Software Defect Prediction on Unlabelled Datasets: A Comparative Study

Elisabetta Ronchieri[1(✉)], Marco Canaparo[1], and Mauro Belgiovine[2]

[1] INFN-CNAF, Viale Berti Pichat 6/2, Bologna, Italy
elisabetta.ronchieri@cnaf.infn.it

[2] Elettrical and Computer Engineering Department, Northeastern University, Boston, USA

Abstract. Background: Defect prediction on unlabelled datasets is a challenging and widespread problem in software engineering. Machine learning is of great value in this context because it provides techniques - called unsupervised - that are applicable to unlabelled datasets. Objective: This study aims at comparing various approaches employed over the years on unlabelled datasets to predict the defective modules, i.e. the ones which need more attention in the testing phase. Our comparison is based on the measurement of performance metrics and on the real defective information derived from software archives. Our work leverages a new dataset that has been obtained by extracting and preprocessing its metrics from a C++ software. Method: Our empirical study has taken advantage of CLAMI with its improvement CLAMI+ that we have applied on high energy physics software datasets. Furthermore, we have used clustering techniques such as the K-means algorithm to find potentially critical modules. Results: Our experimental analysis have been carried out on 1 open source project with 34 software releases. We have applied 17 ML techniques to the labelled datasets obtained by following the CLAMI and CLAMI+ approaches. The two approaches have been evaluated by using different performance metrics, our results show that CLAMI+ performs better than CLAMI. The predictive average accuracy metric is around 95% for 4 ML techniques (4 out of 17) that show a Kappa statistic greater than 0.80. We applied K-means on the same dataset and obtained 2 clusters labelled according to the output of CLAMI and CLAMI+. Conclusion: Based on the results of the different statistical tests, we conclude that no significant performance differences have been found in the selected classification techniques.

Keywords: Unlabelled dataset · Defect prediction · Unsupervised methods · Machine learning

1 Introduction

A large number of software defect prediction models have been proposed over time to enable developers to determine the most critical software modules to

Supported by organization INFN.

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12250, pp. 333–353, 2020.
https://doi.org/10.1007/978-3-030-58802-1_25

which they should pay more attention and track carefully [1,2]. Consequently, a prompt recognition of these modules can allow developers to effectively allocate their limited reviewing and testing resources and may suggest architectural improvements. In the last decade, researchers have striven to employ machine learning (ML) techniques in many software engineering (SE) problems, such as defective prediction [3].

In many approaches described in existing literature prediction models learn from historical labelled data within a software project in a supervised way [4]: the model is trained on labelled data (e.g. where defectiveness information is used) and performs prediction on test data. Although supervised approaches are the vast majority, in practice, software projects usually lack classification labels to train a supervised defect prediction model, because it is considered a time and effort consuming activity [5]. Due to the difficulty in collecting label information before building prediction models, unsupervised methods have begun to draw the attention of researchers [6].

Finding the best way to classify unlabelled data is a challenging task. The current approaches use various experiment configurations in terms of software metrics (e.g., belonging to size and complexity category) and performance metrics (e.g., f-measure and accuracy). Li et al. [7] cathegorize them into clustering and non-clustering techniques, some of them are summarized in Table 1.

Table 1. Unsupervised software defect prediction approaches

Category	Approach	Description
Clustering	k-Partition	It is based on the distance data point [8–11]
	Density	It exploits core objects with dense neighborhoods [12]
	Fuzzy	It determines in which a data point can belong to two or more clusters [13,14]
	Spectral clustering	It is based on eigenvalues of the similarity matrix [15,16]
Non-clustering	Expert	In this case expert determine the labels directly [9]
	Threshold	It uses some thresholds to classify instances directly [8,14]

Once clustered the instances of the datasets, the labelling phase is a necessary step in software defect prediction techniques: the expert-based approach labels a cluster by exploiting the decision of an expert [9]; the threshold-based approach decides the cluster label by making use of particular metric threshold values [8,14]; the top half approach labels defective half of the top clusters [17]; the distribution approach is based on the number of nodes of a cluster, the smaller are considered defect prone, the larger non defect-prone [18]; the majority vote uses the most three similar clustering centers [14]; the cross project exploits metrics from user projects [15].

The labelling techniques are characterized by having some limitations, such as relying on experts' information and using suitable metrics' thresholds, and some assumptions like dealing with datasets which include the same features in cross project defect prediction models. Furthermore, it has been shown that their results are uneasy to compare [19].

Previous studies on unlabelled datasets have employed datasets, like the NASA dataset and the clean version of the NASA dataset [20] and the PROMISE dataset [21]. All these datasets are labelled and include different metrics. Therefore, to allow researches to be able to steer their choices when dealing with unlabelled datasets, we have decided to provide a comparative analysis of approaches that are able to overcome the previous limitations by using a new unlabelled dataset built for C++ software.

Our comparative analysis exploits CLAMI [22] that is made up of 4 phases: Clustering, LAbeling, Metrics selection and Instance selection. Its key idea is to label unlabelled datasets by exploiting metrics' magnitude (e.g. median, mean, quantiles), that is metrics' cutoff threshold. In addition to CLAMI, we have explored CLAMI+ [23] an improvement of CLAMI characterized by a different procedure in the metrics' selection phase (see Sect. 3). All the approaches have been applied on a new dataset that has been obtained by extracting and preprocessing its metrics from its software archives. Differently from the majority of previous analysed datasets that are related to C or Java code, ours is the result of a metrics' extraction of a software written in C++ [24]. Finally, we have used the K-means algorithm as another technique to find potentially critical modules.

In this study, we consider a high energy physics (HEP) software project, called Geant4 [25], characterized by a long development history. It represents a valid candidate to systematic comparison approaches applied on unlabelled datasets. For this software we have measured its software characteristics [26] (by using the Imagix 4D framework [27]) to determine its quality evolution over releases and to build a complete set of data on top of which assessing the applicability of software metrics. Considering the amount of data available - i.e., 34 different datasets each for different software modules (such as class, namespace, function, file and variable) - this analysis may help to use these approaches and to concentrate effort only to the most critical software modules of a software project. Our comparison relies on performance metrics, such as accuracy, f-measure and Kappa statistics, and difference tests. Furthermore, we have manually extracted information about defectives of software modules from their software archives to be able to conduct a comparison with real data.

In the remaining parts of this work, we first introduce the terminology Sect. 2. Section 3 introduces the methodology of this comparison, including the research questions, the data used, the frameworks adopted for the analysis, the unsupervised approaches and the ML techniques used. Section 4 further presents the experimental results. Section 5 discusses the results. At last, the work is concluded with future work in Sect. 6.

2 Terminology

Within this section, we introduce a set of definitions we use in the paper, the workflow of unsupervised approaches, the software metrics we have collected and considered in the software datasets, the performance metrics we have used for the discussion and a set of statistical tests we have used to detect differences amongst the various ML techniques.

Definitions - A *software project* contains different releases of the software. Each *software release* is coded by a unique identifier that is composed of one or more numbers. Semantic versioning [28] uses three digits that respectively identify major, minor and patch. Developers increment the major number when they introduce breaking changes; a minor number when they add functionality in a backward compatible way; a patch number when they make all other non-breaking changes. A *module* identifies an element of the software, such as a class, a file, and a function. An *unlabelled* dataset is a set of software metrics for various modules of a given software release. The modules are not labelled with respect to release defects and they are marked with the key term?. A *labelled* dataset is a set of software metrics for various modules of a given software release. The modules are labelled with defective and non defective key terms, such as buggy (B) and clean (C) respectively, according to the unsupervised approaches.

General Workflow - Figure 1 shows the general workflow of software defect prediction on unlabelled datasets.

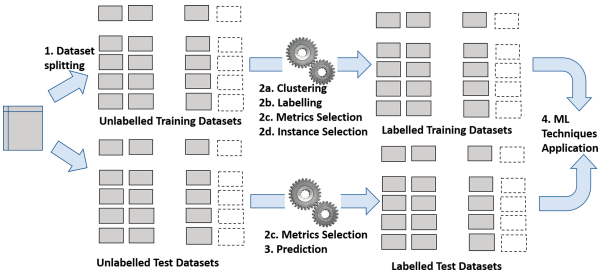


Fig. 1. Workflow.

The *unlabelled dataset* contains for the various modules of a given software releases a set of software metrics. Each software release-specific dataset is split in the training and test datasets respectively to train the model and predict defects: the training dataset starts unlabelled and ends labelled after a set of preprocessing operations related to the adopted unsupervised approaches; the test dataset remains unlabelled. The *training dataset* contains modules labelled as defective (i.e. buggy) or non defective (i.e. clean) according to the unsupervised approaches adopted. Furthermore, each module contains information of some software metrics. Modules with label and software metrics are used to train ML classifiers and to predict new modules as buggy or clean.

Software Metrics - Software metrics provide information on various aspects of software through quantitative measurements [29]. Aspects of software metrics may include: its structure, coupling, maintainability. Metrics can refer to a file, a class, a function or any key software elements that characterize a software.

There are different families of software metrics. In this work we consider size, complexity and object orientation categories. The *size* metrics help to quantify the software size and complexity and suggest the quality of the implementation. Their aim is to identify files, classes or methods that can be hard to understand, reuse, test and modify. For instance, class size metrics can reflect the effort required to build, understand and maintain a class. These metrics go from the comment ratio and size lines of source code up to the size number of statements. The *complexity* metrics help to detect complex code that is difficult to test and maintain. Metrics used to measure this software characteristic are: size metrics (already described), McCabe metrics [30] and Halstead metrics [31]. There are various *object orientation* metrics suites. The most studied one is the Chidamber and Kemerer (CK) metrics suite [32] that aims at measuring design complexity in relation to their impact on external quality attributes, such as maintainability and reusability. The CK metrics are: number of children and depth of inheritance tree related to inheritance and abstraction; weighted methods per class and lack of cohesion in methods pertaining to the internal class structure; coupling between objects and response for a class that are related to relationships among classes.

Performance Metrics - We detail the performance metrics [33] used to assess the considered ML techniques. The metrics are based on the confusion matrix that describes the complete performance of the model by considering the following concepts: a defective instance is defective (i.e. *tp* as true positive); a defective instance is non defective (i.e. *fn* as false negative); a non defective instance is defective (i.e. *tn* as true negative); a non defective instance is non defective (i.e. *fp* as false positive). In this work, we consider the recall metric, the precision metric, the accuracy metric, the f-measure metric, the AUC metric and the Kappa statistic. The *recall* metric $\frac{tp}{tp+fn}$, also called *Probability of Detection* (PD), measures how many of the existing defects are found. The *precision* metric $\frac{tp}{tp+fp}$ measures how many of the found results are actually defects. The *accuracy* metric $\frac{tp+tn}{tp+fn+tn+fp}$ measures the percentage of the correct predictions. The *f-measure* metric $2 \cdot \frac{recall \cdot precision}{recall+precision}$ is the harmonic mean between *recall* and *precision*. The *Probability of False Alarm* (PF) $\frac{fp}{fp+tn}$ is defined as the ratio of false positive to all non-defective modules. Ideally, when used as a performance metric, it is desired to be equal to zero. The *Kappa statistic* $\frac{accuracy-randaccuracy}{1-randaccuracy}$ [34], with the *randaccuracy* equals to $\frac{(tn+fp)\cdot(tn+fn)+(fn+tp)\cdot(fp+tp)}{accuracy \cdot accuracy}$, 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. The *Receiver Operating Characteristics (ROC) curve*, in the field of software prediction, is defined as a tradeoff between the prediction ability to correctly detect defective modules (PD or recall) and non-defective modules (PF) [35]. From a graphical point of view, PD is represented on the Y-axis and PF on the X-axis, while predictor's threshold is varying. The performance of a predictor is better as the curve is closer to coordinates PD = 1 and PF = 0. The *Area Under the Curve (AUC)* metric is the area under the ROC curve and it is commonly

used to compare performance of predictors. It measures a model's discriminatory power which represents the area under receiver operating characteristic (ROC) curve [23,36]. Compared to other metrics, AUC has two advantages: it is independent on the definition of a metric threshold and it is robust towards imbalanced datasets. With other performance metrics, a classifier must rely on the determination of a threshold whose value is usually set as the average of all metric' values, this choice may affect the final results of the classifier. Precision and recall are examples of metrics which are highly influenced by imbalanced datasets, making it difficult to conduct a fairly models' comparison.

Performance Differences Tests - We statistically compare the performances achieved by the experimented prediction models. We are going to detect groups of differences between the ML techniques we have considered in this study. These groups are called *blocks* in statistics and are usually linked to the problems met in the experimental study [37]. For example, in a comparative analysis of multiple datasets, each block is related to the results computed on a specific dataset. When dealing with multiple comparison tests, each block is composed of various results each linked to the couple algorithm, datasets. The *Friedman test* [38] is the most common procedure for testing the differences among two o more related samples which represent the performance metrics of the techniques measured across the same datasets. The objective of this test is to determine if we may conclude from a sample of results that there is difference among treatment effects [37]. It is composed of two steps: the first step converts the original results in ranks by calculating a test statistic; the second step ranks the techniques for each problem separately where the best performing techniques is given rank 1. Iman and Davenport [39] showed that Friedman test have a conservative behaviour and proposed a different statistic. Post hoc procedures are employed to obtain what classifier performances better than the one proposed. They usually follows the application of the Friedman test since they take into account the rankings generated by the Friedman procedure. According to [40], Friedman test with Iman and Davemport extension is the most popular omnibus test and is a good choice for comparing more than five techniques. About post hoc tests, comparing all the techniques with one proposed requires the simple but least powerful Bonferroni test, on the contrary, when it is necessary to conduct pairwise comparison, Hommel [41] and Rom are the two most powerful procedures. The *Nemenyi post-hoc test* [42] is used when every classifier is compared to one another. Two classifiers achieve significant different performance if the corresponding average ranks differ by at least the critical difference (CD) $q_\alpha \sqrt{\frac{k(k+1)}{6N}}$ where k is in the number of ML techniques, N is the number of the datasets and q_α is derived from the studentized range statistic divided by $\sqrt{2}$. The Scott-Knott Effect Size Difference (ESD) test [43] is a mean comparison approach based on a hierarchical clustering analysis to partition the set of treatment means into statistically distinct groups. It corrects the non-normal distribution of an input dataset, and merges any two statistically distinct groups that have a negligible effect size into one group.

3 Comparison Methodology

We describe the methodology that we followed to perform our comparison.

Research Questions - We formulate the research questions (RQ) that we answer through this comparison. *RQ1*: which unsupervised approach performs best in terms of our performance metrics? *RQ2*: which supervised technique performs best in terms of our performance metrics in the two unsupervised approaches? *RQ3*: what is the impact of using different cutoff thresholds (i.e. quantiles) on the comparison results? The reason for *RQ1* is that we intend to test the known unsupervised approaches employed both in previous literature and in our work to find which one is performing better than the others. To do so we will compute their results in term of performance metrics. The analysis of *RQ3* provides insights into how the dataset distribution affects the overall performance of our defect prediction experiments. On the one hand, larger datasets entail more generalization and better performance, however, less data for training may have less variance which can cause an improvement in the performance.

Data - Our comparison assesses the CLAMI and CLAMI+ approaches on a multi-release dataset. It is related to the scientific widespread Geant4 software due to its peculiarity: the first 0.0 release was delivered the 15th of December 1998 and it is still under development by an international collaboration that have produced 10 major releases and at least 34 minor releases with various patches. The development time frames sampled in the study ranges 21 years, covering almost a quarter of century of development history. Figure 2 shows the Geant4 software releases considered for this study over years: for each release number, the $+$ shows the year when the release was delivered, the x shows the year when the related patch was delivered. At the moment, the dataset consists of 34 software releases delivered in almost 20 years of development. There are 66 software metrics for 482 C++ classes, measured for the various releases by using the Imagix 4D tool [27]. The resulting dataset is composed of 34 x 482 classes, 482 are repeated 33 releases. The defect labels are not included in the dataset: we have extracted them by using the release notes that accompany the corresponding software releases.

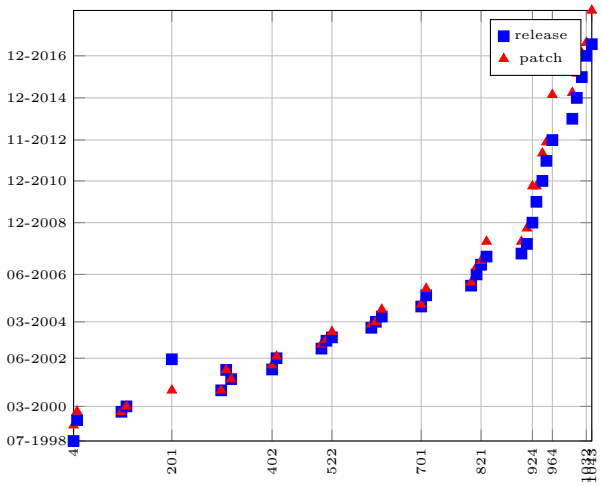


Fig. 2. The Geant4 software releases

Framework - In order to conduct this study, we exploit various available frameworks running in Java, Python and R. We have assessed these frameworks leveraging how much effort they required us to be used: Weka [44] is the easiest to use, while Theano [45] requires more expertise.

Unsupervised Approaches - We apply three unsupervised approaches to the examined dataset. The CLAMI approach conducts the prediction on unlabelled dataset by learning from itself. It creates a labelled training sample by using the magnitude of metric values (i.e. the metrics that are greater than of a specific cutoff threshold) and build the model by supervised learning to predict unlabelled modules. This approach consists of clustering, labelling, metrics selection and instance selection phases: clustering groups the modules; labelling estimates the label of groups; metric selection and instance selection select more informative features (i.e., software metrics) and produce training set. In the labeling, CLAMI measures the count of violation (i.e., a metric value is greater than a certain threshold) of a module. The CLAMI+ approach extends the CLAMI one. The main difference concerns the clustering and labelling phases. CLAMI+ uses the violation degree (i.e., transforming the difference between the metric value and the threshold to a probabilistic value) to replace the boolean representation in CLAMI. In this case, how much the instance violated on a metric is considered. The resulting training set is more informative than the one of CLAMI. K-means is a clustering algorithm [46] that depends upon iterative location that partitions dataset into K number of clusters by standard Euclidean distance. There are various variants of the algorithm, we have used the ones implemented in scikit-learn that is based on either Llody's or Elkan's algorithm [47,48]. Since it is simple and flexible these algorithms are very popular in statistical analysis [49]. The defect prediction is performed within-project where the training and test sets are from the same project.

ML Techniques - Aiming at comparing several classification algorithms, we have considered a total of seventeen classifiers, each of which can be categorized according to its statistical approach. We have identified the following categories: statistical classifiers that construct a Bayes optimal classifier; nearest-neighbor methods that classify a module by considering the k most similar samples, where the similarity definition differs among algorithms; neural networks methods that depicts a network structure to define a concatenation of weighting, aggregation and thresholding functions in order to determine a software module probability of being fp; support vector machines classifiers that aim at optimizing a linear decision function to discriminate between fp and non fp modules; tree-based methods that recursively partition the training data; ensembles methods that embody several base classifiers, built independently and put together to obtain a final class prediction. Table 2 provides, for each category, the considered classifiers. A detailed description is available in general textbooks [50].

Evaluation Strategy and Remarks - The focus of our comparison is not an advancement or improvement of existing approaches. We apply them to an unlabelled dataset related to Geant4 software, a stone in the Monte Carlo simulation

Table 2. Classification models

Category	Model	Acronym
Statistical classifiers	Naïve Bayes	NB
	Logistic Regression	LR
	Bayesian Network	BN
Nearest neighbor methods	k-Nearest Neighbor	k NN
Neural networks	Multi-Linear Programming	MLP
	Radial Basis Function Network	RBF net
Support vector machine classifiers	Support Vector Machine	SVM
	Linear Programming	LP
	Least Squares SVM	LS SVM
	Lagrangian SVM	L SVM
Decision tree approaches	C 4.5 Decision Tree	C 4.5
	J48	J48
	Classification and Regression Tree	CART
Ensemble methods	Random Forest	RF
	Logistic Model Tree	LMT
	Adaboost	Adaboost
	Bagging	Bagging

in order to provide an alternative way to control and monitor its development process. The resulting datasets have been checked against the existing documentation in the release note of each new software release. We have been able to trace modules that were changed for bug fixes, minor fixes, warning, and major fixes, finding a correspondence between the documentation and the labelling activity performed by the considered unsupervised approaches.

4 Results

In this section we provide an analysis of the obtained results. To conduct a reasonable comparison among the various releases we have considered the modules that were present in all the releases. This filtering operation has led to a considerable reduction of our training set. More in detail, while the common modules of all the releases were 482, the releases 4, 11 and 1004 for example contain respectively 891, 915 and 2544 modules. In the following, we present the results relative to the various steps of the considered approaches.

Selected Metrics - Table 3 shows the level, the category (i.e. Complexity, Size, Object Orientation), the complete name and the description of all metrics selected either in the CLAMI or the CLAMI+ approach. The file level metrics have been computed both on header (i.e. the .h extension) and source files (i.e. the .cpp extension). Halstead's metrics have been calculated both at file

Table 3. Metrics' description

Level	Category	Acronym	Name	Description
Class (CL)	Complexity	CLhpd	Halstead Program Difficulty (hpd)	Measure of how compactly the class implements its algorithms (Halstead D). This is the inverse of the Halstead Program Level (of abstraction)
Class (CL)	Complexity	CLhpv	Halstead Program Volume (hpv)	Measure of the information content of the class (Halstead V)
Class (CL)	Complexity	CLhme	Halstead Mental Effort (hme)	Measure of the number of elemental mental discriminations necessary to create; or understand; the class (Halstead E)
Class (CL)	Complexity	CLhic	Halstead Intelligent Content	Language-independent measure of the amount of content (complexity) of the class (Halstead I)
Class	Object Orientation	chlvc	Class Hierarchy and Local Vars Coupling	Depth of the hierarchy of derived classes of the class
Class	Object Orientation	ckrfc	CK Response For a Class	Number of methods called by the class's methods; a measure of the class's response (Chidamber and Kemerer RFC)
Class	Object Orientation	ckwmc	CK Weighted Methods per Class	Total cyclomatic complexity for the class's methods (Chidamber and Kemerer WMC)
Class	Object Orientation	mci	Methods Called - Internal	Number of internal methods called by class methods
Class	Object Orientation	mce	Methods Called - External	Number of external methods called by class methods
Class	Object Orientation	emc	External Methods and Coupling	Total number of external classes used and external methods called by class member methods. This is the sum of coupling between classes and external methods called
Class	Size	ntm	Number of Total Members	Total number of members of the class
Class	Object Orientation	cklcm	CK Lack of Cohesion of Methods	Lack of Cohesion of Methods is a measure of the cohesion of the member functions of the class (Chidamber and Kemerer LCOM; 1994 revised definition)
Class	Size	nmm	Number of Member Methods	Number of methods (member functions) of the class
Class	Size	nmpm	Number of Member Private Methods	Number of local methods (private member functions) of the class
Class	Size	nma	Number of Member Attributes	Number of attributes (member variables) of the class
Class	Size	ntm	Number of Total Members	Total number of members of the class
File	Complexity	Hmctcc	McCabe Total Cyclomatic Complexity (mctcc)	Total cyclomatic complexity for all functions defined in the header (H) file or implementation (I) file
File	Complexity	Imcacc	McCabe Average Cyclomatic Complexity	Average cyclomatic complexity for all functions defined in the header (H) file or implementation (I) file
File	Complexity	Hmi, Imi	Maintainability Index (mi)	Measure of the maintainability of the header (H) file or implementation (I) file (Welker MI)
File	Complexity	Ihpd	Halstead Program Difficulty	Measure of how compactly the header (H) file or implementation (I) file implements its algorithms (Halstead D). This is the inverse of the Halstead Program Level (of abstraction)

(*continued*)

Table 3. (*continued*)

Level	Category	Acronym	Name	Description
File	Complexity	Ihpv	Halstead Program Volume	Measure of the information content of the header (H) file or implementation (I) file (Halstead V)
File	Complexity	Ihic	Halstead Intelligent Content	Language-independent measure of the amount of content (complexity) of the header (H) file or implementation (I) file (Halstead I)
File	Complexity	Ihme	Halstead Mental Effort	Measure of the number of elemental mental discriminations necessary to create; or understand; the header (H) file or implementation (I) file (Halstead E).
File	Complexity	Ihpd	Halstead Program Difficulty	Measure of how compactly the header (H) file or implementation (I) file implements its algorithms (Halstead D). This is the inverse of the Halstead Program Level (of abstraction)
File	Size	Hsb, Isb	Size Bytes (sb)	Size of the header (H) or implementation (I) file in bytes
File	Size	Hidif	Inclusion - Directly Included Files	Number of header files in the project database which are directly included by the header (H) file or implementation (I) file
File	Size	Hitil, Iitil	Inclusion - Transitively Included Lines	Number of lines in header files in the project database which are transitively included by the header (H) file or implementation (I) file.
File	Size	Iitif	Inclusion - Transitively Included Files	Number of header files in the project database which are transitively included by the header (H) file or implementation (I) file
File	Size	Islf	Size - Lines in File	Number of lines in the header (H) file or implementation (I) file.
File	Size	Hslc, Islc	Size - Lines of Comments	Number of lines of comments in the header (H) file or implementation (I) file.
File	Size	Isns	Size - Number of Statements	Number of statements in the header (H) file or implementation (I) file

Table 4. Common metrics between the CLAMI and CLAMI+ approaches over the various quantiles

Acronym	1	2	3	4	5	6	7	8	9	Acronym	1	2	3	4	5	6	7	8	9	Acronym	1	2	3	4	5	6	7	8	9
chlvc	✓		✓		✓		✓	✓		Iitif	✓	✓								Ihic	✓	✓							
ckrfc				✓	✓	✓	✓	✓	✓	Iitil	✓	✓								Ihme	✓	✓							
ckwmc				✓	✓	✓	✓	✓	✓	Ihpd	✓	✓								Isb	✓	✓							
CLhpd	✓		✓		✓	✓	✓	✓	✓	Ihpd	✓									Imi	✓	✓							
CLhpv	✓		✓	✓	✓	✓	✓	✓	✓	Islc	✓									Imctcc									✓
CLhme	✓		✓		✓	✓	✓	✓	✓	ntm					✓			✓	✓	mce									✓
Hmctcc	✓	✓		✓						nma								✓	✓	Isns									✓
Hsb	✓	✓								chlvc									✓	Hslc									✓
Hidif	✓	✓								cklcm									✓	emc									✓
Hmi	✓	✓								CLmcmcc									✓	mce									✓
Hitil	✓	✓								CLhic									✓	Islf	✓	✓							
Imcacc	✓	✓								nmpm									✓	Ihpv	✓	✓							
mci									✓	nmm									✓										

and class levels: metrics CLhpd, CLhpv, CLhme, CLhic are at class level, while metrics Ihpd, Ihpv, Ihic, Ihme, Ihpd are at file level for the source files. Table 4 indicates the metrics selected both in the CLAMI and CLAMI+ approaches for the different quantiles (cutoff thresholds) among the various releases. Quantiles' numbers from 1 to 9 correspond to 10%, ..., 90% respectively. Quantiles 1 and 2 have focused more on metrics on file level, while quantiles from 3 to 9 have selected object orientation metrics. Metrics CLhpd, CLhpv, CLhme, ckrfc, ckwmc are the ones that share more quantiles from 5 to 9, and they belong to the object orientation or complexity category. Table 5 displays the total number of metrics selected in the CLAMI and CLAMI+ approaches for the different quantiles among the various releases. They go from a minimum of 18 in the 4, 5 and 6 quantiles to a maximum of 52 in the 9 quantile.

Labelled Modules - The percentage of buggy modules on the total modules over the releases for each quantile has been measured. The quantiles are related to the different cut-off values chosen for the metrics. For example, for the release 701 and quantile 5 (which corresponds to the median value of the metrics), CLAMI computes about 60% of buggy modules. The difference between CLAMI and CLAMI+ of the percentage of buggy modules over the various releases for each quantile has been measured. As an example, in release 511 and quantile 5, the values of the difference is about 10. Furthermore, CLAMI predicts 50% of buggy modules, consequently, CLAMI+ predicts about 40% of buggy modules.

Table 5. Total metrics per approach - CLAMI and CLAMI+ - over the various quantiles

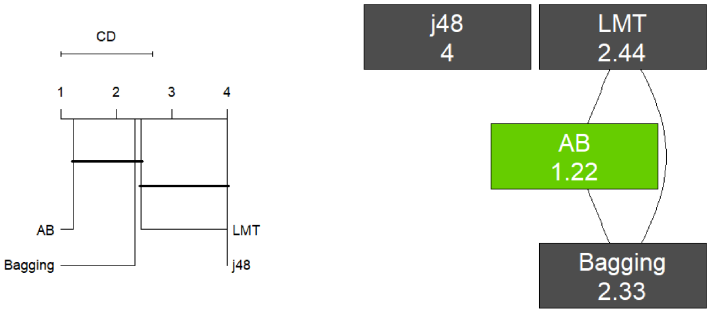
	1	2	3	4	5	6	7	8	9
CLAMI	37	36	35	32	18	18	21	23	47
CLAMI+	36	35	34	18	23	23	47	49	52

It is worth noticing that our manual work has covered just 380 modules, this has reduced the size of our training set. K-means has provided us with two clusters made up of different classes. We checked the different classes of these clusters against the documentation of defectiveness. Afterwards, we collected how CLAMI and CLAMI+ have labelled each class. Both of them have labelled 179 modules as defective in cluster 1, while in cluster 2 CLAMI has labelled 9617 modules as defective and CLAMI+ 9747.

ML Techniques - In our study, we have carried out a comparative analysis of all the ML techniques using the average rank calculated on the various performance metrics, such as accuracy, precision, recall, f-measure and AUC. In order to avoid overfitting we have exploited 10 cross-validation [51–53], furthermore, we have excluded all the prediction models whose Kappa statistic scored less than 0.80 in order to have a good agreement. Table 6 presents the different values of average accuracy computed for 4 ML techniques, such as J48, LMT, AdaBoost and Bagging obtained by using ten cross validation in order to avoid overfitting. The best accuracy values are achieved by all the techniques for the first quantile; CLAMI+ approach reaches high scores also in the ninth quantile. The technique that performs the best regardless of the two approaches and quantiles is Adaboost that scores 95.3508 in terms of average accuracy. Regardless of the specific techniques, CLAMI+ performs better than CLAMI in every quantile except the second, third and fourth. Table 7 displays the different values of

average buggy precision computed for 4 ML techniques i.e. J48, LMT, AdaBoost and Bagging. The technique that performs the best in terms of precision and regardless of the quantiles and approaches is LMT that scores 0,9385 in terms of average precision. Except for quantile one, two and nine, CLAMI performs better than CLAMI+ regardless of the specific technique. Table 8 illustrates the different values of average buggy recall computed for 4 ML techniques i.e. J48, LMT, AdaBoost and Bagging. The technique that performs the best in terms of recall and regardless of the quantiles and approaches is Adaboost that scores 0,9328 in terms of average recall. Except for quantile three and five, CLAMI performs better than CLAMI+ regardless of the specific technique. Table 9 shows the different values of average buggy f-measure computed for 4 ML techniques i.e. J48, LMT, AdaBoost and Bagging. The technique that performs the best in terms of f-measures and regardless of the quantiles and approaches is LMT that scores 0,9343. Except for quantile one, CLAMI performs better than CLAMI+ in terms of f-measure regardless of the specific technique. Table 10 displays the different values of average buggy AUC computed for 4 ML techniques i.e. J48, LMT, AdaBoost and Bagging. The technique that performs the best in terms of AUC and regardless of the quantiles and approaches is Adaboost that scores 0.9830. CLAMI+ performs better than CLAMI in quantiles 1, 2, 5 and 7 in terms of AUC regardless of the specific technique.

ML Statistical Comparison - With the aim of comparing the various algorithms to determine whether and to what extent one outperforms the others, we have employed some statistical tests on the values of the ML performance metrics. All the differences obtained have been tested by considering each pair of algorithms and their resulting p-values have been corrected. The differences for the



(a) (b)

Fig. 3. (a): Critical difference plot for the average buggy AUC in the CLAMI approach (the Nemenyi test at p-value < 0.05): classifiers that are no significantly different are connected by an horizontal line, (b): Algorithm graph for the average buggy AUC in the CLAMI approach (the Friedman post-hoc test with Bergmann and Hommel's correction at p-value < 0.05): classifiers that are not significantly different are connected

Table 6. Average accuracy of the ML techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging	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
	Average rank	3.0000	1.8888	2.3333	2.7777		Average rank	3.0000	2.4444	1.4444	3.1111

Table 7. Average buggy precision of the ML techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging	Quantile	Approach	J48	LMT	AdaBoost	Bagging
1	CLAMI	0.9884	0.9863	0.9885	0.9849	1	CLAMI+	0.9933	0.9935	0.9926	0.9921
2	CLAMI	0.9804	0.9794	0.9779	0.9760	2	CLAMI+	0.9821	0.9786	0.9816	0.9747
3	CLAMI	0.9720	0.9718	0.9695	0.9657	3	CLAMI+	0.9600	0.9633	0.9609	0.9544
4	CLAMI	0.9495	0.9539	0.9465	0.9419	4	CLAMI+	0.9292	0.9364	0.9327	0.9302
5	CLAMI	0.9274	0.9422	0.9244	0.9275	5	CLAMI+	0.9208	0.9370	0.9269	0.9264
6	CLAMI	0.9204	0.9371	0.9318	0.9228	6	CLAMI+	0.9059	0.9181	0.9135	0.9111
7	CLAMI	0.9125	0.9292	0.9237	0.9244	7	CLAMI+	0.8827	0.8875	0.8886	0.8802
8	CLAMI	0.8779	0.9179	0.9073	0.8944	8	CLAMI+	0.8344	0.8555	0.8697	0.8280
9	CLAMI	0.8662	0.8895	0.9008	0.8914	9	CLAMI+	0.8484	0.9166	0.9116	0.8931
	Average rank	2.7777	1.6666	2.4444	3.1111		Average rank	3.1111	1.4444	1.8888	3.5555

Table 8. Average buggy recall of the ML techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging	Quantile	Approach	J48	LMT	AdaBoost	Bagging
1	CLAMI	0.9918	0.9931	0.9914	0.9866	1	CLAMI+	0.9944	0.9939	0.9952	0.9926
2	CLAMI	0.9833	0.9878	0.9831	0.9895	2	CLAMI+	0.9829	0.9829	0.9850	0.9838
3	CLAMI	0.9691	0.9662	0.9673	0.9705	3	CLAMI+	0.9682	0.9640	0.9685	0.9738
4	CLAMI	0.9497	0.9467	0.9569	0.9582	4	CLAMI+	0.9481	0.9325	0.9408	0.9478
5	CLAMI	0.9367	0.9194	0.9293	0.9375	5	CLAMI+	0.9297	0.9274	0.9332	0.9330
6	CLAMI	0.9311	0.9191	0.9265	0.9333	6	CLAMI+	0.9060	0.8993	0.9208	0.9113
7	CLAMI	0.9047	0.9047	0.9067	0.9056	7	CLAMI+	0.8760	0.8903	0.9069	0.8840
8	CLAMI	0.8779	0.8908	0.8809	0.8820	8	CLAMI+	0.8802	0.8604	0.8796	0.8877
9	CLAMI	0.8682	0.8539	0.8619	0.8471	9	CLAMI+	0.8018	0.9166	0.8578	0.8367
	Average rank	2.4444	3.0000	2.6666	1.8888		Average rank	2.7777	3.3333	1.6666	2.2222

various performance metrics have been checked with Iman and Davenport (ID)'s omnibus test. For CLAMI we have obtained a p-value < 0.05 just for the AUC metric, while for CLAMI+ we have obtained a p-value < 0.05 for all performance metrics. For these results, there is at least one algorithm that performs differently than the rest and therefore the post-hoc analysis can be performed for further investigation. More precisely, we have applied Nemenyi's test and Friedman's post-hoc test that computes the raw p-value for each pair of algorithms and their correction using Bergmann and Hommel's correction.

Table 9. Average buggy F-measure of the ML techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging	Quantile	Approach	J48	LMT	AdaBoost	Bagging
1	CLAMI	0.9901	0.9897	0.9899	0.9857	1	CLAMI+	0.9938	0.9937	0.9939	0.9924
2	CLAMI	0.9818	0.9836	0.9805	0.9827	2	CLAMI+	0.9825	0.9807	0.9833	0.9792
3	CLAMI	0.9705	0.9690	0.9684	0.9681	3	CLAMI+	0.9640	0.9636	0.9647	0.9639
4	CLAMI	0.9495	0.9502	0.9516	0.9499	4	CLAMI+	0.9384	0.9342	0.9365	0.9388
5	CLAMI	0.9320	0.9306	0.9266	0.9324	5	CLAMI+	0.9251	0.9321	0.9299	0.9295
6	CLAMI	0.9255	0.9279	0.9290	0.9278	6	CLAMI+	0.9058	0.9085	0.9170	0.9111
7	CLAMI	0.9084	0.9166	0.9149	0.9147	7	CLAMI+	0.8787	0.8886	0.8974	0.8815
8	CLAMI	0.8777	0.9039	0.8937	0.8878	8	CLAMI+	0.8553	0.8569	0.8728	0.8547
9	CLAMI	0.8667	0.8711	0.8801	0.8680	9	CLAMI+	0.8232	0.9166	0.8835	0.8561
	Average rank	3.0000	1.8888	2.2222	2.8888		Average rank	3.0000	2.5555	1.4444	3.0000

Table 10. Average buggy AUC of the ML techniques

Quantile	Approach	J48	LMT	AdaBoost	Bagging	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
	Average rank	4.0000	2.4444	1.2222	2.3333		Average rank	4.0000	2.3333	1.2222	2.4444

Considering the CLAMI approach, Fig. 4 shows the results obtained with Nemenyi's test for the AUC metric, according to which there is no significant differences between methods that are grouped together by a horizontal line: the top line represents the axis on which the average ranks of methods (see Table 10) is plotted; the highest (best) ranks are to the right; the critical difference is also shown above the graph. Still for CLAMI, Fig. 3 shows the results obtained with Friedman's post-hoc test with Bergmann and Hommel's correction for the AUC metric, according to which there is no significance differences between methods that have connected nodes: each node reporting the method name with its average rank (see Table 10) is plotted; the highest (best) ranks are not connected. Compared with the critical difference plot, the graph shows more differences between methods. We have performed the same plots for the other performance metrics. According to the ScottKnott ESD test, we have obtained a non negligible effect size for the AUC metric either in the CLAMI and CLAMI+ approaches as shown in Fig. 4 and 5. In both the approaches Bagging is the most influential ML technique.

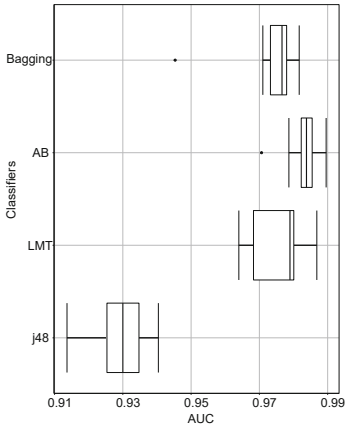


Fig. 4. AUC performance difference for CLAMI

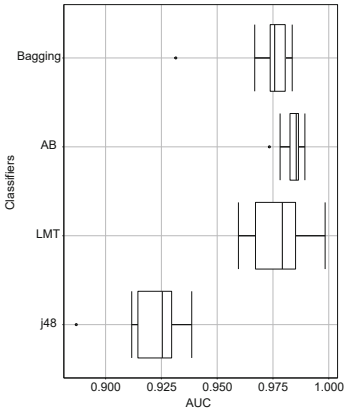


Fig. 5. AUC performance difference for CLAMI+

5 Discussion

In this section we present the answers of our research questions. In regards to RQ1 and RQ2 we employed different performance metrics and statistical tests. In detail, we have used accuracy, precision, recall, f-measure and the area under the receiver operating characteristics (AUC). Accuracy measures the percentage of the correct predictions among all the predictions; precision measures rate of the correctly predicted defective instances among all the instances predicted as defective; recall measures the rate of correctly predicted defective instances among all the actual defective instances. F-measure represents the harmonic mean of precision and recall, AUC measures the area under the ROC curve that is plotted using recall and false positive rate by changing software metrics' cutoff thresholds. All the above metrics have widely used in previous literature in the field of software defect prediction [6,22,23,54,55]. For statistical tests, we have conducted Iman and Davenport (ID) omnibus test, Nemenyi tests, the Friedman post hoc test with Bergman and Hommel's correction and Scott-KnottESD test. We have followed the guidelines of Demšar [42] for the visual representation of the Nemenyi and Friedman tests.

RQ1: which unsupervised approach performs best in terms of our performance metrics? Two main unsupervised approaches have been involved for the comparison: CLAMI and CLAMI+. They entail varying their metrics' cutoff thresholds and, consequently, the creation of 9 quantiles to conduct the labelling operation, and afterwards, the application of supervised ML techniques - more in detail, we have considered all the ML techniques whose Kappa statistic reached a value superior to 0.80. To find the best approach we have computed the average of all the values of one performance metric over the techniques and then quantiles, giving the same weight to each of them. In regards to accuracy CLAMI+ scores

95.6928 and does better than CLAMI that scores 94.7845. CLAMI achieves a higher result than CLAMI+ in precision (0.9389 vs 0.9253), recall (0.9335 vs 0.9276), f-measure (0.9360 vs 0.9258) and AUC (0.9647 vs 0.9636).

RQ2: which supervised technique performs best in terms of our performance metrics in the two unsupervised approaches? Table 11 introduces the average performance metrics values per techniques over approaches and quantiles. Adaboost achieves the best score in accuracy (95.3508), recall (0.9329), and AUC (0.9830), whilst LMT performs best in precision (0.9385) and f-measure (0.9343). Concerning the CLAMI and CLAMI+ approaches and the AUC performance metric, J48 has achieved the best average rank. Figure 4 shows that there are no statistical differences within the pairs LMT/Bagging, LMT/Adaboost and Bagging/Adaboost. For CLAMI+ and the accuracy performance metric, J48 and Bagging have achieved the best average rank: there are no statistical differences within the pairs J48/LMT, LMT/Adaboost, LMT/Bagging and J48/Bagging. For CLAMI+ and the precision performance metric, J48 and Bagging have achieved the best average rank: there are no statistical differences within the pairs J48/Bagging and LMT/Adaboost. For CLAMI+ and the recall performance metric, J48 and Bagging have achieved the best average rank: there are no statistical differences within the pairs J48/LMT, J48/Adaboost, LMT/Bagging, Adaboost/Bagging and J48/Bagging. For CLAMI+ and the f-measure performance metric, J48 and Bagging have achieved the best average rank: there are no statistical differences within the pairs J48/LMT, LMT/Adaboost, Adaboost/Baggin, LMT/Bagging and J48/Bagging.

Table 11. Average performance metrics per techniques over approaches and quantiles

	J48	LMT	Adaboost	Bagging
Accuracy	95.1296	95.3388	95.3508	95.1354
Precision	0.92508	0.9385	0.93602	0.92884
Recall	0.9278	0.9305	0.9329	0.9312
F-measure	0.9260	0.93431	0.9341	0.9291
AUC	0.9246	0.9768	0.9830	0.9722

RQ3: what is the impact of using different cutoff thresholds (i.e. quantiles) on the comparison results? Quantiles values affects the resulting performance metrics, as a consequence, metrics' thresholds have to be taken into consideration since the data distribution is not Gaussian.

6 Conclusion

In this paper, we have conducted a comparative analysis of different unsupervised approaches: CLAMI, CLAMI+ and K-means. Once obtained a labelled dataset by exploiting these approaches, we have carried out a second comparison amongst various ML supervised techniques. Starting from an unlabelled software dataset, CLAMI and CLAMI+ allow users to label each instance by exploiting their metrics magnitude. We have applied different supervised ML techniques on the resulting datasets and detected the best ones by using statistical comparisons, such as Iman and Davenport's test, Nemenyi's test, Friedman's post-hoc test

with Bergmann and Hommel's correction and Scott-KnottESD test. Among the used 17 ML techniques, we have selected the ones which scored more than 0.80 in the Kappa statistic performance metric: J48, Adaboost, LMT and Bagging. The computed predictive average accuracy metric is around 95% for all the ML techniques. Furthermore, our study has shown that, for CLAMI and CLAMI+, the J48 method performs best according to the average AUC performance metric. In regards to CLAMI+, J48 is followed by: the Bagging method for the average AUC, average accuracy, average precision and average f-measure metrics; and by the LMT method for the average recall metric. Based on the results of our empirical analysis, we conclude that no significant performance differences have been found in the selected classification techniques. Future works based on this study could involve all the datasets for each release of Geant4 (now only 482 modules for each release are considered) and the employment of other clustering unsupervised techniques such as Fuzzy CMeans and Fuzzy SOMs.

Acknowledgment. The authors thank the Imagix Corporation for providing an extended free license of Imagix 4D to perform this work.

References

1. Arar, O.F., Ayan, K.: Software defect prediction using cost-sensitive neural network. Appl. Softw. Comput. **33**, 263–277 (2015)
2. Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng. **33**, 2–13 (2007)
3. Ronchieri, E., Canaparo, M.: Metrics for software reliability: a systematic mapping study. J. Integr. Des. Process Sci. **22**, 5–25 (2018)
4. Malhotra, R., Bansal, A.J.: Cross project change prediction using open source projects. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE (2014). https://doi.org/10.1109/ICACCI.2014.6968347
5. Zhong, S., Khoshgoftaar, T.M., Seliya, N.: Unsupervised learning for expert-based software quality estimation. In: Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering. IEEE (2004). https://doi.org/10.1109/HASE.2004.1281739
6. Yang, J., Qian, H.: Defect prediction on unlabeled datasets by using unsupervised clustering. In: Proceedings of the IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS) (2016)
7. Li, N., Shepperd, M.J., Guo, Y.: A systematic review of unsupervised learning techniques for software defect prediction. Inf. Softw. Technol. **122**, 106287 (2020)
8. Catal, C., Sevim, U., Diri, B.: Clustering and metrics thresholds based software fault prediction of unlabeled program modules. In: 2009 Sixth International Conference on Information Technology: New Generations, pp. 199–204 (2009)
9. Zhong, S., Khoshgoftaar, T.M., Seliya, N.: Analyzing software measurement data with clustering techniques. IEEE Intell. Syst. **19**(2), 20–27 (2004). https://doi.org/10.1109/MIS.2004.1274907

10. Bishnu, P.S., Bhattacherjee, V.: Software fault prediction using quad tree-based k-means clustering algorithm. IEEE Trans. Knowl. Data Eng. (2012). https://doi.org/10.1109/TKDE.2011.163
11. Aleem, S., Capretz, L.F., Ahmed, F.: Benchmarking machine learning techniques for software defect detection. Int. J. Softw. Eng. Appl. 6(3) (2015). https://doi.org/10.5121/ijsea.2015.6302
12. Alsawalqah, H., Hijazi, N., Eshtay, M., et al.: Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Appl. Sci. **10**(1745) (2020). https://doi.org/10.3390/app10051745
13. Yang, B., Zheng, X., Guo, P.: Software metrics data clustering for quality prediction. In: Huang, D.-S., Li, K., Irwin, G.W. (eds.) ICIC 2006. LNCS (LNAI), vol. 4114, pp. 959–964. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-540-37275-2_121
14. Abaei, G., Selamat, A.: Increasing the accuracy of software fault prediction using majority ranking fuzzy clustering. In: Lee, R. (ed.) Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. SCI, vol. 569, pp. 179–193. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10389-1_13
15. Zhang, F., Zheng, Q., Zou, Y., Hassan, A.E.: Cross-project defect prediction using a connectivity-based unsupervised classifier. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 309–320 (2016). https://doi.org/10.1145/2884781.2884839
16. Chang, R., Shen, X., Wang, B., Xu, Q.: A novel method for software defect prediction in the context of big data. In: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), pp. 100–104 (2017). https://doi.org/10.1109/ICBDA.2017.8078785
17. Yan, M., Yang, M., Liu, C., Zhang, X.: Self-learning change-prone class prediction. In: The 28th International Conference on Software Engineering and Knowledge Engineering, SEKE 2016, Redwood City, San Francisco Bay, USA, 1–3 July 2016, pp. 134–140 (2016). https://doi.org/10.18293/SEKE2016-039
18. Park, M., Hong, E.: Software fault prediction model using clustering algorithms determining the number of clusters automatically. Int. J. Softw. Eng. Appl. **8**, 199–204 (2014)
19. Herbold, S., Trautsch, A., Grabowski, J.: A comparative study to benchmark cross-project defect prediction approaches. IEEE Trans. Softw. Eng. **44**(9), 811–833 (2017)
20. Shepperd, M., Song, Q., Sun, Z., Mair, C.: Data quality: some comments on the NASA software defect datasets. IEEE Trans. Softw. Eng. **39**(9), 1208–1215 (2013). https://doi.org/10.1109/TSE.2013.11
21. Peters, F., Menzies, T., Gong, L., Zhang, H.: Balancing privacy and utility in cross-company defect prediction. IEEE Trans. Softw. Eng. **39**(8), 1054–1068 (2013). https://doi.org/10.1109/TSE.2013.6
22. Nam, J., Kim, S.: CLAMI: defect prediction on unlabeled datasets (T). In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE (2015). https://doi.org/10.1109/ASE.2015.56
23. Yan, M., Zhang, X., Liu, C., et al.: Automated change-prone class prediction on unlabeled dataset using unsupervised method. Inf. Softw. Technol. **92**, 1–16 (2017)
24. Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: IEEE/ACM 37th International Conference of Software Engineering (2015). https://doi.org/10.1109/ICSE.2015.91

25. Agostinelli, S., Allison, J., Amako, K., et al.: GEANT4 - a simulation toolkit. Nucl. Instrum. Methods Phys. Res. Sect. A **506**(3), 250–303 (2003)
26. Ronchieri, E., Pia, M.G.: Assessing software quality in high energy and nuclear physics: the geant4 and root case studies and beyond. In: Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), Sydney, Australia, Australia (2018)
27. Imagix: Reverse Engineering Tools - C, C++, Java - Imagix. https://www.imagix.com/
28. Preston-Werner, T.: Semantic Versioning 2.0.0 (2013). https://semver.org/spec/v2.0.0.html
29. Fenton, N.E., Neil, M.: A critique of software defect prediction models. IEEE Trans. Softw. Eng. **25**(5), 675–689 (1999)
30. McCabe, T.: A complexity measure. IEEE Trans. Softw. Eng. SE **2**(4), 308–320 (1976)
31. Halstead, M.H.: Elements of Software Science (1975)
32. Chidamber, S.R., Kemerer, C.F.: Metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
33. Zhang, H., Zhang, X.: Comments on data mining static code attributes to learn defect prediction. IEEE Trans. Softw. Eng. **33**(9), 635–636 (2007)
34. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics **33**, 159–174 (1977)
35. Yucalar, F., Ozcift, A., Borandag, E., Kilinc, D.: Multiple-classifiers in software quality engineering: combining predictors to improve software fault prediction ability. Eng. Sci. Technol. Int. J. (2019). https://doi.org/10.1016/j.jestch.2019.10.005
36. Yan, M., Xia, X., Shihab, E., et al.: Automating change-level self-admitted technical debt determination. IEEE Trans. Softw. Eng. **45**(12), 1211–1229 (2019). https://doi.org/10.1109/TSE.2018.2831232
37. Garcìa, S., Fernandez, A., Luego, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. Inf. Sci. **180**, 2044–2064 (2009)
38. Friedman, M.: A comparison of alternative tests of significance for the problem of M rankings. Annal. Math. Stat. **11**(1), 86–92 (1940). https://www.jstor.org/stable/2235971
39. Iman, R.L., Davenport, J.M.: Approximations of the critical region of the friedman statistic. Commun. Stat. **9**, 571–595 (1980)
40. Calvo, B., Santafé, G.: scmamp: statistical comparison of multiple algorithms in multiple problems. R J. **8**(1), 248–256 (2016)
41. Bergmann, B., Hommel, G.: Improvements of general multiple test procedures for redundant systems of hypotheses. Mult. Hypotheses Test. **70**, 100–115 (1988)
42. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
43. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. **43**(1), 1–18 (2017). https://doi.org/10.1109/TSE.2016.2584050
44. Azeem, N., Usmani, S.: Analysis of data mining based software defect prediction techniques. Glob. J. Comput. Sci. Technol. **11** (2011)
45. Wang, J., Ma, Y., Zhang, L., Gao, R., Wu, D.: Deep learning for smart manufacturing: methods and applications. J. Manuf. Syst. **48**, 144–156 (2017)
46. Sculley, D.: Web-scale K-means clustering. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, New York, NY, USA, pp. 1177–1178. ACM (2010). https://doi.org/10.1145/1772690.1772862

47. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theor. **28**(2), 129–137 (2006). https://doi.org/10.1109/TIT.1982.1056489
48. Elkan, C.: Using the triangle inequality to accelerate k-means. In: Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML 2003, pp. 147–153. AAAI Press (2003), http://dl.acm.org/citation.cfm?id=3041838.3041857
49. Kaur, D., Kaur, A., Gulati, S., Aggarwal, M.: A clustering algorithm for software fault prediction. In: International Conference on Computer and Communication Technology (ICCCT) (2010). https://doi.org/10.1109/ICCCT.2010.5640474
50. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, Cham (2002)
51. Usama, M., Qadir, J., Raza, A., et al.: Unsupervised machine learning for networking: techniques. applications and research challenges. IEEE Access **7**, 65579–65615 (2019). https://doi.org/10.1109/ACCESS.2019.2916648
52. Domingos, P.: A few useful things to know about machine learning. Commun. ACM **55**(10), 78–87 (2012). https://doi.org/10.1145/2347736.2347755
53. Srivastava, N., Krizhevsky, G.H.A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
54. Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., Ubayashi, N.: An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, New York, NY, USA, pp. 172–181. ACM (2014). https://doi.org/10.1145/2597073.2597075
55. Jing, X.Y., Ying, S., Zhang, Z.W., Wu, S.S., Liu, J.: Dictionary learning based software defect prediction. In: Proceedings of the 36th International Conference on Software Engineering, pp. 414–423 (2014). https://doi.org/10.1145/2568225.2568320