Feature Selection Techniques to Counter Class Imbalance Problem for Aging Related Bug Prediction*

Aging Related Bug Prediction†

Lov Kumar
Thapar University
Patiala, Punjab, India
lovkumar505@gmail.com

Ashish Sureka
Ashoka University
Sonepat, Haryana, India
ashish.sureka@ashoka.edu.in

ABSTRACT

Aging-Related Bugs (ARBs) occur in long running systems due to error conditions caused because of accumulation of problems such as memory leakage or unreleased files and locks. Aging-Related Bugs are hard to discover during software testing and also challenging to replicate. Automatic identification and prediction of aging related fault-prone files and classes in an object oriented system can help the software quality assurance team to optimize their testing efforts. In this paper, we present a study on the application of static source code metrics and machine learning techniques to predict aging related bugs. We conduct a series of experiments on publicly available dataset from two large open-source software systems: Linux and MySQL. Class imbalance and high dimensionality are the two main technical challenges in building effective predictors for aging related bugs.

We investigate the application of five different feature selection techniques (OneR, Information Gain, Gain Ratio, RELEIF and Symmetric Uncertainty) for dimensionality reduction and five different strategies (Random Under-sampling, Random Oversampling, SMOTE, SMOTEBoost and RUSBoost) to counter the effect of class imbalance in our proposed machine learning based solution approach. Experimental results reveal that the random under-sampling approach performs best followed by RUSBoost in-terms of the mean AUC metric. Statistical significance test demonstrates that there is a significant difference between the performance of the various feature selection techniques. Experimental results shows that Gain Ratio and RELEIF performs best in comparison to other strategies to address the class imbalance problem. We infer from the statistical significance test that there is no difference between the performances of the five different learning algorithms.

KEYWORDS

Aging Related Bugs, Imbalance Learning, Empirical Software Engineering, Feature Selection Techniques, Machine Learning, Predictive Modeling, Software Maintenance, Source Code Metrics

*Aging Related Bug Prediction
†Aging Related Bug Prediction

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ISEC'2018, , Hyderabad, India
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6398-3…$15.00
https://doi.org/10.475/123_4

ACM Reference Format:
Lov Kumar and Ashish Sureka. 2018. Feature Selection Techniques to Counter Class Imbalance Problem for Aging Related Bug Prediction: Aging Related Bug Prediction. In *Proceedings of (ISEC'2018).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.475/123_4

1 RESEARCH MOTIVATION AND AIM

Aging-Related Bugs (ARBs) are defined as those bugs which are caused by error-conditions associated with software aging. An example of ARB is a system crash or hang due to gradual depletion of resources (such as memory leak in a web server) caused because of a long running system [7][9][8]. ARB happens when the execution time or period of the system is long as faults due to error conditions like memory leakage or unreleased files and locks that accumulate over a period of time before a failure is triggered [7][9][8].

It is hard to uncover aging related bugs during system testing and also it is non-trivial to replicate such bugs. Early prediction of such bugs and identification of files or classes in a software system which are prone to aging related faults can help a quality assurance team in optimizing their testing efforts and resources. Recent research shows that static source code metrics can be used as predictors for aging related bugs within a machine learning framework [7][9][8]. However there are *two main technical challenges* in building predictive models for gaining related bugs.

Imbalanced Data: One of the technical challenges in building an ARB classifier is that the training dataset is highly imbalanced or skewed. A data is considered as balanced when the instances of the target class are approximately evenly distributed across varies categories of the target class. An imbalance dataset is a dataset in which the instances of the target class are unevenly distributed across the target class categories. In a binary classification problem (for example aging related bug prediction), the class of interest is in minority which poses technical challenges is building an effective classifier. For example, in a binary class problem, if the class of interest (aging related bugs) has only 1% instances in comparison to 99% instances of the majority class then the dataset is considered to be highly imbalanced.

High-Dimensional Data Research shows that a high dimensional feature space consisting of irrelevant and redundant features decreases the performance of the classifiers [11] [12] [19]. The presence of large number of features (in our case, static source code features) poses intrinsic challenges to machine learning classification algorithms. Dimensionality reduction and identification of important features is a technical challenge in the context of ARB

prediction.

Which classes or files in a given object oriented software system are fault-prone from the perspective of aging related issue or likely to contain gaining related defect? is a problem encountered by software practitioners and the work presented in this paper is motivated by the need of building tool support for software developers for identifying aging related fault-prone classes. Automatic identification or prediction of aging related defect-prone classes in an object oriented software system is an area that has attracted several researchers attention (refer to the Related Work Section of the paper: Section 2). However, a large scale study (involving several software systems) on the application of several feature selection techniques, application of different types of strategies to counter the effect of imbalance dataset and comparing the performance of various machine learning algorithms is relatively unexplored. Our literature survey reveals lack of in-depth and focused empirical studies on the relative performance and impact of various types of data-level and algorithm-level methods to address the issue of imbalance dataset which is an important issue while learning a classifier for aging related bug prediction. Furthermore, there are lack of statistical analysis based studies examining the effectiveness of various dimensionality or feature reduction technique which is also a hard problem and poses one of the major technical challenges in the domain of aging related bug prediction using source code metrics. We frame several research questions aimed at filling some of the research gaps. The work presented in this paper is an extension of our previous work on Aging Related Bugs [15]. The work presented in this paper is paper is guided by the following research questions:

RQ 1: Is there a statistically significant difference in the prediction performance of five different feature ranking techniques? Which feature ranking technique(s) yields the best result?

RQ 2: Does removing a significant percentage of features improves the predictive performance substantially or does the model performance remains similar even after applying feature ranking or selection techniques?

RQ 3: Is there a statistically significant difference in the prediction performance of five different techniques applied to address the class imbalance problem? Which of the five technique(s) yields the best result?

RQ 4: What is the relative performance of the five different types of learning algorithms and classifiers in-terms of the AUC, accuracy and f-measure metrics? Is there a statistically significant difference in the prediction performance of five different classifiers?

2 RELATED WORK

Cotroneo et al. presents analysis of software gaining phenomenon at the Operating System (OS) level and investigate the software aging sources inside the Linux OS kernel [9]. They use a kernel tracing tool to examine the behavior of the kernel over long running executions and infer potential sources of aging in the kernel system [9]. Qin et al. investigate if aging related bugs can be identified using cross-project prediction approaches [22]. They apply a transfer learning based approach and validate the effectiveness of their method by conducting experiments on two real world software systems [22]. Cotroneo et al. conduct an empirical study to investigate the relationship between static features of the software and aging [7]. Their analysis reveals that static features of software can be used as predictors for identifying aging related bugs [7]. Lal et al. analyze the class-imbalance problem for logged code construct prediction [17]. They apply machine-learning models to predict logged code construct [17]. Wang et al. study the issue of if and how class imbalance learning methods can benefit software defect prediction with the aim of finding better solutions [27].

The most closely related work to the study presented in this paper is the work by Cotroneo et al. [8] on predicting aging related bugs using software complexity metrics. We use the dataset uploaded by Cotroneo et al. [8] to the PROMISE [1] repository. We make several extensions to the work by Cotroneo et al. [8] in-terms of the feature selection techniques, strategies to address the class imbalance problem and application of several different machine learning classifiers. Carrozza et al. conduct an analysis of Mandelbugs in an industrial software system [2].

3 RESEARCH CONTRIBUTIONS

In context to existing work, the study presented in this paper makes the following novel and unique research contributions:

(1) The study presented in this paper is the *first study* on the application of five different feature selection techniques, five different strategies to counter the effect of imbalance data and five different machine learning algorithm for predicting aging related bugs in seven sub-systems of two large open-source software projects using 82 source code metrics as predictors.
(2) We conduct a series of experiments to measure the performance of various techniques using metrics such as accuracy, f-measure and AUC and present results showing the effectiveness of various approaches. We frame *four research questions* and conduct statistical significance test to answer the stated research questions on identifying the best feature selection technique, imbalance learning strategy and classification algorithm.

4 RESEARCH FRAMEWORK

Figure 1 shows the research framework consisting of a multi-step process. As shown in Figure 1, there are four main steps in the proposed approach and framework. The four steps are: (1) Feature ranking and selection (2) Learning from imbalanced dataset (3) Predictive model building using machine learning classifiers (4) Classifier evaluation using performance metrics and conducting statistical significance test. Each of the four steps are described below in detail. Furthermore, we follow a research methodology in which we formulate *four grounded research questions* which guides our study. We apply the Sandberg et al. approach of formulating research questions based on existing literature and theories [23]. Our primary object is to formulate research questions and answer them to facilitate development of tools and creation of methods on

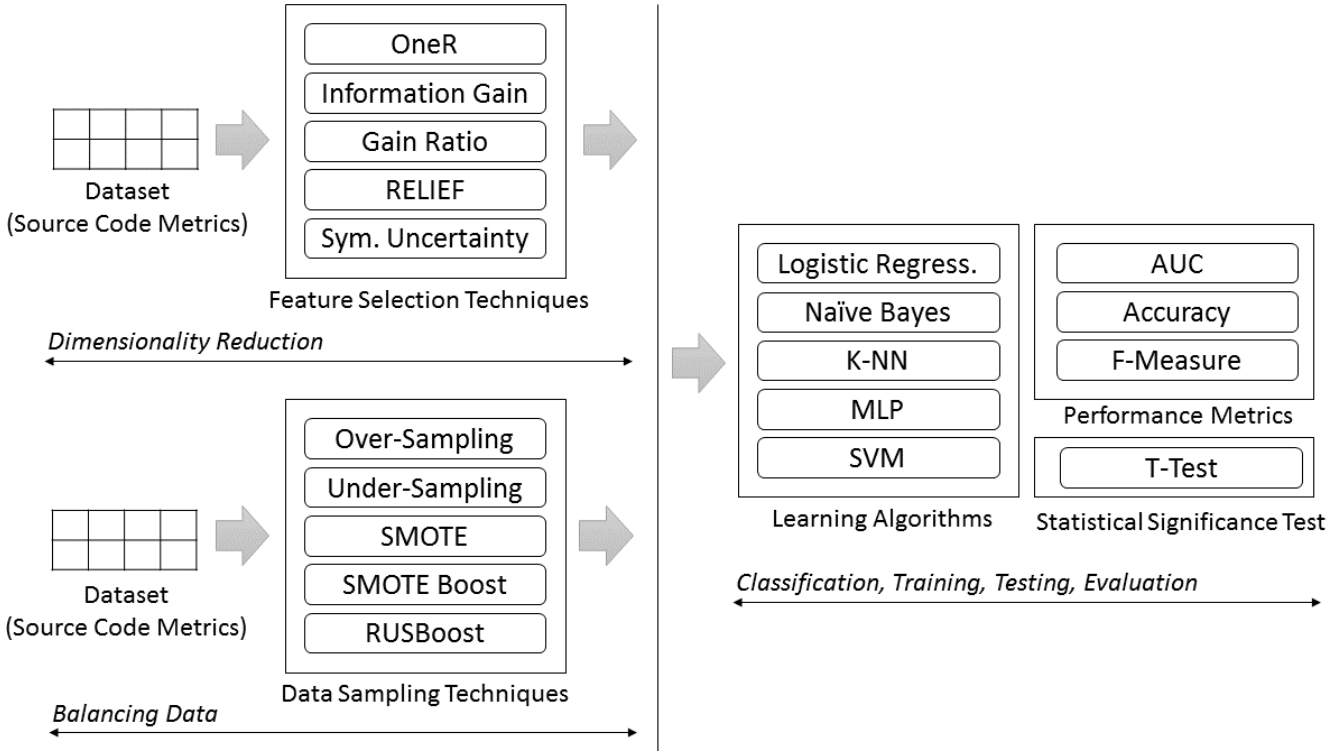

Figure 1: Research Framework and Solution Approach

tool support for automatic identification of prediction of gaining related defect prone classes and files.

Feature Ranking and Selection: We apply feature ranking and attribute selection to reduce the dimensionality of the input software metrics. Feature ranking is a pre-processing step applied before training or building a predictive model. The primary purpose of attribute ranking and subset selection is to reduce the number of redundant and irrelevant features and noise in the data. Feature selection improves the performance of the classifier and learning algorithm in-terms of speed, storage requirement and predictive accuracy. Lower number of relevant and important features also makes the result easy to interpret and helps in understanding the target concept. We apply a feature-ranking approach in which every feature is ranked and then we define a criteria to select the top k ranked features. The higher ranked features have better prediction capability than the features which are lower ranked. The value of k in the top k feature is pre-defined and finally a subset of features are selected based on the k value and feature ranks.

There are two approaches for feature selection: filter based and wrapper based approaches. Filter based approaches consists of applying correlation coefficient between variables for ranking whereas wrapper based approaches assesses the usefulness of features according to a given predictor or learning algorithm [11][14][16]. Liu et al. present a survey of several feature selection algorithms [19]. We apply five different feature selection approaches (refer to Figure 1): OneR, Information Gain (IG), Gain Ratio (GR), RELEIF (RF) and Symmetric Uncertainty (SU). We also consider no feature ranking or subset selection (all metrics or features). OneR system consists of creating rules based on a single feature and then calculating the classification accuracy for each rule and feature [12]. The ranking of each feature is derived based on the classification score for each rule. Information Gain (IG) and Gain Ratio (GR) are based on ideas from information theory and probabilistic reasoning. RELIEF (RF) method applies the concept of instance based learning to assign a relevance score to each of the feature [12]. Symmetric Uncertainty (SU) based method consists of measuring the fitness of all the features by calculating the symmetric uncertainty between two variables as a function of mutual information and entropy of the two variables. The feature which has a higher value of symmetric uncertainty is regarded as more important [12].

Learning from Imbalanced Dataset: We address the issue of class imbalance through two approaches: data-level and classification algorithm level. The data level approach consists of oversampling the rare or minority class (the class which is of interest) or under-sampling the majority class. The oversampling or undersampling can be random or informed. An informed over-sampling or under-sampling is not random and rather guided by a heuristic or criteria [5][13]. Oversampling and under-sampling can also be combined or blended. We apply five different techniques to address the class imbalance problem (refer to Figure 1): Random Under-sampling (USAM), Random Oversampling (OSAM), SMOTE, SMOTEBoost and RUSBoost.

Every strategy has their own advantages and shortcomings and hence our objective is to investigate which technique is more effective for the problem of ARB prediction. For example, one of the short-comings of random oversampling of minority class can lead to over-fitting. However previous research shows that the technique has been effective in countering the effects of imbalance dataset for problems in several domains [5][13].

SMOTE (Synthetic Minority Over-sampling Technique) algorithm blends under-sampling of the majority class with a special form of over-sampling the minority class [6]. Seiffert et al. propose a method called as RUSBoost for mitigating the class-imbalance problem [24]. RUSBoost is a method which combines both boosting and data sampling [24]. RUSBoost uses a technique called as boosting which is also used by algorithms such as AdaBoost consisting of building an ensemble of models creating a collection of weighted classifiers for classifying instances [24][25]. RUSBoost has been shown to perform well on skewed training data [24][25]. SMOTEBoost is a combination of the SMOTE (Synthetic Minority Over-sampling Technique) algorithm and the boosting procedure [4]. SMOTEBoost consists of creates synthetic examples from the rare or minority class and thereby compensating for skewed distribution [4].

Learning Algorithms: As shown in Figure 1, we apply five different types of learning algorithms. Lim et al. present a comparison of the predictive accuracy, complexity and training time of 33 learning algorithms [18]. We apply both classical algorithms as well as relatively modern and new statistical algorithms to investigate the generalizability of our results and identify the best performing classification algorithm. Caruana present an empirical comparison of the AUC performance of seven supervised learning methods including SVMs, neural nets, decision trees, k-nearest neighbor [3]. They conduct a study in which a total of 2000 models are evaluated in-terms of AUC [3]. Their study reveals a wide variance in the performance of the various learning algorithms on the same dataset. We *generate a total of* $5 * 5 * 5 * 7 = 875$ *predictive models* by varying four parameters: learning algorithm, dataset, imbalance learning strategy and feature selection techniques.

Performance Evaluation: There are several different performance metrics for measure the effectiveness of a classifier. Sokolova present a systematic analysis of the performance measures for classification task [26]. We use Area under the ROC (Receiver Operating Characteristics) Curve (AUC), f-measure and accuracy to measure classifier performance. We use f-measure as it incorporates both precision and recall. Similarly, we use AUC as AUC has an attractive property of being insensitive to changes in the class distribution [26]. We apply cross validation technique to remove bias while evaluating the predictive models.

5 EXPERIMENTAL DATASET

We use a publicly available dataset available at tera-PROMISE Repository[1] for our experiments. The tera-PROMISE website is a well-known repository consisting of several software engineering research datasets on code analysis, defects, effort, refactoring and test generation [1]. We use a dataset from the tera-PROMISE repository so that our experiments can be easily replicated and can be used for benchmarking and comparison by other researchers. We conduct experiments on a dataset containing information on aging-related bugs for two large, complex and long-living software systems. The two open-source projects are the Linux kernel and the MySQL DBMS. The dataset by Roberto et al. [20] has been used in the past for building aging-related defect prediction models using software complexity metrics as features in a machine learning framework [8]. The dataset contains several types of metrics (82 metrics - refer to Table 2) such as program size related metrics, McCabe's Cyclomatic complexity, Halstead metrics and aging-related metrics [20]. The metrics are independent variables and the number of aging-related bugs found in each file is the dependent variable.

Table 1 displays the experimental dataset details. The dataset consists of seven different projects. Four projects are Linux subsystems and three projects are MySQL sub-systems. The largest of the seven projects is Linux driver net consisting of 2292 records out of which 9 records (0.39%) are ARBs. The smallest project in the dataset consists of 29 records in which the percentage of the minority class is 17.24%. Table 1 reveals that the dataset for all the projects is highly imbalanced. The percentage of minority class varies from 0.39% to 17.24%.

6 EXPERIMENTAL RESULTS

Feature Ranking: Table 2 displays the output of OneR and Information Gain (IG) feature ranking technique. We select top k features which is a logarithmic function of the total number of features. Table 2 reveals the ranking of each feature from 1 to 82 for each of the 7 projects for the two feature ranking techniques. Similarly, we compute the rank of each feature for Gain Ratio (GR),

[1] http://openscience.us/repo/

Table 1: Experimental Dataset Description

ID	Name	Total Records	Majority Class	Minority Class	% Majority Class	% Minority Class
Proj1	**Linux driver net**	2292	2283	9	99.61	0.39
Proj2	**Linux driver scsi**	962	958	4	99.58	0.42
Proj3	**Linux ext3**	29	24	5	82.76	17.24
Proj4	**Linux ipv4**	117	115	2	98.29	1.71
Proj5	**MySQL innodb**	402	370	32	92.04	7.96
Proj6	**MySQL optimizer**	36	33	3	91.67	8.33
Proj7	**MySQL replication**	32	28	4	87.5	12.5

Table 2: Several Features or Source Code Metrics Ranked by the OneR and Information Gain Feature Selection Technique for Seven Different Software Sub-Systems

	OneR							Information Gain						
	Proj1	Proj2	Proj3	Proj4	Proj5	Proj6	Proj7	Proj1	Proj2	Proj3	Proj4	Proj5	Proj6	Proj7
AltAvgLineBlank	82	82	72	81	4	72	39	37	82	82	82	34	10	82
AltAvgLineCode	30	30	33	30	53	40	50	54	30	35	30	44	33	50
AltAvgLineComment	29	29	32	29	40	56	48	39	1	34	29	47	32	49
AltCountLineBlank	28	28	77	28	66	57	69	3	29	32	28	1	30	24
AltCountLineCode	27	27	31	27	37	68	81	32	27	29	27	10	27	34
AltCountLineComment	26	26	79	26	78	65	65	9	24	31	26	35	29	48
AvgCyclomatic	23	23	30	23	35	81	66	58	26	30	23	39	8	51
AvgCyclomaticModified	24	24	29	24	36	66	23	40	25	36	24	42	28	45
AvgCyclomaticStrict	31	31	34	31	43	70	24	59	31	33	31	40	7	53
AvgEssential	32	32	28	32	44	80	14	51	28	37	32	43	3	56
AvgLine	33	33	35	33	60	42	15	1	32	43	33	41	34	55
AvgLineBlank	38	38	37	38	3	71	31	36	38	45	38	29	9	54
AvgLineCode	40	40	42	40	8	78	58	50	40	44	40	36	35	47
AvgLineComment	39	39	41	39	5	58	61	38	39	42	39	48	36	46
CountClassBase	37	37	40	37	12	45	22	42	37	39	37	54	37	42
CountClassCoupled	34	34	39	34	31	46	21	43	34	27	34	55	42	41
CountClassDerived	36	36	26	36	22	47	19	45	36	40	36	64	41	40
CountDeclClass	35	35	38	35	49	14	18	49	35	38	35	65	40	38
CountDeclClassMethod	25	25	36	25	21	7	33	48	33	28	25	66	39	43
CountDeclClassVariable	21	21	27	21	20	15	32	47	22	24	21	75	25	39
CountDeclFunction	4	4	76	4	55	16	71	24	13	26	4	9	31	31
CountDeclInstanceMethod	20	20	24	20	23	27	13	57	21	18	20	78	26	44
CountDeclInstanceVariable	9	9	25	9	19	28	27	61	10	17	9	79	21	57
CountDeclInstanceVariablePrivate	8	8	13	8	24	32	30	73	9	14	8	82	12	52
CountDeclInstanceVariableProtected	7	7	12	7	26	51	29	75	7	13	7	81	13	59
CountDeclInstanceVariablePublic	6	6	14	6	29	22	38	74	2	19	6	76	17	60
CountDeclMethod	5	5	15	5	28	20	37	77	6	16	5	74	18	74
CountDeclMethodAll	2	2	17	2	27	18	36	78	3	22	2	67	19	73
CountDeclMethodConst	3	3	22	3	25	19	35	79	5	23	3	73	20	76
CountDeclMethodFriend	10	10	23	10	17	23	28	80	11	15	10	68	23	75
CountDeclMethodPrivate	11	11	21	11	10	24	26	81	8	21	11	69	24	77
CountDeclMethodProtected	12	12	18	12	16	25	20	76	12	25	12	70	11	81
CountDeclMethodPublic	17	17	20	17	7	26	25	72	18	20	17	71	15	80
CountInput	19	19	19	19	6	29	17	82	20	46	19	72	22	79
CountLine	18	18	9	18	76	67	80	25	19	41	18	3	14	35
CountLineBlank	16	16	74	16	69	62	78	4	17	48	16	2	16	23
CountLineCode	13	13	10	13	50	21	6	29	14	66	13	7	43	17
CountLineCodeDecl	15	15	43	15	58	64	16	33	16	72	15	49	38	7
CountLineCodeExe	14	14	8	14	56	10	3	28	15	71	14	5	45	4
CountLineComment	41	41	45	41	79	74	64	8	41	12	41	8	55	26
CountLineInactive	42	42	44	42	68	5	76	14	42	47	42	38	82	58
CountLinePreprocessor	43	43	80	43	9	6	79	35	43	70	43	45	72	29
CountOutput	46	46	66	53	14	4	34	70	46	69	46	62	71	78
CountPath	71	71	64	70	15	3	40	62	71	68	71	63	69	72
CountSemicolon	70	70	68	69	63	8	11	18	70	6	70	6	66	2
CountStmt	68	68	7	67	57	9	9	22	68	73	68	4	68	9
CountStmtDecl	65	65	62	64	13	11	82	27	65	8	65	18	67	33
CountStmtEmpty	67	67	63	66	11	73	1	44	67	67	67	46	73	10
CountStmtExe	66	66	5	65	67	13	8	21	66	74	66	23	74	18
Cyclomatic	72	72	67	71	30	12	57	69	72	76	72	80	75	71
CyclomaticModified	69	69	65	72	18	30	56	68	69	81	69	60	76	70
CyclomaticStrict	73	73	69	73	32	17	59	67	73	80	73	61	81	63
Essential	79	79	71	76	33	31	60	66	79	79	79	57	80	62
Knots	81	81	73	80	46	43	55	63	81	78	81	53	79	61
MaxCyclomatic	80	80	70	79	54	79	75	64	80	77	80	16	2	30
MaxCyclomaticModified	78	78	61	78	71	1	72	71	78	75	78	14	5	25
MaxCyclomaticStrict	75	75	53	77	59	76	63	65	75	65	75	15	1	36
MaxEssentialKnots	77	77	59	74	47	49	62	60	77	57	77	56	78	64
MaxInheritanceTree	76	76	60	75	45	50	54	46	76	64	76	51	77	65
MaxNesting	74	74	50	68	48	52	47	52	74	54	74	52	70	66
MinEssentialKnots	64	64	48	63	38	55	53	56	64	53	64	58	65	69
PercentLackOfCohesion	63	63	46	62	42	54	52	53	63	52	63	77	64	68
RatioCommentToCode	62	62	47	61	34	53	44	55	62	51	62	28	63	67
SumCyclomatic	51	51	4	50	81	48	4	10	51	50	51	20	52	6

RELEIF and Symmetric Uncertainty (SU) ranking techniques. Table 2 shows that the rank of the feature UniqueDerefSet is 1 for Projects 1, 2 and 4 when OneR feature ranking technique is applied. We observe that there are some feature which are ranked highly across projects and across feature selection techniques while there are some features which are ranked higher for a particular project or ranking technique. Table 2 shows that different feature selection techniques produces different output. An interesting result is that few metrics are highly ranked for some projects while the same

Table 3: Experimental Results for OneR Feature Ranking Technique Measured using Accuracy, F-Measure and AUC for Multiple Classifiers and Imbalance Learning Techniques

	AUC					Accuracy					F-Measure				
Undersampling															
	LOGR	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**
Linux driver net	0.81	0.82	0.75	0.78	0.54	92.03	93.33	90.81	85.58	98.56	0.96	0.97	0.95	0.92	0.99
Linux driver scsi	0.66	0.62	0.51	0.57	0.48	72.33	83.63	81.87	72.85	94.82	0.84	0.91	0.9	0.84	0.97
Linux ext3	0.54	0.6	0.48	0.62	0.5	63.33	73.33	66.67	63.33	70	0.76	0.83	0.79	0.74	0.82
Linux ipv4	0.34	0.39	0.4	0.38	0.45	66.09	77.39	78.26	73.91	88.7	0.8	0.87	0.88	0.85	0.94
MySQL innodb	0.68	0.73	0.75	0.67	0.69	80.25	83.25	79.5	76.5	88.5	0.89	0.9	0.88	0.86	0.94
MySQL optimizer	0.8	0.8	0.73	0.83	0.86	80	80	82.5	85	90	0.88	0.88	0.9	0.91	0.94
MySQL replication	0.67	0.65	0.67	0.77	0.67	85.71	82.86	85.71	88.57	85.71	0.92	0.9	0.92	0.93	0.92
Oversampling															
	LOGR	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**
Linux driver net	0.86	0.72	0.65	0.55	0.85	92.33	93.51	90.68	99.04	89.24	0.96	0.97	0.95	1	0.94
Linux driver scsi	0.66	0.53	0.47	0.5	0.47	91.71	84.87	94.2	98.76	93.16	0.96	0.92	0.97	0.99	0.96
Linux ext3	0.6	0.82	0.68	0.7	0.72	73.33	83.33	73.33	76.67	80	0.83	0.89	0.83	0.85	0.88
Linux ipv4	0.42	0.39	0.43	0.48	0.41	81.74	77.39	85.22	94.78	80	0.9	0.87	0.92	0.97	0.89
MySQL innodb	0.82	0.8	0.74	0.48	0.82	91.75	89	88.75	86.5	91.5	0.95	0.94	0.94	0.93	0.95
MySQL optimizer	0.8	0.79	0.53	0.66	0.86	95	77.5	77.5	85	90	0.97	0.86	0.87	0.91	0.94
MySQL replication	0.67	0.87	0.87	0.87	0.87	85.71	91.43	91.43	91.43	91.43	0.92	0.95	0.95	0.95	0.95
SMOTE															
	LOGR	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**
Linux driver net	0.5	0.74	0.5	0.49	0.5	99.56	97.17	99.56	98.04	99.56	1	0.99	1	0.99	1
Linux driver scsi	0.5	0.5	0.5	0.99	0.5	99.48	99.48	99.48	98.45	99.48	1	1	1	0.99	1
Linux ext3	0.7	0.3	0.7	0.8	0.3	50	50	50	66.67	50	0.57	0.67	0.57	0.75	0.67
Linux ipv4	0.5	0.5	0.5	0.5	0.5	91.3	91.3	91.3	91.3	91.3	0.95	0.95	0.95	0.95	0.95
MySQL innodb	0.78	0.76	0.54	0.61	0.7	87.5	83.75	85	83.75	87.5	0.93	0.91	0.92	0.91	0.93
MySQL optimizer	0.5	0.5	0.5	0.5	0.5	87.5	87.5	87.5	87.5	87.5	0.93	0.93	0.93	0.93	0.93
MySQL replication	0.92	0.92	0.92	0.42	0.92	85.71	85.71	85.71	71.43	85.71	0.91	0.91	0.91	0.83	0.91
SMOTEBoost															
	LOGR	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**
Linux driver net	0.95	0.5	0.49	0.48	0.95	90.63	98.91	97.82	96.51	90.85	0.95	0.99	0.99	0.98	0.95
Linux driver scsi	0.93	0.5	0.5	0.5	0.95	86.01	99.48	99.48	99.48	90.67	0.92	1	1	1	0.95
Linux ext3	0.3	0.5	0.9	0.9	0.8	50	83.33	83.33	83.33	66.67	0.67	0.91	0.89	0.89	0.75
Linux ipv4	0.45	0.5	0.5	0.5	0.68	82.61	91.3	91.3	91.3	82.61	0.9	0.95	0.95	0.95	0.9
MySQL innodb	0.7	0.54	0.44	0.54	0.63	86.25	86.25	81.25	85	87.5	0.92	0.93	0.9	0.92	0.93
MySQL optimizer	0.93	0.5	0.93	0.93	1	87.5	87.5	87.5	87.5	100	0.92	0.93	0.92	0.92	1
MySQL replication	0.5	1	0.5	1	1	85.71	100	85.71	100	100	0.92	1	0.92	1	1
RUSBoost															
	LOGR	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**	**LOGR**	**NBC**	**MLP**	**KNN**	**SVM**
Linux driver net	0.95	0.96	0.96	0.96	0.74	90.63	92.81	91.72	92.37	97.39	0.95	0.96	0.96	0.96	0.99
Linux driver scsi	1	0.93	0.5	0.48	0.5	99.48	86.53	99.48	96.37	99.48	1	0.93	1	0.98	1
Linux ext3	0.4	0.5	0.5	0.5	0.5	66.67	83.33	83.33	83.33	83.33	0.8	0.91	0.91	0.91	0.91
Linux ipv4	0.7	0.48	0.7	0.5	0.5	86.96	86.96	86.96	91.3	91.3	0.93	0.93	0.93	0.95	0.95
MySQL innodb	0.79	0.76	0.81	0.64	0.81	88.75	83.75	92.5	76.25	92.5	0.94	0.91	0.96	0.86	0.96
MySQL optimizer	0.36	0.79	0.86	0.5	1	62.5	62.5	75	87.5	100	0.77	0.73	0.83	0.93	1
MySQL replication	0.5	0.83	0.5	0.92	0.92	85.71	71.43	85.71	85.71	85.71	0.92	0.8	0.92	0.91	0.91

Table 4: Descriptive Statistics of the Relative Performance of Various Techniques in-terms of Accuracy, F-Measure and AUC

	Min	Max	Mean	Median	Std Dev	Q1	Q3
AUC							
USAM	0.30	1.00	0.72	0.76	0.16	0.63	0.83
OSAM	0.38	0.95	0.68	0.68	0.16	0.53	0.82
SMOTE	0.30	1.00	0.69	0.66	0.21	0.50	0.92
SMOTEBoost	0.30	1.00	0.66	0.50	0.22	0.50	0.93
RUSBoost	0.20	1.00	0.70	0.72	0.22	0.50	0.92
OneR	0.30	1.00	0.66	0.65	0.19	0.50	0.81
IG	0.30	1.00	0.68	0.68	0.20	0.50	0.87
GR	0.33	1.00	0.71	0.72	0.20	0.50	0.91
RF	0.30	1.00	0.71	0.71	0.19	0.50	0.87
SU	0.30	1.00	0.70	0.71	0.20	0.50	0.88
ALL	0.20	1.00	0.68	0.65	0.21	0.50	0.87
LOGR	0.30	1.00	0.68	0.67	0.20	0.50	0.86
NBC	0.20	1.00	0.69	0.70	0.19	0.50	0.86
MLP	0.20	1.00	0.68	0.69	0.19	0.50	0.86
KNN	0.30	1.00	0.68	0.63	0.20	0.50	0.89
SVM	0.30	1.00	0.71	0.71	0.20	0.50	0.90
Accuracy							
USAM	56.52	100.00	82.72	84.48	8.59	78.50	88.57
OSAM	56.67	99.27	88.25	90.00	8.18	84.75	93.91
SMOTE	50.00	100.00	89.84	91.62	12.57	85.71	99.48
SMOTEBoost	50.00	100.00	91.01	91.30	9.40	86.01	99.48
RUSBoost	33.33	100.00	86.37	87.50	11.25	83.33	92.23
OneR	50.00	100.00	85.83	87.50	10.65	82.67	91.72
IG	50.00	100.00	88.21	88.57	9.72	84.35	96.43
GR	50.00	100.00	88.87	90.00	10.06	85.71	97.32
RF	56.52	100.00	88.14	89.54	10.29	82.89	97.74
SU	50.00	100.00	88.42	88.57	8.86	83.33	94.30
ALL	33.33	100.00	86.36	87.56	12.92	81.13	98.10
LOGR	50.00	100.00	85.09	86.25	11.53	80.25	92.23
NBC	33.33	100.00	86.96	87.50	10.28	83.33	93.64
MLP	33.33	100.00	87.71	87.50	10.49	83.33	96.30
KNN	50.00	100.00	88.66	89.76	10.25	83.33	98.45
SVM	50.00	100.00	89.77	91.13	9.48	85.71	97.50
F-Measure							
USAM	0.67	1.00	0.90	0.91	0.06	0.87	0.93
OSAM	0.71	1.00	0.93	0.94	0.05	0.91	0.97
SMOTE	0.57	1.00	0.93	0.96	0.10	0.91	1.00
SMOTEBoost	0.67	1.00	0.95	0.95	0.06	0.92	1.00
RUSBoost	0.50	1.00	0.92	0.93	0.08	0.91	0.96
OneR	0.57	1.00	0.91	0.93	0.08	0.90	0.96
IG	0.57	1.00	0.93	0.93	0.07	0.91	0.98
GR	0.57	1.00	0.93	0.94	0.07	0.92	0.99
RF	0.71	1.00	0.93	0.94	0.07	0.90	0.99
SU	0.67	1.00	0.93	0.93	0.06	0.91	0.97
ALL	0.50	1.00	0.92	0.93	0.09	0.89	0.99
LOGR	0.57	1.00	0.91	0.92	0.08	0.89	0.96
NBC	0.50	1.00	0.92	0.93	0.07	0.90	0.97
MLP	0.50	1.00	0.93	0.93	0.07	0.90	0.98
KNN	0.57	1.00	0.93	0.94	0.07	0.91	0.99
SVM	0.67	1.00	0.94	0.95	0.06	0.92	0.99

metrics is ranked low for other projects within the context of a particular feature selection technique. For example, CountDeclMethod and CountDeclMethodAll are ranked highly for Projects 1, 2 and 4 but low for Projects 5, 6 and 7. We observe that there can be a wide variance in-terms of the ranking for the same metrics across different projects. For example, AltAvgLineBlank metric is ranked as 4 for Project 5 and 82 for Project 1 and Project 3.

Detailed Analysis for OneR Feature Ranking: While Figure 2, 3 and 4 and Table 4 shows summary results, Table 3 displays complete and detailed results for OneR feature ranking technique. We present detailed result for one of the technique and summary for all other techniques due to limited space in the paper. Table 3 shows the experimental results for all the seven projects in-terms of accuracy, f-measure and AUC performance metrics across all classifiers and across all the five imbalance learning strategies. From Table 3, we observe that there is no one learning algorithm or imbalanced dataset sampling approach which dominates all other approaches.

The characteristics of the dataset and learning algorithm influences the performance of a given feature selection technique and it is a combination of quality of features, input data and classification algorithm characteristics which determines the overall effectiveness of predictive model. Table 3 shows that for SVM classifier using SMOTE technique the AUC is only 0.3 for the Linux ext3 project whereas it is 0.92 for the MySQL replication project. Such a wide variance shows that the project data characteristics plays an important role and it is not just the sampling technique, classification algorithm or the feature ranking technique which influences the outcome. Table 3 reveals that OneR feature ranking technique, SMOTEBoost using NBC, KNN and SVM results in a perfect AUC score of 1.00 for the MySQL replication project. In case of RUSBoost, we obtain a perfect AUC score of 1.00 for Linux drive scsi using the LOGR, MLP and SVM classification algorithm. In-terms of the f-measure, SMOTE gives encouraging result when used in conjunction with the KNN classifier.

Relative Comparison of Techniques: Table 4 displays the descriptive statistics of the relative performance of various techniques in-terms of accuracy, f-measure and AUC. There are 7 projects, 5 different techniques for feature selection, 5 strategies for imbalance learning and 5 learning algorithms. Hence we generate $5*5*7 = 175$ data points to compute the mean value of any feature selection technique, imbalance learning strategy or classification algorithm. Table 4 reveals that the mean AUC value of USAM is best in comparison to the other imbalance learning strategies. RUSBoost is the second best technique. The mean AUC value of USAM is 0.72 and the mean AUC value of RUSBoost is 0.70. However, the accuracy and f-measure value shows a different trend. According to the f-measure value, SMOTEBoost performs best followed by RUSBoost.

From Table 4, we infer that SVM has a mean AUC value of 0.71. The mean AUC value of NBC is 0.69 and the mean AUC value of LOGR, MLP and KNN is 0.68 each. Hence, SVM performance best followed by NBC. In-terms of f-measure, the mean f-measure value of SVM is 0.94. The mean f-measure value of KNN and MLP is 0.93 which is higher than the mean f-measure value of NBC which is 0.92 and 0.91 for LOGR. Table 4 reveals the relative performance of the five feature selection technique in-terms of the accuracy, f-measure and AUC. Table 4 also shows the performance of the all metrics case (no feature selection technique applied). GR and RF performs equally well in-terms of the mean AUC value. The mean AUC value of SU is 0.70 which is second best. In terms of the f-measure value, IG, GR, RF and SU performs equally well having a

mean value of 0.93. The mean f-measure value if all metrics (ALL) is 0.92 which is slightly lower than the mean f-measure values of IG, GR, RF and SU. The Q3 value for the mean AUC for SMOTEBoost is 0.93 which means that 25% of the predictive models with this setting has an AUC value of more than 0.93. The lowest Q3 value for the mean AUC is for OSAM.

Distribution Comparison and Visualization using Boxplot: Figure 2, 3 and 4 shows the boxplots for comparing the degree of dispersion, interquartile range, outliers and skewness in the accuracy, f-measure and AUC values for all the techniques across all the projects. The red line in the boxplot marks the midpoint of the data (median) and it divides the box into two parts. Figure 2 shows that the median AUC value of USAM is higher than the corresponding values for other imbalance learning techniques. From the distributional characteristics of data in Figure 3, we observe that the inter-quartile range (the middle box representing the middle 50% of the values of the variable) range for GR is more than the corresponding values for OneR, IG, RF, SU and ALL.

The boxplot for the AUC values for NBC and MLP is comparatively tall and hence shows that there is more variation in the AUC value obtained from 125 executions. We see some obvious differences in the boxplots for the f-measure metric (refer to Figure 4). We observe that for some of the boxplots for the f-measure even if the medians at the same level (such as OneR, IG, SU and ALL), the distribution is quite different. The upper and lower whiskers as well as the outliers in Figure 4 shows how many observations are outside the middle 50%. From Figure 3, we study the variability and dispersion in the accuracy values. Figure 3 shows that the span between the first and third quartile is more for SMOTE and SMOTEBoost than USAM, OSAM and RUSBoost. Similarly, we observe that the span between the first and third quartile for RF and ALL is more for than the span for OneR, IG, GR and SU.

Null Hypothesis Statistical Significance Testing: We apply the nonparametric Wilcoxon signed-rank test with a Bonferroni correction to compare whether the two populations are different. It is a type of paired t-test used to check if the null hypothesis is true or false [28]. Figure 5 shows the output of the hypothesis testing consisting of a comparison between five feature selection techniques, five imbalance learning strategies and five classifiers based on AUC performance metrics. In Figure 5, a red dot means that H_0 is rejected and a green dot means that H_0 is accepted and there is no statistically significant difference between the two techniques under comparison. We use a standard cut-off of 0.05 (the null hypothesis is rejected when $p < 0.05$ and not rejected when $p > 0.05$). Figure 5 reveals several red dots which shows that there is a statistically significant difference between the performances of the various techniques.

Figure 5 reveals that there is a *statistically significant difference between SMOTEBoost and RUSBoost, OSAM and USAM and USAM and SMOTEBoost.* There are a total of 10 pair-wise comparisons for a matrix consisting of 5 different techniques. For the analysis involving comparing feature selection techniques, we observe that the null hypothesis H_0 is rejected for 3 out of 10 significance tests. From Figure 5, we observe that the null hypothesis H_0 is rejected for 3 out of 10 significance tests involving pairwise comparison between the imbalance learning techniques. Figure 5 reveals that there is a *statistically significant difference between OneR and Gain Ratio, OneR and RELIEF and OneR and Symmetric Uncertainty (SU).* However, we observe that there is *no statistical significance difference between the five different classification algorithms* based on the nonparametric Wilcoxon signed-rank test.

7 ANSWER TO RESEARCH QUESTIONS

Following are the answers to the four research questions framed in Section 1:

Answer RQ 1: There is a statistically significant difference between 3 out of 10 pair-wise comparisons (5 different techniques excluding the all metrics). There is a statistically significant difference between OneR and Gain Ratio, OneR and RELIEF and OneR and Symmetric Uncertainty (SU). In-terms of the mean AUC, GR and RF performs best and performs equally well. In-terms of the f-measure also GR and RF performs best but IG and SU also performs equally well.

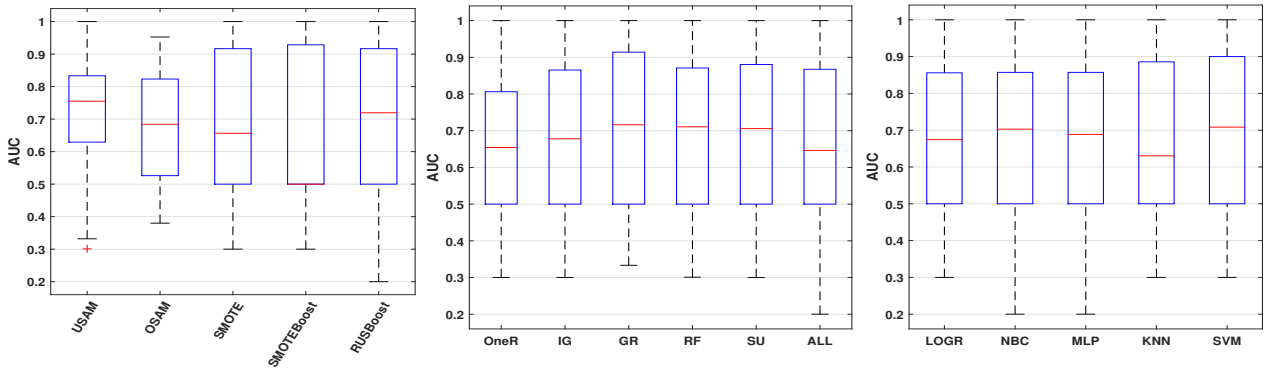

Figure 2: Boxplot Showing the Degree of Dispersion, Interquartile Range, Outliers and Skewness in the AUC Value

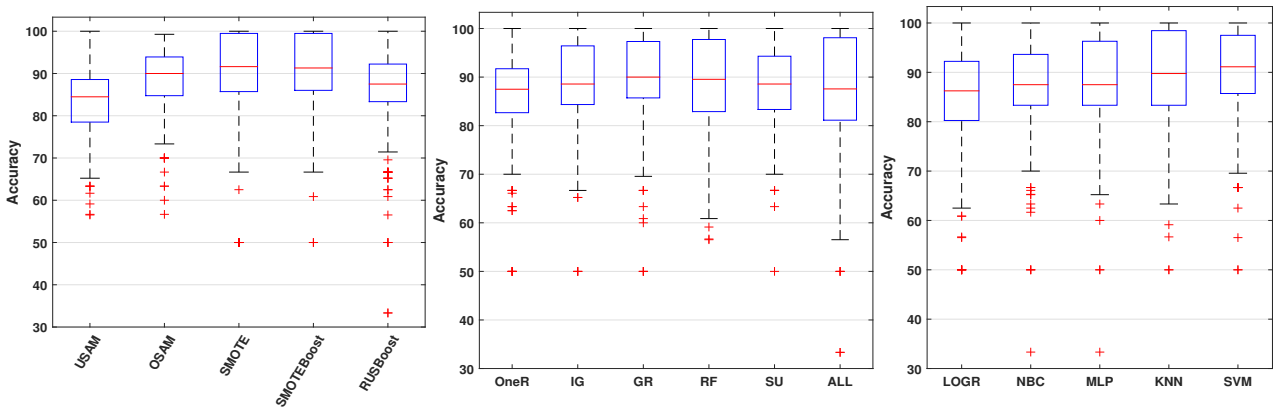

Figure 3: Boxplot Showing the Degree of Dispersion, Interquartile Range, Outliers and Skewness in the Accuracy Value

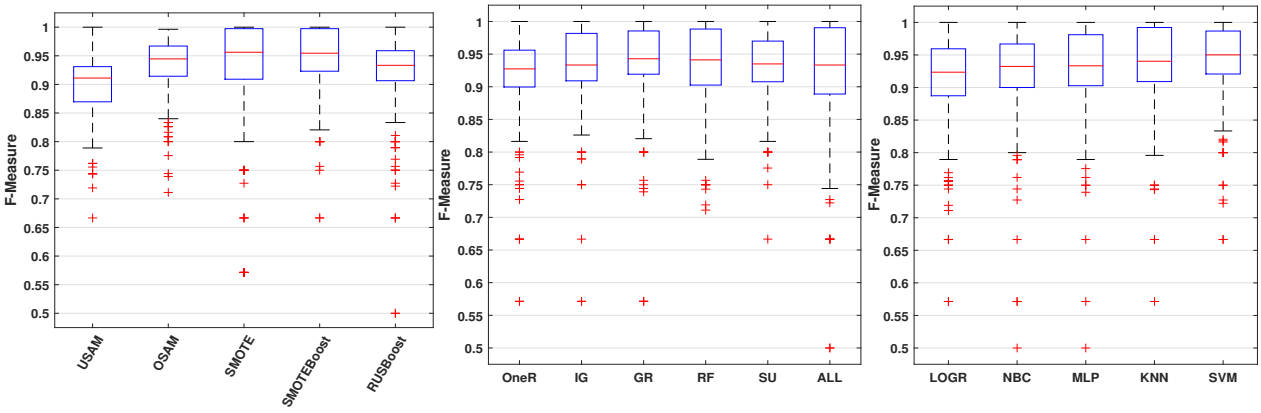

Figure 4: Boxplot Showing the Degree of Dispersion, Interquartile Range, Outliers and Skewness in the F-Measure Value

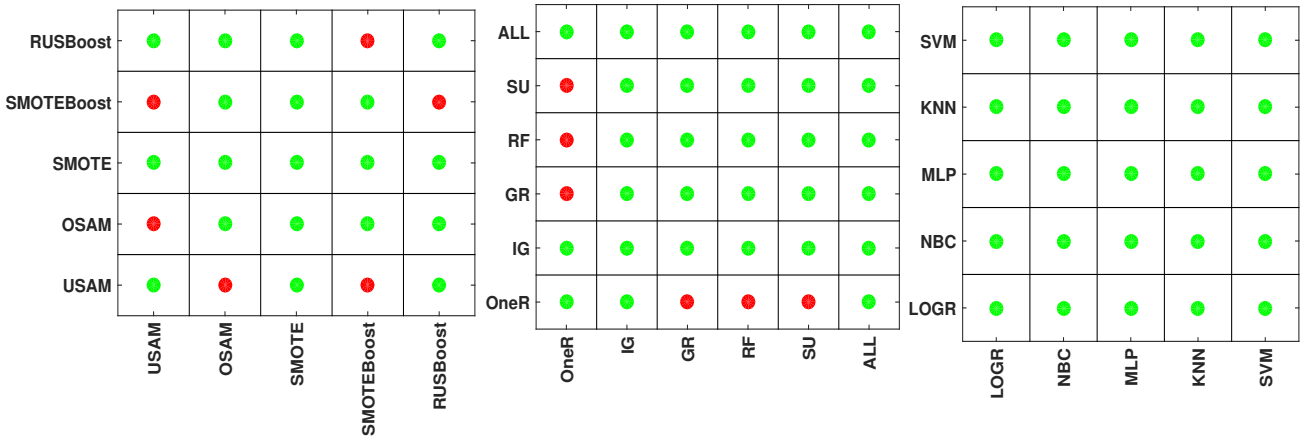

Figure 5: Results of the Statistical Significance Wilcoxon Signed-Rank Test with a Bonferroni Correction. A Red Dot Means that H_0 is Rejected

Answer RQ 2: There is no statistically significant difference between the performance of the all metrics (no filter selection technique) and the 5 different types of feature selection technique according to the Wilcoxon signed-rank test. However, based on the 125 data points, feature selection technique results in performance

improvement in comparison to all metrics.

Answer RQ 3: There is a statistically significant difference between 3 out of 10 pair-wise comparisons. There is a statistical difference between SMOTEBoost and RUSBoost, OSAM and USAM and USAM and SMOTEBoost. According to the f-measure value, SMOTEBoost performs best followed by RUSBoost. According to AUC, USAM performs best followed by RUSBoost.

Answer RQ 4: There is no statistically significant difference between the performance of 5 different classification algorithm according to the Wilcoxon signed-rank test. However, based on the data points generated as a result of our experiments on seven projects, SVM performs best in-terms of f-measure and AUC. NBC performs second best in-terms of AUC. KNN and MLP performance is equal and perform second best in-terms of f-measure.

8 THREATS TO VALIDITY

The study presented in this paper is empirical software engineering research and has several possible threats to validity. In this Section, we report our validity analysis and provide justification on how we overcame or minimized the possible four different types of threats to validity [10][21] in our experiments. We apply statistical significance test (Wilcoxon Signed-Rank Test) to mitigate conclusion validity and present evidences to show that the performance improvement is actual and not by chance. However, the p-value considered is a standard value (0.05) and a more stringent p-value (for example 0.01 or even lower) can have an effect on the hypothesis testing outcome. We conduct experiments on dataset belonging to seven projects (seven sub-systems of two large software systems: Linux and MySQL) to investigate if our results are generalizable and thus mitigate external validity. However, we believe that more experiments needs to be conducted on additional projects to ensure that the results obtained holds true for other situations also. More experiments needs to be conducted using additional dataset, feature selection techniques, imbalance learning strategies and classification algorithms.

There is a possibility of threats to internal validity associated with making accurate measurements. We downloaded a aging related bugs dataset from PROMISE repository which is manually validated and of high quality to ensure that there are no annotation or measurement errors. We applied cross-validation approach and executed the experiments more than once to ensure that there are no errors while measuring and observing data. However, there are still possibilities of threats to internal validity as the impact on the dependent variable may not be completely attributed to the changes in the independent variable due to overfitting of the predictive model as well as small number of examples of the minority class (aging related bugs). While our results shows relationship between the dependent (aging related fault-proneness) and independent variable (source code metrics), we believe that more experiments with a variety of feature selection and extraction techniques and learning algorithm is required to confirm that the variables accurately model the hypothesis.

9 CONCLUSION

We present a focused empirical study on identifying aging related bugs using source code metrics by applying a machine learning framework consisting of five feature ranking techniques, five imbalance learning strategies and five classification algorithms on seven projects resulting in 875 unique combinations of predictive models. Our main conclusions are that static source code metrics can be used as predictors for aging related bugs. We conclude that while there are several source code metrics which can be extracted from the software system, not all are equally important as several metrics are redundant and irrelevant. Our results reveal that some of the feature ranking techniques for dimensionality reduction improves the performance of the predictive model in comparison to using all metrics. Imbalanced and highly skewed data is one of the major challenges in learning a predictive model and classifying unseen instanced. We apply five different strategies to counter the effect of imbalanced data distribution and our empirical result shows that SMOTEBoost performs best followed by RUSBoost and there is a statistically significant difference between these techniques and other imbalance learning approaches. There is no statistically significant difference between the performance of 5 different classification algorithm according to the Wilcoxon signed-rank test.

REFERENCES

[1] 2015. The Promise Repository of Empirical Software Engineering Data. (2015).
[2] Gabriella Carrozza, Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. 2013. Analysis and prediction of mandelbugs in an industrial software system. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*. IEEE, 262–271.
[3] Rich Caruana and Alexandru Niculescu-Mizil. 2006. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 161–168.
[4] Nitesh Chawla, Aleksandar Lazarevic, Lawrence Hall, and Kevin Bowyer. 2003. SMOTEBoost: Improving prediction of the minority class in boosting. *Knowledge Discovery in Databases: PKDD 2003* (2003), 107–119.
[5] Nitesh V Chawla. 2005. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*. Springer, 853–867.
[6] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
[7] Domenico Cotroneo, Roberto Natella, and Roberto Pietrantuono. 2010. Is software aging related to software metrics?. In *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second International Workshop on*. IEEE, 1–6.
[8] Domenico Cotroneo, Roberto Natella, and Roberto Pietrantuono. 2013. Predicting aging-related bugs using software complexity metrics. *Performance Evaluation* 70, 3 (2013), 163–178.
[9] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. 2010. Software aging analysis of the linux operating system. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE, 71–80.
[10] Robert Feldt and Ana Magazinius. 2010. Validity Threats in Empirical Software Engineering Research-An Initial Survey.. In *SEKE*. 374–379.
[11] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, Mar (2003), 1157–1182.
[12] Mark Andrew Hall. 1999. Correlation-based feature selection for machine learning. (1999).
[13] T Ryan Hoens and Nitesh V Chawla. 2013. Imbalanced datasets: from sampling to classifiers. *Imbalanced Learning: Foundations, Algorithms, and Applications* (2013), 43–59.
[14] Lov Kumar, Santanu Kumar Rath, and Ashish Sureka. 2017. Empirical Analysis on Effectiveness of Source Code Metrics for Predicting Change-Proneness.. In *ISEC*. 4–14.
[15] Lov Kumar and Ashish Sureka. 2017. Aging Related Bug Prediction using Extreme Learning Machines. *IEEE India Council International Conferece (INDICON)* (2017).
[16] Lov Kumar and Ashish Sureka. 2017. Using Structured Text Source Code Metrics and Artificial Neural Networks to Predict Change Proneness at Code Tab and Program Organization Level.. In *ISEC*. 172–180.
[17] Sangeeta Lal, Neetu Sardana, and Ashish Sureka. 2016. Improving Logging Prediction on Imbalanced Datasets: A Case Study on Open Source Java Projects.

International Journal of Open Source Software and Processes (IJOSSP) 7, 2 (2016), 43–71.

[18] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. 2000. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning* 40, 3 (2000), 203–228.

[19] Huan Liu and Lei Yu. 2005. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering* 17, 4 (2005), 491–502.

[20] Roberto Natella. 2017. Aging-related bugs and software complexity metrics Aging-related bugs and software complexity metrics. (May 2017). https://doi.org/10.5281/zenodo.581659

[21] Dewayne E Perry, Adam A Porter, and Lawrence G Votta. 2000. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*. ACM, 345–355.

[22] Fangyun Qin, Zheng Zheng, Chenggang Bai, Yu Qiao, Zhenyu Zhang, and Cheng Chen. 2015. Cross-Project Aging Related Bug Prediction. In *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on*. IEEE, 43–48.

[23] Jörgen Sandberg and Mats Alvesson. 2011. Ways of constructing research questions: gap-spotting or problematization? *Organization* 18, 1 (2011), 23–44.

[24] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. 2008. RUSBoost: Improving classification performance when training data is skewed. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE, 1–4.

[25] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. 2010. RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 40, 1 (2010), 185–197.

[26] Marina Sokolova and Guy Lapalme. 2007. Performance measures in classification of human communications. *Advances in Artificial Intelligence* (2007), 159–170.

[27] Shuo Wang and Xin Yao. 2013. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability* 62, 2 (2013), 434–443.

[28] RF Woolson. 2008. Wilcoxon Signed-Rank Test. *Wiley encyclopedia of clinical trials* (2008).