Lessons Learned from the Assessment of Software Defect Prediction on WLCG Software

A Study with Unlabelled Datasets and Machine Learning Techniques

Elisabetta Ronchieri[1,*], *Marco* Canaparo[1,**], *Mauro* Belgiovine[1,***], *Davide* Salomoni[1,****], and *Barbara* Martelli[1,†]

[1]INFN CNAF, Bologna, Italy

Abstract.
Software defect prediction is an activity that aims at narrowing down the most likely defect-prone software modules and helping developers and testers to prioritize inspection and testing. This activity can be addressed by using Machine Learning techniques applied to software metrics datasets that are usually unlabelled, i.e. they lack modules classification in terms of defectiveness. To overcome this limitation, in addition to the usual data pre-processing operations to manage mission values and/or to remove inconsistencies, researches have to adopt an approach to label their unlabelled software datasets. The extraction of defectiveness data to label all the instances of the datasets is an extremely time and effort consuming operation. In literature, many studies have introduced approaches to build a defect prediction models on unlabelled datasets.
In this paper, we describe the analysis of new unlabelled datasets from WLCG software, coming from HEP-related experiments and middleware, by using Machine Learning techniques. We have experimented new approaches to label the various modules due to the heterogeneity of software metrics distribution. We discuss a number of lessons learned from conducting these activities, what has worked, what has not and how our research can be improved.

1 Background

Machine learning (ML) as a means to help in different Software Engineering (SE) tasks, such as software defects prediction and test code generation, has been often considered in research studies in the last decades [1–5]. ML techniques are fed with input software data properly processed and collected in datasets that are composed of instances, i.e. software modules (such as files, classe and functions), and features, i.e. software metrics [6]. For the software defect prediction, the actual defect information of instances is also mandatory in supervised ML techniques; nevertheless, it may be not enough in the software archives of new or recent software projects, or it can not have been traced properly in already existing software projects

[*]e-mail: elisabetta.ronchieri@cnaf.infn.it
[**]e-mail: marco.canaparo@cnaf.infn.it
[***]e-mail: mauro.belgiovine@cnaf.infn.it
[****]e-mail: davide.salomoni@cnaf.infn.it
[†]e-mail: barbara.martelli@cnaf.infn.it

[7]. This constitutes a serious limitation in the utilization of supervised-based ML techniques [8].

To address the limitation of supervised-based learning techniques in constructing defect prediction models by using unlabelled datasets, researches have proposed various approaches which can be categorized in five groups.

1. The within-project defect prediction (WPDP) is a typical prediction process built for a specific project and is based on supervised ML. This approach is characterized by a high precision, however, its prediction model can hardly be used for other projects' prediction being built on a single software project [9].

2. The cross-project defect prediction (CPDP) builds a prediction model using a labelled datasets. Then, it uses the same model to predict if an instance of another software project is defective or not. This approach might be useful in case of new projects or projects with limited defect information, however, it assumes that the two datasets have the same set of metrics and that they have the same probability distribution [10].

3. The expert-based defect prediction first employs a clustering algorithm, like K-means, to cluster the unlabelled instances, then it relies on a human expert for labelling each cluster as defective or not [11]. The major limitation of this approach is that it requires human experts to categorize cluster as defective.

4. The threshold-based defect prediction approach predicts an instance as buggy when any metric value is greater than the given metric's threshold. This approach can be automatized, however, defectiveness prediction is dependent on metrics thresholds which must be established in advance [12].

5. The Clustering, LAbelling, Metric selection, Instance selection (CLAMI) approach is based on a four-step procedure to be applied to the instances of an unlabelled dataset. It is an automatizable approach, which does not involve human effort and relies on metrics' values which may not always be comparable and may introduce bias. CLAMI is dependent on metric thresholds [13]. CLAMI+ is an evolution of the CLAMI approach: it employs a different procedure in the metrics' selection phase. CLAMI+ is still dependent on thresholds, but it normalizes metrics' values [14].

The extraction of the complete set of features (metrics and labels) is time and effort consuming: moreover, the selection of the right tool for metrics' extraction can be difficult to conduct. For these reasons, unlabelled datasets are the vast majority of software datasets. To perform defect prediction with unlabelled datasets it is necessary to find an automatizable way to label instances.

2 Experimental Setup

The Worldwide LHC Computing Grid (WLCG) [15] employs a wide variety of software whose vast majority has been using devops procedures in their development and maintenance phases. The adopted tools collect software metrics, e.g. cyclomatic complexity or lines of code [16, 17], over the releases, that can be used to build software datasets. In our study, we have found that software projects have documentation related to code changes, like release notes, which can be exploited to provide an assessment of the defectiveness prediction in software.

Our work aims at testing the usefulness of ML techniques in WLCG domain in terms of the identification of the pieces of code, which require particular attention during the development and maintenance phases of sofware. ML techniques can help in selecting software modules that should be examined with greater care by developers. To achieve this goal, we have constructed a defect prediction model by exploiting the unlabelled software datasets of Geant4 that is one of the most rigorously validated software packages for the simulation of the passage of particles through matter [18]. Amongst the different ML methodologies, we have selected CLAMI [13] and CLAMI+ [14] in order to label the instances in the software datasets. In addition, we have applied a large set of ML techniques to predict defect-prone modules.

Our approach (summarized in Figure 1) uses as input a subset of the Geant4 software dataset composed of 482 modules with 66 software metrics and 34 releases. This dataset has been obtained by applying a tool for the static analysis - Imagix 4D tool [19] - to the various modules of several software releases [20]. Imagix 4D tool's output has been preprocessed in order to keep only the common software modules and software metrics among the Geant4 software versions. This preprocessing activity has produced a 34 multiversions dataset each composed of 482 software modules and 66 software metrics. The used 34 releases begin with Geant4 0.0.4 to Geant 10.0.4.

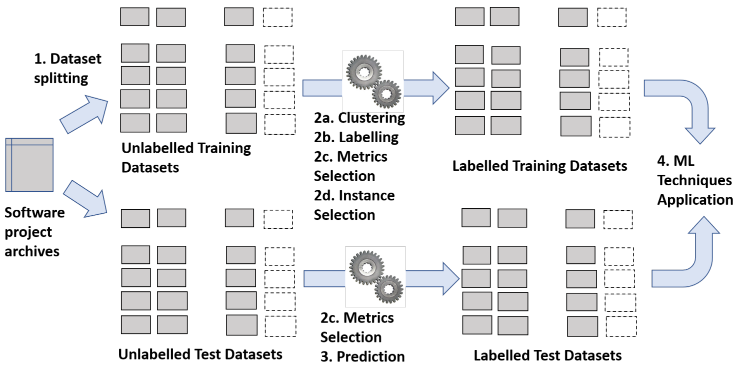

Figure 1. Workflow

The input unlabelled dataset is turned into labelled dataset in 4 steps, 3 out of 4 are based on the CLAMI approach. In the first step we have split the unlabelled dataset into a training and a test dataset. The training dataset is composed of the 67% of the total instances, the remaining instances have been included in the test dataset. With the aim of labelling the training dataset, we have applied the CLAMI approach.

We have used an unsupervised algorithm to cluster instances by relying on the magnitude of their metrics. More in detail, in the clustering phase, we have identified, for each instance, the metrics that are greater than a specific cutoff threshold (e.g. the median value) and then determined the number of metrics K, whose values have exceeded the threshold. Afterwards, the instance have been clustered according to their K values. By relying on previous literature, it is known that instances with larger values on their metrics are more likely defective [21–23]. Therefore, we have discriminated the top half cluster as defective code and the bottom half as non defective code.

The metrics selection phase considers the exclusion of the metrics that violate the defect proneness tendency [24]. The violation occurs, for example, when in a defective-labelled instance, a metric does not exceed its cutoff threshold, or, on the contrary, when, in a non defective-labelled instance, a metric exceeds its cutoff threshold. For each metric, we have computed the metric violation score (MVS) that is the ratio between the number of violations and the number of metric's values. The metrics with lower MVS are selected for the training dataset. The last CLAMI phase considers to remove all the instances with any violated metric values. If this operation produces a training dataset without either defective or non defective instances, then another MVS value is to be chosen and the last step need to be reiterate. The CLAMI+ approach differs from CLAMI in the Clustering and Labelling steps (2a and 2b in Figure 1). More in detail, CLAMI+ transforms the Boolean representation in CLAMI of metrics' violation into a probabilistic value based on the difference between the metric value and the threshold. Consequently, CLAMI+ considers how much an instance violated on a metric and leads to a different selection of the final training set that is expected to be more informative than that built by CLAMI [14].

Once obtained the labelled dataset, we have applied various ML techniques, previously selected among those that have already been used in the Software Engineering field, and, more in detail, in the software defect prediction problem, as presented in previous literature [25]: AdaBoost (AB) [26], Boosted Logistic Regression (BLR) [21, 27], J48 [28], Cost-Sensitive C5.0 (C5.0 Cost) [29], Logistic Model Tree (LMT) [30], Multilayer Perceptron (MLP) [31], Support Vector Machines with Radial Basis Function Kernel (SVM Radial) [32], Partial Least Squares (PLS) [33], Boosted Tree (BT) [34] and Random Forest (RF) [35]. In order to compare the different ML techniques, we have employed the most common performance indicators detailed in literature. For a question of space, the indicators that we have shown in section 3 are:

- Accuracy that measures the percentage of instances correctly classified as either defective or non defective;
- Kappa statistics [36], whose value ranges from 0 to 1, that determines how much better a classifier is performing over the performance of a classifier that simply guesses at random. When Kappa's value is between 0.81 to 0.99, this value indicates an almost perfect agreement.
- Area Under the ROC (Receiver Operating Characteristic) Curve (AUC) [37] that is able to consider the ability of a classifier to differentiate between the two classes. AUC has lower variance and is more reliable than other performance metrics for software defect prediction [38].

The assessment of our predictions has been conducted by comparing our results against the software documentation like release notes.

3 Lessons Learned

In this section, we will discuss the key lessons that we have learned from our experience and also propose activities for future research. It is worth highlighting that our approach uses as input a subset of the Geant4 software dataset that is composed of 482 modules with 66 software metrics and 34 releases.

In the labeling phase, we have determined the defect-prone modules for each release and different quantiles according to CLAMI and CLAMI+ approaches. Lower cutoff values identify less defect prone software modules and, on the other hand, modules with larger values in all metrics are more likely defective [22]. This study considers just a subset of modules available for each release, therefore even though the labeled modules as defectives have found a correspondence in the Geant4 documentation, we believe the current results may envelope a bias for which we have a further investigation.

In the metrics selection phase, we have removed from 33% to 55% of the total number of metrics because our choice was to keep the metrics that were in common to the various releases and quantiles (i.e. cut-off values choose for the metrics). Therefore, the average number of the selected metrics is 38 out of 66 and, more in detail, this resulting set was composed of metrics belonging to the size, complexity, maintainability and object orientation categories.

For the defect prediction, we have applied various classification and regression techniques on training datasets with 10-fold cross validation and assessed them on test datasets. We have noticed that Kappa static value inferior to 0.81 corresponded to values of Accuracy inferior to 90%. We have excluded all the prediction models whose Kappa statistic scored less than 0.81 in order to consider a good agreement between the observed and the expected accuracy. Table 1 and Table 2 show which ML techniques perform best, in terms of accuracy and AUC, by using either the CLAMI approach or the CLAMI+ approach, over the various quantiles. Quantiles' numbers from 1 to 9 correspond to 10%, ..., 90% cutoff values respectively.

Table 1. Average Accuracy of the ML Techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging
1	CLAMI	98.1670	98.1072	98.1381	97.3868
1	CLAMI+	98.8291	98.8101	98.8471	98.5651
2	CLAMI	96.8106	97.1064	96.6285	96.9970
2	CLAMI+	96.9223	96.6182	97.0572	96.3726
3	CLAMI	95.3838	95.1111	95.0132	94.9443
3	CLAMI+	94.6002	94.5518	94.6830	94.5434
4	CLAMI	93.0748	93.2507	93.3463	93.1027
4	CLAMI+	92.7295	92.3045	92.5200	92.7422
5	CLAMI	92.4850	92.4551	91.8897	92.5070
5	CLAMI+	93.2537	93.9593	93.7013	93.6975
6	CLAMI	93.2719	93.5851	93.6380	93.4789
6	CLAMI+	93.7195	93.9813	94.4491	94.0966
7	CLAMI	93.8482	94.4191	94.2796	94.2807
7	CLAMI+	94.9153	95.1850	95.5844	94.9700
8	CLAMI	94.2239	95.5005	95.0125	94.7273
8	CLAMI+	96.6170	96.7744	96.9310	96.2294
9	CLAMI	94.7243	95.0025	95.4197	94.9242
9	CLAMI+	98.7572	99.3769	99.1762	98.8717

Table 2. Average defect-prone AUC of the ML Techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging
1	CLAMI	0.9321	0.9820	0.9895	0.9453
1	CLAMI+	0.9117	0.9792	0.9864	0.9757
2	CLAMI	0.9146	0.9729	0.9823	0.9779
2	CLAMI+	0.9297	0.9671	0.9855	0.9668
3	CLAMI	0.9347	0.9678	0.9822	0.9766
3	CLAMI+	0.9254	0.9660	0.9781	0.9737
4	CLAMI	0.9300	0.9682	0.9786	0.9732
4	CLAMI+	0.9291	0.9595	0.9733	0.9740
5	CLAMI	0.9253	0.9639	0.9706	0.9734
5	CLAMI+	0.9385	0.9780	0.9831	0.9804
6	CLAMI	0.9404	0.9789	0.9841	0.9816
6	CLAMI+	0.9295	0.9815	0.9854	0.9821
7	CLAMI	0.9355	0.9800	0.9837	0.9786
7	CLAMI+	0.9246	0.9851	0.9892	0.9836
8	CLAMI	0.9275	0.9868	0.9856	0.9777
8	CLAMI+	0.9144	0.9875	0.9886	0.9778
9	CLAMI	0.9136	0.9791	0.9854	0.9710
9	CLAMI+	0.8868	0.9983	0.9826	0.9314

With regard to internal validity, each Geant4 release contains release note that includes information about new development, change, bug fixes, performance improvement and so on.

According to this documentation the modules traced for changes and included in the small dataset have also been labelled as defect-prone during the labelling phase. We think that those modules may contain false positives and true negatives, and thus future work would involve investigating those defect-prone modules and confirming their validity. Geant4 is on board by many years: it is important to understand what is an improvement and a bug fix. Furthermore, the studied dataset may not be representative of all Geant4, and further investigation is need to confirm our findings.

In terms of external validity, we have considered 34 Geant4 versions, which differ in size, complexity, popularity and revision history. Our small dataset may not take the place of all kinds of WLCG software, and we have to extend our study to other types of source code, such as ROOT.

4 Conclusion

We have reported our experience which includes the labelling of a Geant4 multi-release unlabelled software dataset and the application of several Machine Learning techniques. The supervised machine learning algorithms explored take as input a representation for source code in terms of code metrics, and predict if a module is defective-prone or not. We have selected the machine learning techniques that scored more than 0.80 in the Kappa statistic performance metric: J48, Adaboost, LMT and Bagging. We have discovered that our findings on defect prone modules have a correspondence in the analysis of the software code documentation (e.g. release notes).

This research is at an early stage of effort. The lessons we have learned will lead to different improvements in order to make it more accesibile to readers in terms of explainability, scalability and usefulness.

We plan to extend our approach to the whole Geant4 datasets and apply this approachto other types of WLCG software. In addition, we would like to explore how our approach can be extended to assessing different types of defects such as bug fix, performance improvement and so on.

References

[1] S. Jha, R. Kumar, L.H. Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma, H.V. Long, IEEE Access: special section on new trends in brain signal processing and analysis **7**, 61840 (2019)

[2] N. Pritam, M. Khari, L.H. Son, R. Kumar, S. Jha, I. Priyadarshini, M. Abdel-Basset, H.V. Long, IEEE Access: New Trends in Brain Signal Processing and Analysis **7**, 37414 (2019)

[3] L. Wei, W. Luo, J. Weng, Y. Zhong, X. Zhang, Z. Yan, IEEE Access: Internet-of-Things (IoT) Big Data Trust Management **5**, 25591 (2017)

[4] F. Wu, X.Y. Jing, Y. Sun, L. Huang, F. Cui, Y. Sun, IEEE Transaction on Reliability **67**, 581 (2018)

[5] Z.W. Zhang, X.Y. Jing, F. Wu, IETSoftware **12**, 527 (2018)

[6] J. Ge, J. Liu, W. Liu, *Comparative Study on Defect Prediction Algorithms of Supervised Learning Software Based on Imbalanced Classification Data Sets*, in *19th IEEE/ACIS International Conference on Softtware Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (IEEE, 2018)

[7] Meiliana, S. Karim, H.L.H.S. Warnars, F.L. Gaol, E. Abdurachman, B. Soewito, *Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset*, in *IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)* (IEEE, 2017)

[8] C. Catal, B. Diri, *A Fault Prediction Model with Limited Fault Data to Improve Test Process*, in *Product-Focused Software Process Improvement*, edited by A. Jedlitschka, O. Salo (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), pp. 244–257, ISBN 978-3-540-69566-0

[9] X. Xuan, D. Lo, X. Xia, Y. Tian, *Evaluating Defect Prediction Approaches Using a Massive Set of Metrics: An Empirical Study*, in *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (Association for Computing Machinery, New York, NY, USA, 2015), SAC '15, pp. 1644–1647, ISBN 9781450331968, `https://doi.org/10.1145/2695664.2695959`

[10] L. Goel, D. Damodaran, S.K. Khatri, M. Sharma, *A literature review on cross project defect prediction*, in *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)* (2017), pp. 680–685

[11] Shi Zhong, T.M. Khoshgoftaar, N. Seliya, *Unsupervised learning for expert-based software quality estimation*, in *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.* (2004), pp. 149–155

[12] A. Boucher, M. Badri, *Using Software Metrics Thresholds to Predict Fault-Prone Classes in Object-Oriented Software*, in *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science Engineering (ACIT-CSII-BCD)* (2016), pp. 169–176

[13] J. Nam, S. Kim, *CLAMI: Defect Prediction on Unlabeled Datasets*, in *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (IEEE, Lincoln, NE, USA, 2016)

[14] M. Yan, X. Zhang, C. Liu, L. Xu, M. Yang, D. Yang, Information and Software Technology **92**, 1 (2017)

[15] D. Bonacorsi, T. Ferrari, Italian Meeting on High Energy Physics (2006)

[16] S. Huda, S. Alyahya, M. Mohsin Ali, S. Ahmad, J. Abawajy, H. Al-Dossari, J. Yearwood, IEEE Access **6**, 2844 (2018)

[17] H. Zhang, X. Zhang, M. Gu, *Predicting Defective Software Components from Code Complexity Measures*, in *13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)* (2007), pp. 93–96

[18] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nucl. Instrum. Methods Phys. Res., Sect. A **506**, 250 (2003)

[19] Imagix, *Imagix - Source Code Analysis*, `https://www.imagix.com/products/source-code-analysis.html`

[20] E. Ronchieri, M.G. Pia, T. Basaglia, M. Canaparo, Journal of Physics: Conf. Series **898** (2017)

[21] W. Rhmann, B. Pandey, G. Ansari, D.K. Pandey, Journal of King Saud UNiversity - Computer and Information Sciences (2019)

[22] T. Menzies, J. Greenwald, A. Frank, IEEE Trans. Softw. Eng. **33**, 2 (2007)

[23] K. Herzig, S. Just, A. Rau, A. Zeller, *Predicting defects using change genealogies*, in *24th International Symposium on Software Reliability Engineering (ISSRE)* (IEEE, 2013)

[24] M. D'Ambros, M. Lanza, R. Robbes, Empirical Software Engineering **17** (2012)
[25] Y.A. Alshehri, K. Goseva-Popstojanova, D.G. Dzielski, T.Devine, *Applying Machine Learning to Predict Software Fault Proneness Using Change Metrics, Static Code Metrics, and a Combination of Them*, in *SoutheastCon* (2018)
[26] Y. Gao, C. Yang, *Software Defect Prediction based on Adaboost algorithm under Imbalance Distribution*, in *Proceedings of the 2016 4th International Conference on Sensors, Mechatronics and Automation* ((2016))
[27] J. Otero, L. Sànchez, Soft Computing **10** (2006)
[28] M.C.M. Prasad, L. Florence, A. Arya, Internation Journal of Database Theory and Applkcation **8** (2015)
[29] M.J. Siers, M.Z. Islam, Information Systems **51** (2015)
[30] N. Landwehr, M. Hall, E. Frank, Machine Learning **59** (2005)
[31] M. Gayathri, A. Sudha, International Journal of Recent Technology and Engineering (IJRTE) (2014)
[32] P.A. Selvaraj, D.P. Thangaraj, International Journal of Engineering & Technology Research **1** (2013)
[33] G. Luo, Y. Ma, K. Qin, IEICE Transactions on Information and Systems (2012)
[34] F. Wang, J. Huang, Y. Ma, *A Top-k Learning to Rank Approach to Cross-Project Software Defect Prediction*, in *25th Asia-Pacific Software Engineering Conference* (2018)
[35] R.M. Magal, S.G. Jacob, International Journal of Computer Applications **117** (2015)
[36] J.R. Landis, G.G. Koch, Biometrics **33**, 159 (1977)
[37] T. Fawcett, Pattern Recognition **28**, 861 (2006)
[38] Y. Jiang, J. Lin, B. Cukic, T. Menzies, *Variance analysis in software fault prediction models*, in *International Symposium in Software Reliability Engineering* (2009), pp. 99–108