# Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction

Daniel Rodriguez
Dept of Computer Science
University of Alcala
28871 Alcalá de Henares,
Madrid, Spain
daniel.rodriguezg@uah.es

Israel Herraiz
Technical Univ of Madrid
(UPM)
28040 Madrid, Spain
(Now at AMADEUS)
isra@herraiz.org

Rachel Harrison
School of Technology
Oxford Brookes University
Oxford OX33 1HX, UK
rachel.harrison@brookes.ac.uk

Javier Dolado
Dept of Computer Science
Univ of the Basque Country
20018 Donostia, Spain
javier.dolado@ehu.es

José C. Riquelme
Dept of Computer Science
University of Seville
41012 Seville, Spain
riquelme@us.es

## ABSTRACT

Imbalanced data is a common problem in data mining when dealing with classification problems, where samples of a class vastly outnumber other classes. In this situation, many data mining algorithms generate poor models as they try to optimize the overall accuracy and perform badly in classes with very few samples. Software Engineering data in general and defect prediction datasets are not an exception and in this paper, we compare different approaches, namely sampling, cost-sensitive, ensemble and hybrid approaches to the problem of defect prediction with different datasets preprocessed differently. We have used the well-known NASA datasets curated by Shepperd et al. There are differences in the results depending on the characteristics of the dataset and the evaluation metrics, especially if duplicates and inconsistencies are removed as a preprocessing step.

Further Results and replication package:
`http://www.cc.uah.es/drg/ease14`

## Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

## General Terms

Measurement, Experimentation

## Keywords

Defect Prediction, Imbalanced data, Data Quality


*EASE* '14, May 13 - 14 2014, London, England, BC, United Kingdom

## 1. INTRODUCTION

Most publicly available datasets in software defect prediction are highly imbalanced, i.e., samples of non-defective modules vastly outnumber the defective ones. In this situation, many data mining algorithms generate poor models because they try to optimize the overall accuracy but perform badly in classes with very few samples. For example, if the number of non-defective samples outnumbers the defective samples by 95%, an algorithm that always predicts a module as non-defective will obtain a very high accuracy. As a result, many data mining algorithms obtain biased models that do not take into account the minority class (defective modules). When the problem of imbalanced data is not considered, many learning algorithms generate distorted models for which (i) the impact of some factors can be hidden and (ii) the prediction accuracy can be misleading. This is due to the fact that most data mining algorithms assume balanced datasets. The imbalance problem is known to affect many machine learning algorithms such as decision tress, neural networks or support vectors machines [22]. When dealing with imbalanced datasets, there are different alternatives, either bootstrapping, sampling or balancing techniques, cost-sensitive evaluation, ensembles to wrap multiple classifiers making them more robust to the imbalance problem or hybrid techniques.

In this paper, we compare different types of algorithms to deal with imbalanced data in the domain of defect prediction using well-known NASA datasets considering (i) different evaluation metrics including the Matthew's Correlation Coefficient which has not been used in the domain of software defect prediction and (ii) with different cleaning processes (in particular considering the removal of duplicates and inconsistencies).

The rest of the paper is organised as follows. Section 2 describes the related work. Next, Section 3 briefly describes the approaches to deal with imbalanced data, followed by the experimental work in Section 4. Finally, Section 5 concludes the paper and outlines future work.

## 2. RELATED WORK

Defect prediction has been an important research topic

for more than a decade with an increasing number of papers including two recent systematic literature reviews [5, 19].

Many classification techniques for defect prediction have been proposed, including techniques which originated from the field of statistics (regression [2], and Support Vector Machines [12], etc.), machine learning techniques (such as classification trees [23]), neural networks [24], probabilistic techniques (such as Naïve Bayes [34] and Bayesian networks), ensembles of different techniques and metaheuristic techniques such as ant colonies [45], etc. However, there are discrepancies among the outcomes as: (i) no classifier is consistently better than the others; (ii) there is no optimum metric to evaluate and compare classifiers as highlighted in the following papers [31, 48, 34]; (iii) there are quality issues regarding the data such as imbalanced class overlaps, outliers, transformation issues, etc.

Seiffert et al. [41] analysed 15 software-quality data sets of different sizes and levels of imbalance (including CM1, KC1, KC2, KC3, MW1 and PC1 from NASA datasets). For the NASA datasets, the authors used Halstead and McCabe base metrics together with some lines of code metrics, i.e., attributes were manually selected. The authors concluded that boosting almost always outperforms sampling techniques using the area-under-the-curve (AUC) and Kolmogorov-Smirnov (K-S) statistic as performance measure.

Khoshgoftaar et al. [25] also highlighted the problem of imbalanced datasets when dealing with defect prediction and used Case-Based Reasoning to deal with this problem.

In relation to the performance and evaluation of the classifiers, there is no technique that has consistently performed better than others. Several papers have compared multiple techniques with multiple evaluation measures. For example, Peng [37] proposes a performance metric to evaluate the merit of classification algorithms using a broad selection of classification algorithms and performance measures. The experimental results, using 13 classification algorithms with 11 measures over 11 software defect datasets, indicate that the classifier which obtains the best result for a given dataset according to a given measure may perform poorly with a different measure. The results of the experiment indicate that Support Vector Machines, the $k$-nearest neighbor algorithm and the C4.5 algorithm were ranked as the top three classifiers. Also Peng et al. [38] used ten NASA datasets to rank classification algorithms, showing that a CART boosting algorithm and the C4.5 decision-tree algorithm with boosting are ranked as the optimum algorithms for defect prediction. Lessman et al. [26] also compared 22 classifiers grouped into statistical, nearest neighbour, neural networks, support vector machine, decision trees and ensemble methods over ten datasets from the NASA repository. The authors discuss several performance metrics such as $TP_r$ and $FP_r$ but advocate the use of Area Under the ROC (AUC) as the best indicator for comparing the different classifiers. This is known not to be optimal in the case of highly imbalanced datasets.

Arisholm et al. [1] compared a classification tree algorithm (C4.5), a coverage rule algorithm (PART), logistic regression, back–propagation neural networks and Support Vector Machines over 13 releases of a Telecom middleware system developed in Java using three types of metrics: (i) object-oriented metrics, (ii) churn (delta) metrics between successive releases, and (iii) process management metrics from a configuration management system. The authors concluded that although there are no significe differences regarding the techniques used, large differences can be observed depending on the criteria used to compare them. The authors also propose a new cost-effectiveness metric based on AUC and number of statements so that larger modules are more expensive to test. The same approach of considering module size in conjunction with the AUC as evaluation metrics has been explored by Mende and Koschke [32] using NASA datasets and three versions of Eclipse with random forests [4] as the classification technique.

# 3. APPROACHES TO DEAL WITH IMBALANCED DATA

We can classify the different approaches to deal with imbalanced data as sampling, cost-sensitive, ensemble approaches or hybrid approaches. We next briefly describe the techniques used in this work.

## 3.1 Sampling Techniques

Sampling techniques are classified as oversampling or undersampling and are based on adding or removing instances of the training dataset as a preprocessing step. The simple procedure of replicating instances from the minority class towards a more balanced distribution is called Random Over-Sampling (ROS), whereas Random Under-Sampling (RUS) removes instances from the majority class.

There are more sophisticated or intelligent approaches to the generation of new artificial instances rather than the replication of instances. The most popular technique is called SMOTE (Synthetic Minority Over-sampling Technique) [8] which generates new instances based on a number of nearest neighbours. Other techniques that remove instances intelligently include the Edited Nearest Neighbour (ENN) and Wilson's Editing that remove instances in which close neighbours belong to a different class [46].

## 3.2 Cost-Sensitive Classifiers

Cost-Sensitive Classifiers (CSC) adapt classifiers to handle imbalanced datasets by either (i) adding weights to instances (if the base classifier algorithm allows this) or resampling the training data according to the costs assigned to each class in a predefined cost matrix, or (ii) generating a model that minimises the expected cost (multiplying the predicted probability distribution with the misclassification costs). The idea is to penalise differently the different types of error (in binary classification, the false positives and false negatives).

The problem with CSC is defining the cost matrix as there is no systematic approach to do so. However, it is common practice to set the cost to equalize the class distribution.

## 3.3 Ensembles

Ensembles or meta-learners combine multiple models to obtain better predictions. They are typically classified as *Bagging*, *Boosting* and Stacking (Stacked generalization). We have used Bagging and Boosting algorithms in this work.

Bagging [3] (also known as Bootstrap aggregating) is an ensemble technique in which a base learner is applied to multiple equal size datasets created from the original data using bootstraping. Predictions are based on voting of the individual predictions. An advantage of bagging is that it does not require any modification to the learning algorithm and

takes advantage of the instability of the base classifier to create diversity among individual ensembles so that individual members of the ensemble perform well in different regions of the data. Bagging does not perform well with classifiers if their output is robust to perturbation of the data such as nearest-neighbour (NN) classifiers.

Boosting techniques generate multiple models that complement each other inducing models that improve regions of the data where previous induced models preformed poorly. This is achieved by increasing the weights of instances wrongly classified, so new learners focus on those instances. Finally, classification is based on a weighted voted among all members of the ensemble. In particular, AdaBoost.M1 [15] is a popular boosting algorithm for classification. The set of training examples is assigned an equal weight at the beginning and the weight of instances is either increased or decreased depending on whether the learner classified that instance incorrectly or not. The following iterations focus on those instances with higher weights. AdaBoost.M1 can be applied to any base learner.

Rotation Forests [40] combine randomly chosen subsets of attributes (*random subspaces*) and bagging approaches with principal components feature generation to construct an ensemble of decision trees. Principal Component Analysis is used as a feature selection technique combining subsets of attributes which are used with a bootstrapped subset of the training data by the base classifier.

A problem with ensembles is that their models are difficult to interpret (they behave as blackboxes) in comparison to decision trees or rules which provide an explanation of their decision making process.

## 3.4 Hybrid Approaches

The SMOTEBoost goal is to reduce the bias inherent in the learning procedure due to the class imbalance, and increase the sampling weight for the minority class. Introducing SMOTE [6] in each round of boosting will enable each learner to be able to sample more of the minority class cases, and also learn better and broader decision regions for the minority class. Using SMOTE in each round of boosting enhances the probability of selection for the difficult minority class cases that are dominated by the majority class points [7]. SMOTEBoost is a combination of SMOTE and a boosting procedure, in this case, a variant of the AdaBoost.M2 procedure [14].

RUSBoost [42] is based on the AdaBoost.M2 and so SMOTEBoost and RUSBoost are similar. In SMOTEBoost, SMOTE is applied to the training data during each round of boosting. The difference is that RUSBoost applies Random Under Sampling instead of SMOTE. The application of SMOTE at this point has two drawbacks that RUSBoost is designed to overcome. First, it increases the complexity of the algorithm, SMOTE must find the $k$ nearest neighbours of the minority class examples, and extrapolate between them to make new synthetic examples. RUS, on the other hand, simply deletes the majority class examples at random. Second, since SMOTE is an oversampling technique, it results in longer model training times. On the other hand, RUS produces smaller training data sets and, therefore, shorter model training times [42]. Like SMOTEBoost, RUSBoost combines Boosting and filtering, but it uses RUS instead of the SMOTE as filter.

MetaCost [11] combines bagging with cost-sensitive classification. Bagging is used to relabel training data so that each training instance is assigned the prediction that minimizes the expected cost. Based on the modified training data, MetaCost induces a single new classifier based on the new relabeled data which provides information about how a decision was reached.

# 4. EXPERIMENTAL WORK

## 4.1 Datasets

In this paper, we have used available software defect prediction datasets generated from projects carried out at NASA. These datasets are available in two different versions from the PROMISE repository[1][33] and by Shepperd et al.[2] [43] who analysed different problems and differences with these datasets and curated the repository.

Table 1 shows some basic statistics for both versions of the datasets, the original one (referred to as MDP) and the PROMISE version. Both versions have been preprocessed by Shepperd *et al.* generating two new datasets for each version in which instances presenting problems such as implausible values have been removed. The difference between D' and D" is that duplicates and inconsistencies have been removed in D". Table 1 shows number of instances for each dataset, their imbalance ratio (IR), percentage of duplicates, inconsistencies (equal values for all attributes of an instance but the class) and other problems (for a complete description of problems we refer to [43]). It can be observed that most datasets are highly imbalanced, varying from less than 1% to 30% and there are large numbers of duplicates and inconsistencies in some of the datasets which seems to be the biggest problem.

All datasets contain attributes mainly composed of different McCabe [30], Halstead [20] and count metrics. The last attribute represent the class (*defective*) which has 2 possible values (whether a module has reported defects or not). The McCabe metrics are based on the count of the number of paths contained in a program based on its graph. Halstead's *Software Science* metrics are based on simple counts of tokens grouped into (i) *operators* such as keywords from programming languages, arithmetic operators, relational operators and logical operators and (ii) *operands* that include variables and constants.

These sets of metrics (both McCabe and Halstead) have been used for QA during development, testing and maintenance. Generally, the developers or maintainers use threshold values. For example, if the cyclomatic complexity ($v(g)$) of a module is between 1 and 10, it is considered to have a very low risk of being defective; however, any value greater than 50 is considered to have an unmanageable complexity and risk. Although these metrics have been used for long time, there are no generally agreed thresholds.

Table 1 shows that there are large differences in the level of imbalance or Imbalance Ratio (IR). After the cleaning process data is more balanced. In the original datasets there are two datasets (PC2, MC1) with less than 1% of the positive cases (defective modules). But most of the problems seem to come from the fact that some datasets have many duplicates and inconsistencies. If problematic cases are removed, the percentage of imbalance is affected as is the case

[1] `https://code.google.com/p/promisedata/`

[2] `http://nasa-softwaredefectdatasets.wikispaces.com/`

**Table 1: Number of Instances, Imbalance and Problems of the Datasets**

| MDP | # Inst | % IR | % Dupl | %Incons | # Inst D' | % Probl Inst | %IR D' | # Inst D" | % Probl Inst D" | %IR D" |
|---|---|---|---|---|---|---|---|---|---|---|
| CM1 | 505 | 9.5 | 5.15 | 0 | 344 | 31.88 | 12.21 | 327 | 35.25 | 12.84 |
| JM1 | 10878 | 19.32 | 24.16 | 8.17 | 9593 | 11.83 | 18.34 | 7720 | 29.03 | 20.88 |
| KC1 | 2107 | 15.42 | 50.78 | 12.01 | 2095 | 0.57 | 15.51 | 1162 | 44.85 | 25.3 |
| KC3 | 458 | 9.39 | 2.62 | 0 | 200 | 56.33 | 18 | 194 | 57.64 | 18.56 |
| KC4 | 125 | 48.8 | 8 | 7.2 | n.a | 100 | n.a | n.a | 100 | n.a |
| MC1 | 9466 | 0.72 | 84.22 | 1.12 | 8737 | 51.14 | 0.78 | 1952 | 80.49 | 1.84 |
| MC2 | 161 | 32.3 | 2.48 | 0 | 127 | 21.12 | 34.65 | 124 | 22.36 | 35.48 |
| MW1 | 403 | 7.69 | 3.72 | 1.24 | 264 | 34.49 | 10.23 | 250 | 37.72 | 10 |
| PC1 | 1107 | 6.87 | 7.68 | 1.17 | 759 | 32.07 | 8.04 | 679 | 37.13 | 8.1 |
| PC2 | 5589 | 0.41 | 17.61 | 0 | 1493 | 72.55 | 1.07 | 722 | 86.87 | 2.22 |
| PC3 | 1563 | 10.24 | 5.05 | 0.38 | 1125 | 28.41 | 12.44 | 1053 | 31.35 | 12.35 |
| PC4 | 1458 | 12.21 | 11.39 | 0.21 | 1399 | 7.68 | 12.72 | 1270 | 12.48 | 13.86 |
| PC5 | 17186 | 3 | 91.53 | 10.04 | 16962 | 10.37 | 2.96 | 1694 | 90.23 | 27.04 |
| Avg | | 13.53 | 24.18 | 3.2 | | 35.26 | 12.25 | | 51.18 | 15.71 |
| PROMISE | # Inst | % IR | % Dupl | %Incons | # Inst D' | % Probl Inst | %IR D' | # Inst D" | % Probl Inst D" | %IR D" |
| CM1 | 498 | 9.84 | 18.88 | 0.4 | 495 | 0.6 | 9.7 | 437 | 12.25 | 10.53 |
| JM1 | 10885 | 19.35 | 24.14 | 8.17 | 9591 | 11.89 | 18.34 | 7720 | 29.08 | 0.28 |
| KC1 | 1783 | 18.28 | 60.01 | 14.19 | 2095 | 0.79 | 15.51 | 1162 | 53.11 | 25.3 |
| KC2 | 522 | 20.5 | 34.87 | 22.61 | 484 | 7.28 | 20.66 | 325 | 37.74 | 28.31 |
| KC3 | 458 | 9.39 | 37.12 | 0.44 | 458 | 6.33 | 9.39 | 324 | 31 | 12.96 |
| MC1 | 9398 | 0.72 | 84.83 | 1.13 | 8737 | 51.51 | 0.78 | 1952 | 81.07 | 1.84 |
| MC2 | 161 | 32.3 | 3.73 | 1.24 | 159 | 0 | 32.7 | 155 | 3.11 | 32.9 |
| MW1 | 403 | 7.69 | 8.93 | 1.74 | 402 | 0 | 7.71 | 375 | 6.7 | 7.47 |
| PC1 | 1109 | 6.94 | 21.64 | 1.17 | 1083 | 6.67 | 6.65 | 919 | 17.67 | 6.53 |
| PC2 | 5589 | 0.41 | 82.68 | 1.79 | 5356 | 20.81 | 0.43 | 1362 | 76.88 | 1.54 |
| PC3 | 1563 | 10.24 | 12.09 | 0.58 | 1535 | 3.45 | 10.29 | 1409 | 8.83 | 10.5 |
| PC4 | 1458 | 12.21 | 11.39 | 0.21 | 1379 | 7.68 | 12.91 | 1270 | 12.48 | 13.86 |
| PC5 | 17186 | 3 | 91.53 | 10.04 | 16962 | 10.37 | 2.96 | 1694 | 90.23 | 27.04 |
| Avg | | 11.61 | 37.83 | 4.9 | | 9.8 | 11.39 | | 35.4 | 13.77 |

**Table 2: Attribute Statistics and Problems of the Datasets**

| MDP | # Att | # Probl | % Prob | # Att D' | Removed |
|---|---|---|---|---|---|
| CM1 | 41 | 6 | 14.63 | 38 | 3 |
| JM1 | 22 | 9 | 40.91 | 22 | 0 |
| KC1 | 22 | 4 | 18.18 | 22 | 0 |
| KC3 | 41 | 3 | 7.32 | 40 | 1 |
| KC4 | 41 | 30 | 73.17 | 0 | 41 |
| MC1 | 40 | 5 | 12.5 | 39 | 1 |
| MC2 | 41 | 2 | 4.88 | 40 | 1 |
| MW1 | 41 | 4 | 9.76 | 38 | 3 |
| PC1 | 41 | 8 | 19.51 | 38 | 3 |
| PC2 | 41 | 8 | 19.51 | 37 | 4 |
| PC3 | 41 | 7 | 17.07 | 38 | 3 |
| PC4 | 41 | 11 | 26.83 | 38 | 3 |
| PC5 | 40 | 5 | 12.5 | 39 | 1 |
| Avg | | | 21.29 | | |
| PROMISE | Att | Prob | %Prob | Att D' | Removed |
| CM1 | 22 | 15 | 68.18 | 21 | 1 |
| JM1 | 22 | 16 | 72.73 | 22 | 0 |
| KC1 | 22 | 16 | 72.73 | 22 | 0 |
| KC2 | 22 | 15 | 68.18 | 22 | 0 |
| KC3 | 40 | 1 | 2.5 | 40 | 0 |
| MC1 | 39 | 4 | 10.26 | 39 | 0 |
| MC2 | 40 | 0 | 0 | 40 | 0 |
| MW1 | 38 | 0 | 0 | 38 | 0 |
| PC1 | 22 | 15 | 68.18 | 22 | 0 |
| PC2 | 37 | 3 | 8.11 | 23 | 14 |
| PC3 | 38 | 3 | 7.89 | 38 | 0 |
| PC4 | 38 | 8 | 21.05 | 38 | 0 |
| PC5 | 39 | 4 | 10.26 | 39 | 0 |
| Avg | | | 31.54 | | |

of KC1 and PC5 in both MDP and Promise repositories (see Table 1). However, we believe that there is no reason to remove duplicate and inconsistent instances provided they come from the real distribution, i.e., data points were properly collected from the data. PC5 seems to be a special case as it has many duplicates. Features with a single value can be safely removed as they do not provide any predictive capability for the classifiers. On the contrary, they *confuse* the machine learning algorithms and the learning of the model takes longer.

In both repositories after the cleaning process the percentage of positive cases tends to increase (data becomes more balanced) but in both repositories, some datasets seem to be *problematic* (also reported by Shepperd et al). In the case of KC4, it had no instances left after the cleaning process. There were 26 numerical attributes out of 41 in which all instances had 0 as values (including the Halstead base measures and and as a result all their derived metrics). Perhaps zero here meant 'missing' and a different result could have been obtained if the values had been removed before considering them as inconsistencies. This dataset was not included in the Promise repository.

## 4.2 Evaluation Measures

With dichotomous or binary classifiers, many of the performance measures can be obtained from the confusion matrix (Table 3). A widely used metric, the predictive *accuracy* (*Acc*) defined by Eq. 1 should not be used when data are highly imbalanced as it does not take into account the number of correct labels of different classes.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Other metrics should be considered. From the confusion matrix, we can obtain the *true positive rate* ($TP_r = \frac{TP}{TP+FN}$) is the proportion of positive instances correctly classified

**Table 3: Confusion Matrix for Two Classes**

| | | *Act* | | |
|---|---|---|---|---|
| | | *Pos* | *Neg* | |
| *Pred* | *Pos* | True Pos ($TP$) | False Pos ($FP$) Type I error (False alarm) | $PPV = Conf = Prec = \frac{TP}{TP+FP}$ |
| | *Neg* | False Neg ($FN$) Type II error | True Neg ($TN$) | $NPV = \frac{TN}{FN+TN}$ |
| | | $Recall = Sens = TP_r = \frac{TP}{TP+FN}$ | $Spec = TN_r = \frac{TN}{FP+TN}$ | |

(also called *recall* or *sensitivity*); the *False negative rate* ($FN_r = \frac{FN}{TP+FN}$) is the proportion of positive instances misclassified as belonging to the negative class; the *True negative rate* ($TN_r = \frac{TN}{FP+TN}$) is the proportion of negative instances correctly classified (*specificity*); and finally, the *false positive rate* ($FP_r = \frac{FP}{FP+TN}$) is the proportion of negative cases misclassified (also called *false alarm rate*). The *Positive Prediction Value*, also known as *Confidence* or *Precision* ($\frac{TP}{TP+FP}$) or the *Negative Predicted Value* ($\frac{TN}{FN+TN}$) do not either consider the $TN$ or the $TP$ respectively.

There is a trade-off between the *true positive rate* and *true negative rate* as the objective is to maximise both metrics. Another widely used metric when measuring the performance of classifiers applied to highly imbalance data is the *f-measure* ($f1$) [47] which is the harmonic median of precision and recall (Eq. 2). However, there are also some criticisims to the *f-measure* metric as it does not take into the $TN$ (negative cases).

$$f - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (2)$$

where *precision* ($\frac{TP}{TP+FP}$) is the proportion of positive predictions that are correct and *recall* is the $TP_r$ previously defined.

A more suitable and interesting performance metric for binary classification when data are imbalanced is Matthew's Correlation Coefficient ($MCC$) [29]. $MCC$ can also be calculated from the confusion matrix as shown in Eq. (3) and is simple to understand. Its range goes from -1 to +1; the closer to one the better as it indicates perfect prediction whereas a value of 0 means that classification is not better than random prediction and negative values mean that predictions are worst than random.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (3)$$

Another evaluation technique to consider when data is imbalanced is the *Receiver Operating Characteristic* ($ROC$) [13] curve which provides a graphical visualisation of the results

The Area Under the ROC Curve (AUC) also provides a quality measure between positive and negative rates with a single value. It can be calculated as Eq.(4).

$$AUC = \frac{1 + TP_r - FP_r}{2} \quad (4)$$

Similarly to ROC, another widely used evaluation technique is the Precision-Recall Curve (PRC), which depicts a trade off between precision and recall and can also be summarised into a single value as the Area Under the Precision-Recall Curve (AUPRC) [9].

## 4.3 Base Classifiers

In this work, we have used the following base learners implemented in Weka:

- C4.5 [39] (known as J48 in Weka) is a decision tree. Decision trees are constructed in a top-down approach. The leaves of the tree correspond to classes, nodes correspond to features, and branches to their associated values. To classify a new instance, one simply examines the features tested at the nodes of the tree and follows the branches corresponding to their observed values in the instance.

- The Naïve Bayes (NB) [36] uses the Bayes theorem to predict the class for each case, assuming that the predictive attributes are independent given a category. A Bayesian classifier assigns a set of attributes $A_1, \ldots, A_n$ to a class $C$ such that $P(C|A_1, \ldots, A_n)$ is maximum, that is the probability of the class description value given the attribute instances, is maximal.

## 4.4 Empirical Results

We have generated multiple results for all the techniques and metrics previously mentioned using Weka's Experimenter using 5 times 5 Cross Validation (5x5CV) over all algorithms and datasets. We next present the most relevant results together with a discussion leaving other material on the companion Web site. With respect to the algorithms, RUS and ROS were used to balance the data so that the majority and minority class represented the 75% and 25% respectively (in our case, exact balancing, 50:50 distribution, in both ROS and RUS did not perform as well as a more moderated balance). It is not trivial to set the optimal balance level. The SMOTE filter increases the minority class by 200%. It worth noting that we used Weka's filtered classifier, so that these sampling filters are applied only to the training folds of the stratified CV folds, not the whole dataset in advance (the latter approach will provide over optimistic results). For the cost-sensitive classifiers (including the *MetaCost* classifier), we used cost matrix that penalises false positives (defective modules classified as non-defective) by ten times the cost of false negatives (non-defective modules classified as defective). We choose this value as the average Imbalance Ratio (IR) is close to 10%. We also used the default parameters for the metalearners algorithms as in general they seem to work well.

Tables 4 and 5 show the results of using as base learner the C4.5 algorithm (called J48 in Weka) with the default parameters (using pruning, with the minimum number of instances per leaf as two and no Laplace smoothing). The first numerical column shows the results for the base classifier and the rest of the columns show the results of the different algorithms and whether they are statistically significant using the $t$-test at 0.10 significance level. These results were obtained with Weka's experimenter tool and we must perform further statistical analyses in our future work. There are big differences in some datasets. For example, when using J48, MC1 and PC5. In some cases such as PC2, after removing all *problematic* instances, the cost algorithms failed (we believe that after the cross validation some folds had no

instances and empty matrices operations with zero values could not be performed).

Table 5 shows the results for the PROMISE dataset. J48 results show slight differences because in this repository some attributes were removed from the most common datasets used in the literature. Consequently, the number of duplicates and inconsistencies is also affected. Again, these differences are not large as between D' and D" (with or without duplicates).

There are also differences on the results depending on the base classifier. Table 6 shows the results using Naïve Bayes as base Learner, and although there are slight improvements (not statistically significant) using sampling based techniques, the rest of the classifiers do not systematically improve the results, on the contrary. A hypothesis is that we did not carry out any kind of feature selection procedure and there is a large number of attributes in these datasets that are not independent and highly correlated. While decision trees are the most popular technique in this type of work and the feature selection is embedded as part of the algorithm (the splitting nodes), naïve Bayes could benefit of using feature selection as preprocessing technique. In any case, naïve Bayes performs quite well by itself (also highlighted by Menzies et al. [35]).

Although in general metalearners are the ones achieving the best performance, simple RUS and ROS as sampling techniques behaved well in general, especially when there are no duplicates or inconsistencies (D"). RUS has the problem that if there is high level of imbalance and a large percentage of the majority cases are removed, information is lost. However, as an advantage the algorithm's training time improves. On the other hand, ROS, replicating instances, can lead to overfitting and training takes longer. Adjusting the level of imbalance greatly affects the results (Van Hulse [44] also analysed several sampling techniques). In general, it is known that ensemble methods outperform other techniques [16] and our experiments also showed that ensemble methods (including SMOTEBoost and RUSBoost) provided better results than sampling or cost sensitive methods. However ensembles do not provide inside information about how the decision making process was reached. Knowing how the decision was reached can help to identify important metrics or which metrics (or combination of metrics) are capable of identifying defective modules. As a drawback, ensemble techniques are computationally very expensive compared with sampling or cost approaches.

There are several issues that we need to consider to extend this work:

- The level of imbalance is not the only factor that hinders the performance of the classifiers, and other factors such as the degree of data overlapping (represented as duplicates) among the classes is another factor that lead to the decrease in performance of learning algorithms. As stated by López et al. [28, 27] there are other problems: dataset shift (training and test data follow different distributions), distribution of the cross validation data, small disjuncts, the lack of density or small sample size, the class overlapping, the correct management of borderline examples or noisy data. Many of these problems are related to how to measure these data characteristics and the quality of data. For instance, Van Hulse and Khoshgoftaar [21] have looked into the how the level of noise in data (quality) impact the performance of the classifiers. As we have previously discussed, we believe that there is no reason to remove duplicates and inconsistencies if that is the real distribution of the data. However, in this case the number of duplicates and inconsistencies seems to be too large in some datasets. Furthermore, we need to analyse the duplicates as it seems that there are many duplicates within the duplicates.

- We have used the datasets without applying feature selection techniques. It is embedded in some algorithms such as C4.5 and Rotation Forest algorithms but it adds complexity to the analysis.

- There are many more base learners that we need to compare. For instance, Nearest Neighbour techniques are quite robust to the imbalance problem (as no model is in fact learnt and the classification is done based on similar instances); however, these techniques should not be used within ensembles or bagging because their output usually will not change even if the training data is modified by sampling.

- Explore different evaluation metrics and graphical representations to evaluate and compare classifiers in the case of imbalance datasets, in this work for example, MCC improvements are not as large are as with AUC. We also plan to study the evaluation measures tests to deal with the evaluation and comparison of classifiers dealing with imbalanced data.

- We have used the $t$-test to compare the performance of the classifiers. It is a standard technique provided by Weka's Experimenter, however, using parametric tests to compare classifiers is currently debated and non-parametric tests could also be used (e.g. [10, 31, 1]).

- The problem with imbalanced data is an area of active research and newer and more complex techniques and algorithms are being proposed, for instance, using genetic algorithms [17, 18].

# 5. CONCLUSIONS AND FUTURE WORK

In this paper, we compared different machine learning approaches and evaluation metrics to deal with imbalanced data in software defect prediction taking into account the quality of data which contains many duplicate instances. We used two different version of datasets originated by NASA projects which have cleaned and preprocessed by Shepperd et al. removing errors and considering the data with and without duplicates. The results showed that the algorithms to deal with imbalanced datasets (grouped into sampling, cost-sensitive, ensembles and hybrid approaches) enhanced the correct classification of the minority class (defective cases). However, the improvement was affected by the preprocessing of the data (especially if duplicates and inconsistencies were removed) and the characteristics of the datasets in addition to the level of imbalance. We believe that duplicates should not be removed if data was properly collected. However, it is not known in this case as the source code is not provided.

Future work to deal with imbalanced data will include the analysis of further algorithms as well as ways to optimise their parameters, explore different measures to both

**Table 4: Results with the MDP D' and D" Datasets**

| | | J48 | RUS | ROS | SMOTE | SMOTEBoost | RUSBoost | MetaCost | CSC-Resamp | CSC-MinCost | AdaBoostM1 | Bagging | RF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D' MCC | CM1 | 0.10 | 0.18 | 0.17 | 0.17 | 0.16 | 0.16 | 0.23 | 0.23 | 0.13 | 0.19 | 0.12 | 0.04 |
| | JM1 | 0.23 | 0.24 | 0.23 | 0.24 | 0.26 | 0.24 | 0.25 | 0.24 | 0.21 | 0.24 | 0.26 | 0.18 • |
| | KC1 | 0.28 | 0.32 | 0.30 | 0.31 | 0.33 | 0.34 ○ | 0.33 | 0.32 | 0.21 | 0.31 | 0.36 ○ | 0.31 |
| | KC3 | 0.22 | 0.25 | 0.24 | 0.29 | 0.29 | 0.26 | 0.22 | 0.24 | 0.18 | 0.24 | 0.30 | 0.28 |
| | MC1 | 0.44 | 0.20 • | 0.40 | 0.43 | 0.42 | 0.17 • | 0.35 | 0.44 | 0.41 | 0.59 ○ | 0.45 | 0.45 |
| | MC2 | 0.21 | 0.21 | 0.21 | 0.20 | 0.34 | 0.36 | 0.16 | 0.16 | 0.18 | 0.32 | 0.33 | 0.38 |
| | MW1 | 0.32 | 0.22 | 0.10 • | 0.15 | 0.19 | 0.27 | 0.22 | 0.20 | 0.31 | 0.25 | 0.20 | 0.30 |
| | PC1 | 0.24 | 0.29 | 0.24 | 0.26 | 0.29 | 0.33 | 0.29 | 0.30 | 0.25 | 0.23 | 0.25 | 0.22 |
| | PC2 | 0.00 | 0.16 ○ | 0.07 | 0.09 | 0.08 | 0.12 | 0.11 | 0.09 | 0.00 | 0.01 | 0.01 | 0.00 |
| | PC3 | 0.24 | 0.25 | 0.22 | 0.22 | 0.30 | 0.31 | 0.32 ○ | 0.29 | 0.29 | 0.29 | 0.23 | 0.19 |
| | PC4 | 0.51 | 0.52 | 0.47 | 0.52 | 0.56 | 0.55 | 0.53 | 0.51 | 0.54 | 0.53 | 0.51 | 0.54 |
| | PC5 | 0.50 | 0.52 | 0.51 | 0.54 ○ | 0.55 ○ | 0.48 | 0.56 ○ | 0.52 | 0.52 | 0.52 | 0.52 | 0.52 |
| | Avg | 0.27 | 0.28 | 0.26 | 0.29 | 0.31 | 0.30 | 0.30 | 0.29 | 0.27 | 0.31 | 0.30 | 0.28 |
| D' ROC | CM1 | 0.56 | 0.62 | 0.56 | 0.59 | 0.73 ○ | 0.74 ○ | 0.68 ○ | 0.64 | 0.57 | 0.73 ○ | 0.77 ○ | 0.75 ○ |
| | JM1 | 0.67 | 0.65 | 0.60 • | 0.66 | 0.70 ○ | 0.70 ○ | 0.67 | 0.66 | 0.63 • | 0.69 | 0.72 ○ | 0.73 ○ |
| | KC1 | 0.67 | 0.70 | 0.62 | 0.69 | 0.77 ○ | 0.77 ○ | 0.73 | 0.66 | 0.64 | 0.75 ○ | 0.81 ○ | 0.82 ○ |
| | KC3 | 0.59 | 0.61 | 0.60 | 0.65 | 0.72 ○ | 0.71 ○ | 0.65 | 0.67 | 0.59 | 0.71 | 0.69 | 0.72 |
| | MC1 | 0.77 | 0.88 ○ | 0.80 | 0.81 | 0.96 ○ | 0.93 ○ | 0.74 | 0.81 | 0.65 • | 0.94 ○ | 0.91 ○ | 0.88 ○ |
| | MC2 | 0.62 | 0.62 | 0.62 | 0.61 | 0.73 ○ | 0.73 ○ | 0.59 | 0.58 | 0.59 | 0.72 ○ | 0.72 ○ | 0.75 ○ |
| | MW1 | 0.58 | 0.63 | 0.55 | 0.59 | 0.69 | 0.72 | 0.67 | 0.63 | 0.64 | 0.67 | 0.73 ○ | 0.74 ○ |
| | PC1 | 0.70 | 0.73 | 0.59 | 0.68 | 0.83 ○ | 0.82 ○ | 0.70 | 0.68 | 0.66 | 0.82 ○ | 0.83 ○ | 0.84 ○ |
| | PC2 | 0.52 | 0.77 ○ | 0.53 | 0.56 | 0.79 ○ | 0.89 ○ | 0.63 | 0.56 | 0.50 | 0.76 ○ | 0.78 ○ | 0.70 |
| | PC3 | 0.65 | 0.68 | 0.59 | 0.64 | 0.80 ○ | 0.81 ○ | 0.72 ○ | 0.68 | 0.68 | 0.80 ○ | 0.81 ○ | 0.83 ○ |
| | PC4 | 0.77 | 0.79 | 0.70 | 0.75 | 0.93 ○ | 0.92 ○ | 0.84 | 0.81 | 0.81 | 0.92 ○ | 0.92 ○ | 0.94 ○ |
| | PC5 | 0.77 | 0.91 ○ | 0.64 • | 0.79 | 0.95 ○ | 0.96 ○ | 0.89 ○ | 0.67 • | 0.80 | 0.95 ○ | 0.96 ○ | 0.96 ○ |
| | Avg | 0.65 | 0.72 | 0.61 | 0.67 | 0.80 | 0.81 | 0.71 | 0.67 | 0.65 | 0.79 | 0.81 | 0.80 |
| D' f1 | CM1 | 0.17 | 0.29 | 0.27 | 0.28 | 0.26 | 0.24 | 0.33 | 0.33 | 0.25 | 0.24 | 0.15 | 0.06 |
| | JM1 | 0.32 | 0.37 ○ | 0.37 | 0.36 ○ | 0.38 ○ | 0.36 | 0.41 ○ | 0.40 ○ | 0.39 ○ | 0.35 | 0.33 | 0.14 • |
| | KC1 | 0.35 | 0.42 ○ | 0.40 | 0.42 ○ | 0.42 | 0.44 ○ | 0.44 ○ | 0.44 ○ | 0.37 | 0.40 | 0.41 ○ | 0.31 |
| | KC3 | 0.32 | 0.37 | 0.36 | 0.41 | 0.40 | 0.37 | 0.39 | 0.40 | 0.36 | 0.34 | 0.34 | 0.34 |
| | MC1 | 0.39 | 0.11 • | 0.38 | 0.43 | 0.42 | 0.12 • | 0.35 | 0.43 | 0.37 | 0.56 ○ | 0.39 | 0.38 |
| | MC2 | 0.45 | 0.45 | 0.45 | 0.48 | 0.57 | 0.56 | 0.54 | 0.54 | 0.52 | 0.54 | 0.52 | 0.55 |
| | MW1 | 0.33 | 0.29 | 0.19 | 0.24 | 0.25 | 0.33 | 0.30 | 0.28 | 0.34 | 0.29 | 0.23 | 0.31 |
| | PC1 | 0.27 | 0.34 | 0.30 | 0.33 | 0.34 | 0.38 | 0.34 | 0.36 | 0.29 | 0.28 | 0.26 | 0.20 |
| | PC2 | 0.00 | 0.10 ○ | 0.08 | 0.09 | 0.08 | 0.10 | 0.12 | 0.09 | 0.00 | 0.02 | 0.01 | 0.00 |
| | PC3 | 0.31 | 0.35 | 0.32 | 0.32 | 0.38 | 0.40 | 0.40 ○ | 0.39 | 0.39 ○ | 0.35 | 0.27 | 0.18 • |
| | PC4 | 0.56 | 0.58 | 0.54 | 0.59 | 0.62 | 0.61 | 0.57 | 0.57 | 0.60 | 0.58 | 0.56 | 0.57 |
| | PC5 | 0.51 | 0.49 | 0.52 | 0.55 ○ | 0.56 ○ | 0.46 | 0.55 ○ | 0.53 | 0.52 | 0.53 | 0.53 | 0.51 |
| | Avg | 0.33 | 0.35 | 0.35 | 0.38 | 0.39 | 0.36 | 0.39 | 0.40 | 0.37 | 0.37 | 0.33 | 0.30 |
| D" MCC | CM1 | 0.11 | 0.12 | 0.13 | 0.16 | 0.16 | 0.18 | 0.19 | 0.17 | 0.08 | 0.13 | 0.09 | 0.05 |
| | JM1 | 0.19 | 0.21 | 0.20 | 0.20 | 0.22 | 0.21 | 0.21 | 0.18 | 0.07 • | 0.20 | 0.22 | 0.17 |
| | KC1 | 0.23 | 0.23 | 0.23 | 0.23 | 0.29 | 0.22 | 0.19 | 0.18 | 0.07 • | 0.26 | 0.28 | 0.27 |
| | KC3 | 0.23 | 0.25 | 0.24 | 0.24 | 0.30 | 0.23 | 0.27 | 0.25 | 0.18 | 0.20 | 0.33 | 0.29 |
| | MC1 | 0.07 | 0.13 | 0.22 | 0.18 | 0.29 ○ | 0.14 | 0.15 | 0.23 | 0.08 | 0.28 | 0.07 | 0.08 |
| | MC2 | 0.21 | 0.21 | 0.21 | 0.18 | 0.35 | 0.31 | 0.21 | 0.13 | 0.15 | 0.30 | 0.36 | 0.35 |
| | MW1 | 0.16 | 0.27 | 0.18 | 0.17 | 0.31 | 0.32 | 0.26 | 0.17 | 0.24 | 0.25 | 0.37 | 0.26 |
| | PC1 | 0.23 | 0.26 | 0.27 | 0.31 | 0.33 | 0.31 | 0.27 | 0.30 | 0.22 | 0.31 | 0.26 | 0.28 |
| | PC3 | 0.22 | 0.26 | 0.21 | 0.25 | 0.30 | 0.26 | 0.29 | 0.26 | 0.28 | 0.23 | 0.22 | 0.16 |
| | PC4 | 0.46 | 0.50 | 0.45 | 0.50 | 0.54 | 0.53 | 0.50 | 0.51 | 0.51 | 0.51 | 0.52 | 0.53 |
| | PC5 | 0.33 | 0.33 | 0.33 | 0.34 | 0.39 ○ | 0.37 | 0.34 | 0.33 | 0.29 | 0.37 | 0.37 | 0.36 |
| | Avg | 0.22 | 0.25 | 0.24 | 0.25 | 0.32 | 0.28 | 0.26 | 0.25 | 0.20 | 0.28 | 0.28 | 0.25 |
| D" ROC | CM1 | 0.52 | 0.58 | 0.56 | 0.56 | 0.73 ○ | 0.71 ○ | 0.64 | 0.60 | 0.55 | 0.66 ○ | 0.72 ○ | 0.72 ○ |
| | JM1 | 0.63 | 0.63 | 0.59 | 0.63 | 0.67 ○ | 0.67 ○ | 0.63 | 0.65 | 0.55 • | 0.66 ○ | 0.69 ○ | 0.69 ○ |
| | KC1 | 0.61 | 0.61 | 0.61 | 0.62 | 0.70 ○ | 0.67 | 0.63 | 0.64 | 0.54 • | 0.69 ○ | 0.70 ○ | 0.71 ○ |
| | KC3 | 0.58 | 0.58 | 0.60 | 0.61 | 0.71 ○ | 0.68 | 0.67 | 0.68 | 0.59 | 0.68 | 0.73 ○ | 0.70 |
| | MC1 | 0.53 | 0.67 ○ | 0.61 | 0.62 | 0.83 ○ | 0.79 ○ | 0.60 | 0.63 | 0.52 | 0.80 ○ | 0.74 ○ | 0.72 ○ |
| | MC2 | 0.62 | 0.62 | 0.62 | 0.58 | 0.75 ○ | 0.74 ○ | 0.62 | 0.57 | 0.57 | 0.75 ○ | 0.75 ○ | 0.77 ○ |
| | MW1 | 0.48 | 0.66 | 0.57 | 0.59 | 0.71 ○ | 0.73 ○ | 0.69 ○ | 0.60 | 0.61 ○ | 0.68 ○ | 0.76 ○ | 0.74 ○ |
| | PC1 | 0.65 | 0.70 | 0.61 | 0.70 | 0.84 ○ | 0.82 ○ | 0.73 | 0.67 | 0.64 | 0.82 ○ | 0.83 ○ | 0.86 ○ |
| | PC3 | 0.62 | 0.66 | 0.57 | 0.62 | 0.80 ○ | 0.79 ○ | 0.69 | 0.66 | 0.67 | 0.78 ○ | 0.81 ○ | 0.82 ○ |
| | PC4 | 0.76 | 0.76 | 0.69 | 0.73 | 0.92 ○ | 0.91 ○ | 0.83 | 0.81 | 0.79 ○ | 0.91 ○ | 0.92 ○ | 0.93 ○ |
| | PC5 | 0.66 | 0.66 | 0.66 | 0.68 | 0.79 ○ | 0.78 ○ | 0.69 | 0.73 ○ | 0.66 | 0.78 ○ | 0.79 ○ | 0.80 ○ |
| | Avg | 0.61 | 0.65 | 0.61 | 0.63 | 0.77 | 0.76 | 0.67 | 0.66 | 0.61 | 0.75 | 0.77 | 0.77 |
| D" f1 | CM1 | 0.18 | 0.22 | 0.25 | 0.27 | 0.27 | 0.27 | 0.32 | 0.29 | 0.22 | 0.20 | 0.12 | 0.07 |
| | JM1 | 0.28 | 0.34 | 0.34 ○ | 0.34 ○ | 0.37 ○ | 0.34 ○ | 0.41 ○ | 0.40 ○ | 0.36 ○ | 0.34 ○ | 0.30 | 0.14 • |
| | KC1 | 0.37 | 0.37 | 0.37 | 0.43 | 0.45 ○ | 0.35 | 0.45 ○ | 0.44 ○ | 0.41 | 0.42 | 0.40 | 0.33 |
| | KC3 | 0.32 | 0.38 | 0.35 | 0.38 | 0.41 | 0.35 | 0.42 | 0.41 | 0.37 | 0.30 | 0.38 | 0.33 |
| | MC1 | 0.06 | 0.11 | 0.23 ○ | 0.18 | 0.28 ○ | 0.14 | 0.16 ○ | 0.24 ○ | 0.07 | 0.26 | 0.05 | 0.06 |
| | MC2 | 0.46 | 0.46 | 0.46 | 0.49 | 0.59 ○ | 0.52 | 0.56 | 0.53 | 0.50 | 0.51 | 0.54 | 0.51 |
| | MW1 | 0.21 | 0.34 | 0.26 | 0.25 | 0.37 | 0.38 | 0.34 | 0.27 | 0.29 | 0.30 | 0.39 | 0.28 |
| | PC1 | 0.26 | 0.32 | 0.33 | 0.36 | 0.37 | 0.36 | 0.31 | 0.35 | 0.27 | 0.34 | 0.25 | 0.25 |
| | PC3 | 0.30 | 0.36 | 0.31 | 0.34 | 0.38 | 0.35 | 0.38 | 0.36 | 0.37 | 0.30 | 0.25 | 0.13 • |
| | PC4 | 0.53 | 0.57 | 0.53 | 0.57 | 0.61 | 0.60 | 0.56 | 0.58 | 0.58 | 0.57 | 0.58 | 0.56 |
| | PC5 | 0.49 | 0.49 | 0.49 | 0.53 | 0.56 ○ | 0.52 | 0.54 ○ | 0.54 ○ | 0.51 | 0.53 | 0.51 | 0.47 |
| | Avg | 0.31 | 0.36 | 0.36 | 0.38 | 0.42 | 0.38 | 0.40 | 0.40 | 0.36 | 0.37 | 0.34 | 0.28 |

○, • statistically significant improvement or degradation

**Table 5: Results using Promise D' using J48**

| | | J48 | RUS | ROS | SMOTE | SMOTEBoost | RUSBoost | MetaCost | AdaBoostM1 | Bagging | RF |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MCC | CM1 | 0.10 | 0.13 | 0.13 | 0.13 | 0.09 | 0.15 | 0.17 | 0.11 | 0.06 | 0.00 |
| | JM1 | 0.21 | 0.24 | 0.23 | 0.23 | 0.26 | 0.25 | 0.24 | 0.25 | 0.26 | 0.16 • |
| | KC1 | 0.28 | 0.32 | 0.30 | 0.33 | 0.34 | 0.35 ○ | 0.35 ○ | 0.32 | 0.34 ○ | 0.33 ○ |
| | KC2 | 0.41 | 0.40 | 0.42 | 0.44 | 0.32 | 0.30 | 0.44 | 0.45 | 0.45 | 0.44 |
| | KC3 | 0.26 | 0.28 | 0.24 | 0.26 | 0.30 | 0.20 | 0.30 | 0.24 | 0.22 | 0.21 |
| | MC1 | 0.43 | 0.20 • | 0.39 | 0.45 | 0.39 | 0.13 • | 0.04 • | 0.58 ○ | 0.45 | 0.44 |
| | MC2 | 0.18 | 0.18 | 0.18 | 0.25 | 0.36 ○ | 0.30 | 0.17 | 0.29 | 0.23 | 0.27 |
| | MW1 | 0.27 | 0.25 | 0.19 | 0.18 | 0.26 | 0.26 | 0.23 | 0.24 | 0.20 | 0.25 |
| | PC1 | 0.34 | 0.31 | 0.33 | 0.32 | 0.35 | 0.29 | 0.32 | 0.36 | 0.38 | 0.33 |
| | PC2 | 0.00 | 0.11 ○ | 0.06 | 0.11 | 0.09 | 0.08 | 0.05 | 0.05 | 0.00 | 0.00 |
| | PC3 | 0.18 | 0.26 | 0.24 | 0.22 | 0.31 ○ | 0.32 ○ | 0.31 ○ | 0.23 | 0.20 | 0.18 |
| | PC4 | 0.46 | 0.51 | 0.45 | 0.51 | 0.55 | 0.53 | 0.52 | 0.52 | 0.52 | 0.53 |
| | PC5 | 0.50 | 0.53 | 0.52 | 0.54 | 0.56 ○ | 0.46 | 0.55 ○ | 0.52 | 0.52 | 0.53 |
| | Avg | 0.28 | 0.29 | 0.28 | 0.31 | 0.32 | 0.28 | 0.29 | 0.32 | 0.29 | 0.28 |
| ROC | CM1 | 0.55 | 0.60 | 0.57 | 0.57 | 0.69 ○ | 0.74 ○ | 0.66 | 0.68 ○ | 0.71 ○ | 0.75 ○ |
| | JM1 | 0.66 | 0.66 | 0.60 • | 0.66 | 0.70 ○ | 0.69 ○ | 0.66 | 0.69 ○ | 0.72 ○ | 0.73 ○ |
| | KC1 | 0.68 | 0.70 | 0.63 | 0.68 | 0.77 ○ | 0.77 ○ | 0.73 | 0.75 ○ | 0.80 ○ | 0.81 ○ |
| | KC2 | 0.73 | 0.71 | 0.70 | 0.72 | 0.75 | 0.77 | 0.77 | 0.76 | 0.83 ○ | 0.83 ○ |
| | KC3 | 0.60 | 0.67 | 0.53 | 0.64 | 0.79 ○ | 0.77 ○ | 0.74 ○ | 0.71 | 0.78 ○ | 0.79 ○ |
| | MC1 | 0.75 | 0.86 ○ | 0.78 | 0.81 | 0.93 ○ | 0.91 ○ | 0.51 • | 0.93 ○ | 0.91 ○ | 0.90 ○ |
| | MC2 | 0.60 | 0.60 | 0.60 | 0.64 | 0.76 ○ | 0.72 ○ | 0.62 | 0.71 ○ | 0.69 ○ | 0.71 ○ |
| | MW1 | 0.54 | 0.66 | 0.62 | 0.61 | 0.75 ○ | 0.75 ○ | 0.65 | 0.76 ○ | 0.75 ○ | 0.72 ○ |
| | PC1 | 0.65 | 0.75 ○ | 0.67 | 0.70 | 0.85 ○ | 0.82 ○ | 0.76 ○ | 0.85 ○ | 0.86 ○ | 0.84 ○ |
| | PC2 | 0.50 | 0.74 ○ | 0.38 | 0.59 | 0.74 ○ | 0.83 ○ | 0.55 | 0.75 ○ | 0.75 ○ | 0.60 |
| | PC3 | 0.62 | 0.66 | 0.57 | 0.65 | 0.81 ○ | 0.80 ○ | 0.73 ○ | 0.79 ○ | 0.81 ○ | 0.82 ○ |
| | PC4 | 0.78 | 0.79 | 0.70 | 0.74 | 0.92 ○ | 0.92 ○ | 0.85 | 0.92 ○ | 0.92 ○ | 0.94 ○ |
| | PC5 | 0.77 | 0.91 ○ | 0.65 • | 0.81 | 0.95 ○ | 0.96 ○ | 0.88 ○ | 0.94 ○ | 0.96 ○ | 0.96 ○ |
| | Avg | 0.65 | 0.72 | 0.62 | 0.68 | 0.80 | 0.80 | 0.70 | 0.79 | 0.81 | 0.80 |

○, • statistically significant improvement or degradation

**Table 6: Results with the Naïve Bayes as Base Learner**

| | | NB | RUS | ROS | SMOTE | SMOTEBoost | RUSBoost | MetaCost | CSC-Resamp | CSC-MinCost | AdaBoostM1 | Bagging | RF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MCC | CM1 | 0.21 | 0.21 | 0.21 | 0.21 | 0.17 | 0.18 | 0.20 | 0.21 | 0.21 | 0.21 | 0.22 | 0.20 |
| | JM1 | 0.22 | 0.23 | 0.22 | 0.23 ○ | 0.18 • | 0.23 | 0.24 ○ | 0.23 | 0.23 | 0.22 | 0.22 | 0.21 |
| | KC1 | 0.29 | 0.30 | 0.30 | 0.31 ○ | 0.26 | 0.27 | 0.33 | 0.31 ○ | 0.31 ○ | 0.29 | 0.30 | 0.31 |
| | KC3 | 0.26 | 0.26 | 0.28 | 0.28 | 0.27 | 0.26 | 0.21 | 0.29 | 0.29 | 0.24 | 0.29 | 0.29 |
| | MC1 | 0.20 | 0.18 | 0.19 • | 0.18 • | 0.15 • | 0.14 | 0.17 | 0.19 • | 0.19 • | 0.20 | 0.19 | 0.18 |
| | MC2 | 0.31 | 0.31 | 0.31 | 0.33 | 0.31 | 0.24 | 0.32 | 0.32 | 0.32 | 0.38 | 0.33 | 0.33 |
| | MW1 | 0.32 | 0.31 | 0.31 | 0.31 | 0.30 | 0.24 • | 0.23 • | 0.31 | 0.31 | 0.33 | 0.32 | 0.33 |
| | PC1 | 0.28 | 0.26 | 0.27 | 0.27 | 0.16 • | 0.16 • | 0.16 • | 0.27 | 0.27 | 0.26 | 0.28 | 0.27 |
| | PC2 | 0.08 | 0.13 | 0.09 | 0.09 | 0.03 | 0.13 | 0.15 | 0.08 | 0.08 | 0.06 | 0.08 | 0.09 |
| | PC3 | 0.15 | 0.23 | 0.14 | 0.13 | 0.05 | 0.08 | 0.02 • | 0.11 • | 0.11 • | 0.16 | 0.18 | 0.14 |
| | PC4 | 0.32 | 0.31 | 0.34 | 0.38 ○ | 0.28 | 0.05 • | 0.25 • | 0.35 | 0.35 | 0.37 | 0.34 | 0.28 |
| | PC5 | 0.42 | 0.44 | 0.42 | 0.42 | 0.27 • | 0.21 • | 0.41 | 0.42 | 0.42 | 0.42 | 0.42 | 0.41 |
| | Avg | 0.25 | 0.26 | 0.26 | 0.26 | 0.20 | 0.18 | 0.22 | 0.26 | 0.26 | 0.26 | 0.26 | 0.25 |
| ROC | CM1 | 0.70 | 0.71 | 0.71 | 0.71 | 0.57 • | 0.61 • | 0.70 | 0.70 | 0.61 • | 0.66 | 0.73 | 0.71 |
| | JM1 | 0.69 | 0.69 | 0.69 | 0.69 | 0.58 • | 0.60 • | 0.69 | 0.69 | 0.58 • | 0.59 • | 0.69 ○ | 0.66 • |
| | KC1 | 0.79 | 0.79 | 0.79 | 0.79 | 0.73 • | 0.63 • | 0.78 • | 0.79 | 0.65 • | 0.79 | 0.79 | 0.79 |
| | KC3 | 0.68 | 0.68 | 0.67 | 0.68 | 0.66 | 0.65 | 0.67 | 0.68 | 0.65 | 0.64 | 0.68 | 0.70 |
| | MC1 | 0.91 | 0.91 | 0.91 | 0.91 | 0.86 • | 0.75 • | 0.90 • | 0.91 | 0.78 • | 0.88 | 0.91 | 0.91 |
| | MC2 | 0.73 | 0.73 | 0.73 | 0.73 | 0.72 | 0.65 • | 0.69 • | 0.73 | 0.64 • | 0.70 | 0.73 | 0.73 |
| | MW1 | 0.75 | 0.75 | 0.75 | 0.76 | 0.75 | 0.71 | 0.72 | 0.75 | 0.71 | 0.74 | 0.75 | 0.76 |
| | PC1 | 0.78 | 0.78 | 0.79 | 0.78 | 0.67 • | 0.58 • | 0.73 • | 0.78 | 0.64 • | 0.77 | 0.80 ○ | 0.78 |
| | PC2 | 0.83 | 0.86 | 0.83 | 0.81 | 0.59 | 0.79 | 0.87 | 0.83 | 0.60 • | 0.71 | 0.86 | 0.83 |
| | PC3 | 0.77 | 0.77 | 0.76 | 0.76 | 0.51 • | 0.63 • | 0.61 • | 0.77 | 0.57 • | 0.77 | 0.77 | 0.74 |
| | PC4 | 0.83 | 0.82 | 0.83 | 0.84 ○ | 0.73 | 0.56 • | 0.74 • | 0.83 | 0.67 • | 0.84 | 0.82 | 0.82 |
| | PC5 | 0.94 | 0.94 | 0.94 | 0.94 | 0.90 • | 0.75 • | 0.94 • | 0.94 | 0.73 • | 0.94 | 0.94 | 0.94 |
| | Avg | 0.78 | 0.79 | 0.78 | 0.78 | 0.69 | 0.66 | 0.75 | 0.78 | 0.65 | 0.75 | 0.79 | 0.78 |

○, • statistically significant improvement or degradation

deal with the quality and characteristics of the data and to compare algorithms and statistical analyses. It is also necessary to study other repositories (e.g., open source systems) in order to study the quality of such repositories, validate generic claims found in the literature and provide guidelines for the problem of imbalance in defect prediction.

## Acknowledgments

The authors thank S. Garcia for helping with the implementation of some algorithms and the anonymous reviewers for their comments and suggestions. This research has been partially supported by European Commission funding under the 7th Framework Programme IAPP Marie Curie program for project ICEBERG no. 324356, and projects TIN2011-68084-C02-00 and TIN2013-46928-C3-2-R.

## 6. REFERENCES

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[2] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahvas. Software defect prediction using regression via classification. In *IEEE International Conference on Computer Systems and Applications (AICCSA 2006)*, pages 330–336, 8 2006.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346 – 7354, 2009.

[6] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[7] N. Chawla, A. Lazarevic, L. Hall, and K. Bowyer. Smoteboost: Improving prediction of the minority class in boosting. In *7th European Conference on Principles and Practice of Knowledge Discovery in Databases(PKDD 2003)*, pages 107–119, 2003.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 2002.

[9] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning (ICLM'06*, ICML'06, pages 233–240, New York, NY, USA, 2006. ACM.

[10] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, Dec. 2006.

[11] P. Domingos. Metacost: a general method for making classifiers cost-sensitive. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, pages 155–164, New York, NY, USA, 1999. ACM.

[12] K. O. Elish and M. O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.

[13] T. Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[14] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[15] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco, 1996. Morgan Kaufmann.

[16] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(4):463–484, 2012.

[17] M. Galar, A. Fernández, E. Barrenechea, and F. Herrera. EUSboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*, null(null), May 2013.

[18] S. García, R. Aler, and I. M. Galván. Using evolutionary multiobjective techniques for imbalanced classification data. In K. Diamantaras, W. Duch, and L. S. Iliadis, editors, *Artificial Neural Networks - ICANN 2010*, volume 6352 of *Lecture Notes in Computer Science*, pages 422–427. Springer Berlin Heidelberg, 2010.

[19] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Transactions on Software Engineering*, In Press – 2011.

[20] M. Halstead. *Elements of software science*. Elsevier Computer Science Library. Operating And Programming Systems Series; 2. Elsevier, New York ; Oxford, 1977.

[21] J. V. Hulse and T. Khoshgoftaar. Knowledge discovery from imbalanced and noisy data. *Data & Knowledge Engineering*, 68(12):1513–1542, 2009.

[22] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, Oct. 2002.

[23] T. M. Khoshgoftaar, E. Allen, and J. Deng. Using regression trees to classify fault-prone software modules. *IEEE Transactions on Reliability*, 51(4):455–462, 2002.

[24] T. M. Khoshgoftaar, E. Allen, J. Hudepohl, and S. Aud. Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks*, 8(4):902–909, 1997.

[25] T. M. Khoshgoftaar and N. Seliya. Analogy-based practical classification rules for software quality estimation. *Empirical Software Engineering*, 8(4):325–350, 2003.

[26] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, July-Aug. 2008.

[27] V. López, A. Fernández, and F. Herrera. On the importance of the validation technique for

classification with imbalanced datasets: Addressing covariate shift when data is skewed. *Information Sciences*, 257:1–13, 2014.

[28] V. López, A. Fernández, J. G. Moreno-Torres, and F. Herrera. Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics. *Expert Systems with Applications*, 39(7):6585–6608, June 2012.

[29] B. W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et biophysica acta*, 405(2):442–451, Oct. 1975.

[30] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[31] T. Mende and R. Koschke. Revisiting the evaluation of defect prediction models. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering (PROMISE'09)*, pages 1–10, New York, NY, USA, 2009. ACM.

[32] T. Mende and R. Koschke. Effort-aware defect prediction models. In *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering (CSMR'10)*, CSMR'10, pages 107–116, Washington, DC, USA, 2010. IEEE Computer Society.

[33] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012.

[34] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald. Problems with precision: A response to comments on data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(9):637–640, 2007.

[35] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 2007.

[36] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[37] Y. Peng, G. Kou, G. Wang, H. Wang, and F. Ko. Empirical evaluation of classifiers for software risk management. *International Journal of Information Technology & Decision Making (IJITDM)*, 08(04):749–767, 2009.

[38] Y. Peng, G. Wang, and H. Wang. User preferences based software defect detection algorithms selection using MCDM. *Information Sciences*, In Press.:–, 2010.

[39] J. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, California, 1993.

[40] J. Rodriguez, L. Kuncheva, and C. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, oct 2006.

[41] C. Seiffert, T. Khoshgoftaar, and J. Van Hulse. Improving software-quality predictions with data sampling and boosting. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(6):1283–1294, 2009.

[42] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Napolitano. RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(1):185–197, 2010.

[43] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215, 2013.

[44] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on Machine learning (ICLM)*, ICML '07, pages 935–942, New York, NY, USA, 2007. ACM.

[45] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen. Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software*, 81(5):823–839, 2008.

[46] D. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man and Cybernetics*, (3):408–421, 1972.

[47] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*. Morgan Kaufmann, 2011.

[48] H. Zhang and X. Zhang. Comments on "data mining static code attributes to learn defect predictors". *IEEE Transactions on Software Engineering*, 33(9):635–637, 2007.