The Jinx on the NASA Software Defect Data Sets

Jean Petrić
Science and Technology Research Institute
University of Hertfordshire
Hatfield, Hertfordshire
AL10 9AB, UK
j.petric@herts.ac.uk

David Bowes
Science and Technology Research Institute
University of Hertfordshire
Hatfield, Hertfordshire
AL10 9AB, UK
d.h.bowes@herts.ac.uk

Tracy Hall
Department of Computer Science
Brunel University London
Uxbridge, Middlesex
UB8 3PH, UK
tracy.hall@brunel.ac.uk

Bruce Christianson
Science and Technology Research Institute
University of Hertfordshire
Hatfield, Hertfordshire
AL10 9AB, UK
b.christianson@herts.ac.uk

Nathan Baddoo
Science and Technology Research Institute
University of Hertfordshire
Hatfield, Hertfordshire
AL10 9AB, UK
n.baddoo@herts.ac.uk

ABSTRACT

Background: The NASA datasets have previously been used extensively in studies of software defects. In 2013 Shepperd et al. presented an essential set of rules for removing erroneous data from the NASA datasets making this data more reliable to use.
Objective: We have now found additional rules necessary for removing problematic data which were not identified by Shepperd et al.
Results: In this paper, we demonstrate the level of erroneous data still present even after cleaning using Shepperd et al.'s rules and apply our new rules to remove this erroneous data.
Conclusion: Even after systematic data cleaning of the NASA MDP datasets, we found new erroneous data. Data quality should always be explicitly considered by researchers before use.

Keywords

Data quality, software defect prediction, machine learning

1. INTRODUCTION

Software defect prediction (SDP) uses historical software data to predict locations in code likely to have defects before the software is released. Defects may cause software behave in unintended ways deviating from requirements. To find defects, SDP researchers use quantitative measures of software, which are considered to correlate with parts of the software that are likely to be defective. Various, usually automated, approaches are used to learn from historical data and make predictions on new data.

Most automated defect prediction is performed using various machine learning techniques [7, 9]. Regression and classification are the two most commonly used approaches in defect prediction. Regression techniques predict the density or number of possible defects for each module, whilst classification techniques only give categorical information (i.e. a module is defective or non-defective). Gray et al. argue that regression techniques should be preferred over classification techniques since they can provide a priority list of potentially defective modules [2]. However, according to Wahono, 77% of studies have used classification techniques, compared to 14% that used regression techniques [9].

Historical software data is usually fed into machine learners for the purpose of their training. The training data contain quantitative measures for each module, along with the number or label depicting whether a module is defective or not. Machine learners then search for patterns in data and derive mathematical rules for predicting defectiveness. Therefore, the accuracy of predictions highly relies on the quality of historical data. Lessmann et al. conducted a comprehensive study benchmarking over 20 different machine learning algorithms on the NASA MDP datasets [6]. The authors concluded that the top performing 17 classifiers do not result in significantly different prediction performances. However, after Gray et al.'s [3] and Shepperd et al.'s [8] efforts on data cleansing, Ghotra et al. [1] performed a similar study to Lessmann et al. on the cleaned version of the NASA MDP datasets. In this case, the authors concluded that the classifiers' prediction performances are significantly different, and therefore that the choice of a classifier matters. Hence, poor data quality may significantly affect the conclusions that researchers derive.

In the early 2000s, the lack of data availability was a great challenge [5]. However, NASA stepped in and published the NASA Metrics Data Program datasets for researchers to use. These datasets soon became very popular among researchers, since the data in its original form can easily be used for doing defect prediction. According to Hall et al., the NASA MDP datasets have been used in 62 out of 208 software prediction studies from 2000 to 2010 [4]. However,

EASE '16, June 01-03, 2016, Limerick, Ireland

DOI: http://dx.doi.org/10.1145/2915970.2916007

not much contextual information was released along with the datasets. Without articulating sufficient contextual information it becomes more difficult for researchers to understand their results. Ambiguity in the metric definitions makes it difficult or even impossible to verify the consistency of these metrics. Therefore, future effort in collecting defect data should ensure that enough contextual information is also retrieved.

Shepperd et al.'s paper: "Data Quality: Some Comments on the NASA Software Defect Datasets" was published in the September 2013 issue of the IEEE Transactions on Software Engineering journal [8]. Their paper, which extends the work of Gray et al. [2, 3], presents a set of cleaning steps for removing erroneous data from the *NASA Metrics Data Program* datasets. The aim of this paper is to point out additional inconsistency in some of the NASA MDP datasets. The contribution of this paper is two-fold. First, we introduce additional integrity checks essential for cleaning the NASA MDP data before use. Second, we show to what extent the NASA MDP data is affected by applying these additional integrity checks.

In the next section we present related work aimed at cleaning NASA MDP datasets. Following the related work, we show our findings and point out some additional integrity checks that should be addressed when cleaning the defect prediction data. Our conclusions are set out in the final section.

2. RELATED WORK

Gray et al. systematically questioned the quality of the NASA MDP datasets [2, 3]. Gray et al. pointed out a series of problems in the data and offered a solution for dealing with those problems. They identified several concerns that can be summarised as:

- *repeated or inconsistent instances*: multiple software modules contain the same attribute (i.e. static code metric) and class values (i.e. defective label). This situation is clearly possible in the real world, however repeated data points can cause over-optimistic performances when used in the machine learning context. Inconsistent instances happen when multiple software modules have the same attribute values, but different class values. Again, such situation is possible in the real world context, however it can potentially have a negative impact on machine learners;

- *data integrity*: for example, 1.1 lines of code is a clear example of such integrity check;

- *constant and repeated attributes*: attributes that do not contain any variance are of no use for machine learners. Repeated attributes, on the other hand, contain the same values for each instance, which can harm predictions having some attribute over-represented;

- *missing values*: can either be harmful or completely ignored by classification methods.

Shepperd et al. have built on the work of Gray et al. producing a comprehensive set of rules for data cleansing [8]. Their most significant extension from Gray et al.'s rules was in data integrity. They compiled a list of 18 different referential integrity checks that can test the validity of data instances. Overall, they found a significant number of erroneous data points, which they divided into two domains. The first domain contains only the problematic data, i.e. the data with impossible values. The other domain deals with the data that is not problematic, but does not help defect prediction (e.g. repeated attributes). All instances that fit in either of these two categories were removed. The authors provided the cleaned versions of the NASA datasets for both domains to the scientific community.

3. INVESTIGATION AND RESULTS

Our analysis is based on the Shepperd et al.'s cleaned versions of the NASA datasets from the tera-PROMISE repository[1], namely DS′ and DS″. DS′ denotes data with conflicting attribute values and implausible values removed, whilst DS″ is a dataset from which data have been removed that are not problematic but which do not help improve defect prediction (e.g., attributes with constant values, as defined by Shepperd et al. [8]). The cleaned versions of the NASA datasets were not publicly available from the Shepperd et al. original site[2] at the time of our analysis. Consequently, we used the Shepperd et al.'s cleaned version of the NASA datasets from the tera-PROMISE repository. The tera-PROMISE repository did not contain all 14 datasets initially published by the NASA MDP. Datasets KC1 and KC4 were not available in the tera-PROMISE repository, and dataset KC2 did not contain Shepperd et al.'s cleaned versions of the data. The remaining 11 NASA datasets used in this study were at revision number 7[3] in the tera-PROMISE SVN repository. Although the Shepperd et al. and tera-PROMISE versions of the cleaned datasets may differ, we used the tera-PROMISE version as this was the only version now available. However, the tera-PROMISE version is frequently used in software defect prediction studies.

Table 1 provides definitions and acronyms of the *lines of code* (LOC) metrics used in this study. These definitions were available on the now defunct MDP site[4] in the original *NASA Metrics Data Program* documents. For the sake of simplicity and space, we use the letters a to e to denote the LOC metrics used and replace the *number of lines* metric with the letter N (as shown in Table 1). We introduce a new variable, called ξ, which quantifies the missing lines in a module (also shown in Table 1).

Table 1 shows that N counts all LOC between open and close brackets in a module. Because all of these 11 NASA systems were written in either C/C++ or JAVA, only a limited number of code structures are allowed to occur between open and close brackets. In particular, a module may contain a number of: blank lines (variable d), comment lines (variable b) and lines containing code. In the NASA data sets lines containing code are divided into either code and comment on the same line (variable a) or only executable code on a line (variable c). Consequently N is equal to:

$$N = a + b + c + d + \xi \qquad \text{(IC1)}$$

where we should have $\xi = 0$ provided that the variable a is

[1]http://openscience.us/repo/
[2]http://j.mp/scvvIU
[3]The most recent revision at the time of conducting our analysis
[4]http://mdp.ivv.nasa.gov

Table 1: Definition of lines of code metrics in NASA MDP datasets

Acronym	Metric	Definition
a	LOC_CODE_AND_COMMENT	The number of lines which contain both code & comment in a module.
b	LOC_COMMENTS	The number of lines of comments in a module.
c	LOC_EXECUTABLE	The number of lines of executable code for a module (not blank or comment)
d	LOC_BLANK	The number of blank lines in a module.
e	LOC_TOTAL	The total number of lines for a given module.
N	NUMBER_OF_LINES	Number of lines in a module. Pure, simple count from open bracket to close bracket. Includes every line in between, regardless of character content.
ξ	IRREGULARITY_COUNT	The number of unexpected missing lines in a module. We add this metric to support a novel rule for removing erroneous data.

Table 2: Results of erroneous data in the NASA defect datasets violating two new integrity checks

Dataset	DS'						DS"					
	IC1 partition				IC1 violation (P1 + P4)	IC2 violation	IC1 partition				IC1 violation (P1 + P4)	IC2 violation
	P1	P2	P3	P4			P1	P2	P3	P4		
CM1	5	1	311	27	32 (9.3%)	0%	4	1	295	27	31 (9.48%)	0%
JM1	-	-	-	-	-	9555 (99.6%)	-	-	-	-	-	7753 (99.63%)
KC3	0	0	128	72	72 (36%)	0%	0	0	123	71	71 (36.6%)	0%
MC1	0	0	3552	5725	5725 (61.71%)	0%	0	0	115	1873	1873 (94.22%)	0%
MC2	0	0	1	126	126 (99.21%)	0%	0	0	1	124	124 (99.2%)	0%
MW1	0	0	264	0	0 (0%)	0%	0	0	253	0	0 (0%)	0%
PC1	12	1	711	35	47 (6.19%)	0%	12	1	660	32	44 (6.24%)	0%
PC2	-	-	-	-	-	0%	-	-	-	-	-	0%
PC3	2	3	1087	33	35 (3.11%)	0%	2	3	1040	32	34 (3.16%)	0%
PC4	0	0	275	1124	1124 (80.34%)	0%	0	0	228	1059	1059 (82.28%)	0%
PC5	0	0	1504	15497	15497 (91.15%)	0%	0	0	94	1617	1617 (94.51%)	0%

not subsumed in c, and similarly the variable b is not subsumed in a. To verify that the variable a is not subsumed in c we found multiple data points where $a > c$. The same verification test was used to confirm that the variable b is not subsumed in a. Both verification tests were performed after the datasets had been cleaned using our integrity checks. C/C++ programming languages allow the use of preprocessor directives that could have been ignored by the metric extraction tool and not counted in any of the LOC metrics described above. However, we confirmed that this is not the case since dataset KC3 is written in Java and some of KC3 instances also violate the (IC1) rule.

Table 3: All possible outcomes of violating the IC1 rule

Partition	Rule	Description
P1	$\xi < 0$	Implausible values. It is impossible to have more lines of code than total number of lines in a module.
P2	$\xi = 0$	Expected values. Number of lines in a module matches the sum of lines of the code metrics.
P3	$\xi = 1$	Out by one. This is the most common of these issues in the NASA datasets.
P4	$\xi > 1$	The ξ are the missing lines in the dataset.

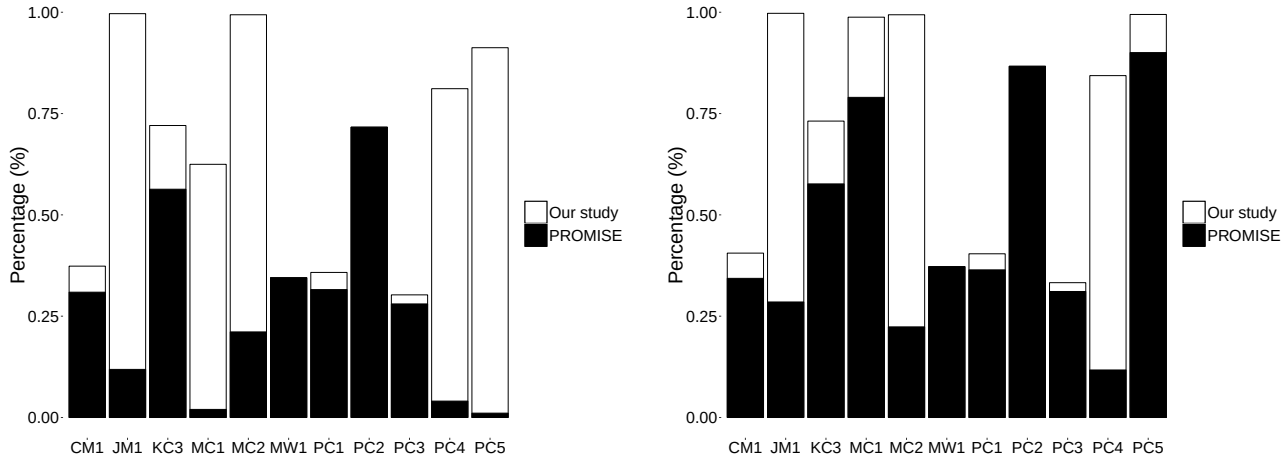
Although all data should obey equation (IC1) with $\xi = 0$, this is not the case for all of the 11 NASA datasets we analysed. Table 3 presents all possible outcomes of calculating the integrity check (IC1). Partition *P2* is the ideal situation where the data complies with (IC1), and therefore to the equation (IC1). Partition *P3* is where the result is out by one. This is a commonly occurring situation in these datasets and could be explained by tools not counting the last line in a module. If the last line of a module is just a bracket, the tool may count this as an additional line (N) but not as a blank line (d), because the new line character comes after the close bracket. This is not the case for the beginning of a module, because the new line character comes after the open bracket. However we cannot verify our suspicion as no code is available with the NASA datasets. We do not remove these out by one instances and accept the data that are present in the partition *P3* set. Partition *P1* presents an impossible situation, because N must always be equal to or greater than the sum of metrics from which it is derived. In *P4* missing lines of code occur, which we capture with the ξ variable. But, because we cannot know for certain the source of ξ and what effect it may have on defect prediction, we should probably avoid such occurrences. Therefore, our opinion is that the data in the *P1* and *P4* sets are violating the (IC1) rule.

The total number of lines in a module (e) for all investigated NASA datasets was equal to:

$$e = a + c \qquad \text{(IC2)}$$

except for the JM1 dataset. We derived the IC2 rule by checking its validity on 10 out of 11 NASA datasets we used. Unfortunately, due to non-availability of the NASA source code and information about the metric tools used, we could not check the reason for this anomaly. However, it is likely that problems were encountered during the collection of metrics for the JM1 dataset because it is the only instance of the NASA datasets that violates the (IC2) rule.

Figure 1: The erroneous instances discarded by Shepperd et al. (PROMISE data) and by (Our study)



(a) Problematic data (DS′)

(b) Data that do not add to defect prediction (DS″)

We also encountered problems with data availability. Datasets JM1 and PC2 were missing N and d metrics, respectively, so we could not check the (IC1) rule for these two NASA datasets.

Table 2 presents the overall results of applying the (IC1) and (IC2) rules to Shepperd et al. cleaned versions of the NASA datasets. Table 2 shows that some datasets are affected dramatically by applying the (IC1) rule. For example, more than 60% of the data in MC1, MC2, PC4 and PC5 breaks the (IC1) rule. Table 2 also shows that there is more erroneous data in the DS″ dataset than the DS′ dataset. This is because the DS″ cleaning procedure reduces the amount of data affected by Shepperd et al.'s rules, whilst not removing the data affected by our rules. Table 2 shows that JM1 and MC2 datasets are particularly problematic. After cleaning, insufficient data remains in the JM1 and MC2 datasets, rendering them poor candidates for defect prediction. Additionally, the post-cleaning of MC1 and PC4 datasets removed all defective data points, making them unusable for defect prediction.

Finally, Figure 1 compares the amount of instances discarded by applying the rules from Shepperd et al. and by our rules. The black bar denotes the instances removed from the tera-PROMISE version of the NASA MDP datasets by using Shepperd et al.'s cleaning rules. Similarly, the white bar denotes the instances eliminated by using our (IC1) and (IC2) rules. The figures clearly show that JM1, MC1, MC2, PC4 and PC5 NASA MDP datasets are highly affected by our rules and not by [8]. Furthermore, in the case of DS″ the JM1, MC1, MC2 and PC5 datasets remain with less than 5% of the original data points.

4. CONCLUSION

Software defect prediction models rely on the quality of the datasets on which they are built. NASA data has been used frequently in previous defect prediction studies. The quality of the NASA data underpins the confidence that we can have in the results of studies using this data. Because data collection mistakes are inevitable, it is essential that the quality of data is explicitly considered and that data is cleaned before use. It is critical that the data cleaning presented by Shepperd et al. and extended here by us is applied to the NASA data before it is used in future studies.

5. ACKNOWLEDGEMENTS

The authors would like to thank Dr David Gray and Professor Martin Shepperd on their valuable comments and recommendations. This work was partly funded by a grant from the UK's Engineering and Physical Sciences Research Council under grant number: EP/L011751/1.

6. REFERENCES

[1] B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *37th Int. Conf. on Software Engineering (ICSE)*, 2015.

[2] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the NASA metrics data program data sets for automated software defect prediction. In *Evaluation Assessment in Software Engineering (EASE 2011)*, pages 96–103, 2011.

[3] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. Reflections on the NASA MDP data sets. *Software, IET*, 6(6):549–558, Dec 2012.

[4] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, Nov 2012.

[5] Y. Kamei and E. Shihab. Defect prediction: Accomplishments and future challenges. In *Software Analysis, Evolution and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, 2016.

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

[7] R. Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504 – 518, 2015.

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

[9] R. S. Wahono. A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks. *Journal of Software Engineering*, 1(1):1–16, 2015.