Machine Learning Techniques for Software Analysis of Unlabelled Program Modules

Elisabetta Ronchieri[*]
INFN CNAF, Bologna Italy
E-mail: elisabetta.ronchieri@cnaf.infn.it

Marco Canaparo
INFN CNAF, Bologna Italy
E-mail: marco.canaparo@cnaf.infn.it

Davide Salomoni
INFN CNAF, Bologna Italy
E-mail: davide.salomoni@cnaf.infn.it

Machine Learning (ML) has proven to be of great value in a variety of Software Engineering (SE) tasks to conduct, e.g., software defect prediction and estimation and test code generation. To accomplish these tasks, software datasets (i.e. collections of the various modules, such as files and classes, with features, such as software metrics and defective data) have to be gathered and properly preprocessed before the application of ML techniques.
In SE practice, software datasets may lack some features' classification data, e.g. defective data are not included being difficult to collect in new projects or in projects with partial historical data. These datasets are called unlabelled datasets and are the vast majority of software datasets. The extraction of the complete set of features (defectiveness included) and the classification of the various instances imply effort and time.
In literature, there exist various approaches to build a prediction model on unlabelled datasets that entail a high number of time-consuming permutations. Cloud computing infrastructure, GPU-equipped resources and adequate ML framework can give the chance to overcome this problem.
In this study, we are going to present the usefulness of the Clustering, LAbeling, Metric selection, Instance selection approach in high energy physics by applying them to a Geant4 software unlabelled dataset as a case study; by implementing models in different available frameworks, such as TensorFlow and Keras; and by running them in Java, Python and R. We intend to reduce the distance between theory and practice by providing strengths and limitations of the considered frameworks to enable users to assess suitability for their requirements.

International Symposium on Grids & Clouds 2019, ISGC2019
31st March - 5th April, 2019
Academia Sinica, Taipei, Taiwan

[*]Speaker.

1. Introduction

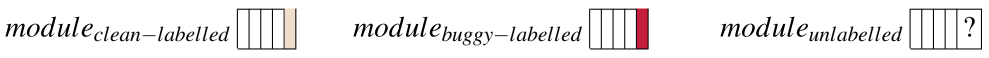
Software analysis is of vital importance in the assessment of software characteristics. It is usually based on software measurement and techniques derived from both statistics and Machine Learning (ML).

ML has been widely adopted in the field of Software Engineering (SE), whose tasks can be formulated as learning problems and approached through learning techniques. Over time, ML has proven to enable computer engineers e.g. to predict defects [1] in software [2] and generate source test code [3], offering a viable alternative to existing approaches in addressing SE issues. To accomplish these tasks, ML techniques normally require labelled datasets that are collections of features (like software metrics [1]) with various instances - i.e. modules (such as files and classes) - opportunely classified according to the considered task. These datasets have to be gathered and properly preprocessed before the application of ML techniques: typical data preprocessing operations may include replacement of missing values and/or removal of inconsistencies (like the total number of lines in a file is smaller than the number of code lines in a file).

Figure 1 shows an example of a labelled dataset composed of N modules and M metrics. Each cell <module, metric> contains the metric value of the specific module. Among the metrics, there is at least one that can be used to classify each module for example in terms of defectiveness, as shown in Figure 2: Figure 2a shows a module identified as clean (i.e. no defects), Figure 2b shows a module identified as buggy (i.e. with defects) and Figure 2c shows a module identified as unlabelled (i.e. no information available coded with the ? symbol). Defect prediction models are trained with the labelled modules and also tested with the unlabelled modules, i.e. modules whose defectiveness is unknown.

	$Metric_1$	$Metric_2$	$Metric_3$	$Metric_{...}$	$Metric_M$
$module_1$					🟥
$module_2$					▨
$module_3$					▨
$module_{...}$					?
$module_N$					?

Figure 1: Sheet of Software Labelled Dataset.



(a) Clean Module (b) Buggy Module (c) Unlabelled Module

Figure 2: Example of Module Classification for Defect Prediction.

In SE practice, software datasets may lack some features' classification data, such as the software module defectiveness, which are mandatory for the application of supervised learning techniques and are not easy to find either for new projects or in projects with partial historical data.

These datasets are called unlabelled datasets and they are the vast majority of software datasets. Figure 3 shows an example of an unlabelled dataset composed of N modules and M metrics, where all modules are not classified. The extraction of the complete set of features (defectiveness included) and the classification of the various instances imply effort and time, penalizing a real application of ML techniques to predict modules defectiveness.

	$Metric_1$	$Metric_2$	$Metric_3$	$Metric_{...}$	$Metric_M$
$module_1$					?
$module_2$					?
$module_3$					?
$module_{...}$					?
$module_N$					?

Figure 3: Sheet of Software Unlabelled Dataset.

In the last decade, unlabelled datasets have been explored with the purpose of conducting analysis and predictions [4, 5]. In literature, various approaches exist to build a prediction model on unlabelled datasets, such as cross-project defect prediction [6], expert-based defect prediction [5] and Clustering, LAbeling, Metric selection, Instance selection (CLAMI) [7]. All these approaches present some limitations mainly related to the dataset characteristics, the need of human experts and the selection of metrics thresholds. In this study, we have chosen CLAMI, because it is independent on the metrics thresholds, it does not rely on expert software knowledge and can be easily automated.

The resulting prediction models entail a high number of permutations that lead to a greater resources' consumption. Cloud computing infrastructure, GPU-equipped resources and adequate ML framework can constitute a helping hand to overcome this problem. Therefore, we have decided to explore the adoption of ML framework on (also GPU-equipped) cloud computing infrastructure to determine the best models according to intrinsic ML evaluation indicators, such as Accuracy.

In this study, we are going to present the usefulness of the CLAMI approach in high energy physics (HEP) by applying it to a Geant4 [8] unlabelled dataset as a case study and implementing models in different available frameworks. We have evaluated frameworks, such as TensorFlow [9] and Keras [10], and running them in Java, Python and R, by considering four aspects: extensibility, hardware utilization, speed [11] and their learning curve. Figure 4 shows the learning curve for the various frameworks we have experimented: Weka [12] is the easiest to use, while Theano [13] requires more expertise.

Due to the lack of a comprehensive study about practical aspects of software analytic models, we aim at providing a procedure to perform software defect prediction in a scientific environment in order to minimize human effort. Furthermore, we intend to reduce the distance between theory and practice by providing strengths and limitations of the considered frameworks to enable users to assess suitability according to their requirements.

The reminder of this paper is organized as follows. Section 2 presents the background in this context. Section 3 describes the experimental settings of this study. Section 4 discusses the results.

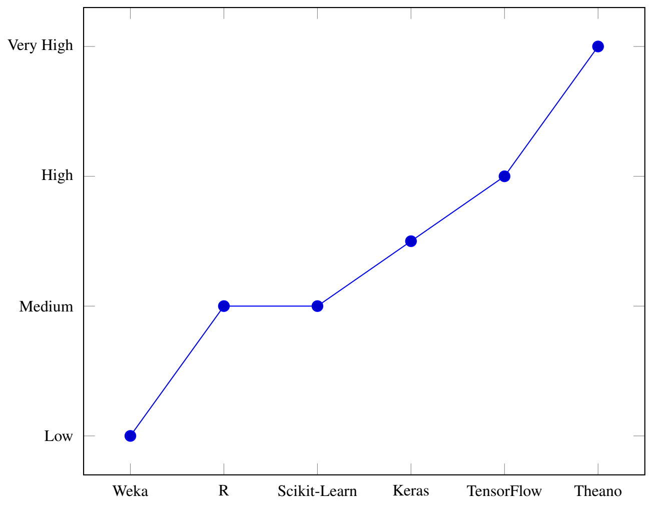


Figure 4: ML Framework Learning Curve.

Finally, Section 5 draws some conclusions and provides recommendations for further research.

2. Related Works

Computer scientists have tried to tackle the issue of building defect prediction models on new software projects or projects with limited historical data. In the following some approaches are provided.

Nam et al. [7] proposed CLAMI technique with the aim of automatizing defect prediction on unlabeled datasets by using the magnitude of metric values. Their approach consisted of 4 steps: Clustering of the instances of the unlabeled dataset; LAbeling of the obtained clusters in buggy or clean; Metrics selections based on the metric violation scores and, finally, Instance selection. By testing CLAMI on seven open source projects' datasets, the authors showed that it obtained promising prediction performance.

Catal et al. [4] explored two fault prediction approaches. The first one is a two stage approach consisting of clustering and checking against the metrics thresholds. The second approach uses the metrics thresholds standalone to identify the fault-prone and not fault-prone software modules. The authors concluded that the second approach was easier to conduct and still effective. The metrics thresholds were determined based on 'Experience and Hints from Literature', past defect prone modules and analysis of past version of the project.

Zhong et al. [5] proposed a clustering and expert-based approach. The first step consists of clustering software modules into groups by using some clustering techniques such as k-means, the second step involves software quality experts that labels each cluster as either fault-prone or not fault-prone based on their domain knowledge and data statistics.

Seliya et al. [14] introduced a semisupervised clustering scheme for software quality analysis of program modules without defect data. They employed K-means clustering method and showed that this approach helped experts in making better prediction with respect to predictions obtained by unsupervised learning algorithm.

Deshmukh et al. [15] tackled the problem of using unlabeled data in transfer learning. The proposed algorithm leverages partially labeled training datasets that consist of a small number of labeled instances and a large number of unlabeled instances. The solution proposed is a kernel and graph Laplacian based approach which is demonstrated to improve prediction as much as 5.9% on the sample dataset.

In this study, we have chosen the CLAMI approach because it is independent on the metrics thresholds, it does not rely on expert software knowledge and can be easily automated.

3. Experimental Settings

This study has been conducted by following a process made up of five steps to follow after identifying the unlabelled datasets and the ML techniques. Our process requires two inputs and provides two outcomes.

Input :

1. U = set of unlabelled instances
2. C = set of ML techniques

Process :

1. Repeat 2-5 T times for each $u \in U$ to conduct P predictions
2. Randomly split dataset in training (67%) dataset (with labelled defective instances) and test (33%) dataset
3. Construct classifier by applying $c \in C$ to training dataset
4. Assess classifier
5. Predict test dataset

Output :

1. Average I (I = set of performance indicators)
2. Test dataset prediction

3.1 Input sets

The software unlabelled datasets and the ML techniques are the input to our process.

The unlabelled datasets derive from the Geant4 [8] software - a toolkit for the simulation of the passage of particles through matter. For this software, effort has been dedicated to collect software metrics for the different modules and 34 software releases [16], and to analyse them with statistical methods [17].

Several software metrics unlabelled datasets have been built by using Imagix 4D tool [18]. Software metrics are a quantitative measurement that assigns numbers to attributes of the measured module [19]: an attribute is a property of a module, e.g. size. There are several different families of software metrics. Those that are measured by the tool belong to size (e.g., comment ratio and lines of code), object orientation (e.g., decision depth and class coupling from Chidamber and Kemerer metric suite [20]) and complexity (e.g., McCabe cyclomatic complexity [21] and Halsted complexity [22]) categories.

For this study, we have considered the same set of modules (i.e. classes) over the various Geant4 releases. Table 1 shows the characteristics of the major release 10 of Geant4 at class level. The percentage of buggy instances is not available (NA) since their datasets are unlabelled.

Release	#Classes	Buggy (%)	#Metrics
10.4.0	482	NA	66
10.2.3	482	NA	66
10.1.3	482	NA	66
10.0.4	482	NA	66

Table 1: Summary of the Geant4 Datasets for the Major Release 10.

In relation to the ML techniques, we have considered the ones used in the SE field according to existing literature [23]. Focusing on defect prediction problem several ML techniques have been investigated: AdaBoost (AB) [24], Boosted Logistic Regression (BLR) [25, 26], J48 [27], Cost-Sensitive C5.0 (C5.0 Cost) [28] and Logistic Model Tree (LMT) [29] as classification techniques; Multilayer Perceptron (MLP) [30], Support Vector Machines with Radial Basis Function Kernel (SVM Radial) [31], Partial Least Squares (PLS) [32], Boosted Tree (BT) [33] and Random Forest (RF) [34] as classification and regression techniques.

3.2 Output sets

The process outcomes are the average of performance indicators and the prediction on test datasets.

Each indicator can be defined in terms of: True Positive (TP) i.e. all the instances predicted as buggy that are actually buggy; True Negative (TN) i.e. all the instances predicted as clean and that are actually clean; False Positive (FP) i.e. all the instances predicted as buggy and that are actually clean and finally, False Negative (FN) i.e. all the instances predicted as clean and that are actually buggy.

As performance indicators we have considered Accuracy and Kappa statistic:

- Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$: it is the percentage of instances correctly classified as either buggy or non-buggy (i.e. clean).
- Kappa = $\frac{Accuracy-randomAccuracy}{1-randomAccuracy}$ with randomAccuracy = $\frac{(TN+FP)*(TN+FN)+(FN+TP)*(FP+TP)}{Accuracy*Accuracy}$: Kappa statistic [35] compares the observed accuracy with the expected accuracy and its value $\in [0,1]$. If Kappa statistic $\in [0.81, 0.99]$, then the value indicates an almost perfect agreement.

Concerning the prediction on test datasets, it can be used to identify the piece of code that may contain problems. Furthermore, this information can be checked against existing documentation.

3.3 Process

Our research procedure (summarized in Figure 5) aims at creating a software defect prediction model based on supervised learning techniques starting from an unlabelled dataset.

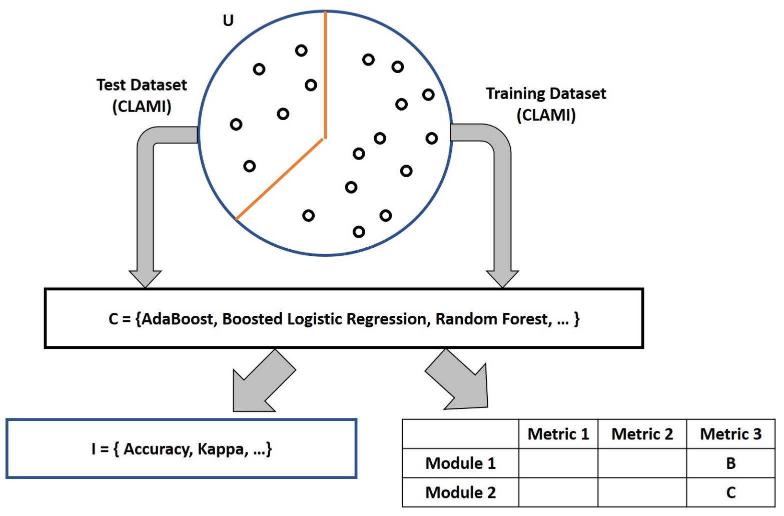

U
Test Dataset (CLAMI)
Training Dataset (CLAMI)
C = {AdaBoost, Boosted Logistic Regression, Random Forest, ... }
I = { Accuracy, Kappa, ...}
Metric 1
Metric 2
Metric 3
Module 1
B
Module 2
C

Figure 5: Research Procedure Summary.

We have followed the CLAMI approach [7] that is composed of various steps: splitting the unlabelled dataset in training and test, applying unsupervised clustering and labelling to the training dataset, removing all noisy metrics and instances and, finally, constructing the models by using supervised learning techniques to predict unlabelled instances.

In the clustering phase, we have first identified metric values that are greater than a specific cutoff threshold (e.g. the median value) and then determined the number of metrics K for each instance whose values are greater than the given metric threshold. In the labelling phase, we have categorized the clusters according to their K values and then divided them into two groups [36]: a bottom half for the clean instances, and a top half for the buggy instances. The instances in the top half cluster are labelled as buggy since the instances with larger values on all metrics are more likely to be defective [25, 37, 38].

Figure 6 shows an example of a training dataset with 7 modules and 6 metrics. An example of a cutoff threshold is given by the Median_j value, where j is the metric index. The yellow $\text{cell}_{i,j}$ represents the j-th metric value of i-th module greater than Median_j, where i is the module index. For reasons of simplicity, the cutoff threshold in the example above is the median of metric values because we made the assumption that instances are clustered 50% in buggy and 50% in clean, ignoring the actual data distribution. However, in our study, we have considered 9 different cutoff thresholds to decide the higher metric values, each threshold is related to the *p-th* percentile with $p \in 10, 20, ..., 90$.

K represents, for each instance, the number of metrics whose values are greater than the median for each metric. Figure 7 shows in different colours the four clusters defined according to the K values. The created clusters are then split into two groups:

1. Clean (C) for K $\in$ {0,1,2} (a bottom half)

2. Buggy (B) for K=3 (a top half)

Modules	$Metric_1$	$Metric_2$	$Metric_3$	$Metric_4$	$Metric_5$	$Metric_6$
A	10	11	4	6	8	?
D	23	10	15	14	10	?
E	15	17	4	8	5	?
F	9	10	9	6	3	?
G	11	13	15	5	8	?
H	14	10	17	9	0	?
I	7	9	21	13	9	?
Median	11	10	15	8	8	

Figure 6: Example of Unlabelled Training Dataset with the Metrics' Median.

Modules	$Metric_1$	$Metric_2$	$Metric_3$	$Metric_4$	$Metric_5$	$Metric_6$	*K*
A	10	11	4	6	8	?	$K = 1$
D	23	10	15	14	10	?	$K = 3$
E	15	17	4	8	5	?	$K = 2$
F	9	10	9	6	3	?	$K = 0$
G	11	13	15	5	8	?	$K = 1$
H	14	10	17	9	0	?	$K = 3$
I	7	9	21	13	9	?	$K = 3$

Figure 7: The *K* Values for the Various Modules.

In the cleaning/selecting phase, all the metrics that violate the defect proneness tendency are removed [36]. A violation occurs e.g. when an instance has been labelled as buggy but one of its metric value is not greater than the metric threshold, or when it has been labelled as clean but one of its metric value is greater than the metric threshold.

In Figure 8, the gray $\text{cell}_{i,j}$ represents the metrics whose value violates the defect-proneness tendency:

- D is Buggy, but $\text{Metric}_2 = 10$ is not greater than Median_2
- E is Clean, but $\text{Metric}_1 = 15$ is greater than Median_1

We have selected metrics by relying on the metric violation scores (MVS). MVS_j is the ratio between the number of violations in the j-th metric and the number of metric values in the j-th metric. Figure 9 shows the MVS values of the various metrics.

Metrics with the minimum MVS are selected for the training dataset. Figure 10 shows the resulting training dataset after metrics and instance selections. The test dataset includes the same selected metrics.

Modules	$Metric_1$	$Metric_2$	$Metric_3$	$Metric_4$	$Metric_5$	$Metric_6$
A	10	11	4	6	8	*C*
D	23	10	15	14	10	*B*
E	15	17	4	8	5	*C*
F	9	10	9	6	3	*C*
G	11	13	15	5	8	*C*
H	14	10	17	9	0	*B*
I	7	9	21	13	9	*B*
Median	11	10	15	8	8	

Figure 8: The Metrics Violation - Metric_1, Metric_2 and Metric_3 violate defect-proneness tendency.

Modules	$Metric_1$	$Metric_2$	$Metric_3$	$Metric_4$	$Metric_5$	$Metric_6$
A	10	11	4	6	8	*C*
D	23	10	15	14	10	*B*
E	15	17	4	8	5	*C*
F	9	10	9	6	3	*C*
G	11	13	15	5	8	*C*
H	14	10	17	9	0	*B*
I	7	9	21	13	9	*B*
MVS	$\frac{1}{7}$	$\frac{5}{7}$	$\frac{1}{7}$	$\frac{0}{7}$	$\frac{0}{7}$	

Figure 9: The MVS values.

Modules	$Metric_4$	$Metric_5$	$Metric_6$
A	6	8	*C*
D	14	10	*B*
E	8	5	*C*
F	6	3	*C*
G	5	8	*C*
H	9	0	*B*
I	13	9	*B*

Figure 10: The Final Training Dataset - Metric_1, Metric_2 and Metric_3 have been removed.

According to CLAMI, the instances with any violated metric values must be removed. This operation may lead to a dataset without either buggy or clean instances. In this situation, another MVS value can be chosen to reiterate the metrics and instances selection and to generate another training dataset with both buggy and clean instances.

4. Results

The experimental testbed is composed of a physical machine and a virtual machine on the cloud infrastructure at INFN CNAF [39]. Table 2 shows details about the resources used in this study. At the time of this study it was not possible to have similar resources in terms of operating system, CPU numbers and GPU.

	Physical Machine	**Virtual Machine**
CPU	2xIntel(R)E5-2640v2 2.00 GHz	2 x 12 AMD Opteron(TM) Processor 6238
Number of Cores	32 (HT)	16 V CPU
GPU	2 x NVIDIA TeslaK40m	
Memory	128 GB RAM	32 GB RAM
Operating System	CentOS Linux release 7.4.1708	Ubuntu Linux release 18.04
Python	2.7.5	3.6.7
R		3.5.2
Jupyter-notebook	5.7.8	5.7.4

Table 2: The Resources Characteristics.

The preprocessing activity has been applied to 34 datasets - each one related to the Geant4 software release - composed of the same classes and the same software metrics over the various releases.

Each dataset has been split in training (67%) dataset and test (33%) dataset. The training dataset has been preprocessed by considering 11 different cutoff threshold values for 500 times. On the virtual machine all the datasets have been produced in almost 8 days: in each permutation 374 training datasets and 374 test datasets are generated in almost 23.856 minutes. Figure 11 shows the preprocessing time: eTime − sTime is the time requested to build training and test sets per permutation.

Figure 12 shows the normalized value of the ratio between the number of buggy instances and the number of clean instances (RBC) over the 34 Geant4 releases for 9 different cutoff values calculated on the basis of 10-th, 20-th, ..., 90-th percentiles: the greater RBC, the lower the percentile. A low RBC value identifies classes with higher clean than buggy. We have omitted percentile at 0 and 100 being cause of bias interpretation.

Figure 13 shows the number of the selected metrics over the 34 Geant4 releases for 9 different cutoff values on the basis of 10-th, 20-th, ..., 90-th percentile: the smaller the number of selected metrics, the bigger the percentile. The metrics belong to various categories such as size, complexity, maintainability and object orientation. Over the various datasets, the number of the selected metrics belongs to the range [45%,77%] with an average selected metrics of 38 out of 66.

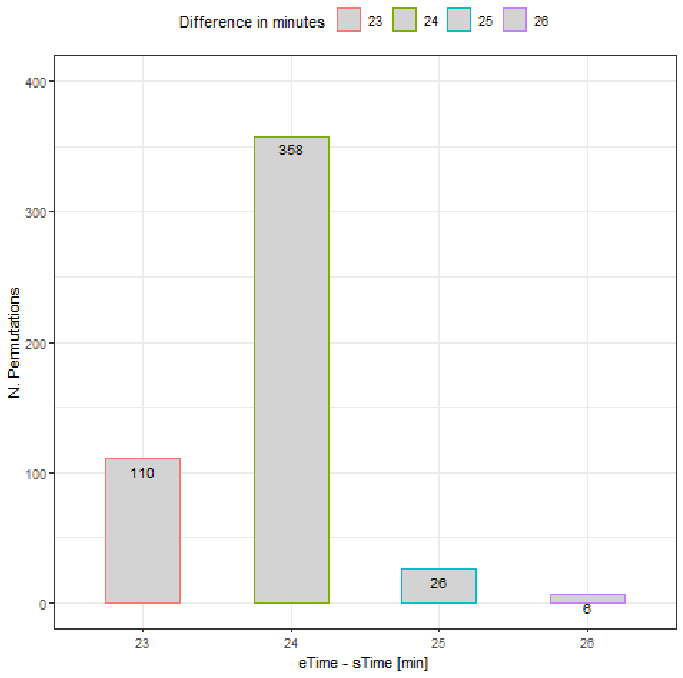


Figure 11: The Preprocessing Time on Virtual Machine.

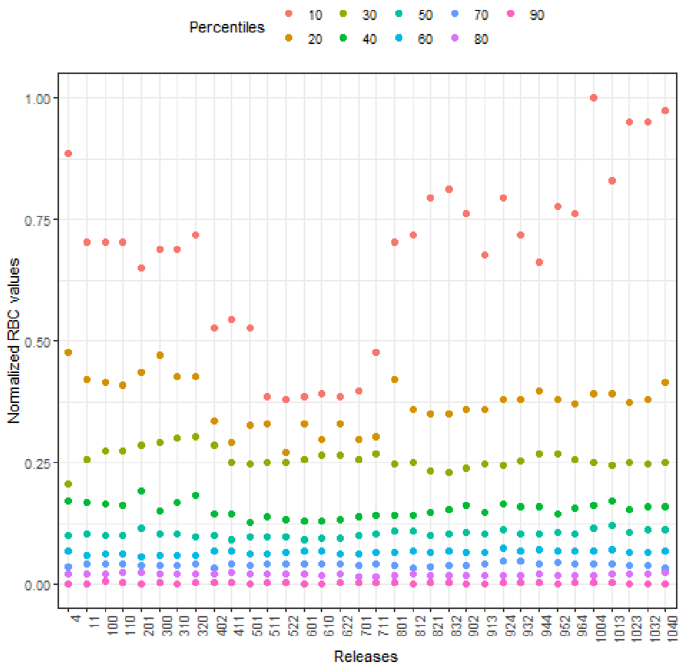


Figure 12: The Normalized RBC = $\frac{N.Buggies}{N.Clean}$ over the Geant4 Releases.

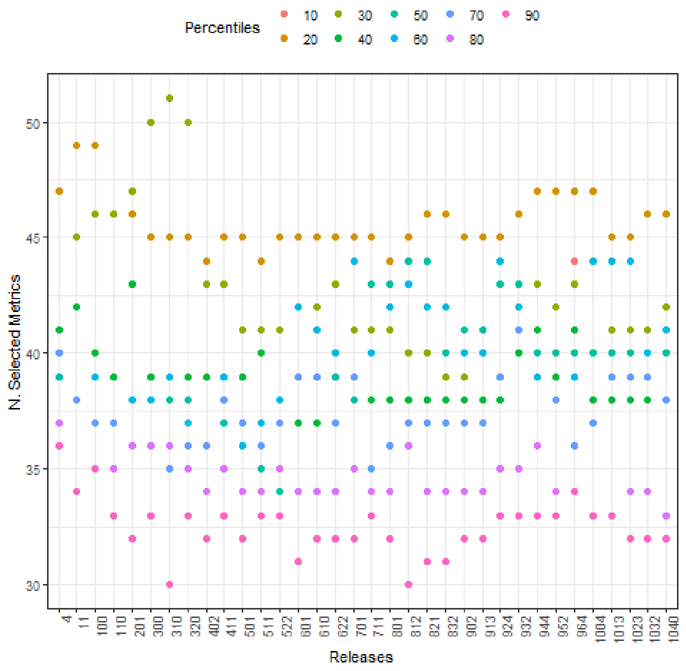


Figure 13: The Number of Selected Metrics over the Geant4 Releases.

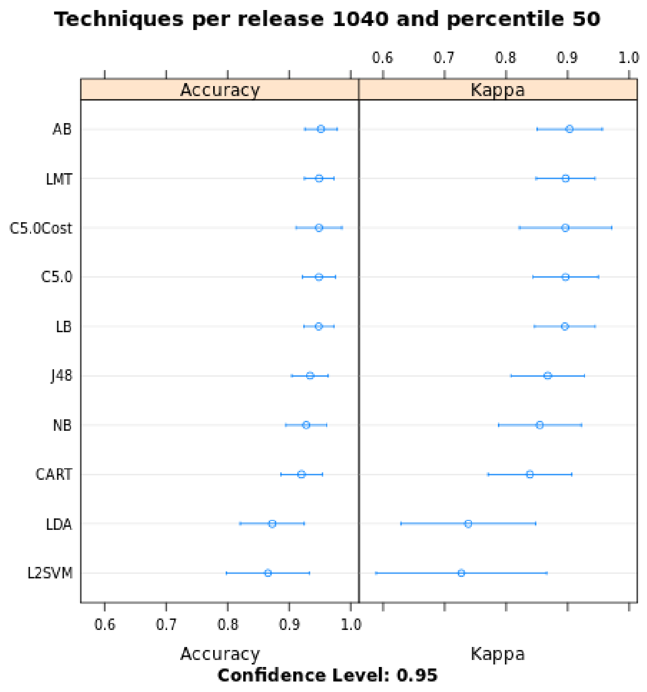


Figure 14: The Comparing Performance Indicators for Ada Boost.

As ML techniques we have used both classification and regression algorithms with 10-fold cross validation and we have employed Weka, R, Python-based framework on virtual machine and TensorFlow 1.13.1 on the GPU-equipped physical machine.

Figure 14 and Figure 15 show the accuracy and Kappa statistic indicators for the Geant4 release 10.4.0 and the 50-th percentile with a 95% confidence level (i.e. p-value < 0.05), respectively produced with the Ada Boost classification technique and the Random Forest classification and regression technique. The techniques have been run on the virtual machine obtaining a processing time of 36 seconds. Kappa statistic provides an almost perfect agreement on the two techniques.

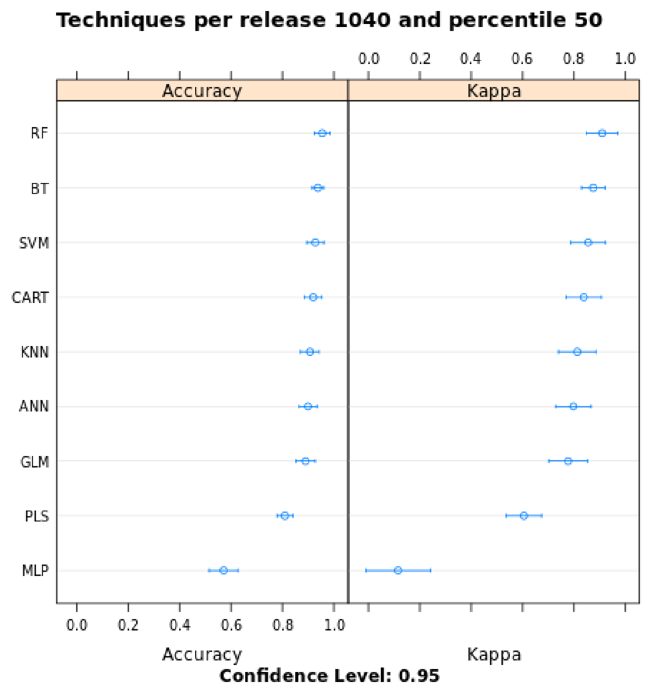


Figure 15: The Comparing Performance Indicators for Random Forest.

Table 3 shows the best ML techniques on the Geant4 10 major release over the 10-th, 20-th, ..., and 90-th percentiles.

Release	10	20	30	40	50	60	70	80	90	Type
10.0.4	**AB**	**AB**	C5.0 Cost	C5.0 Cost	**AB**	**BLR**	**AB**	LMT	**AB**	
10.1.3	LMT	**AB**	**BLR**	**BLR**	**BLR**	J48	C5.0 Cost	LMT	**BLR**	
10.2.3	C5.0 Cost	C5.0 Cost	**BLR**	**AB**	C5.0 Cost	**BLR**	**BLR**	**LB**	**LB**	Class.
10.3.2	**AB**	**AB**	**BLR**	C5.0 Cost	**BLR**	C5.0 Cost	LMT	**AB**	**BLR**	
10.4.0	**AB**	LMT	**BLR**	**BLR**	**AB**	**AB**	**BLR**	**BLR**	**BLR**	
10.0.4	**RF**	**RF**	**RF**	BT	BT	**RF**	**RF**	BT	**RF**	
10.1.3	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	BT	Class.
10.2.3	PLS	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	&
10.3.2	PLS	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	SVM Radial	Regr.
10.4.0	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	**RF**	MLP	SVM Radial	

Table 3: The Best ML Techniques.

Figure 16 shows the accuracy performance indicator for the Geant4 release 10.4.0 and the 50-th percentile produced with the Logistic Regression classification technique. It has been run on the

GPU-equipped physical machine by using the TensorFlow library 1.13.1, obtaining a processing time of 3 seconds with an improvement of a factor 10 compared to the result on the virtual machine. However, the resources used in this study are different, therefore it is not possible to perform an actual comparison of the processing time.

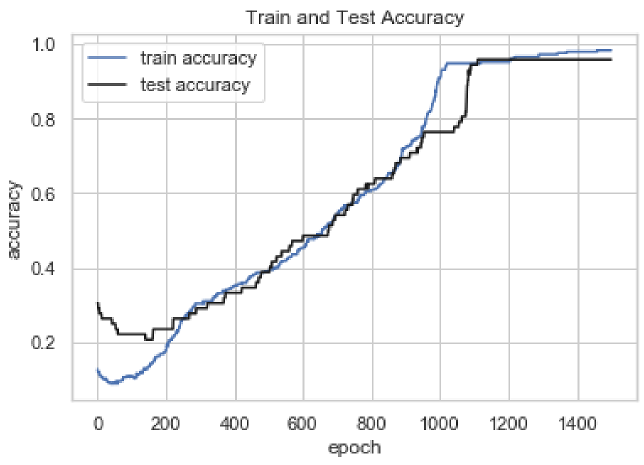


Figure 16: The Accuracy Indicator for Logistic Regression - the epoch represents the entire processing by the learning algorithm.

5. Conclusions and Discussions

Our results show that Adaboost, Boosted Logistic Regression and Random Forest ML techniques have achieved the best average accuracy over the datasets.

The effectiveness of our procedure in detecting defective instances can be assessed by checking existing software documentation and datasets, such as release notes and software metrics. In our study, we have shown that our approach can help detect pieces of software that may require particular attention by combining software metrics datasets, ML techniques and approaches, like CLAMI, that address the issue of unlabelled metrics datasets. Furthermore, ML techniques may be complementary to existing SE tools to address SE issues: SE tools are mainly responsible to calculate software metrics values or to perform statistical analysis; ML techniques can provide an objective interpretation of the analysed software just with the use of software metrics datasets and help to identify metrics that are suitable for a given software release.

What are the pros and cons of using the R or python-based framework? It depends on data, problem to be solved, hardware available, data preparation time. According to our experience, it is preferable to start exploring ML techniques by using the easiest framework, like Weka. Later, it is reasonable to use R and/or scikit-learn. TensorFlow may be helpful to exploit GPU-based systems.

Our approach is based on the CLAMI methodology that enables developers to build a prediction model on unlabelled datasets in an automated manner. Once obtained a labelled training dataset, one can employ all the other supervised and semi-supervised techniques to detect defective instances on test datasets. This approach entails a number of selection processes that may affect the final results. For example, the process of metrics' selection according to metrics' violations may decrease the size of the training dataset and, as a consequence, of the test dataset, penalizing both performance indicators and prediction.

The advantage of the CLAMI approach with respect to other techniques for unlabeled datasets is two-fold: firstly, it does not require software experts to determine defective modules or to select metrics and thresholds' values to construct both the training and test datasets; secondly, it can be easily automatized and used by developers without knowing all its details.

Software developers may rely on CLAMI-based software tools to determine which software modules they are working on need more attention and tests.

Our next steps involve the employment of other approaches, such as CLAMI+, and other clustering techniques, like K-means, to identify the best solution for defect prediction on unlabelled datasets in HEP software. Moreover, we are going to check our findings against existing documentation of Geant4 in order to obtain a semi-labelled dataset on which employ semi-supervised learning techniques.

References

[1] E. Ronchieri and M. Canaparo, *Metrics for Software Reliability: a Systematic Mapping Study*, *Journal of Integrated Design and Process Science* **22** (2018) 1.

[2] H. Tong, B. Liu and S. Wang, *Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning*, *Information and Software Technology* **96** (2018) .

[3] A. Joulin and T. Mikonov, *Inferring algorithmic patterns with stack-augmented recurrent nets*, in *Proceedings of 28th International Conference on Neural Information Processing Systems*, vol. 1, pp. 190–198, MIT Press Cambridge, MA, USA, (2015).

[4] C. Catal, U. Sevim and B. Diri, *Clustering and Metrics Thresholds Based Software Fault Prediction of Unlabeled Program Modules*, in *Proceedings of Sixth International Conference on Information Technology: New Generations*, IEEE, (2009), DOI.

[5] S. Zhong, T. M. Khoshgoftaar and N. Seliya, *Unsupervised learning for expert-based software quality estimation*, in *Proceedings of the 8th IEEE Internation Symposium on High Assurance Systems Engineering*, IEEE, (2004), DOI.

[6] S. J. Pan and Q. Yang, *A survey on transfer learning*, *IEEE Transactions on Knowledge and Data Engineering* **22** (2010) 1345.

[7] J. Nam and S. Kim, *CLAMI: Defect Prediction on Unlabeled Datasets (T)*, in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, (2015), DOI.

[8] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce et al., *GEANT4 - a simulation toolkit*, *Nucl. Instrum. Methods Phys. Res., Sect. A* **506** (2003) 250.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean et al., *TensorFlow: A system for large-scale machine learning*, in *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, (Savannah, GA, USA), pp. 265–283, 2016.

[10] J. Li, P. He, J. Zhu and M. R. Lyu, *Software Defect Prediction via Convolutional Neural Network*, in *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, (2017), DOI.

[11] S. Bahrampour, N. Ramakrishnan, L. Schott and M. Shah, *Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning*, *CoRR* **abs/1511.06435** (2015) .

[12] N. Azeem and S. Usmani, *Analysis of Data Mining Based Software Defect Prediction Techniques*, *Global Journal of Computer Science and Technology* **11** (2011) .

[13] J. Wang, Y. Ma, L. Zhang, R. Gao and D. Wu, *Deep learning for smart manufacturing: Methods and applications*, *Journal of Manufactoring Systems* (2017) .

[14] N. Seliya and T. M. Khoshgoftaar, *Software Quality Analysis of Unlabeled Program Modules With Semisupervised Clustering*, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* **37** (2007) .

[15] A. A. Deshmukh and E. Laftchiev, *Semi-supervised transfer learning using marginal predictors*, in *Proceedings of IEEE Data Science Workshop (DSW)*, pp. 160–164, IEEE, (2018), DOI.

[16] E. Ronchieri, M. G. Pia, T. Basaglia and M. Canaparo, *Assessment of Geant4 Maintainability with respectto software engineering references*, *Journal of Physics: Conf. Series* **898** (2017) .

[17] M. G. Pia and E. Ronchieri, *Measurements and Trends of Geant4 Software Evolution*, in *Proceedings of IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, 2017.

[18] I. 4D. https://www.imagix.com/.

[19] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach - 3rd Edition*, Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. CRC Press, 2014.

[20] S. R. Chidamber and C. F. Kemerer, *Metrics suite for object oriented design*, *IEEE Transactions on Software Engineering* **20** (1994) 476.

[21] T. McCabe, *A complexity measure*, *IEEE Transactions on Software Engineering* **Se-2** (1976) 308.

[22] M. H. Halstead, *Elements of Software Science*. 1975.

[23] Y. A. Alshehri, K. Goseva-Popstojanova, D. G. Dzielski and T.Devine, *Applying Machine Learning to Predict Software Fault Proneness Using Change Metrics, Static Code Metrics, and a Combination of Them*, in *SoutheastCon*, 2018.

[24] Y. Gao and 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), DOI.

[25] W. Rhmann, B. Pandey, G. Ansari and D. K. Pandey, *Software fault prediction based on change metrics using hybrid algorithms: An empirical study*, *Journal of King Saud UNiversity - Computer and Information Sciences* (2019) .

[26] J. Otero and L. Sànchez, *Induction of descriptive fuzzy classifiers with the Logitboost algorithm*, *Soft Computing* **10** (2006) .

[27] M. C. M. Prasad, L. Florence and A. Arya, *A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques*, *Internation Journal of Database Theory and Applkcation* **8** (2015) .

[28] M. J. Siers and M. Z. Islam, *Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem*, *Information Systems* **51** (2015) .

[29] N. Landwehr, M. Hall and E. Frank, *Logistic Model Trees*, *Machine Learning* **59** (2005) .

[30] M. Gayathri and A. Sudha, *Software Defect Prediction System using Multilayer Perceptron Neural Network with Data Mining*, *International Journal of Recent Technology and Engineering (IJRTE)* (2014) .

[31] P. A. Selvaraj and D. P. Thangaraj, *Support Vector Machine for Software Defect Prediction*, *International Journal of Engineering & Technology Research* **1** (2013) .

[32] G. Luo, Y. Ma and K. Qin, *Asymmetric Learning Based on Kernel Partial Least Squares for Software Defect Prediction*, *IEICE Transactions on Information and Systems* (2012) .

[33] F. Wang, J. Huang and Y. Ma, *A Top-k Learning to Rank Approach to Cross-Project Software Defect Prediction*, in *25th Asia-Pacific Software Engineering Conference*, 2018, DOI.

[34] R. M. Magal and S. G. Jacob, *Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques*, *International Journal of Computer Applications* **117** (2015) .

[35] J. R. Landis and G. G. Koch, *The Measurement of Observer Agreement for Categorical Data*, *Biometrics* **33** (1977) .

[36] M. D'Ambros, M. Lanza and R. Robbes, *Evaluating defect prediction approaches: a benchmark and an extensive comparison*, *Empirical Software Engineering* **17** (2012) .

[37] T. Menzies, J. Greenwald and A. Frank, *Data Mining Static Code Attributes to Learn Defect Predictors*, *IEEE Trans. Softw. Eng.* **33** (2007) 2.

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

[39] https://www.cnaf.infn.it/en/.