Published in IET Software
Received on 31st July 2011
Revised on 3rd February 2012
doi: 10.1049/iet-sen.2011.0132

ISSN 1751-8806

Reflections on the NASA MDP data sets

D. Gray D. Bowes N. Davey Y. Sun B. Christianson

Computer Science Department, University of Hertfordshire, UK
E-mail: d.gray@herts.ac.uk

Abstract: Background: The NASA metrics data program (MDP) data sets have been heavily used in software defect prediction research. Aim: To highlight the data quality issues present in these data sets, and the problems that can arise when they are used in a binary classification context. Method: A thorough exploration of all 13 original NASA data sets, followed by various experiments demonstrating the potential impact of duplicate data points when data mining. Conclusions: Firstly researchers need to analyse the data that forms the basis of their findings in the context of how it will be used. Secondly, the bulk of defect prediction experiments based on the NASA MDP data sets may have led to erroneous findings. This is mainly because of repeated/duplicate data points potentially causing substantial amounts of training and testing data to be identical.

1 Introduction

Modern software defect prediction research typically involves classification and/or regression algorithms being used to predict the presence of non-syntactic implementational errors in software source code. To make these predictions such algorithms attempt to generalise upon software fault data, observations of software product and/or process metrics coupled with a 'level of defectiveness' value. This value typically takes the form of a 'number of faults reported' metric, for a given software unit after a given amount of time (either post code development or system deployment). Note that in this paper we use the following terms interchangeably: defect, error, fault and bug.

The predictions made by defect predictors may be continuous (level of defectiveness) or categorical [often set membership of either: ('defective', 'non-defective')]. The current trend by researchers is typically to report the latter (categorical predictions) only. However, this may not be the best approach, as continuous predictions allow software units to be ranked according to their predicted quality factor (this is known as module-order modelling, see [1]). A software quality ranking system has the real-world benefit of producing an ordered list of the seemingly most error-prone code units. These code units can then be subjected to some form of further inspection, in descending order of predicted level of defectiveness, for as long as resources allow.

In order to carry out a software defect prediction experiment there is a requirement for reasonable quality data. However, software fault data is very difficult to obtain. Commercial software development companies often do not have a fault measurement program in place. Moreover, even if such a program is in place, it is typically undesirable from a business perspective to publicise fault data. This is particularity true for systems where quality has been a serious problem, that is, where it would be most useful to publicise such data and give researchers an opportunity to discover why this was the case.

Open-source systems are frequently used by researchers to construct their own fault data sets [2–6]. Such systems are often developed while using a bug-tracking system to record the faults encountered by developers. If bug information has been correctly and consistently entered into version control commit messages, it is then possible to autonomously locate the fault-fixing revisions. From here it is possible to (fairly accurately) map fault-fixing revisions back to where the fault was first introduced (the bug-introducing change, see [5–7]). The major problem with constructing fault data from open-source systems is that it can be a very time consuming task to do accurately. This is because human intervention is often required to check the validity of the automated mappings. For systems of even moderate size, the quantity of these mappings can make this task infeasible.

Thus difficulty in obtaining software fault data is the major factor why public domain fault data repositories, such as those hosted by NASA and PROMISE, have become so popular among researchers. These repositories host numerous data sets, which require no data analysis and little or no pre-preprocessing, before machine learning tools such as Weka will classify them. The ease of this process can be dangerous to the inexperienced researcher. Results can be obtained without any scrutiny of the data. Furthermore, researchers may naively assume the NASA metrics data program (MDP) data sets are of reasonable quality for data mining. This issue is worsened by the hosting sites not indicating the main problems, and by so many researchers using these data sets inappropriately. The aim of this study is therefore to illuminate: the data quality issues present in these data sets, and the problems that can arise when they are used (as they often are) in a binary classification context. It is hoped that this study will encourage

researchers to take data quality seriously, and to question the results of some studies based on these data sets.

The NASA MDP Repository was previously available at http://mdp.ivv.nasa.gov/ and is currently available in a more basic, less documented form at http://filesanywhere.com/fs/v.aspx?v=896a648c5e5e6f799b. The repository currently contains 13 module-level data sets explicitly intended for software metrics research. Each data set represents a NASA software system/subsystem and contains static code metrics and fault data for each comprising module. Note that 'module' in this case refers to either a function, procedure or method. The static code metrics recorded include lines-of-code (LOC)-count, Halstead [8] and McCabe [9]-based measures. The primary fault data takes the form of an error-count metric, which was reportedly calculated from the number of error reports issued for each module via a bug-tracking system. From the details given at the original NASA MDP Repository, it is unclear precisely how these error reports were mapped back to the individual modules; however, it was stated that 'If a module is changed due to an error report (as opposed to a change request), then it receives a one up count. It cannot receive more than a one up for a given error report.' It was also stated that the error-count metric describes 'the number of changes due to error'. The originating source code for these data sets is entirely closed-source, making the validation of data integrity more difficult. A substantial amount of research based wholly or partially on these data sets has been published over the last decade, including [10–55]. Note that in this study we focus solely on the data sets just described, because they are widely used; we do not make use of the available object-oriented metrics data set, or any other data set affiliated with NASA.

The most common usage for the NASA data sets (as reported in the literature) is in binary classification experiments. Typically, a classifier is trained on binary-labelled data, and then each new set of module metrics is predicted as belonging to either a 'faulty' module, or a 'non-faulty' module. This is clearly a huge simplification of the real world, for two main reasons. Firstly, fault quantity is disregarded: there is typically no distinction between a module with one reported fault and a module with 31 reported faults, they are both simply labelled as 'faulty'. Secondly, fault severity is disregarded: there is typically no distinction between a trivial fault and a life-threatening fault. Despite these crude simplifications, binary classification defect prediction studies continue to be very prolific.

It is widely accepted by the data mining community that in order to accurately assess the potential real-world performance of a classification model, the model must be tested against entirely different data from that upon which it was trained [56]. This is why there is a distinction between a training set and a testing set. A testing set is also referred to as an independent test set, as it is intended to be independent from the training set (i.e. models should be tested against 'unseen data', see [56]). This is very basic data mining knowledge, and is no surprise to the defect prediction community. In 2004 Menzies *et al.* [11] state: 'if the goal of learning is to generate models that have some useful future validity, then the learned theory should be tested on data not used to build it. Failing to do so can result in a excessive over-estimate of the learned model…'. Despite this fact being well-known, numerous studies based on the NASA MDP data sets (henceforth, NASA data sets) have potentially had high proportions of data points common to both their training and testing sets. This is because the NASA data sets contain varied quantities of repeated data points, observations of module metrics and their corresponding fault data occurring more than once. Thus, when this data is used in a classification context, the separation into training and testing sets may result in both sets containing large proportions of common data points. This can yield the aforementioned excessive estimate of performance, as classifiers can memorise rather than generalise. This is very serious, as when data mining 'it is important that the test data was not used in any way to create the classifier' [56].

In this study we thoroughly analyse all 13 of the original NASA data sets. We are interested in data quality in terms of noise, inaccurate/incorrect data (see [57]). Additionally, because these data sets are typically used in binary classification experiments, we are also interested in the issues specific to this context. Firstly, we highlight the basic data quality problems via our novel data cleansing process. This process is for removing noise, and for preparing the data sets for binary classification. Next, we present the more complex issues still remaining after the cleansing process, including the issue of repeated data points. We discuss at length the potential problems caused by repeated data points when data mining, and why using lower level metrics (such as character counts) in fault data sets may alleviate these problems, by helping to distinguish non-identical modules.

The rest of this paper is presented as follows: in the next section we discuss related work, papers where problems with the NASA data sets have been documented or discussed. In Section 3 we document our novel data cleansing process in incremental stages. Section 4 contains a discussion of the issues still remaining after our cleansing process, including a demonstration of the potential effects of repeated data points during classification experiments. A new method to address these issues is proposed at the end of Section 4. Our conclusions are presented in Section 5.

2 Related studies

The major problem when using the NASA data sets in a classification context is that repeated data points may result in training data inadvertently being included in testing sets, potentially invalidating the experiment. This is not a new finding; however, we believe it needs spelling out to researchers, as previous studies mentioning this issue seem to have been ignored. In this section the most relevant studies surrounding this issue are discussed.

The earliest mention of repeated data points in NASA data sets that we can find was made in [58]. The authors state that they eliminated 'redundant data', but give no further explanation as to why. The data set used was NASA data set KC2, which is not available from the NASA MDP Repository. Although this data set is currently available from the PROMISE Repository, we did not use it in our study in an effort to use only the original, unmodified data.

In [33] five NASA data sets were used in various classification experiments. The author states that 'data pre-processing removes all duplicate tuples from each data set along with those tuples that have questionable values (e.g. LOC equal to 1.1).' Interestingly, it is only the PROMISE versions of the NASA data sets that contain these clearly erroneous non-integer LOC-count values. The author goes into detail on repeated data points, stating that 'to avoid building artificial models, perhaps the best

doi: 10.1049/iet-sen.2011.0132

approach would be not to allow duplicates within datasets'. One of the experiments carried out was intended to show the effect of the repeated data points in the five NASA data sets used. This was in a 10-fold cross-validation classification experiment with a C4.5 decision tree. The claimed result was that the data sets with the repeats included achieved significantly better performance than those without. Although this result is to be expected, there was an unfortunate technical shortcoming in the experimental design. When reporting the performance of classifiers on test sets with imbalanced class distributions, 'accuracy' (or its inverse: 'error rate') should not be used [59]. In addition to this, care is required when performing such an experiment, as the proportion of repeated data points in each class is not related to the class distribution. Therefore, post the removal of repeated data points, the data sets could have substantially different class distributions. This may boost or reduce classifier performance, because of the class imbalance problem (see [60, 61]).

Classification experiments utilising probabilistic outputs were carried out in [39]. Here the authors used five of the original NASA data sets and state that they removed both 'redundant and inconsistent patterns'. Inconsistent data points are another of the problems when data mining with the NASA data sets. They occur when repeated feature vectors (module metrics) describe data points with differing class labels. Thus, in this domain they occur where the same set of metrics is used to describe both a module labelled as 'defective' and a module labelled as 'non-defective'. We believe the removal of such instances was first carried out in [25].

The work described here differs from that previously described, as it is not based on classification experiments. It is instead based on the analysis and cleansing of data. This study demonstrates: the poor quality of the NASA data sets; the extent to which repeated data points disseminate into training and testing sets; and the effect of testing sets containing seen data during classification experiments.

3 Method: data cleansing

The NASA data sets are available from the aforementioned NASA MDP and PROMISE repositories. For this study, we used the original versions of the data sets from the NASA MDP Repository. Note, however, that the main issues also apply to the PROMISE versions of these data sets (available at http://promisedata.org/), which are for the most part simply the same data in a different format. Also note that updated versions of the data sets have recently been uploaded at PROMISE. These new versions address many of the issues pointed out in our earlier work (see [62]).

3.1 Initial pre-processing: binarisation of class variable and removal of module identifier and extra error data attributes

In order to be suitable for binary classification, the error-count attribute is commonly reported in the literature (see [13, 29, 55] for example) as being binarised as follows

$$\text{defective?} = (\text{error\_count} \geq 1)$$

It is then necessary to remove the 'unique module identifier' attribute, as this gives no information towards the defectiveness of a module. Lastly, it is necessary to remove all other error-based attributes, to make the classification task worthwhile. This initial pre-processing is summarised in Fig. 1. As the NASA data are often reportedly used post this initial pre-processing, we present an overview of each data set in Table 1.

3.2 Stage 1: removal of constant attributes

A numeric attribute which has a constant/fixed value throughout all instances is easily identifiable as it will have a variance of zero. Such attributes contain no information with which to discern modules apart, and are at best a waste of classifier resources. Each data set had from 0 to 10% of their total attributes removed during this stage, with the exception of data set KC4. This data set has 26 constant attributes out of a total of 40, thus 65% of available data contains no information with which to train a classifier.

This stage removes data that may be genuine, but in the context of machine learning it is of no use and is therefore discarded. Regarding data set KC4, it appears as though many of the metrics have not been collected; instead of leaving them out of the data set originally however, they were instead included with all values equal to zero.

An additional note regarding data set KC4 is that two of its attributes: 'essential complexity' and 'essential density' have two unique values each, but in each case, one of the values occurs just once. This data may be valid, but after the data divide into training and testing set, it may be that the training data contains a constant attribute. This can be problematic for some learning techniques, and is therefore something that researchers should be aware of.

```
rmAttributes = [ MODULE, ERROR_DENSITY, ERROR_REPORT_IN_6_MON,
                 ERROR_REPORT_IN_1_YR, ERROR_REPORT_IN_2_YRS ]

for dataSet in dataSets:
    for rmAttribute in rmAttributes:
        if rmAttribute in dataSet:
            dataSet = dataSet - rmAttribute
    dataSet.binarise(ERROR_COUNT)
    dataSet.rename(ERROR_COUNT, DEFECTIVE)
```

Fig. 1 *Initial pre-processing pseudo-code*

Table 1 Details of the NASA data sets post initial pre-processing

Name	Language	Features	Instances	Defective instances, %
CM1	C	40	505	10
JM1	C	21	10 878	19
KC1	C++	21	2107	15
KC3	Java	40	458	9
KC4	Perl	40	125	49
MC1	C and C++	39	9466	0.7
MC2	C	40	161	32
MW1	C	40	403	8
PC1	C	40	1107	7
PC2	C	40	5589	0.4
PC3	C	40	1563	10
PC4	C	40	1458	12
PC5	C++	39	17 186	3

3.3 Stage 2: removal of repeated attributes

In addition to constant attributes, repeated attributes occur where two or more attributes have identical values for each instance. Such attributes are therefore fully correlated, which may effectively result in a single attribute being over-represented. Among the NASA data sets there are two repeated attributes (post stage 1), namely the 'number of lines' and 'loc total' attributes in data set KC4. The difference between these two metrics is poorly defined at the NASA MDP Repository. However, they may be identical for this data set as (according to the metrics) there are no modules with any lines either containing comments or which are empty. For this data-cleansing stage we removed one of the attributes so that the values were only being represented once. We chose to keep the 'loc total' attribute label as this is common to all 13 NASA data sets.

This stage again removes data that may be genuine, because it can be problematic when data mining. It is interesting that data set KC4 has had so much data removed in these first two stages. Table 1 shows that KC4 is unique in that it is the only data set based on Perl code. Therefore it may be that the metrics collection tool (McCabeIQ 7.1) was more limited in the metrics it could collect for this language.

3.4 Stage 3: replacement of missing values

Missing values may or may not be problematic for learners depending on the classification method used. However, dealing with missing values within the NASA data sets is very simple. Seven of the data sets contain missing values, but all in the same single attribute: 'decision density'. This attribute is defined as 'condition count' divided by 'decision count', and for each missing value both these base attributes have a value of zero. It therefore appears as though missing values have occurred because of a division by zero error. In the remaining data set which contains all three of the aforementioned attributes but does not contain missing values, all instances with 'condition count' and 'decision count' values of zero also have a 'decision density' of zero. Because of this we replace all missing values with zero, ensuring consistency between data sets. Note that in [39] all instances which contained missing values within the NASA data sets were discarded. It is more desirable to cleanse data than to remove it, as the quantity of possible information to learn from will thus be maximised.

This stage adds data via the replacement of missing values, because they are problematic for many learning techniques. Note, however, that some researchers may not wish to carry out this stage, if they are using a learning method that is resilient to missing values (such as naive Bayes). Additionally, some researchers may wish to exclude derived features (such as 'decision density') altogether. There is more discussion on this in Section 4.

3.5 Stage 4: enforce integrity with domain-specific expertise

The NASA data sets contain varied quantities of attributes derived from simple equations of other attributes, which are useful for checking data integrity. In addition, it is possible to use domain-specific expertise to validate data integrity, by searching for theoretically impossible occurrences. The following is a non-exhaustive list of checks that can be carried out for each data point:

- Halstead's length metric (see [8]) is defined as 'number of operators' + 'number of operands'.
- Each token that can increment a module's cyclomatic complexity (see [9]) is counted as an operator according to the original NASA MDP Repository. Therefore the cyclomatic complexity of a module should not be greater than the number of operators + 1. Note that the minimum cyclomatic complexity is 1.
- The number of function calls within a module is recorded by the 'call pairs' metric. A function call operator is counted as an operator according to the original NASA MDP Repository, therefore the number of function calls should not exceed the number of operators.

These three simple rules are a good starting point for removing noise in the NASA data sets. Any data point which does not pass all of the checks contains noise. As the original NASA software systems/subsystems from where the metrics are derived are not publicly available, it is impossible for us to investigate this issue of noise further. The most viable option is therefore to discard each offending instance. Note that a prerequisite of each check is that the data set must contain all of the relevant attributes (post stage 1). Six of the data sets had data removed during this stage, between 1 and 12% of their data points in total.

During this stage, it is possible to not only remove noise (inaccurate/incorrect data), but also problematic data. A module which (reportedly) contains no lines of code and no operands and operators should be an empty module containing no code. Should such a data point be discarded? As it is impossible for us to check the validity of the metrics against the original code, this is a grey area. An empty module may still be a valid part of a system, it may just be a question of time before it is implemented. Furthermore, a module missing an implementation may still have been called by an unaware programmer. As the module is unlikely to have carried out the task its name implies, it may also have been reported to be faulty. Despite this, researchers need to decide for themselves what to do with data that cannot be proved to be noisy, but is nonetheless strange. For example, the original data set MC1 (according to the metrics) contains 4841 modules (51% of modules in total) with no lines of code. We feel that it would therefore not be unreasonable to remove such data points, or even reject the entire data set altogether.

4 Further issues

Our data-cleansing process demonstrates issues with the NASA data sets in terms of noise (stage 4) and classification-specific problems (stages 1–3). Although there are almost certainly more noisy data points that could not be identified using such simple methods, it is difficult for us to explore this further because of the closed-source, proprietary nature of the software. However, there are additional, more serious classification-specific problems, which we now discuss.

The most well-known issue regarding use of the NASA data sets in classification experiments is that of the varied levels of class imbalance (see Table 1). The table shows that data set KC4 has an almost balanced class distribution, whereas data set PC2 has only 0.4% of data points belonging to the minority class. This is an issue that

doi: 10.1049/iet-sen.2011.0132

researchers should be aware of. Learning from imbalanced data is an active area of research within the data mining community, we therefore refer readers to standard texts [56, 60, 61, 63]. Note, however, that defect prediction researchers need to be very careful in the way they assess the performance of their classifiers when using highly imbalanced data (see [64–66]).

Another issue is that, as mentioned previously, there are attributes within the NASA data sets that are simple equations of other attributes. Although useful for checking data integrity, they can be problematic (or simply a waste of computational resources) depending on the learning technique used. For example, support vector machines utilising a Gaussian radial basis kernel will typically not benefit from the inclusion of such attributes, as they will be implicitly calculated. Additionally, other highly correlated attributes can be found within the data sets, which are known to harm classification performance with many learning techniques [67, 68]. Therefore in some contexts, researchers may wish to address these issues. This usually involves removing attributes during pre-processing and/or utilising a feature selection technique on the training data.

The most severe issue when using the NASA data sets for classification experiments is that of repeated data points. Unfortunately, such data points are often ignored in the defect prediction literature. Repeated, redundant or duplicate data points are data points (or instances) that appear more than once within a data set. They occur either because of a data quality problem (a faulty data collection process), or (in this domain) when many modules have the same values for all measured metrics; for example, when they have the same number of: lines of code, lines of comments, blank lines, operands, operators, unique operands, unique operators, function calls and so on. Additionally, these modules have also been assigned the same class label referring to whether they are or are not 'defective'. This situation is clearly possible in the real world; for example, in an object-oriented system, there may be many simple accessor and mutator methods that have not been reported as faulty and share identical metrics. However, such data points may be problematic in the context of machine learning, where is it imperative that classifiers are tested upon data points independent from those used during training [56]. The issue is that when data sets containing repeated data points are split into training and testing sets (e.g. by a x% training, $100 - x$% testing split, or n-fold cross-validation), it is possible for there to be instances common to both sets. With test data included in the training data, the learning task is either simplified or reduced entirely to a task of recollection. Ultimately however, if the experiment is intended to show how well a classifier could generalise upon future, unseen data points, the results will be erroneous as the experiment is invalid. This is because the assumption of unseen data has been violated, as the test data has been contaminated with training data.

Inconsistent (or conflicting) instances are another issue, and are very similar to repeated instances in that both occur when the same feature vectors describe multiple modules. The difference between repeated and inconsistent instances is that with the latter, the class labels differ, thus (in this domain) the same metrics would describe both a 'defective' and a 'non-defective' module. This is again possible in the real world, and while not as serious an issue as the repeated instances (in the case of the NASA data sets), inconsistent data points can be problematic during binary classification tasks. When building a classifier which outputs a predicted class set membership of either 'defective' or 'non-defective', it is illogical to train such a classifier with data instructing that the same set of features is resultant in both classes. We focus more on repeated data points than inconsistent ones in this study, as for most data sets the proportion of repeated instances is considerably larger. Note, however, that it is possible for a data point to be both repeated and inconsistent.

The proportion of repeated data points in each data set (post initial pre-processing, as this is how they are most frequently used) is shown in Fig. 2. Note that in some cases the proportion is very large (89, 79 and 75% for data sets PC5, MC1 and PC2, respectively). In an earlier study [62], we recommended the removal of such instances as part of our data cleansing process, ensuring that each consistent data point is unique. This is a simple and acceptable way to address the issues caused by repeated data points, which has been carried out in prior studies (see [39, 58]). However, in this study we propose a novel and robust approach, where test sets remain unmodified. We come back to this at the end of this section, after addressing the following, more immediate questions: Why are there so many repeated data points and what can be done in future to avoid them? What proportion of seen data points could end up in testing sets if this data were to be used in classification experiments? What effect could having such quantities of seen data points in testing sets have on classifier performance? Each of these questions are addressed in the sections that follow.

4.1 Why are there so many repeated data points and how can they be avoided in future?

As previously stated, the NASA data sets are based on closed-source, proprietary software, so it is impossible for us to validate whether the repeated data points are truly a representation of each software system/subsystem, or whether they are noise. Despite this, a probable factor in why the repeated (and inconsistent) data points are a part of these data sets is because of the poor differential capability of the metrics used. Intuitively, 40 metrics describing each software module seem like a large set. However, many of the metrics are simple equations of other metrics. Because of this, it may be highly beneficial in future to also record lower level metrics, such as character counts. These will help to distinguish modules apart, particularly small

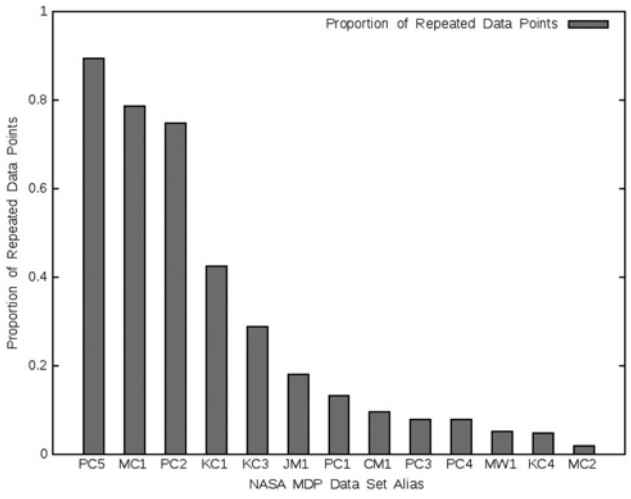

Fig. 2 *Proportion of repeated data points in each NASA data set*

modules, which statistically result in more repeated data points than large modules. Additionally, learners should be able to utilise such low-level data for helping to detect potentially troublesome modules (in terms of error-proneness).

The size of the input space in these data sets and fault data sets in general has not been previously discussed in detail to the best of our knowledge. All base attributes (attributes not derived from simple equations of other attributes) in these data sets contain only discrete values ≥ 0. Therefore the probability of a repeated data point in these data sets is much greater than, for example, a data set with real-valued measurements for the same number of attributes. This is why we believe that in future, researchers should additionally record lower level metrics (such as character counts), to alleviate the issues with repeated data points by helping to distinguish non-identical modules.

4.2 What proportion of seen data points could end up in testing sets if this data were to be used in classification experiments?

In order to find the answer to this question, a small Java program was developed utilising the Weka machine learning tool's libraries (version 3.7.5). These libraries were chosen as they have been heavily used in defect prediction research (see [13, 33, 36], for example). In this experiment a standard stratified 10-fold cross-validation was carried out. During each of the 10-folds, the number of instances in the testing set which were also in the training set were counted. After all 10-folds, the average number of shared instances in each testing set was calculated. This process was repeated 10 000 times with different pseudo-random number seeds to defend against order effects. For this experiment, we used the NASA data sets post basic pre-processing (see Table 1). We did this because it was as representative as possible of what will have happened in many previous studies. It is for the same reason that we chose stratified 10-fold cross-validation, the Weka default. The results from this experiment are shown in Figs. 3*a* and 3*b*. These figures show that for each data set, the average proportion of seen data points in the testing sets was greater than the proportion of repeated data points in total. Additionally, this relationship can be seen in Fig. 3*b* to have a strong positive correlation. It is worth emphasising that in some cases the average proportion of seen data points in the testing sets was very large (91, 84 and 82% for data sets PC5, MC1 and PC2, respectively).

4.3 What effect could having such quantities of seen data points in testing sets have on classifier performance?

To answer this question we do not use the NASA data sets because of the point regarding class distributions mentioned in Section 2. Instead, for clarity, we construct an artificial data set, with 10 numeric features and 1000 data points. All features were generated by a uniform pseudo-random number generator and have a possible range between 0 and 1 inclusive. The data set has a balanced class distribution, with 500 data points representing each class.

Using the Weka Experimenter, we ran 10 repetitions of a stratified 10-fold cross-validation experiment with the data set just described, and the following variations of it:

- 25% repeats from each class, an extra copy of 125 unique instances in each class, 1250 instances in total.
- 50% repeats from each class, an extra copy of 250 unique instances in each class, 1500 instances in total.
- 75% repeats from each class, an extra copy of 375 unique instances in each class, 1750 instances in total.
- 100% repeats, two copies of every instance, 2000 instances in total.

We used random forest learners for this experiment, with 500 trees and all other parameters set to the Weka defaults. We chose 500 trees as this was the number used in [19, 21, 22]. The results of this experiment are shown in Fig. 4. The mean accuracy levels for each data set: original, 25, 50, 75 and 100% repeats were 47.84, 67.38, 80.22, 88.37 and 95.00. This clearly shows that seen data points can have a huge influence on the performance of classifiers, even with pseudo-random data. As the percentage of duplicates increases, so does the performance of the classifiers. There are several major factors why this is the case, including that each node (or tree) comprising a random forest is

- *Unpruned*: Meaning that the training data is essentially memorised. Therefore in cases where the test data contains seen data points, these points are highly likely to be classified correctly, provided they are consistent in the training data.
- *Trained on a bootstrap sample of the original training data*: Such samples are made with replacement, meaning that many original training instances will be repeated and some not chosen at all. Clearly, if the original training data contains duplicates to begin with, the proportion of repeated instances trained on by each node will likely increase dramatically.
- *Trained on a subset of available features*: Meaning that, with the input space of each data point reduced because of this subset, there is a greater likelihood of repeated (and inconsistent) data.

The results from this experiment are very interesting, as random forests have been reported to work well with the NASA data sets [19–22]. Note that random forests have also been reported to struggle with imbalanced data [69–71]. This makes them not the obvious choice for use with the NASA data sets, as most of them are imbalanced (see Table 1). Despite this, favourable performance has been observed [19–22], which we believe is partly because of the reasons just described. Note that we have confirmed these findings using version 5.1 of Breiman and Cutler's original Fortran code (available at http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm).

It is worth pointing out that the effect of learning on repeated data points is algorithm specific. For example, naive Bayes classifiers have been reported to be fairly resilient to duplicates [72]. We confirm this by repeating the experiment described previously with naive Bayes classifiers. The results are shown in Fig. 5. Although these results are not as striking as those for the random forests, we believe they are still noteworthy, and that in practice the effect may be significant.

4.4 How to address the issues caused by repeated data points

An additional issue with repeated data points, separate to the problem of test set contamination, may occur as a result of

doi: 10.1049/iet-sen.2011.0132

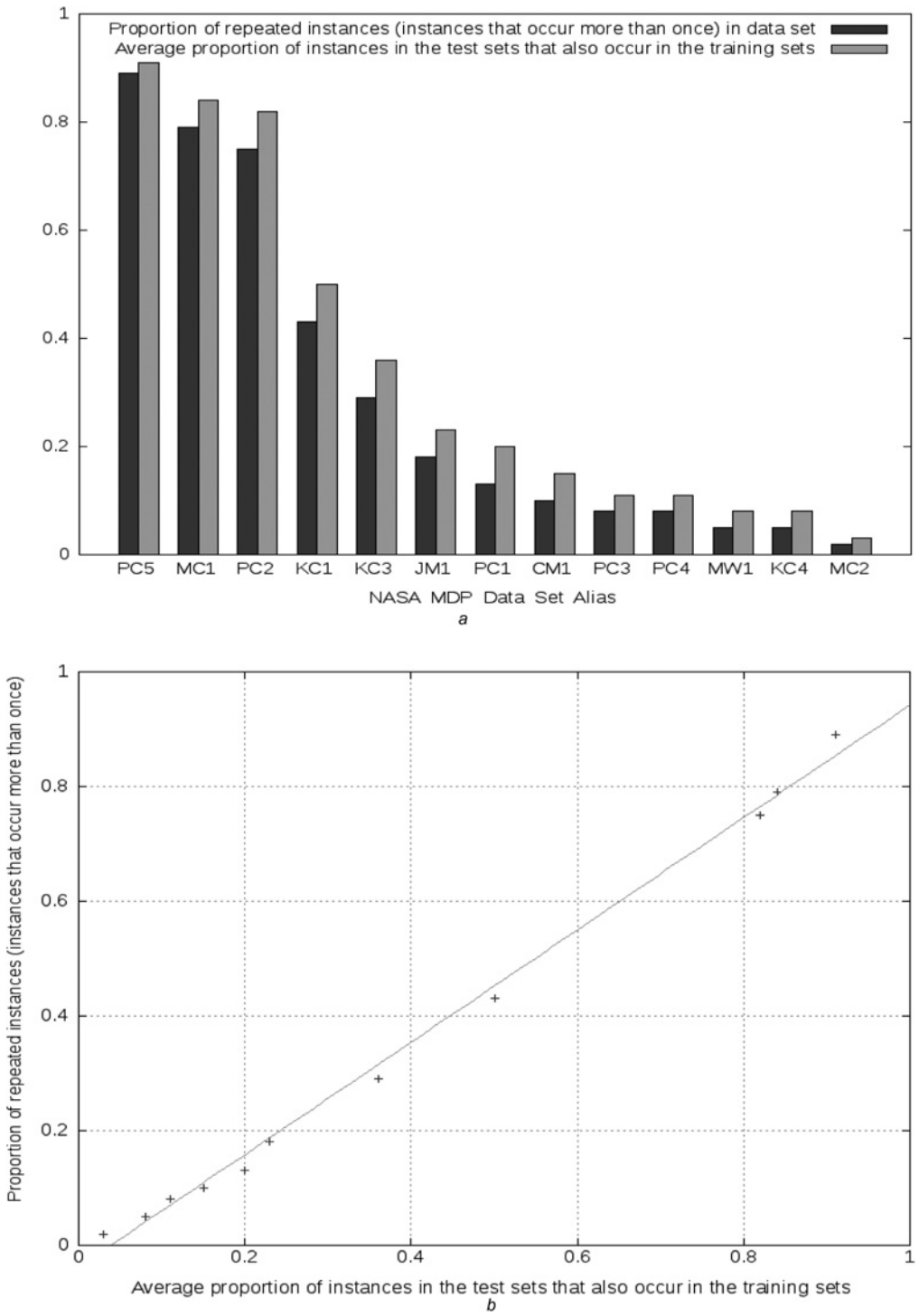

Fig. 3 *Dispersion of repeated data points in the NASA data sets*

a Proportions of repeated data and seen data in testing sets
b Proportions of repeated data against seen data in testing sets

model construction (training and optimisation). Training a model on data containing small proportions of repeated data points is typically non-problematic. For example, a simple oversampling technique is to duplicate minority class data points. Using training sets which contain repeated data points is reasonable, as long as training and testing sets share no common instances. Note however that the issue with excessive oversampling: overfitting, may also occur here. Overfitting can be identified when a model obtains good training performance, but poor performance on unseen test data [61]. It is also possible to cause overfitting by optimising model parameters using a validation set (a withheld subset of the training set) containing duplicate data points. It is for this reason that [72] recommend 'tuning a trained classifier with a duplicate-free sample'. Note that this also applies when optimising using multiple validation sets, for example via *n*-fold cross-validation. It is also worth noting that feature selection techniques can be negatively affected by duplicates [72].

As already mentioned, the simplest way to address the issues caused by repeated data points is to discard them as part of the contextual data-cleansing process, making each

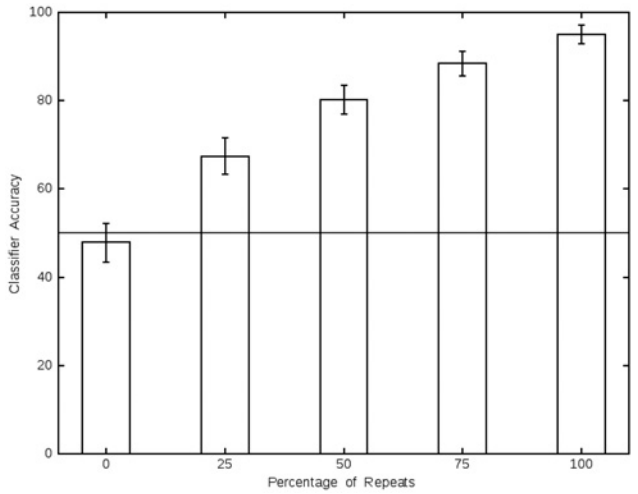

Fig. 4 *Random Forest classifiers with repeated data points*

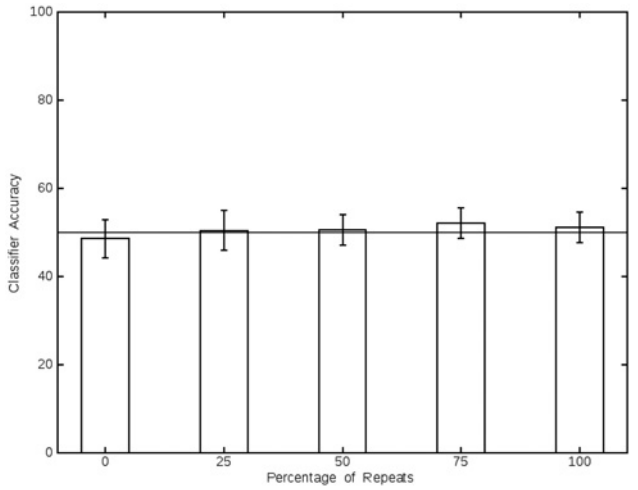

Fig. 5 *Naive Bayes classifiers with repeated data points*

consistent data point unique (for details see our earlier paper [62]). However, carrying out such a pre-processing step may not be the best approach, as repeated data points may occur in the real world. For this reason, we propose the following method for addressing the issues caused by repeated data points in classification experiments:

1. After the initial stratified-divide into training and testing set, discard all training data points with feature vectors common to the testing set. This ensures that performance is measured on unseen data, while the test set remains unmodified.
2. If the class distribution of the training set is adversely altered as a consequence of step 1, consider sampling techniques (see [61, 73]) to help maintain the original (or a more balanced) distribution. If oversampling is required, we recommend the synthetic minority oversampling technique (SMOTE [73]) rather than oversampling by duplication, as it reduces the likelihood of overfitting [73–75].
3. During model optimisation/tuning (if performed), remove all duplicates from the validation set, as recommended by [72]. Next, discard all validation set data points with feature vectors common to the corresponding training set (the training subset). If the class distribution of the validation set is adversely altered, consider the use of sampling techniques, as described in step 2. The purpose of this step is to help avoid overfitting.

This proposed approach is most suitable when researchers believe the repeated data points to be genuine, not noise. This is because test sets remain unmodified. With the simple approach of removing all repeated data points during data cleansing, test sets are indirectly modified before the data separation process has occurred. Therefore researchers who believe the rates of duplication in the NASA data sets to be feasible may wish to use this new approach, whereas researchers believing otherwise may wish to use the more simple approach (described in our earlier paper [62]). Note that a possible addition to the proposed approach is to remove all duplicates from the training set, to further reduce the possibility of overfitting. The best place for this to occur would be between steps 1 and 2.

5 Conclusions

Regardless of whether repeated data points are, or are not noise, it is unsuitable to have seen data in testing sets during experiments intended to show how well a classifier could potentially perform on future, unseen data. This is because seen data points can result in an excessive estimate of performance, occurring because classifiers can, to varying degrees, memorise rather than generalise. There is an important distinction between learning from and simply memorising data: only if you learn the structure underlying the data can you be expected to correctly predict unseen data.

Some researchers may argue that repeated data points should be tolerated and subject of no special treatment, as it is possible for modules with identical metrics to be contained within a software system. However, we have demonstrated that machine learning experiments based on data containing repeated data points can lead to invalid results. This is because it is possible for the test data to be (perhaps inadvertently) contaminated with training data, violating the assumption of unseen data. Even if researchers choose to ignore this fact, it is folly to report the performance of classification experiments with no distinction between performance on seen and unseen data. This is because the generalisation ability of classifiers may be far worse than the performance obtained implies.

If, in the real world, you happen to have a feature vector in your test set which is also contained in your training set, a simple lookup may be all that is required for best performance. A system could therefore be implemented where training data is stored in a hash table, and each test vector checked to see if it is contained within the hash table prior to classification. If so, the class label could be looked up directly with the classification model being unneeded (or some form of ensemble method used). A confidence interval could be derived using the lookup method, with a weight assigned to each training data point based on the number of times it occurs (because of replication) in the training data. This could also be extended to deal with inconsistent instances, perhaps by predicting the most frequently occurring of the classes. If the number of copies of the inconsistent instance in each class is equal, it may be best to report this, and/or make independent use of the classification model(s).

A simple approach to address the issues caused by repeated data points is to discard them prior to classification (for details see our earlier paper [62]). A more sophisticated approach was proposed in this paper (see Section 4.4), which keeps

doi: 10.1049/iet-sen.2011.0132

test sets unmodified, and may be a more realistic approach. Another possible option was proposed in [33], and is to use an extra attribute: number of duplicates. This will help to ensure that information is not lost, and like the approach proposed in this paper, is most viable when repeats are believed to be genuine.

A possible reason why there are so many repeated (and inconsistent) data points within the NASA data sets is because of the poor differential power (and the small input space) of the metrics used. It may be highly beneficial in future to also record lower level metrics (such as character counts), as these will help to distinguish non-identical modules, reducing the likelihood of modules sharing identical metrics. In addition to having lower level metrics to begin with, researchers should be careful when discarding attributes, as this typically reduces the size of the input space, increasing the probability of repeated feature vectors. For example, [13] used eight of the NASA data sets with just two or three of the original 38 features. Removing over 92% of the available features in each of these data sets typically yields dramatic increases in the proportions of repeated and inconsistent instances.

'Data cleaning is a time-consuming and labor-intensive procedure but one that is absolutely necessary for successful data mining … Time looking at your data is always well spent.' [56] We believe the data cleansing process defined in this paper will increase the accuracy of the NASA data sets, and make them more suitable for machine learning. This process may also be a good starting point when using other software fault data sets. Experiments based on the NASA data sets which blindly included the repeated data points may have led to erroneous findings. This is because results are often reported as if they were based on unseen data, when in fact (to varying degrees) they were not. The impression given from the literature is that many defect prediction researchers using this data have not been aware of this issue. Future work may be required to (where possible) repeat these studies with appropriately processed data. Other areas of future work include

- Extending the list of integrity rules described in stage 4 of the cleansing process. This will help to catch as many infeasible feature vectors, sets of metric values that can be proved to be noisy, as possible.
- Analysing other fault data sets to see whether the proportions of repeated data points in the NASA data sets are typical of fault data sets in general. This will help to determine the extent of this problem.
- Experimenting with, improving and refining the proposed new method of addressing the issues caused by repeated data points.

6 References

1 Khoshgoftaar, T.M., Allen, E.B.: 'Ordering fault-prone software modules', *Softw. Qual. Control*, 2003, **11**, pp. 19–37
2 Kim, S., James, E., Whitehead, J., Zhang, Y.: 'Classifying software changes: clean or buggy?', *IEEE Trans. Softw. Eng.*, 2008, **34**, (2), pp. 181–196
3 Schröter, A., Zimmermann, T., Zeller, A.: 'Predicting component failures at design time'. Proc. Fifth Int. Symp. on Empirical Software Engineering, 2006, pp. 18–27
4 Shivaji, S., Whitehead, E.J., Akella, R., Kim, S.: 'Reducing features to improve bug prediction'. 24th IEEE/ACM Int. Conf. Automated Software Engineering, 2009. ASE'09, 2009, pp. 600–604
5 Śliwerski, J., Zimmermann, T., Zeller, A.: 'When do changes induce fixes?', *SIGSOFT Softw. Eng. Notes*, 2005, **30**, (4), pp. 1–5
6 Kim, S., Zimmermann, T., Pan, K., Whitehead, E.J.J.: 'Automatic identification of bug-introducing changes'. ASE'06: Proc. 21st IEEE/ACM Int. Conf. on Automated Software Engineering, Washington, DC, USA, 2006, pp. 81–90
7 Williams, C., Spacco, J.: 'SZZ revisited: verifying when changes induce fixes'. Proc. 2008 Workshop on Defects in Large Software Systems. DEFECTS'08, New York, USA, 2008, pp. 32–36
8 Halstead, M.H.: 'Elements of software science (operating and programming systems series)' (Elsevier Science Inc., New York, USA, 1977)
9 McCabe, T.J.: 'A complexity measure'. ICSE'76: Proc. Second Int. Conf. on Software Engineering, Los Alamitos, CA, USA, 1976, p. 407
10 Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J.: 'A general software defect-proneness prediction framework', *IEEE Trans. Softw. Eng.*, 2011, **37**, (3), pp. 356–370
11 Menzies, T., Stefano, J.S.D., Orrego, A., Chapman, R.: 'Assessing predictors of software defects'. Proc. Workshop on Predictive Software Models, 2004
12 Menzies, T., Stefano, J.S.D.: 'How good is your blind spot sampling policy?'. Proc. Eighth IEEE Int. Symp. on High Assurance Systems Engineering, 2004, pp. 129–138
13 Menzies, T., Greenwald, J., Frank, A.: 'Data mining static code attributes to learn defect predictors', *IEEE Trans. Softw. Eng.*, 2007, **33**, (1), pp. 2–13
14 Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., Jiang, Y.: 'Implications of ceiling effects in defect predictors'. Proc. Fourth Int. Workshop on Predictor Models in Software Engineering. PROMISE'08, New York, USA, 2008, pp. 47–54
15 Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A.: 'Defect prediction from static code features: current results, limitations, new approaches', *Autom. Softw. Eng.*, 2010, **17**, (4), pp. 375–407
16 Zhang, H.: 'An investigation of the relationships between lines of code and defects'. IEEE Int. Conf. on Software Maintenance, 2009. ICSM 2009, 2009, pp. 274–283
17 Zhang, H., Nelson, A., Menzies, T.: 'On the value of learning from defect dense components for software defect prediction'. Proc. Sixth Int. Conf. on Predictive Models in Software Engineering. PROMISE'10, New York, USA, 2010, p. 14:1–14:9
18 Guo, L., Cukic, B., Singh, H.: 'Predicting fault prone modules by the Dempster–Shafer belief networks'. Proc. 18th IEEE Int. Conf. on Automated Software Engineering, 2003, pp. 249–252
19 Guo, L., Ma, Y., Cukic, B., Singh, H.: 'Robust prediction of fault-proneness by random forests'. 15th Int. Symp. on Software Reliability Engineering. ISSRE 2004, 2004, pp. 417–428
20 Jiang, Y., Cukic, B., Menzies, T.: 'Fault prediction using early lifecycle data'. 18th IEEE Int. Symp. on Software Reliability, 2007. ISSRE'07, 2007, pp. 237–246
21 Jiang, Y., Cukic, B., Menzies, T., Bartlow, N.: 'Comparing design and code metrics for software quality prediction'. Proc. Fourth Int. Workshop on Predictor Models in Software Engineering. PROMISE'08, New York, USA, 2008, pp. 11–18
22 Jiang, Y., Cukic, B., Menzies, T.: 'Can data transformation help in the detection of fault-prone modules?'. DEFECTS'08: Proc. 2008 Workshop on Defects in Large Software Systems, New York, USA, 2008, pp. 16–20
23 Jiang, Y., Cukic, B., Ma, Y.: 'Techniques for evaluating fault prediction models', *Empir. Softw. Eng.*, 2008, **13**, (5), pp. 561–595
24 Jiang, Y., Cukic, B.: 'Misclassification cost-sensitive fault prediction models'. Proc. Fifth Int. Conf. on Predictor Models in Software Engineering. PROMISE'09, New York, USA, 2009, p. 20:1–20:10
25 Khoshgoftaar, T.M., Seliya, N.: 'The necessity of assuring quality in software measurement data'. METRICS'04: Proc. Software Metrics, 10th Int. Symp., Washington, DC, USA, 2004, pp. 119–130
26 Zhong, S., Khoshgoftaar, T.M., Seliya, N.: 'Unsupervised learning for expert-based software quality estimation'. Proc. Eighth IEEE Int. Symp. on High Assurance Systems Engineering, 2004, pp. 149–155
27 Seliya, N., Khoshgoftaar, T.M., Zhong, S.: 'Analyzing software quality with limited fault-proneness defect data'. Ninth IEEE Int. Symp. on High-Assurance Systems Engineering, 2005. HASE 2005, 2005, pp. 89–98
28 Liu, Y., Khoshgoftaar, T.M., Seliya, N.: 'Evolutionary optimization of software quality modeling with multiple repositories', *IEEE Trans. Softw. Eng.*, 2010, **36**, (6), pp. 852–864
29 Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: 'Benchmarking classification models for software defect prediction: a proposed framework and novel findings', *IEEE Trans. Softw. Eng.*, 2008, **34**, (4), pp. 485–496
30 Turhan, B., Bener, A.: 'A multivariate analysis of static code attributes for defect prediction'. QSIC'07: Proc. Seventh Int. Conf. on Quality Software, Washington, DC, USA, 2007, pp. 231–237

31 Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J.: 'On the relative value of cross-company and within-company data for defect prediction', *Empir. Softw. Eng.*, 2009, **14**, pp. 540–578
32 Turhan, B., Kocak, G., Bener, A.: 'Data mining source code for locating software bugs: a case study in telecommunication industry', *Expert Syst. Appl.*, 2009, **36**, (6), pp. 9986–9990
33 Boetticher, G.D.: 'Improving credibility of machine learner models in software engineering'. Advanced Machine Learner Applications in Software Engineering (Series on Software Engineering and Knowledge Engineering), 2006, pp. 52–72
34 Mende, T., Koschke, R.: 'Revisiting the evaluation of defect prediction models'. Proc. Fifth Int. Conf. on Predictor Models in Software Engineering. PROMISE'09, New York, USA, 2009, pp. 7:1–7:10
35 Mende, T., Koschke, R.: 'Effort-aware defect prediction models'. European Conf. on Software Maintenance and Reengineering, 2010, pp. 107–116
36 Koru, A.G., Liu, H.: 'An investigation of the effect of module size on defect prediction using static measures', *ACM SIGSOFT Softw. Eng. Notes*, 2005, **30**, (4), pp. 1–5
37 Koru, A.G., Liu, H.: 'Building effective defect-prediction models in practice', *IEEE Softw.*, 2005, **22**, (6), pp. 23–29
38 Tosun, A., Bener, A.: 'Reducing false alarms in software defect prediction by decision threshold optimization'. Proc. 2009 Third Int. Symp. on Empirical Software Engineering and Measurement. ESEM'09, Washington, DC, USA, 2009, pp. 477–480
39 Bezerra, M.E.R., Oliveira, A.L.I., Meira, S.R.L.: 'A constructive RBF neural network for estimating the probability of defects in software modules'. Int. Joint Conf. on Neural Networks, 2007. IJCNN 2007, 2007, pp. 2869–2874
40 Singh, Y., Kaur, A., Malhotra, R.: 'Predicting software fault proneness model using neural network'. Proc. Ninth Int. Conf. on Product-Focused Software Process Improvement. PROFES'08, Berlin, Heidelberg, 2008, pp. 204–214
41 Challagulla, V.U.B., Bastani, F.B., Yen, I.L., Paul, R.A.: 'Empirical assessment of machine learning based software defect prediction techniques'. WORDS'05: Proc. 10th IEEE Int. Workshop on Object-Oriented Real-Time Dependable Systems, Washington, DC, USA, 2005, pp. 263–270
42 Challagulla, V.U.B., Bastani, F.B., Yen, I.L.: 'A unified framework for defect data analysis using the mbr technique'. ICTAI'06: Proc. 18th IEEE Int. Conf. on Tools with Artificial Intelligence, Washington, DC, USA, 2006, pp. 39–46
43 Pelayo, L., Dick, S.: 'Applying novel resampling strategies to software defect prediction'. Annual Meeting of the North American Fuzzy Information Processing Society, 2007. NAFIPS'07, 2007, pp. 69–72
44 Kutlubay, O., Turhan, B., Bener, A.B.: 'A two-step model for defect density estimation'. 33rd EUROMICRO Conf. Software Engineering and Advanced Applications, 2007, pp. 322–332
45 Ma, Y., Guo, L., Cukic, B.: 'A statistical framework for the prediction of fault-proneness', in 'Advances in machine learning application in software engineering' (Idea Group Inc., 2006, vol. 1), pp. 237–265
46 Oral, A.D., Bener, A.B.: 'Defect prediction for embedded software'. 22nd Int. Symp. on Computer and Information Sciences, 2007. ISCIS 2007, 2007, pp. 1–6
47 Rodriguez, D., Ruiz, R., Cuadrado-Gallego, J., Aguilar-Ruiz, J.: 'Detecting fault modules applying feature selection to classifiers'. IEEE Int. Conf. on Information Reuse and Integration. IRI 2007, 2007, pp. 667–672
48 Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., Haesen, R.: 'Mining software repositories for comprehensible software fault prediction models', *J. Syst. Softw.*, 2008, **81**, (5), pp. 823–839
49 Mertik, M., Lenic, M., Stiglic, G., Kokol, P.: 'Estimating software quality with advanced data mining techniques'. Int. Conf. on Software Engineering Advances, 2006, p. 19
50 Cong, J., En-Mei, D., Li-Na, Q.: 'Software fault prediction model based on adaptive dynamical and median particle swarm optimization'. 2010 Second Int. Conf. on Multimedia and Information Technology (MMIT), 2010, vol. 1, pp. 44–47
51 de Carvalho, A.B., Pozo, A., Vergilio, S.R.: 'A symbolic fault-prediction model based on multiobjective particle swarm optimization', *J. Syst. Softw.*, 2010, **83**, (5), pp. 868–882
52 Tao, W., Wei-hua, L.: 'Naive Bayes software defect prediction model'. 2010 Int. Conf. on Computational Intelligence and Software Engineering (CiSE), 2010, pp. 1–4
53 Li, Z., Reformat, M.: 'A practical method for the software fault-prediction'. IEEE Int. Conf. on Information Reuse and Integration. IRI 2007, August 2007, pp. 659–666
54 Vivanco, R.A., Kamei, Y., Monden, A., Matsumoto, K.-i., Jin, D.: 'Using search-based metric selection and oversampling to predict fault prone modules'. IEEE CCECE, 2010, pp. 1–6
55 Elish, K.O., Elish, M.O.: 'Predicting defect-prone software modules using support vector machines', *J. Syst. Softw.*, 2008, **81**, (5), pp. 649–660
56 Witten, I.H., Frank, E.: 'Data mining: practical machine learning tools and techniques', in 'Morgan Kaufmann series in data management systems' (Morgan Kaufmann, 2005, 2nd edn.)
57 Liebchen, G.A., Shepperd, M.: 'Data sets and data quality in software engineering'. PROMISE'08: Proc. Fourth Int. Workshop on Predictor Models in Software Engineering, New York, USA, 2008, pp. 39–44
58 Kaminsky, K., Boetticher, G.: 'Building a genetically engineerable evolvable program (GEEP) using breadth-based explicit knowledge for predicting software defects'. IEEE Annual Meeting of the Fuzzy Information, 2004. Processing NAFIPS'04, 2004, vol. 1, pp. 10–15
59 Nickerson, A.S., Japkowicz, N., Milios, E.: 'Using unsupervised learning to guide resampling in imbalanced data sets'. Proc. Eighth Int. Workshop on AI and Statistics, 2001, pp. 261–265
60 Chawla, N.V., Japkowicz, N., Kolcz, A.: 'Special issue on learning from imbalanced datasets', *SIGKDD Explor. Newsl.*, 2004, **6**, (1), pp. 1–6
61 He, H., Garcia, E.A.: 'Learning from Imbalanced Data', *IEEE Trans. Know. Data Eng.*, 2009, **21**, pp. 1263–1284
62 Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B.: 'The misuse of the NASA metrics data program data sets for automated software defect prediction'. Evaluation and Assessment in Software Engineering (EASE), 2011, pp. 96–103
63 Batista, G.E.A.P.A., Prati, R.C., Monard, M.C.: 'A study of the behavior of several methods for balancing machine learning training data', *SIGKDD Explor. Newsl.*, 2004, **6**, pp. 20–29
64 Davis, J., Goadrich, M.: 'The relationship between Precision-Recall and ROC curves'. Proc. 23rd Int. Conf. on Machine Learning. ICML'06, New York, USA, 2006, pp. 233–240
65 Zhang, H., Zhang, X.: 'Comments on "data mining static code attributes to learn defect predictors"', *IEEE Trans. Softw. Eng.*, 2007, **33**, pp. 635–637
66 Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B.: 'Further thoughts on precision'. Evaluation and Assessment in Software Engineering (EASE), 2011, pp. 129–133
67 Hall, M.A.: 'Correlation-based feature subset selection for machine learning'. Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1999
68 Howley, T., Madden, M.G., O'Connell, M.L., Ryder, A.G.: 'The effect of principal component analysis on machine learning accuracy with high-dimensional spectral data', *Knowl.-Based Syst.*, 2006, **19**, (5), pp. 363–370
69 Chen, C., Liaw, A., Breiman, L.: 'Using random forest to learn imbalanced data', Technical report 666, Department of Statistics, University of California, Berkeley, 2004
70 Segal, M.R.: 'Machine learning benchmarks and random forest regression' (Centre for Bioinformatics and Molecular Biostatistics, UC, San Francisco, 2004)
71 Dudoit, S., Fridlyand, J.: 'Classification in microarray experiments', in 'Statistical analysis of gene expression microarray data' (Chapman and Hall/CRC, 2003)
72 Kołcz, A., Chowdhury, A., Alspector, J.: 'Data duplication: an imbalance problem?'. ICML 2003 Workshop on Learning from Imbalanced Datasets, 2003
73 Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: 'SMOTE: synthetic minority over-sampling technique', *J. Artif. Intell. Res.*, 2002, **16**, pp. 321–357
74 Chawla, N.V., Lazarevic, A., Hall, L.O., Bowyer, K.W.: 'SMOTEBoost: improving prediction of the minority class in boosting'. Proc. Principles of Knowledge Discovery in Databases (PKDD-2003), 2003, pp. 107–119
75 Cieslak, D.A., Chawla, N.V., Striegel, A.: 'Combating imbalance in network intrusion datasets'. 2006 IEEE Int. Conf. Granular Computing, 2006, pp. 732–737

doi: 10.1049/iet-sen.2011.0132