Predicting defects in object-oriented software using cost-sensitive classification

R Malhotra¹ and J Jain²

¹Associate Professor, Delhi Technological University, Delhi, India
²Research Scholar, Delhi Technological University, Delhi, India

¹ruchikamalhotra2004@yahoo.com, ²erjuhijain@gmail.com

Abstract. In this software era, it is vital to produce reliable and good quality software. Early detection of defects aids in building accurate software with reduced cost and other resources. Researchers have a keen interest in producing machine learning models for effective and accurate software defect prediction in the early stages of software development. Object-oriented metrics of the software are used in developing these models. These models may result in biased predictions owing to the class imbalance problem existing in most of the software datasets. This paper provides an effective defect prediction framework for imbalanced data by employing cost-sensitive classifiers and stable performance measures like GMean, Balance, and AUC. Four decision tree-based classifiers with different cost ratios are investigated to predict defects in three Apache projects. The empirical results are statistically validated using the nonparametric Friedman test and Wilcoxon signed-rank test. The results state with 99% confidence that the predictive capability of J48, AdaBoostM1, Bagging, and RandomSubSpace improved after employing cost-sensitive learning for the four classifiers used in this study.

Keywords: software defect prediction, decision tree, class imbalance problem, software metrics, cost-sensitive classification

1. Introduction

With the growing advent of software and technology, software quality assurance is a very critical activity in today’s world. If defects are uncovered in later stages of software development, the cost of detecting them amplifies exponentially. With the constraints of limited time and limited resources, it becomes the utmost requirement of software developers and practitioners to discover these defects early. Finding defects or faults in the early stages of the software development life cycle leads to reduced cost, effort, and resources [1]. Software metrics are widely used for generating defect prediction models. Different object-oriented (OO) metrics define different internal attributes of the software like cohesion, coupling, inheritance, encapsulation, etc. Therefore, these metrics are utilized to envisage whether a software class can be defective or not [2, 3]. Such predictions guide developers to allocate the resources carefully resulting in the proper resource utilization. Various machine learning (ML) have been used for this purpose in literature.

One of the major issues that have been observed in software data is the class imbalance problem. If there is less number of instances of one type of class than that of other class, then data is said to have a class imbalance problem. For our application, if in software defective classes are less than non-
defective classes, then it is said to be imbalanced. The class imbalance problem can be solved by using resampling techniques or cost-sensitive learning. In a cost-sensitive approach, we can employ different meta cost learners in which False Positives are penalized at different cost factors in comparison to False Negatives. This paper aims at proposing effective software defect prediction (SDP) for imbalanced OO softwares using different meta cost learners.

This research study addresses the following research questions (RQ):

RQ1: What is the performance of used ML techniques without involving cost-sensitive classification?

RQ2: Do cost-sensitive classification improve the defect prediction capability of ML models built for imbalanced data?

RQ3: Which ML technique outperforms in predicting the software defects?

This research exploits four ML techniques- J48, AdaBoostM1, Bagging, and RandomSubSpace to predict defects in imbalanced data. Correlation feature selection is used to remove irrelevant features and stratified ten-fold cross-validation is used to train and construct the model. Model prediction capability is analyzed at different cost ratios (MC10, MC15, MC20, MC25, MC30, MC40, MC50) to find the appropriate cost settings. When data is imbalanced, traditional metrics like accuracy gives biased results. Instead, metrics that emphasize on both positive and negative classes must be considered. Therefore, the model performances are evaluated based on stable performance measures like GMean, Balance, and AUC. This study also compares the results based on sensitivity. The empirical evaluation of results advocates the use of a cost-sensitive approach for better defect prediction. Related work is performed primarily for NASA datasets and they didn't explore GMean and Balance metrics that are preferred for imbalanced data. This motivates us to explore cost-sensitive classification for software-oriented datasets like Apache datasets with stable and reliable performance measures.

The remainder paper is organized as: Section 2 summarizes the related research done in the area. The research methodology in section 3 explains the experimental framework required in the execution of the study. Results and their analysis are detailed in section 4 followed by validity threats in section 5. Section 6 concludes the study with future directions.

2. Literature Review

SDP is the hot research area for the last two decades and researchers have provided many solutions to deal with imbalanced nature in software data. Class imbalance problem can be treated either at the data level or algorithm level. At the data level, resampling techniques are employed to balance the data before model construction. For example, Lingden et al. [4] have proposed a modified undersampling technique that combines CFS, RUS, and decision forest that gives promising results. Recently, Bejjanki et al.[5] used eight different classifiers namely AdaBoost, Decision Tree, Extra Tree, Gradient Boosting, KNN, Logistic Regression, Naïve Bayes, and Random Forest to build SDP models with novel class reduction methods where new samples are generated by calculating the centroids of the minority samples and evaluated based on accuracy, precision, sensitivity, F-measure and GMean.

At the algorithm level, cost-sensitive learning is involved. This work focuses on dealing with the class imbalance problem at the algorithm level. Cost-sensitive learning can be done either (1) by assigning weights to minority samples or (2) by penalizing wrongly predicted defective classes in the cost matrix. Most of the related work is done in the first direction. Arar and Ayan [6] exploited neural networks to predict defects in five NASA datasets. They trained their neural network using FNR and FPR classification costs. Siers and Islam [7] designed a reduced cost decision tree-based ensemble solution for SDP on six NASA datasets and used weighted precision and sensitivity metrics. Afterward, they suggested a balanced cost framework to handle class imbalance as well as cost sensitivity [8]. A balanced cost matrix was created using Standoff. Though the cost was reduced, model performances were not evaluated based on any metrics like precision, sensitivity, etc.

This paper contributes to cost-sensitive classification based on cost matrix penalization. Rodriguez [9] performed the empirical comparison of models constructed using J48 and Naïve Bayes with
different resampling techniques, ensemble techniques (AdaBoostM1, Bagging, and RF), and Metacost learner (with cost = 10). They compared performances based on MCC and AUC and found that ensemble-based models performed better than meta cost learners for NASA datasets. They used a parametric t-test for statistical validation of results even though the underlying data did not have a normal distribution. A detailed empirical investigation of oversampling techniques and meta cost learners to deal with class imbalance problem in the SDP domain is conducted by Malhotra and Kamal [10] over well known 12 NASA datasets. They explored the cost ratio of 10, 30, and 50 in meta cost learners and evaluated performance based on AUC, sensitivity, and precision.

The cost-sensitive classification has been primarily performed on NASA datasets. We want to test the suitability of the cost-sensitive classification on software datasets like Apache for SDP in imbalanced software data. Further, this paper will do performance evaluation based on GMean and Balance metrics that are more reliable metrics than accuracy, sensitivity, or precision in the imbalanced data domain.

3. Research Methodology
This section describes the process of collecting data and identifying independent and dependent variables. It summarizes the feature selection technique, ML techniques, cost-sensitive classification approach, performance measures, and validation techniques used in this study.

3.1. Dataset Collection
Three open-source JAVA projects contributed by Jureczko and Madeyski [11] are downloaded by the Promise library [12]. Description of Jedit4.2, Xerces1.3, and Ant1.7 are presented in Table 1. Datasets are organized in increasing order of their defective classes. #TC represents the total number of classes in the software, #DefC signifies defective classes present in the software. Percentage of defective classes(#%ageDef) range from 13.1% to 22.3%. As evident from Table 1, datasets are highly imbalanced and there is a need of balancing the defective and non-defective classes.

| Dataset  | #TC | #DefC | #%ageDef |
|----------|-----|-------|----------|
| Jedit4.2 | 367 | 48    | 13.1     |
| Xerces1.3| 453 | 69    | 15.2     |
| Ant1.7   | 745 | 166   | 22.3     |

3.2. Independent and Dependent Variables
The Independent variables of the study are 20 OO metrics identified by Jureczko and Madeyski [11]. OO metrics comprise of six Chidamber and Kemerer (CK) Metric suite [13], five Quality Model for Object-Oriented Design metric suite (QMOOD) metrics [14], two Martin metrics [15], some metrics from the extended CK metric suite [16], two McCabe metrics (the maximum Cyclomatic complexity (Max_cc), average cyclomatic complexity (Avg_cc)) and Lines of Code (LOC). CK suite includes Coupling Between Objects (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM), Number of Children (NOC), Weighted Methods of a Class (WMC), and Depth of Inheritance Tree (DIT). Selected QMOOD metrics are Cohesion among Methods of a class (CAM) Method of Functional Abstraction (MFA), Measure of Aggression (MOA), Number of Public Methods (NPM), and Data Access Metric (DAM). Efferent Coupling (Ce) and Afferent Coupling (Ca) are Martin metrics used in this study. Metrics taken from the extended CK metric suite are Coupling Between Methods of a Class (CBM), Inheritance Coupling (IC), Average Method Complexity (AMC), and a variant of LCOM (LCOM3).

The dependent variable of this study is a binary variable ‘defect’. It is obtained by converting the continuous attribute ‘Number of Defects’ in the dataset into the binary representation of ‘yes’ or ‘no’. If the attribute has value 0, then replace it with value ‘No’ else replace it with ‘Yes’. The definitions and details of metrics can be referred from http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html.
3.3. Feature Selection
In presence of a large number of metrics, training the model with a complete set of metrics may be a time-consuming task and can even hamper the model performance. This problem of high dimensionality is referred to as the curse of dimensionality [17]. One possible and preferred solution to this problem is to select only important features and remove redundant or irrelevant metrics. This paper uses Correlation Feature Selection (CFS) [18] for selecting important features. CFS is the most preferred feature selection technique [17, 19] in the literature and is based on univariate analysis. Different types of metrics are selected depending on the nature of the software. Features selected by CFS for the considered datasets are mentioned in Table 2.

| Dataset  | CFS selected features                      |
|----------|--------------------------------------------|
| Jedit4.2 | CBO, RFC, Ca, Ce, CBM, LOC, AMC, NPM, LCOM3, LCOM, CAM, MOA, Max_CC |
| Xerces1.3| AMC, WMC, DAM, IC, Ce, CBM, LCOM, MOA       |
| Ant1.7  | RFC, LOC, AMC, Max_CC, LCOM, CAM, Ce, CBO, MOA |

Now, the models built with help of these selected metrics will tend to classify defects appropriately and in less computation time.

3.4. ML Techniques
Various ML techniques have been used in literature. J48 (or C4.5) can generate good prediction models in case of imbalanced data [20] and has been recognized as one of the top 10 ML techniques [21]. Therefore, as the base learner, we opted for J48 and compared it with three ensembles: boosting-based ABM1, bagging-based Bag, and random forest-based RSS.

3.4.1. J48 [22]. It is a JAVA implementation of the C4.5 decision tree. It follows the greedy approach and split on a decision node based on the gain ratio.

3.4.2. AdaboostM1(ABM1) [22, 23]. The paragraph text ABM1 is an ensemble method based on sequential boosting. In each iteration, the base decision tree learns from the previous tree about misclassified instances. Equal weights are assigned to training data. This weight increases or decreases in the next iteration based on whether that instance is correctly classified or not.

3.4.3. Bagging (Bag) [22, 24]. It is also an ensemble method that enhances the predictive capability of base classifiers by making bags of training data. Models work in parallel and their results are averaged. Bagging reduces the variance.

3.4.4. RandomSubSpace (RSS) [25]. RSS generates random feature subsets to build multiple trees. Tree construction is mainly dependent on Bagging with Reptree. Multiple trees are generated in each iteration and the tree with the best performance is selected. This results in an effective random forest.

3.5. Cost-sensitive Classification
Cost-sensitive classification can be conducted in two ways; either by adding weights to samples or by using cost matrix to penalize type-I and type-II error. Meta cost learners were proposed by [26] in which penalization is done in cost matrix and training instance is relabeled based on the majority voting. These errors are actually false positives and false negatives in the model prediction. In this work, meta-cost classifiers are used and the cost penalization of wrongly predicted defective classes is done at different levels- 5, 10,15, 20, 20, 40, and 50 times the cost of wrongly predicted non-defective classes.

3.6. Performance Measures
In the study, SDP is treated as a binary classification problem with two possible outcomes—defective and non-defective class. Though traditional metrics like accuracy, precision are avoided in the case of imbalanced data [27, 28], we still used sensitivity as it represents the ratio of correctly predicted defective classes to the actual defective classes in the software. Sensitivity or True Positive rate (TPR) is defined as

\[
\text{Sensitivity} = \frac{TP}{TP + FN}
\]

Stable metrics like GMean, Balance, and AUC are used. GMean is considered a stable performance measure because it considers both defective and non-defective classes for the correct predictions [29]. GMean is defined as the geometric mean of sensitivity and specificity. Specificity represents the proportion of non-defective classes that are correctly predicted to the actual non-defective classes in software. GMean is defined as

\[
\text{GMean} = \sqrt{\text{Sensitivity} \times \text{Specificity}} \quad \text{where} \quad \text{Specificity} = \frac{TN}{TN + FP}
\]

Balance is also an important performance measure used in the SDP field [30]. It is measured as the Euclidean distance between a pair of TPR (sensitivity) and False Positive Rate (FPR) as given in Eq(3).

\[
\text{Balance} = 1 - \sqrt{\frac{(0 - FPR)^2 + (1 - \text{Sensitivity})^2}{2}} \quad \text{where} \quad FPR = \frac{FP}{FP + TN}
\]

Area Under Curve (AUC) is a stable metric as it depicts how well the model can discriminate between the defective and non-defective classes and, more importantly, it is threshold dependent [31]. This measure is extensively explored in research for it can handle skewed data [32, 33]. The value of AUC ranges from 0 to 1.

### 3.7. Validation Techniques

Data is split into training and testing subsets of samples and model validation are carried out by stratified ten-fold cross-validation. The use of ten-fold cross-validation reduces validation bias [34]. Stratified ten-fold generation emphasizes on equal distribution of minority class instances (defective classes) in all ten folds. One fold act as testing data and the remaining nine folds contains training data. The process is repeated ten times and values are averaged at the end.

Statistical validation of results enhances the result validation. We have used the Friedman test and Wilcoxon signed-rank test for the purpose because both these tests are non-parametric in nature. In nonparametric tests, data does not require to follow any presumptions. These tests work well for data that have outliers or cases where data distribution is not normal [35]. Friedman test [36] uses a ranking method to evaluate performances of different models based on the performance metrics. If the Friedman test shows significant results then post hoc analysis is done using the Wilcoxon signed-rank test [37]. This test aids in eliminating family-wise errors.

### 4. Result and Analysis

This section summarizes the results and provides answers to the research questions raised in the Introduction section.

#### 4.1. RQ1: What is the performance of used ML techniques without cost-sensitive classification?

J48 and three ensemble-based classifiers are used to predict probable defects in imbalanced datasets—Jedit4.2, Xerces1.3, and Ant1.3. Performances of these models are compared on basis of AUC,
Balance, GMean, and sensitivity. These values are recorded in Table 3. The maximum value of a particular performance measure is highlighted in bold typeface for each dataset.

From Table 3 this can be observed that AUC values lie in the range of 0.6-0.84. The median AUC value attained for all the models is 0.79. It can be deduced from Table 3 that the three ensembles performed better than J48. The minimum value of AUC in the case of ensembles is 0.60 whereas the minimum value attained by J48 is 0.60 for Xerces1.3. This confirms a 25% increase in minimum AUC value. RSS achieved maximum AUC value in all the three datasets. For Balance, value ranges from 38.12 to 68.55 with a median value of 56.21. Similarly, GMean displays 0.35 as the minimum value attained in SDP models. This value is comparatively low. Though the maximum GMean achieved is 0.71 but its mean value could reach up to 0.58 only. The range of sensitivity lies between 12.5 and 56.63. One of the primary reasons for low values of sensitivity is the imbalanced class problem.

This can be deduced from Table 3 that Jedit4.2 has comparatively lower values of sensitivity, GMean, and Balance values as compared to Xerces1.3 and Ant1.7. The reason may be that it has the highest imbalance ratio amongst other datasets. It has only 13.1% of defective classes.

### Table 3. Performance Evaluation of J48 and ensemble methods without Cost-sensitive Classification

| Performance Measure | ML Techniques | Jedit4.2 | Xerces1.3 | Ant1.7 |
|---------------------|---------------|----------|-----------|--------|
| **AUC**             | J48           | 0.69     | 0.60      | 0.74   |
|                     | ABM1          | 0.75     | 0.76      | 0.78   |
|                     | BAG           | 0.82     | 0.80      | 0.80   |
|                     | RSS           | **0.84** | **0.83**  | **0.80** |
| **Balance**         | J48           | 45.39    | 54.8      | 68.55  |
|                     | ABM1          | **55.58**| **58.71** | 62.07  |
|                     | BAG           | 51.31    | 56.84     | 63.32  |
|                     | RSS           | 38.12    | 47.73     | 62.13  |
| **GMean**           | J48           | 0.47     | 0.59      | **0.71** |
|                     | ABM1          | **0.59** | **0.63**  | 0.65   |
|                     | BAG           | 0.55     | 0.61      | 0.67   |
|                     | RSS           | 0.35     | 0.51      | 0.66   |
| **Sensitivity**     | J48           | 22.92    | 36.23     | 56.63  |
|                     | ABM1          | **37.50**| **42.03** | 47.59  |
|                     | BAG           | 31.25    | 39.13     | **48.80** |
|                     | RSS           | 12.50    | 26.09     | 46.99  |

4.2. **RQ2: Do cost-sensitive classification improve the defect prediction capability of ML models build for imbalanced data?**

To answer this RQ, the cost matrix of each classifier is penalized. Different cost factors of 5, 10, 15, 20, 30, 40, and 50 are tried and the program was designed to select the cost factor that yields the best defect prediction. Table 4 holds the performance statistics of four classifiers on three datasets corresponding to the best cost factor settings. The results seem promising in the field of SDP for imbalanced data. Now, this can be observed that the pattern of increasing sensitivity, GMean, and Balance based on the percentage of defective classes is missing in Table 4. Meta cost learners have handled the class imbalance issue with cost penalization.

By analyzing Table 4, we found that the mean AUC value becomes 0.81 and AUC ranged between 0.75 and 0.85. The range of Balance now becomes 59.98-79.91. GMean values also show remarkable improvement and ranged from 0.64 to 0.80. Comparison of Table 4 with Table 3 exhibits the improvement in values of all the performance metrics; AUC, Balance, GMean, and sensitivity.

Because of cost-sensitive classification, the minimum value of sensitivity increased by 250%, and the mean value of sensitivity also demonstrated a 100.8% increase. The value of minimum GMean
predicted by models amplified by 82% whereas the minimum Balance value raised by 57.3%. The mean values for GMean and Balance boosted by 25.5% and 30% respectively.

Table 4. Performance Evaluation of J48 and ensemble methods with Cost-sensitive Classification

| Performance Measure | ML Techniques | Jedit4.2 | Xerces1.3 | Ant1.7 |
|---------------------|---------------|---------|-----------|--------|
|                     | J48           | ABM1    | BAG       | RSS    |
| AUC                 | 0.77          | 0.81    | 0.85      | 0.85   |
|                     | 0.78          | 0.81    | 0.83      | 0.83   |
|                     | 0.75          | 0.79    | 0.82      |        |
| Balance             | 75.32         | 73.28   | 68.58     |        |
|                     | 59.98         | 71.24   | 70.25     |        |
|                     | 79.19         | 72.26   | 73.38     |        |
|                     | 77.79         | 74.82   | 67.95     |        |
| GMean               | 0.75          | 0.73    | 0.70      |        |
|                     | 0.64          | 0.72    | 0.72      |        |
|                     | 0.80          | 0.73    | 0.74      |        |
|                     | 0.78          | 0.75    | 0.70      |        |
| Sensitivity         | 77.08         | 75.36   | 83.13     |        |
|                     | 43.75         | 63.77   | 60.84     |        |
|                     | 87.5          | 82.61   | 77.11     |        |
|                     | 79.17         | 81.16   | 87.35     |        |

These results are still required to be statistically validated. We perform a statistical test to compare the cumulative performance of ML techniques without cost-sensitive learning (NotCS) with the cumulative performance of ML techniques with cost-sensitive learning (CS). This requires to compare only two scenarios, we employed the Wilcoxon-signed rank test in the SPSS21 tool and found considerable statistical improvement in SDP models using cost-sensitive classification for all the performance measures. Wilcoxon signed-ranks are scribed in Table 5 with its p-value. A p-value of less than 0.01 indicates that results are 99% statistically significant.

Table 5. Wilcoxon signed-rank results for various performance measures

|        | AUC | GMean | Balance | Sensitivity |
|--------|-----|-------|---------|-------------|
| CS vs NotCS | +  | +  | +      | +           |
| p-value   | 0.002 | 0.002 | 0.003   | 0.002       |

Therefore, it is concluded that cost-sensitive classification enhances the defect prediction capability of the ML models built for imbalanced data.

4.3. RQ3: Which ML technique outperforms in predicting software defects?

If we refer to Table 3 in the NotCS case RSS performed better only in the case of AUC performance measure. For Balance and GMean RSS and Bag did not perform well because of low sensitivity values in these cases. The reason for low sensitivity is less number of defective classes. In contrast, when cost-sensitive classification is performed, Bag and RSS performed slightly better than other ML techniques. All four classifiers belong to the family of decision trees. To see whether there is a statistical change in performances of different classifiers, the Friedman test is performed and their ranks for various performance measures are recorded in Table 6.
Table 6. Friedman rankings of ML techniques

|       | AUC | Balance | GMean | Sensitivity |
|-------|-----|---------|-------|-------------|
| J48   | 1.00| 2.67    | 2.67  | 2.67        |
| ABM1  | 2.00| 3.00    | 3.00  | 3.33        |
| BAG   | 3.00| 3.00    | 3.00  | 3.00        |
| RSS   | 4.00| 1.33    | 1.33  | 1.00        |
| p-value | .029| .334    | .334  | .122        |
| J48_CS| 1.00| 2.33    | 2.00  | 2.33        |
| ABM1_CS| 2.00| 1.67    | 1.33  | 1.00        |
| BAG_CS| 3.67| 3.33    | 2.67  | 3.33        |
| RSS_CS| 3.33| 2.67    | 4.00  | 3.33        |
| p-value | .042| .457    | .072  | .086        |

Higher the Friedman rank, the better the performance of the classifier. The highest rank is boldfaced for every performance measure. The Friedman ranks are computed at 95% confidence level and ranks are statistically significant only when p-value <= 0.5. Analysis of Table 5 enlightens the fact that only AUC has a p-value of less than 0.05 for ML techniques implemented without using cost-sensitive classification (NotCS) and ML techniques with cost-sensitive classification (CS). AUC results are 95% significant but for the rest of the performance measures, there is no statistical difference between the performances of four classifiers. When the cost matrix is penalized, Bag and RSS have better performances but not statistically better.

5. Validity Threats
Stratified ten-fold cross-validation and nonparametric statistical tests are used in this study to alleviate the conclusion validation threat. For construct validity, correct identification of independent and dependent variables along with the relation between them needs to be established carefully and we used OO metrics as independent variables. These datasets and OO metrics have been widely accepted in the research area. This enhances construct validation. The use of CFS and stable metrics helps to mitigate internal validity threats. We have tried to achieve generalization by using Apache datasets instead of NASA datasets.

6. Conclusion
This paper determines the usage of the penalized cost matrix for effective defect prediction in imbalanced software data. The study incorporates model construction with stratified ten-fold validation on four ML methods: J48, AdaboostM1, Bagging and RSS, and three datasets of the imbalanced nature. Important features were retained by employing the CFS technique. The empirical evaluation was executed with the help of AUC, GMean, Balance, and sensitivity.

The main contribution of the paper is
- to develop a useful and efficient SDP model for imbalanced data
- to use stable performance measures (GMean, Balance, and AUC) for model evaluation
- to statistically validate the results using nonparametric tests
- To incorporate CFS in defect prediction models to provide better and unbiased results.

Therefore, this paper helps in understanding the role of cost-sensitive classification and building better SDP models with imbalanced software data.

Future directions include working toward generalization of results by including more datasets and ML techniques. Instead of JAVA, other language projects can also be considered. This cost-sensitive classification can be extended to multiclass problems also.

References
[1] Malhotra R 2016 Empirical research in software engineering: concepts analysis and
applications (Florida: CRC press)

[2] Basili V R, Briand L C and Melo W L 1996 A validation of object-oriented design metrics as quality indicators IEEE Trans. Software Eng. 22(10) pp 751-76

[3] Singh Y, Kaur A and Malhotra R 2010 Empirical validation of object-oriented metrics for predicting fault proneness models Software Qual. J. 18(1) p 3

[4] Lingden P, Alsadoon A, Prasad P W, Alsadoon O H, Ali R S and Nguyen V T 2019 A novel modified undersampling (MUS) technique for software defect prediction Comput. Intell. 35(4) pp 1003-20

[5] Bejjanki KK, Gyani J and Gugulothu N 2020 Imbalance Reduction (CIR): A novel approach to software defect prediction in the presence of class imbalance Symmetry 12 (3) p 407

[6] Arar ÖF and Ayan K 2015 Software defect prediction using cost-sensitive neural network Appl. Soft Comput. 33 pp 263-77

[7] Siers M J and Islam M Z 2015 Software defect prediction using a cost sensitive decision forest and voting and a potential solution to the class imbalance problem Inform. Syst. 51 pp 62-71

[8] Siers M J and Islam M Z 2016 Addressing class imbalance and cost sensitivity in software defect prediction by combining domain costs and balancing costs Proc. 12th Int. Conf. on Advanced Data Mining and Applications (Gold Cost: Springer) pp 156-71

[9] Rodriguez D, Herraiz I, Harrison R, Dolado J and Riquelme J C 2014 Preliminary comparison of techniques for dealing with imbalance in software defect prediction Proc. 18th Int. Conf. on Evaluation and Assessment in Software Engineering (United States: IEEE) pp 1-10

[10] Malhotra R and Kamal S 2019 An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data Neurocomputing 343 pp 120-140

[11] Jureczko M and Madeyski L 2010 Towards identifying software project clusters with regard to defect prediction Proc. 6th Int. Conf. on Predictive Models in Software Engineering (United States: ACM) pp I-10

[12] Menzies T, Krishna R and Pryor D 2015 The promise repository of empirical software engineering data North Carolina State University Department of Computer Science [Online] http://opencursive/rep access 15th May 2017

[13] Chidamber S R and Kemmerer C F 1994 A metrics suite for object oriented design IEEE Trans. Software Eng. 20(6) pp 476-93

[14] Banskiya J and Davis C G 2002 A hierarchical model for object-oriented design quality assessment IEEE Trans. Software Eng. 28(1) pp 4–17

[15] Martin R C 1995 Object oriented design quality metrics: an analysis of dependencies Rep. Object Anal. Design 2(3) pp 1–8

[16] B Henderson-Sellers 1995 Object-oriented metrics: measures of complexity (United States: Prentice-Hall)

[17] Ghota B McIntosh S and Hassan A E 2017 A large-scale study of the impact of feature selection techniques on defect classification models Proc. 14th IEEE/ACM Int. Conf. on Mining Software Repositories (MSR) (Buenos Aires: IEEE press) pp 146-57

[18] Hall M A 1999 Correlation-based feature selection for machine learning PhD dissertation (The University of Waikato Hamilton New Zealand)

[19] Malhotra R 2015 A systematic review of machine learning techniques for software fault prediction Appl. Soft Comput. 27 pp 504–18

[20] Song Q, Yuchen G and Shepperd M 2008 A comprehensive investigation of the role of imbalanced learning for software defect prediction IEEE Trans. Software Eng. 45(12) pp 1253-69

[21] Wu X, Kumar V, Quinlan J R, Ghosh J, Yang Q, Motoda H, McLachlan G J, Ng A, Liu B, Philip S Y and Zhou Z H 2008 Top 10 algorithms in data mining Know. Inf. Syst. 14(1) pp 1–37

[22] Quinlan J 2014 C4.5: programs for machine learning (United States: Elsevier)
[23] Freund Y and Schapire R E 1996 Experiments with a new boosting algorithm Proc. 13th Int. Conf. on Machine Learning (San Francisco: ACM) pp 148-56
[24] Breiman L 1996 Bagging predictors Mach. Learn. 24(2) pp 123-40
[25] Ho T K 1998 The random subspace method for constructing decision forests IEEE T. Pattern Anal. 20(8) pp 832-44
[26] Domingos P 1999 Metacost: a general method for making classifiers cost-sensitive Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (San Diego: ACM) pp 155–64
[27] Menzies T, Dekhtyar A, Distefano J and Greenwald J 2007 Problems with precision: a response to comments on data mining static code attributes to learn defect predictors IEEE Trans. Software Eng. 33(9) pp 637–40
[28] Gao K, Khoshgoftaar T M and Napolitano A 2015 Combining feature subset selection and data sampling for coping with highly imbalanced software data In Proc. of the Software Engineering Knowledge Engineering Conf. pp 439–44
[29] Malhotra R, Khanna M and Raje R R 2017 On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions Swarm Evol. Comput. 32 pp 85-109
[30] Li M, Zhang H, Wu R and Zhou Z H 2012 Sample-based software defect prediction with active and semi-supervised learning Automat. Softw. Eng. 19(2) pp 201-30
[31] Tantithamthavorn C and Hassan A E 2018 An experience report on defect modelling in practice: pitfalls and challenges Proc. 40th Int. Conf. on Software Engineering: Software Engineering in Practice (Gothenburg: ACM) pp. 286-295
[32] Fawcett T 2006 An introduction to ROC analysis Pattern Recogn. Lett. 27(8) pp 861-74
[33] Malhotra R and Khanna M 2017 An empirical study for software change prediction using imbalanced data Empir. Softw. Eng. 22(6) pp 2806-51
[34] Ma Y, Luo G, Zeng X and Chen A 2012 Transfer learning for cross-company software defect prediction Inform Software Tech. 54(3) pp 248–56
[35] Demšar J 2006 Statistical comparisons of classifiers over multiple data sets J. Mach. Learn. Res. 7(1) pp 1–30
[36] Friedman M 1937 The use of ranks to avoid the assumption of normality implicit in the analysis of variance J. Am. Stat. Assoc. 32(200) pp 675-701
[37] Wilcoxon F, Katti S K and Wilcox R A 1970 Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test Selected tables in mathematical statistics 1 pp 171-259