oversampling techniques on cloud-oriented datasets to predict ARBs with various types of metrics including aging-related metrics. It is aimed to evaluate results with the application of more performance measures. Moreover, it is also aimed to balance the cloud-oriented datasets using techniques based on complexity. To further extend this work, supervised or unsupervised learning classifiers will be used in research.

References

Andrade, E, R Pietrantuono, F Machida and D Cotroneo (2021). A comparative analysis of software aging in image classifiers on cloud and edge. *IEEE Transactions on Dependable and Secure Computing*, 20, 563–573.

Chawla, NV, KW Bowyer, LO Hall and WP Kegelmeyer (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321–357.

Chouhan, SS and SS Rathore (2021). Generative adversarial networks-based imbalance learning in software aging-related bug prediction. *IEEE Transactions on Reliability*, 70, 626–642.

Cotroneo, D, R Natella and R Pietrantuono (2010). Is software aging related to software metrics? In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pp. 1–6. IEEE.

Cotroneo, D, R Natella and R Pietrantuono (2013). Predicting aging-related bugs using software complexity metrics. *Performance Evaluation*, 70(3), 163–178.

Feng, S, J Keung, X Yu, Y Xiao, KE Bennin, MA Kabir and M Zhang (2021). COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. *Information and Software Technology*, 129, 106432.

Ficco, M, R Pietrantuono and S Russo (2018). Aging-related performance anomalies in the apache storm stream processing system. *Future Generation Computer Systems*, 86, 975–994.

Herbold, S (2013). Training data selection for cross-project defect prediction. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, pp. 1–10. ACM.

Herbold, S (2017). A systematic mapping study on cross-project defect prediction. arXiv:1705.06429.

Kaur, H and A Kaur (2022). An empirical study of aging related bug prediction using cross project in cloud oriented software. *Informatica*, 46(8), 105–120.

Kumar, L and A Sureka (2018). Feature selection techniques to counter class imbalance problem for aging related bug prediction: Aging related bug prediction. In *Proceedings of the 11th Innovations in Software Engineering Conference*, pp. 1–11. ACM.

Malhotra, R and J Jain (2020). Handling imbalanced data using ensemble learning in software defect prediction. In *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 300–304. IEEE.

Malhotra, R and K Lata (2020). Using hybridized techniques for prediction of software maintainability using imbalanced data. In *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 787–792. IEEE.

Markov, Z and I Russell (2006). An introduction to the WEKA data mining system. *ACM SIGCSE Bulletin*, 38(3), 367–368.

Parri, J, S Sampietro, L Scommegna and E Vicario (2021). Evaluation of software aging in component-based web applications subject to soft errors over time. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 25–32. IEEE.

Qin, F, X Wan and B Yin (2020). An empirical study of factors affecting cross-project aging-related bug prediction with TLAP. *Software Quality Journal*, 28, 107–134.

Qin, F, Z Zheng, C Bai, Y Qiao, Z Zhang and C Chen (2015). Cross-project aging related bug prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pp. 43–48. IEEE.

Qin, F, Z Zheng, Y Qiao and KS Trivedi (2018). Studying aging-related bug prediction using cross-project models. *IEEE Transactions on Reliability*, 68(3), 1134–1153.

Wan, X, Z Zheng, F Qin, Y Qiao and KS Trivedi (2019). Supervised representation learning approach for cross-project aging-related bug prediction. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 163–172. IEEE.

Wu, X, W Zheng, M Pu, J Chen and D Mu (2020). Invalid bug reports complicate the software aging situation. *Software Quality Journal*, 28, 195–220.

Xu, B, D Zhao, K Jia, J Zhou, J Tian and J Xiang (2020). Cross-project aging-related bug prediction based on joint distribution adaptation and improved subclass discriminant analysis. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 325–334. IEEE.