The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction

David Gray, David Bowes, Neil Davey, Yi Sun and Bruce Christianson

***Abstract*—**
Background: The NASA Metrics Data Program data sets have been heavily used in software defect prediction experiments.
Aim: To demonstrate and explain why these data sets require significant pre-processing in order to be suitable for defect prediction.
Method: A meticulously documented data cleansing process involving all 13 of the original NASA data sets.
Results: Post our novel data cleansing process; each of the data sets had between 6 to 90 percent less of their original number of recorded values.
Conclusions:
One: Researchers need to analyse the data that forms the basis of their findings in the context of how it will be used.
Two: Defect prediction data sets could benefit from lower level code metrics in addition to those more commonly used, as these will help to distinguish modules, reducing the likelihood of repeated data points.
Three: The bulk of defect prediction experiments based on the NASA Metrics Data Program data sets may have led to erroneous findings. This is mainly due to repeated data points potentially causing substantial amounts of training and testing data to be identical.

I. Introduction

Automated software defect prediction is a process where classification and/or regression algorithms are used to predict the presence of non-syntactic implementational errors (henceforth; *defects*) 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 (post either code development or system deployment).

The predictions made by defect predictors may be continuous (level of defectiveness) or categorical (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 (Khoshgoftaar & Allen 2003)). 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.

All authors are with the Computer Science Department at the University of Hertfordshire, UK. Respective email addresses are: {d.gray, d.h.bowes, n.davey, y.2.sun, b.christianson}@herts.ac.uk

In order to carry out a software defect prediction experiment there is naturally 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. And, even if such a process 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, i.e. 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 a good place for researchers to construct their own fault data sets. This is simplest when the system has been developed whilst 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 (Kim, Zimmermann, Pan & Whitehead 2006)). 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.

Thus difficulty in obtaining software fault data is the major factor why public domain fault data repositories, such as those hosted by NASA[1] and PROMISE[2], 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[3] 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 previous researchers using these data sets without appropriate pre-processing. The aim of this study is to illuminate why the NASA MDP data sets require significant pre-processing; or contextual *data cleansing,* before they become suitable for data mining. It is hoped that this paper will encourage researchers to take data quality seriously, and to question the results of some studies based on these data sets.

[1]http://mdp.ivv.nasa.gov/
[2]http://promisedata.org/
[3]http://www.cs.waikato.ac.nz/~ml/weka/

The NASA MDP repository currently consists of 13 data sets explicitly intended for software metrics research. Each data set represents a NASA software system/subsystem and contains the static code metrics and corresponding fault data for each comprising module. Note that 'module' in this case can refer to a function, procedure or method. A substantial amount of research based wholly or partially on these data sets has been published over the last decade, including: (Menzies, Stefano, Orrego & Chapman 2004), (Menzies & Stefano 2004), (Menzies, Greenwald & Frank 2007), (Menzies, Milton, Turhan, Cukic, Jiang & Bener 2010), (Jiang, Cukic & Menzies 2008), (Jiang & Cukic 2009), (Khoshgoftaar & Seliya 2004), (Seliya, Khoshgoftaar & Zhong 2005), (Lessmann, Baesens, Mues & Pietsch 2008), (Boetticher 2006), (Mende & Koschke 2009), (Koru & Liu 2005), (Tosun & Bener 2009), (Bezerra, Oliveira & Meira 2007), (Turhan, Menzies, Bener & Di Stefano 2009), (Pelayo & Dick 2007), (Li & Reformat 2007) and (Elish & Elish 2008).

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 different data from that upon which it was trained (Witten & Frank 2005). 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 (Witten & Frank 2005)). This is very basic data mining knowledge, and is no surprise to the defect prediction community. In 2004 Menzies et al. 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…" (Menzies et al. 2004). 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 identical data points in their training and testing sets. This is because the NASA data sets contain varied quantities of *repeated data points;* i.e. observations of module metrics and their corresponding fault data occurring more than once. Thus, when this data is used in a machine learning context, training and testing sets potentially have large proportions of identical data points. This will result in the aforementioned excessive estimate of performance, as classifiers can memorise rather than learn.

In this study we develop and carry out a meticulously documented data cleansing process involving all 13 of the original NASA data sets. The purpose of this data cleansing is both to make the data sets suitable for machine learning, and to remove *noise* (i.e. inaccurate/incorrect data points, see (Liebchen & Shepperd 2008)). We show that after this process each of the data sets had between 6 to 90 percent less of their original recorded values. We then discuss at length the problems caused by repeated data points when data mining, and why using lower level metrics in fault data sets (such as character counts) may alleviate this problem, 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 issues with the NASA data sets have been documented or discussed. In Section III we document our novel data cleansing process in incremental stages. Section IV contains our findings, which include a demonstration of the effect of repeated data points during an artificial classification experiment. Our conclusions are presented in Section V.

II. Related Studies

The major issue with the original NASA data sets is that when they are used in a machine learning context, 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 in NASA data sets that we can find was made in (Kaminsky & Boetticher 2004). In this study the authors state that they eliminated "redundant data", but give no further explanation as to why. The data set used in this study was NASA data set KC2, which is no longer 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 (Boetticher 2006) 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 values. The author goes into detail on repeated data points, stating that "to avoid building artificial models, perhaps the best approach would be not to allow duplicates within datasets." One of the experiments carried out in this study was intended to show the effect of the repeated data in the five NASA data sets used. This was in the context of 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 higher 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 data sets with imbalanced class distributions, 'accuracy' (or its inverse: 'error rate') should not be used (Nickerson, Japkowicz & Milios 2001). In addition to this, care is required when performing such an experiment, as the proportion of repeated data 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 (Chawla, Japkowicz & Kolcz 2004)).

Classification experiments utilising probabilistic outputs were carried out in (Bezerra et al. 2007). Here the authors used all 13 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 labeled as 'defective' and a module labeled as 'non-defective'. We believe the removal of such instances was first carried out in (Khoshgoftaar & Seliya 2004).

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.

III. Method

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 same issues also apply to the PROMISE versions of these data sets, which are for the most part simply the same data in a different format.

A. Initial Pre-processing: Binarisation of Class Variable & 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 (Menzies, Greenwald & Frank 2007), (Lessmann et al. 2008) and (Elish & Elish 2008) for example) as being binarised as follows:

$$defective = (error\_count \geq 1)$$

It is also necessary to remove the 'unique module identifier' attribute as this gives no information toward 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 as follows:

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

for dataSet in dataSets:
    for attribute in attributes:
        if attribute in dataSet:
            dataSet = dataSet - attribute
    dataSet.binarise(error_count)
    dataSet.rename(error_count, defective)
```

The NASA data is often reportedly used in defect prediction experiments post this initial pre-processing. We therefore present an overview of each data set in Table I. In this table the number of original recorded values is defined as the number of attributes (features) multiplied by the number of instances (data points). For simplicity we do not take missing values into account. We use the number of recorded values as a method of quantifying how much data is available in each data set. We shall come back to these values post data cleansing to judge how much data has been removed.

B. Stage 1: Removal of Constant Attributes

An 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 percent 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 percent of available recorded values contain no information upon which to data mine.

C. Stage 2: Removal of Repeated Attributes

In addition to constant attributes, repeated attributes occur where two attributes have identical values for each instance. This effectively results in a single attribute being over-represented. Amongst the NASA data sets there is only one pair of 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.

TABLE I
DETAILS OF THE NASA MDP DATA SETS POST INITIAL PRE-PROCESSING.

Name	Language	Features	Instances	Recorded Values	% Defective Instances
CM1	C	40	505	20200	10
JM1	C	21	10878	228438	19
KC1	C++	21	2107	44247	15
KC3	Java	40	458	18320	9
KC4	Perl	40	125	5000	49
MC1	C & C++	39	9466	369174	0.7
MC2	C	40	161	6440	32
MW1	C	40	403	16120	8
PC1	C	40	1107	44280	7
PC2		40	5589	223560	0.4
PC3		40	1563	62520	10
PC4		40	1458	58320	12
PC5	C++	39	17186	670254	3

D. Stage 3: Replacement of Missing Values

Missing values may or may not be problematic for machine 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. In the remaining NASA 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. This appears logical, and it is clear that missing values have occurred because of a division by zero error. Because of this we replace all missing values with zero. Note that in (Bezerra et al. 2007) 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.

E. Stage 4: Enforce Integrity with Domain Specific Expertise

The NASA data sets contain varied quantities of correlated attributes, which are useful for checking data integrity. Additionally, 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 possible checks that can be carried out for each data point:

- Halstead's length metric (see (Halstead 1977)) is defined as: *'number of operators'* + *'number of operands'*.
- Each token that can increment a module's cyclomatic complexity (see (McCabe 1976)) is counted as an operator according to the NASA MDP repository. Therefore, the cyclomatic complexity of a module should not be greater than the number of operators + 1. Note that 1 is the minimum cyclomatic complexity value.
- The number of function calls within a module is recorded by the *'call pairs'* metric. A function call operator should be counted as an operator, 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. Because 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. Six of the data sets had data removed during this stage, between 1 to 12 percent of their data points in total.

During this stage it may be tempting to not only remove noise (i.e. inaccurate/incorrect data points), but also *outliers*. A module which (reportedly) contains no lines of code and no operands and operators should be an empty module containing no code. So at this stage of data cleansing, should such a module 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.

F. Stage 5: Removal of Repeated and Inconsistent Instances

As previously mentioned, repeated/redundant instances occur when the same feature vectors (module metrics) describe multiple modules with the same class label. While this situation is clearly possible in the real world, such data points are problematic in the context of machine learning, where is it imperative that classifiers are tested upon data points independent from those used during training (see (Witten & Frank 2005)). The issue is that when data sets containing repeated data points are split into training and testing sets (for example by a *x%* training, *1-x%* testing split, or *n-fold cross-validation*), it is possible for identical instances to appear in both sets. This either simplifies the learning task or reduces it entirely to a task of recollection. Ultimately however, if the experiment was intended to show how well a classifier could generalise upon future, unseen data points, the results will be erroneous as the experiment is invalid.

Inconsistent instances are similar to repeated instances, as they also 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, inconsistent data points are problematic during binary classification tasks. When building a classifier which outputs a predicted class set membership of either *'defective'* or *'non-defective'*, it is clearly illogical to train such a classifier with data instructing that the same set of features is resultant in both classes.

Adding all data points into a mathematical set is the simplest way of ensuring that each one is unique. This ensures classifiers will be tested on unseen data. From here it is possible to remove all inconsistent pairs of modules, to ensure that all feature vectors (data points irrespective of class label) are unique. The proportion of instances removed from each data set during this stage is shown in Figure 1. All data sets had instances removed during this stage, and in some cases the proportion removed was very large (90, 78, and 74 percent for data sets PC5, MC1, and PC2, respectively). Note that for most data sets the proportion of inconsistent instances removed is negligible.

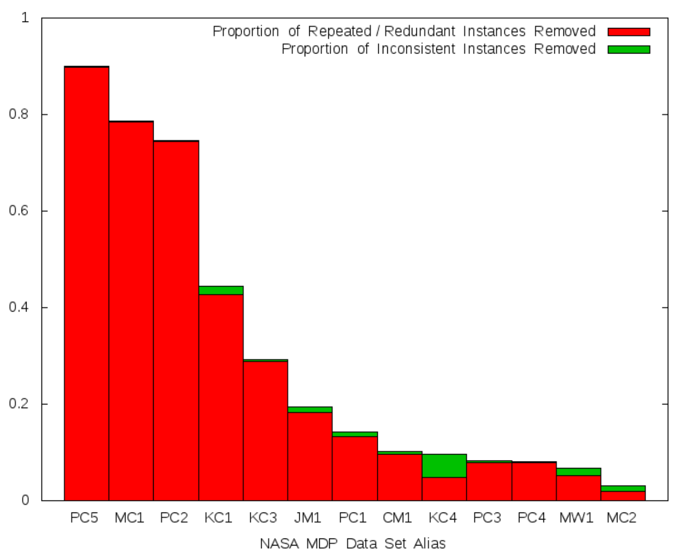

Fig. 1. The proportion of instances removed during stage 5.

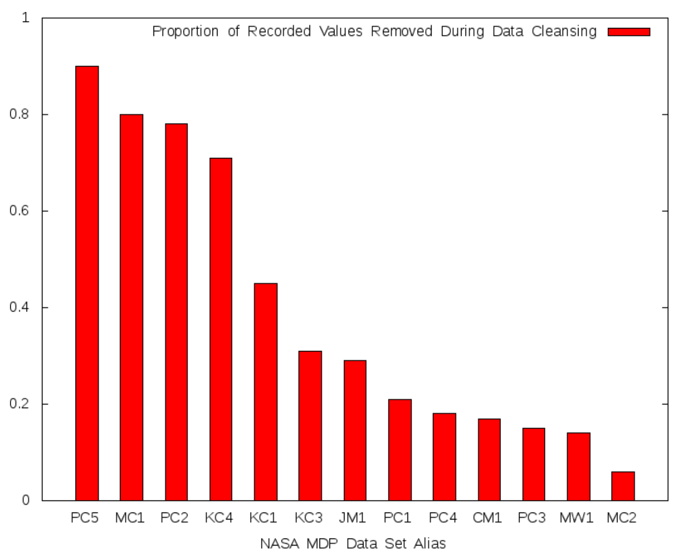

Fig. 2. The proportion of recorded values removed during data cleansing.

IV. Findings

Figure 2 shows the proportion of recorded values removed from the 13 NASA data sets (after basic pre-processing, see Table I) post our 5 stage data cleansing process. Stages 1 and 2 of this process can remove attributes, stage 3 can replace values, and stages 4 and 5 can remove instances (data points). This was the motivation to use the number of recorded values ($attributes * instances$) metric, as it takes both attributes and instances into account. Figure 2 shows that between 6 to 90 percent of recorded values in total were removed from each data set during our data cleansing process.

The purpose of our data cleansing process is to ensure that all data sets are suitable for machine learning (stages 1, 2, 5, and to some extent stage 3), and to remove or repair what can be confidently assumed to be noise (stage 4, and to some extent stage 3). Note however that there are almost certainly noisy data points in the NASA data sets that can not be confidently assumed to be erroneous using such simple methods. But, as the data sets are based on closed source, commercial software, it is impossible for us to investigate the potential issues further. For example, the original data set MC1 (according to the metrics) contains 4841 modules (51% of modules in total) with no lines of code.

Of the data cleansing processes with the potential to reduce the quantity of recorded values, it is the removal of repeated and inconsistent instances (stage 5) that is responsible for the largest average proportions of data removed (see Figure 1). This raises the following questions: *Is the complete removal of such instances really necessary? 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 was 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.

A. Is the complete removal of repeated and inconsistent instances prior to classification really necessary?

Our data cleansing method is a work in progress. Post data cleansing, researchers will be able to use off the shelf data mining tools (such as Weka) to carry out experiments that yield meaningful results (or at least, far more meaningful results than had the issues described not been addressed). However, not every part of the cleansing process will be required in all contexts. In this section we describe the issues that researchers should be aware of with regard to addressing the repeated and inconsistent instances.

Figure 1 shows that the proportion of inconsistent instances removed during stage 5 is negligible. This is partly a consequence of repeated instances being removed before inconsistent ones however, as it is possible for a data point to be both repeated and inconsistent. The complete removal of inconsistent instances prior to classification may not always be necessary or desirable. Because defect prediction data sets typically have an imbalanced class distribution (Menzies, Dekhtyar, Distefano & Greenwald 2007), some researchers may wish to retain each inconsistent minority class data point, in order to keep as much data as possible regarding the minority class (typically modules labeled as ‘defective’). During training, some learning methods (such as various probabilistic learners) may be able to robustly handle conflicting information. Therefore, the inclusion of inconsistent instances in training sets would not be problematic in this context. During testing, some researchers may feel that it is more appropriate to include inconsistent data points, as they may occur in the real world. Note that the inclusion of inconsistent data points in testing sets would typically introduce an upper bound on the level of performance achievable.

The training of learning methods on data containing repeated data points is typically not overly problematic. For example, a very simple oversampling technique is to duplicate minority class data points during training. Using training sets which contain repeated data points is reasonable, *as long as no training data points are included in the testing set.* The same also applies if researchers feel that testing sets should include repeated data points. There are potentially serious issues to be aware of when training sets contain repeated data points however, as pointed out in (Kołcz, Chowdhury & Alspector 2003). When optimising model parameters using a validation set (a withheld subset of the training set) that contains duplicate data points, *over-fitting* may occur, potentially harming performance during assessment. It is for this reason that (Kołcz et al. 2003) recommend "tuning a trained classifier with a duplicate-free sample". Note that this also applies when using multiple validation sets, for example via n-fold cross-validation. Kołcz et al. also point out that feature selection techniques can be affected by duplicate data points. An approach to repeated data points proposed by (Boetticher 2006) is to add an extra attribute: *number of duplicates*. This will ensure data points are unique, and help reduce information being lost.

B. *Why so much repeated data and how can it be avoided?*

As previously stated, the NASA data sets are based on closed source, commercial 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 is a part of these data sets is because of the poor differential capability of the metrics used. Intuitively, 40 metrics describing each software module seems 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 modules, which statistically result in more repeated data points than large modules. Additionally, machine learners may be able to utilise such low level data for helping to detect potentially troublesome modules (in terms of error-proneness).

C. *What proportion of seen data points could end up in testing sets if this data was 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 3.7.1). The Weka libraries were chosen because they have been heavily used in defect prediction experiments (see (Menzies, Greenwald & Frank 2007), (Boetticher 2006) and (Koru & Liu 2005), 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 1000 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 I). We did this because it was as representative as possible of what will have happened in some previous studies. It is for the same reason that we chose 10-fold cross-validation, the Weka *Explorer* default. The results from this experiment are shown in Figures 3 and 4. From these figures can be seen that for each data set, the proportion of seen data points in the testing sets is larger than the proportion of repeated data points in total. Additionally, this relationship can be seen in Figure 4 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 percent for data sets PC5, MC1, and PC2, respectively).

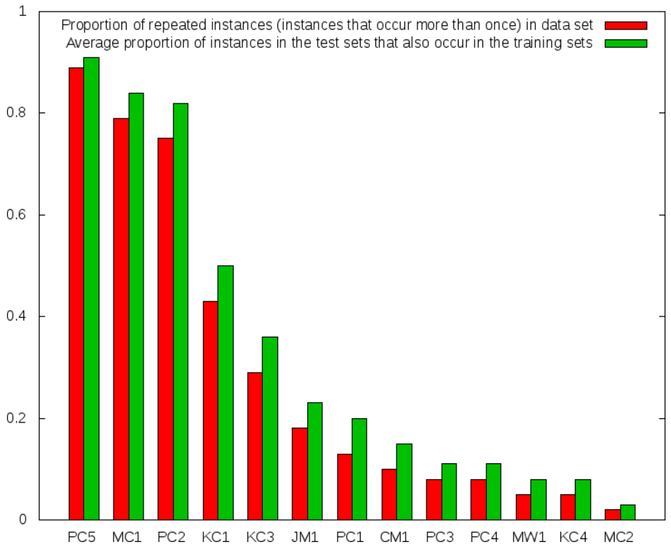

Fig. 3. Proportions of repeated data and seen data in testing sets.

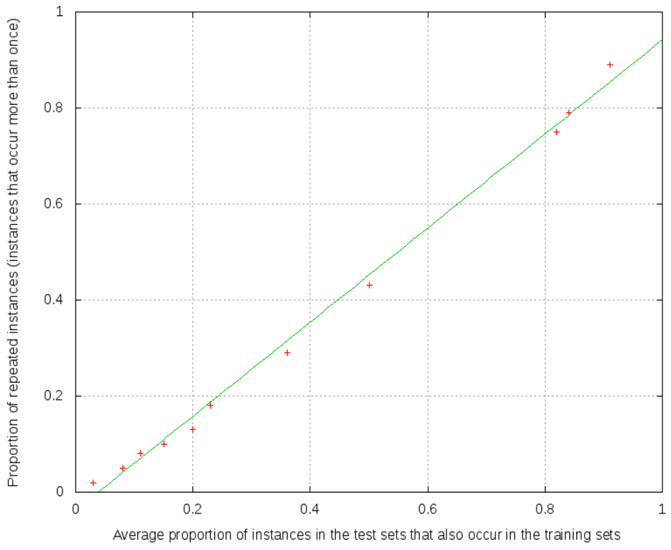

Fig. 4. Proportions of repeated data verses seen data in testing sets.

D. 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 II. Instead, we construct an artificial data set. This data set has 10 numeric features and 1000 data points. The numeric features were all generated by a pseudo-random number generator and have a 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 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 instances in each class, 1250 instances in total.
- 50% repeats from each class, an extra copy of 250 instances in each class, 1500 instances in total.
- 75% repeats from each class, an extra copy of 375 instances in each class, 1750 instances in total.
- 100% repeats, two copies of every instance, 2000 instances in total.

We used a random forest meta decision tree learner for this experiment, with 100 trees and all other parameters set to the Weka defaults. The results from this experiment showed accuracy levels for each data set: original, 25% repeats, 50% repeats, 75% repeats and 100% repeats of 48.30, 65.20, 80.47, 87.49 and 93.50. This clearly shows that repeated data points can have a huge influence on the performance of classifiers, even with pseudo-random data. As the proportion of repeated information increases, so does the performance of the classifier. It is worth pointing out that the severity of repeated data points is algorithm specific. Naive Bayes classifiers have been reported to be fairly resilient to duplicates (Kołcz et al. 2003).

V. Conclusions

Regardless of whether repeated data points are, or are not noise, it is unsuitable to have seen data in testing sets during defect prediction experiments intended to show how well a classifier could potentially perform on future, unseen data points. This is because having identical data points in training and testing sets can result in an excessive estimate of performance, occurring because classifiers can, to varying degrees, memorise rather than learn. 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 as it is possible for modules with identical metrics to be contained within a software system, such data points should be tolerated rather than removed. While the initial part of this argument is true, if, in the real world, you happen to have a data point in your testing set which is also contained in your training set, chance is on your side. However, it is not scientific for chance to be on the side of every researcher experimenting with the NASA data sets, as when unseen data is presented to the classifiers, performance may plummet from the expected.

If researchers believe that repeated data points are a correct representation of the software system (i.e. not noise), there are two options available. Firstly, it is possible to use data containing repeated data points, *so long as there are no common instances shared between training and testing sets*. This may lead researchers to designate the task of ensuring no seen data is contained within testing sets to machine learning software (i.e. during the data separation process; for example during cross-validation). Note however that this complicates the task of stratification. The second option, proposed in (Boetticher 2006), is to use an extra attribute: *number of duplicates*. This will help to ensure that information is not lost, and is most useful when data sets are believed to be of high quality.

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 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.

Data quality is very important during any data mining experiment, time spent analysing data is time well spent. We believe the data cleansing process defined in this paper will ensure that the NASA data sets become 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 included the repeated data points may have led to erroneous findings. 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, and 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.

References

Bezerra, M., Oliveira, A. & Meira, S. (2007), A constructive rbf neural network for estimating the probability of defects in software modules, pp. 2869–2874.

Boetticher, G. (2006), Improving credibility of machine learner models in software engineering, *in* 'Advanced Machine Learner Applications in Software Engineering', Software Engineering and Knowledge Engineering.

Chawla, N. V., Japkowicz, N. & Kolcz, A. (2004), 'Special issue on learning from imbalanced datasets', *SIGKDD Explor. Newsl.* **6**(1).

Elish, K. O. & Elish, M. O. (2008), 'Predicting defect-prone software modules using support vector machines', *J. Syst. Softw.* **81**(5), 649–660.

Halstead, M. H. (1977), *Elements of Software Science (Operating and programming systems series)*, Elsevier Science Inc., New York, NY, USA.

Jiang, Y. & Cukic, B. (2009), Misclassification cost-sensitive fault prediction models, *in* 'Proceedings of the 5th International Conference on Predictor Models in Software Engineering', PROMISE '09, ACM, New York, NY, USA, pp. 20:1–20:10.

Jiang, Y., Cukic, B. & Menzies, T. (2008), Can data transformation help in the detection of fault-prone modules?, *in* 'DEFECTS '08: Proceedings of the 2008 workshop on Defects in large software systems', ACM, New York, NY, USA, pp. 16–20.

Kaminsky, K. & Boetticher, G. (2004), Building a genetically engineerable evolvable program (GEEP) using breadth-based explicit knowledge for predicting software defects, pp. 10–15 Vol.1.

Khoshgoftaar, T. M. & Allen, E. B. (2003), ‘Ordering fault-prone software modules’, *Software Quality Control* **11**, 19–37.

Khoshgoftaar, T. M. & Seliya, N. (2004), The necessity of assuring quality in software measurement data, *in* ‘METRICS ’04: Proceedings of the Software Metrics, 10th International Symposium’, IEEE Computer Society, Washington, DC, USA, pp. 119–130.

Kim, S., Zimmermann, T., Pan, K. & Whitehead, E. J. J. (2006), Automatic identification of bug-introducing changes, *in* ‘ASE ’06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering’, IEEE Computer Society, Washington, DC, USA, pp. 81–90.

Kołcz, A., Chowdhury, A. & Alspector, J. (2003), Data duplication: an imbalance problem?, *in* ‘ICML 2003 Workshop on Learning from Imbalanced Datasets,’.

Koru, A. G. & Liu, H. (2005), ‘An investigation of the effect of module size on defect prediction using static measures’, *ACM SIGSOFT Software Engineering Notes* **30**(4), 1–5.

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

Li, Z. & Reformat, M. (2007), ‘A practical method for the software fault-prediction’, *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on* pp. 659–666.

Liebchen, G. A. & Shepperd, M. (2008), Data sets and data quality in software engineering, *in* ‘PROMISE ’08: Proceedings of the 4th international workshop on Predictor models in software engineering’, ACM, New York, NY, USA, pp. 39–44.

McCabe, T. J. (1976), A complexity measure, *in* ‘ICSE ’76: Proceedings of the 2nd international conference on Software engineering’, IEEE Computer Society Press, Los Alamitos, CA, USA, p. 407.

Mende, T. & Koschke, R. (2009), Revisiting the evaluation of defect prediction models, *in* ‘Proceedings of the 5th International Conference on Predictor Models in Software Engineering’, PROMISE ’09, ACM, New York, NY, USA, pp. 7:1–7:10.

Menzies, T., Dekhtyar, A., Distefano, J. & Greenwald, J. (2007), ‘Problems with precision: A response to “comments on ‘data mining static code attributes to learn defect predictors’”’, *IEEE Transactions on Software Engineering* **33**, 637–640.

Menzies, T., Greenwald, J. & Frank, A. (2007), ‘Data mining static code attributes to learn defect predictors’, *Software Engineering, IEEE Transactions on* **33**(1), 2–13.

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y. & Bener, A. (2010), ‘Defect prediction from static code features: current results, limitations, new approaches’, *Automated Software Engg.* **17**(4), 375–407.

Menzies, T. & Stefano, J. S. D. (2004), ‘How good is your blind spot sampling policy?’, *High-Assurance Systems Engineering, IEEE International Symposium on* **0**, 129–138.

Menzies, T., Stefano, J. S. D., Orrego, A. & Chapman, R. (2004), ‘Assessing predictors of software defects’, *Proceedings of Workshop on Predictive Software Models* .

Nickerson, A. S., Japkowicz, N. & Milios, E. (2001), Using unsupervised learning to guide resampling in imbalanced data sets, *in* ‘In Proceedings of the Eighth International Workshop on AI and Statitsics’, pp. 261–265.

Pelayo, L. & Dick, S. (2007), Applying novel resampling strategies to software defect prediction, pp. 69–72.

Seliya, N., Khoshgoftaar, T. & Zhong, S. (2005), Analyzing software quality with limited fault-proneness defect data, pp. 89–98.

Tosun, A. & Bener, A. (2009), Reducing false alarms in software defect prediction by decision threshold optimization, *in* ‘Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement’, ESEM ’09, IEEE Computer Society, Washington, DC, USA, pp. 477–480.

Turhan, B., Menzies, T., Bener, A. B. & Di Stefano, J. (2009), ‘On the relative value of cross-company and within-company data for defect prediction’, *Empirical Softw. Engg.* **14**, 540–578.

Witten, I. H. & Frank, E. (2005), *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Series in Data Management Systems, second edn, Morgan Kaufmann.