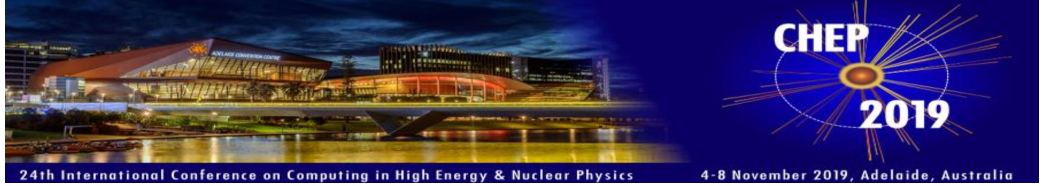


Assessing software defect prediction on WLCG software

A study with Unlabelled datasets and Machine Learning Techniques

Elisabetta Ronchieri, Marco Canaparo, Davide Salomoni, Barbara Martelli

INFN CNAF, Bologna, Italy

CHEP 2019, Adelaide, November 5, 2019

Outline

- Context
- Experimental Settings
- Results
- Conclusions

Machine Learning in Software Engineering (SE)

- Machine Learning (ML) may help in various SE tasks, such as software defects prediction and estimation and test code generation.
- To apply ML techniques, input data have to be properly preprocessed and collected in software datasets.
- Software datasets are a collections of:
  - instances, i.e. modules, such as files, classes and functions;
  - features, i.e. software metrics.
- In SE practice, datasets may lack essential information, such as defectiveness
  - not available in new software projects (historical data and software datasets are not available);
  - not well traced in an existing software projects (e.g. the dataset does not include the class name)

Why this study?

- WLCG uses a variety of software, much of which adopts devops procedures in their development and maintenance phases.
- Traditional software metrics, e.g. cyclomatic complexity metric and lines of code metric, are monitored over releases
  - but no particular analysis is usually performed on these data.
- Documentation related to changes in code, like release notes, is not used to predict defectiveness
  - lack of a comprehensive study about practical aspects of software analytics models

➡ We do have plenty of data, let's try to extract some knowledge from them!

Objectives

- To test the usefulness of ML techniques in WLCG domain:
  - identifying pieces of code that require particular attention
- To build a prediction model on unlabelled datasets:
  - using Geant4 unlabelled datasets collected with Imagix 4D
  - using CLAMI [1] and CLAMI+ [2] approaches that label modules in the datasets
  - using a set of ML techniques to predict defectiveness in modules
- To identify the set of software metrics that can be used without selecting a priori-metric thresholds.

[1] J. Nam, S. Kim, 2015. CLAMI: Defect Prediction on Unlabeled Datasets, In Proc. 30th IEEE/ACM International Conference on Automated Software Engineering
[2] M. Yan, X. Zhang, C. Liu, L. Xu, M. Yang, D. Yang, 2017. Automated change-prone class prediction on unlabeled dataset using unsupervised method, In Information and Software Technology, 92, 1-16

Approaches to Unlabeled Software Datasets Prediction

Approaches	**Strength/**Limitations
Within-project defect prediction	**High precision**/uses a set of metrics specific for the analyzed project -> difficult to generalize
Cross-project defect prediction	**Useful for projects without labeled datasets**/uses metrics from other projects, doesn't consider different probability distributions among datasets
Expert-based defect prediction	**High precision**/always requires human experts
Threshold-based defect prediction	**Some step is automated**/needs to decide metrics thresholds in advance
Clustering, LAbeling, Metric selection, Instance selection (CLAMI)	**Automatic, no manual effort, works with unlabelled datasets/**metric values are not always comparable may introduce bias, depends on thresholds
CLAMI +	**normalizes metrics values/**depends on thresholds

Unlabelled and Labelled Datasets

	McCabe Tot.Complexity	#Lines in Subsystem	Chidamber Kemerer Depth of Inheritance	….	Level in Hierarchy	Label
C++ Class 1	10	11	4	…	6	Buggy
C++ Class 2	23	10	15	…	14	Clean
C++ Class 3	15	17	4	…	8	Clean
….	…	…	…	…	…	…
C++ Class N	7	9	21	…	13	?

Unlabelled datasets are the **vast majority** of software datasets.

- The extraction of the complete set of features (defectiveness included) implies effort and time.
- It is not easy to select tools that measure software characteristics

We need an automated way to label unlabelled datasets in order to be able to apply well established Supervised ML techniques

CLAMI: evaluating metrics and instance labelling

The key idea of the CLAMI approach is to label instances by using the magnitude of metric values

The intuition of this labelling process is based on the defect proneness tendency of typical defect prediction datasets:

- ✓ higher complexity causes more defect proneness

Metric evaluation: cut-off threshold (median)

Identify Higher Values: greater than the threshold (yellow cells)

Instance	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Label
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**	?

Context
Experimental settings
Results
Conclusions

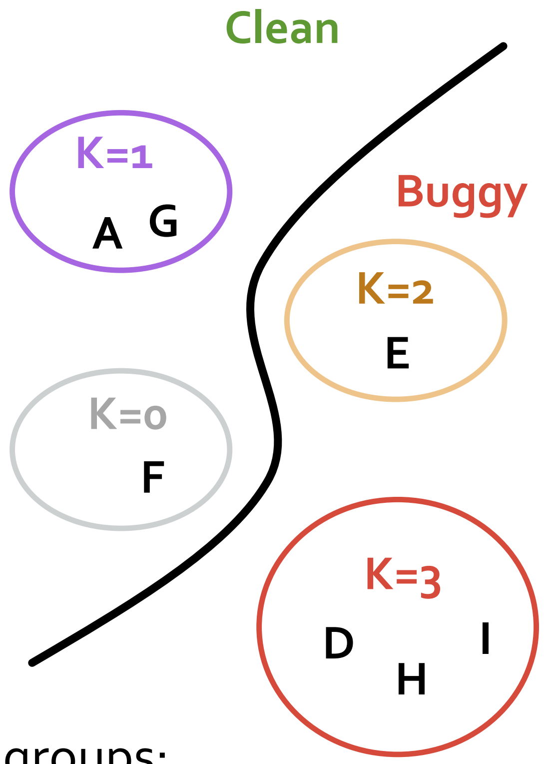
CLAMI: Clustering Instances

K = Number of metrics for each instance whose values are greater than the median for each metric (Higher Values)

Instance	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Label
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**	?

[3] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empirical software Engineering, vol. 17, no. 4{5, pp. 531{577,2012.

Instances	K
A	K= 1
D	K = 3
E	K = 2
F	K = 0
G	K = 1
H	K = 3
I	K = 3



Clusters divided into 2 groups:

1. Clean for K in the bottom half
2. Buggy for K in the top half

The instances that have larger value on all metrics are more likely to be defective [3]

CLAMI: Metric and Instance Selection

We calculate the Metric Violation Score as MVS_j = the ratio between the number of violation in the j-th metric and the number of metric values in the j-th metric

	M 1	M 2	M 3	M 4	M 5
MVS	$\frac{1}{7}$	$\frac{5}{7}$	$\frac{1}{7}$	$\frac{0}{7}$	$\frac{0}{7}$

Metrics with the minimum MVS are selected for the Training dataset.

Metric selection aimed at selecting most informative metrics -> erase metrics that **violate the defect-proneness tendency** (grey cells) [2]:

- D is Buggy, but Metric2 = 10 is not greater than Median2
- E is Clean, but Metric1 = 15 is greater than Median1

Instance	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Label
A	10	11	4	6	8	Clean
D	23	10	15	14	10	Buggy
E	15	17	4	8	5	Clean
F	9	10	9	6	3	Clean
G	11	13	15	5	8	Clean
H	14	10	17	9	0	Buggy
I	7	9	21	13	9	Buggy

Instance	M 4	M 5	Label
A	6	8	Clean
D	14	10	Buggy
E	8	5	Clean
F	6	3	Clean
G	5	8	Clean
H	9	0	Buggy
I	13	9	Buggy

Experimental Configuration

Software Metrics Datasets

GEANT4 — A SIMULATION TOOLKIT

To collect multi-version datasets, 1 for each geant4 release

Imagix

To measure software metrics

Unlabeled to Labeled Datasets Approaches

CLAMI, CLAMI+, and others

To obtain **labeled datasets**

To detect **buggy** and **clean** **modules**

34 Geant4 multi-version datasets (482 modules for each version; 66 software metrics) are considered for the preprocessing activity: datasets contains the same classes and the same software metrics.

Machine Learning

Classifiers
- K-nearest neighbours
- Decision tree
- – – –
- Random forest
- Neural networks
- Logistic regression
- Support Vector Machines

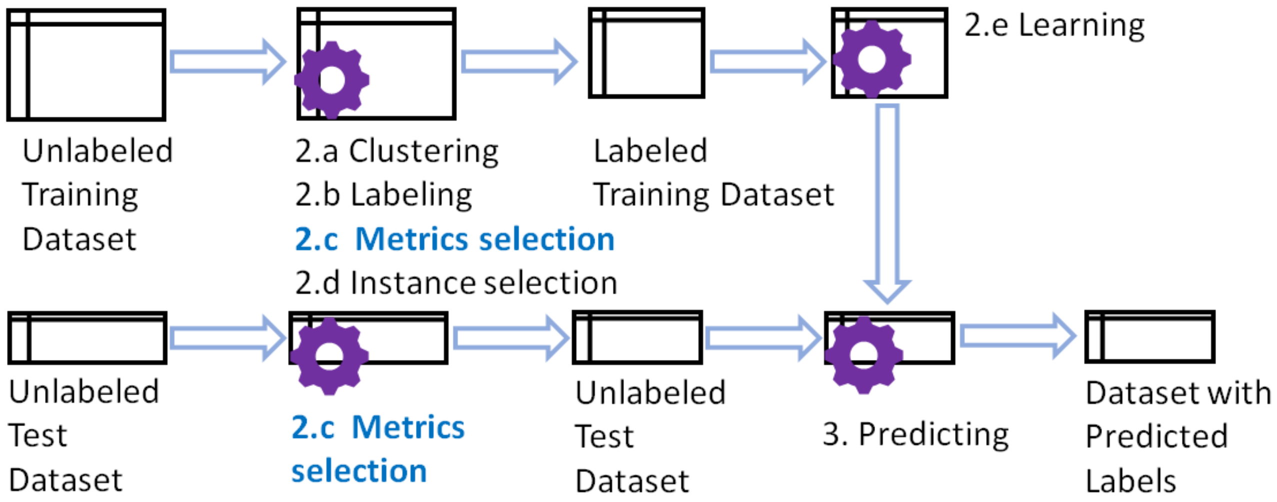
Model Statistics

- •Confusion Matrix
- •TP Rate
- •FP Rate
- •Precision
- •Recall
- •F-measure
- •ROC Area
- •Kappa Statistics

Statistical Tests

Friedman Test
Nemenyi Test
Others

Workflow



Input:

- U = set of unlabelled instances
- C = set of machine learning techniques

Output:

- Average P (P = set of performance indicators)
- Test dataset prediction

Process:

1. Randomly split dataset in training (67%) dataset and test (33%) dataset
2. Apply CLAMI/CLAMI+-based approach to label training dataset
3. Construct classifier by applying c Є C to training dataset
4. Assess classifier
5. Predict test dataset

Context
Experimental settings
Results
Conclusions

Performance Indicators

Each measure can be defined on the basis of the confusion matrix below. Actual Values are derived from the software documentation (e.g. release notes and metrics).

Actual value \ Prediction	**Buggy**	**Clean**
Buggy	True Positive (TP)	False Negative (FN)
Clean	False Positive (FP)	True Negative (TN)

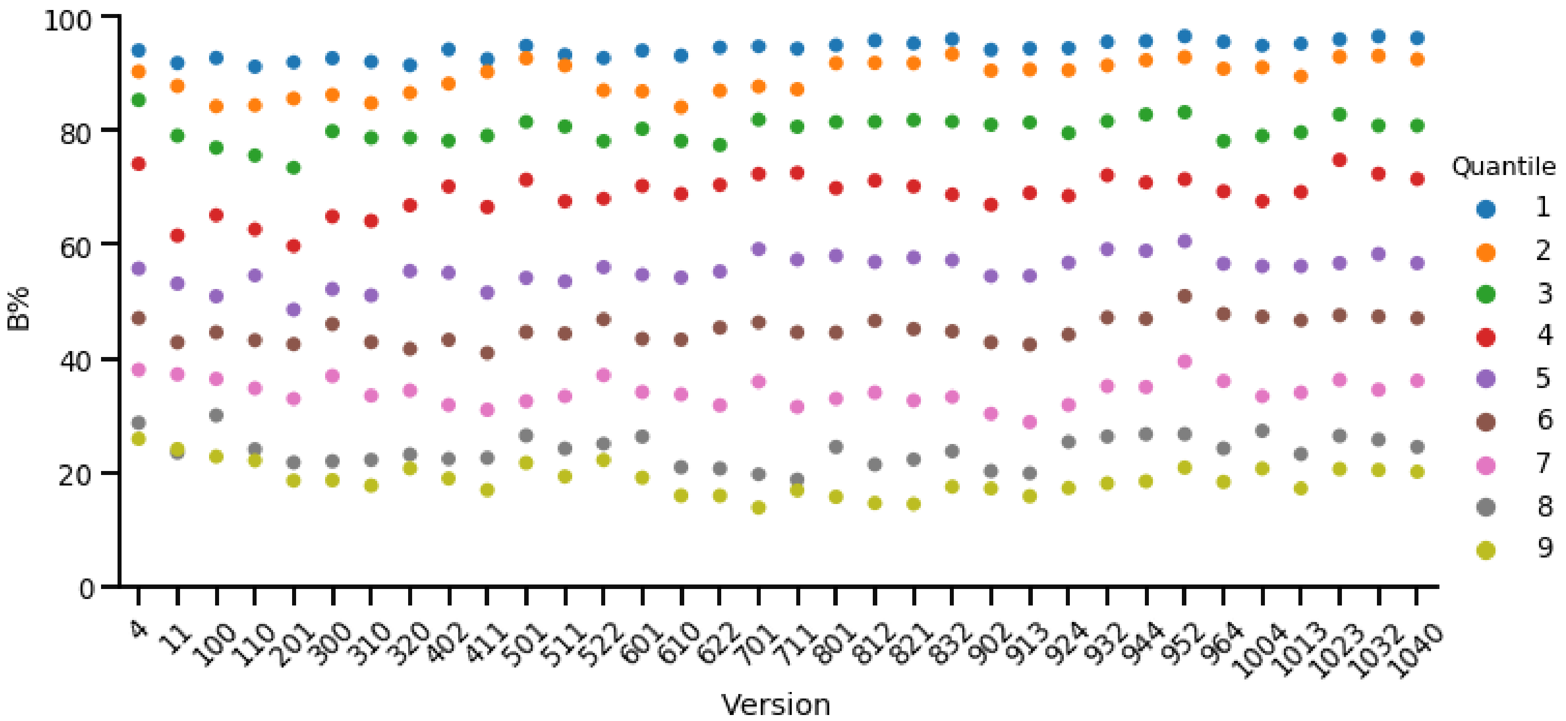
To assess our approach, we have checked the predictions obtained against the software documentation (release notes).

- **Kappa statistic** is a metric (whose value is ∈ [0,1]) that compares an Observed accuracy with the random classifier accuracy [4].
  - It determines how much better a classifier is performing over the performance of a classifier that simply guesses at random.
  - If Kappa statistic ∈ [0.81, 0.99], then the value indicates an almost perfect agreement.
- **Accuracy** is the percentage of instances correctly classified as either buggy or clean

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

[4] Landis, J.R.; Koch, G.G. (1977). The measurement of observer agreement for categorical data. Biometrics 33 (1): 159{174

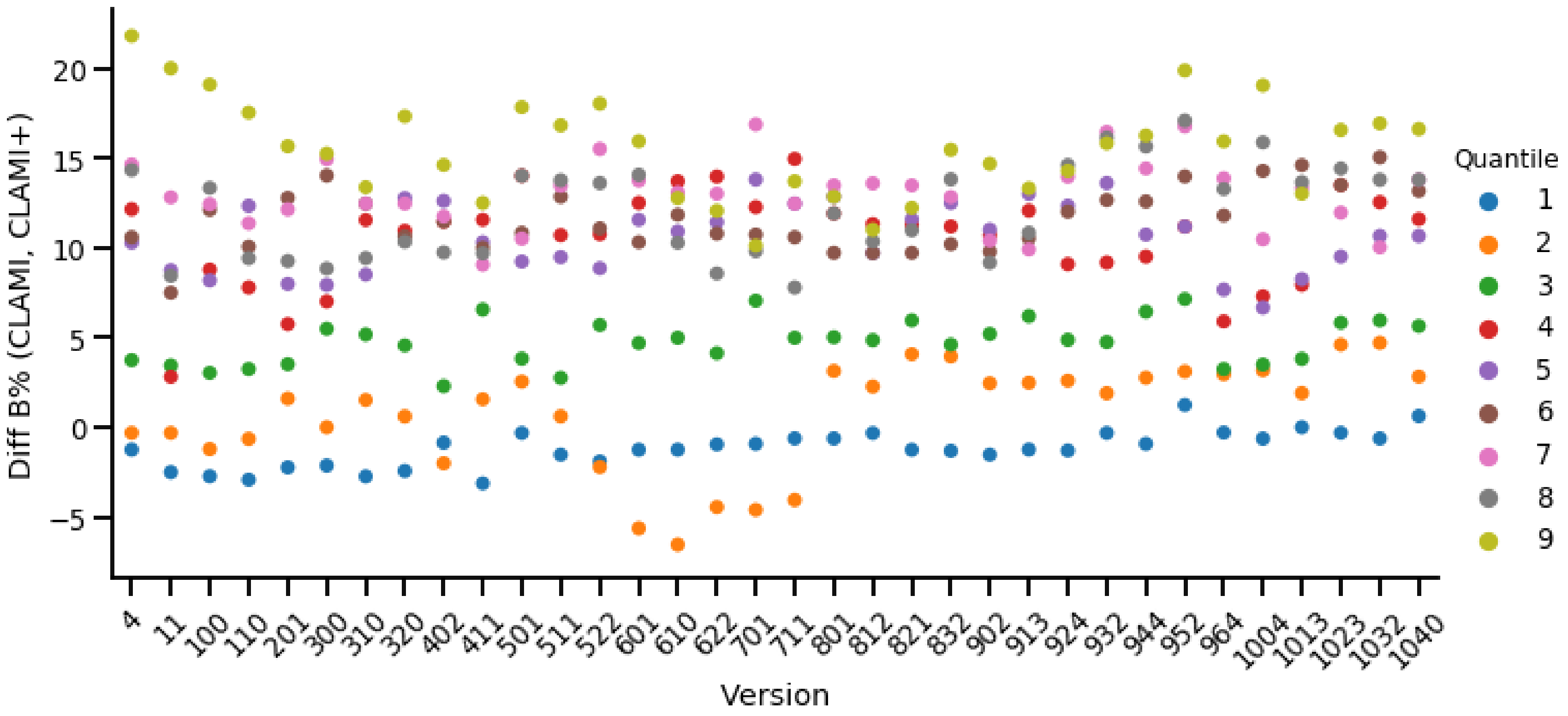
Clustering Phase: CLAMI B%



- B% represents the ratio between the number of buggy modules and total modules.
- Low B% values identify classes with higher clean than buggy.
- The modules characterized by larger values on all metrics are more likely defective.

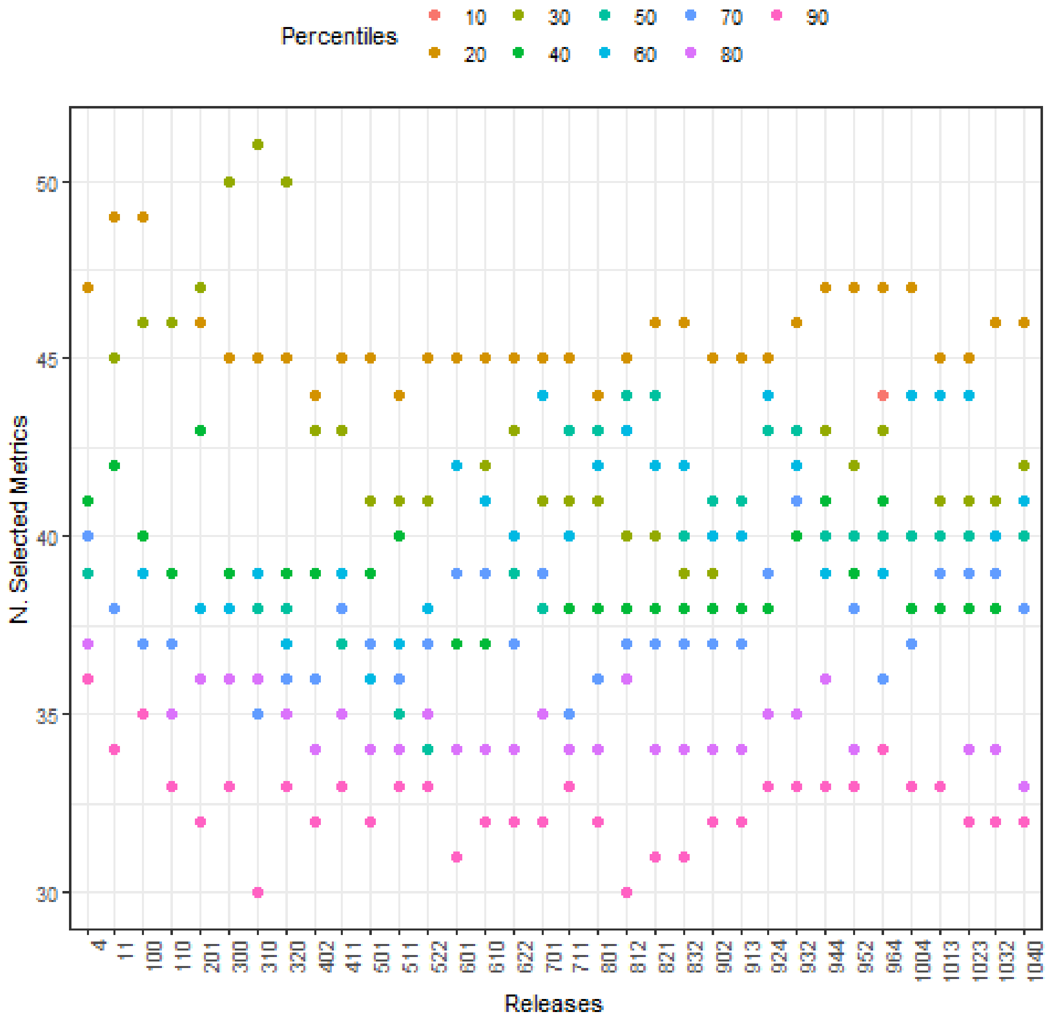
Context
Experimental settings
Results
Conclusions

Clustering Phase: CLAMI B% - CLAMI+ B%



- ClusteringCLAMI : the violation table includes either 0 or 1 values; 1 shows a metric violation according to the metric cut-off value
- ClusteringCLAMI+: the violation table includes continuous values from 0 to 1 determined by a sigmoid function

Selected metrics



- Selected Metrics: [45%,77%]
- Average Selected Metrics: 38 out of 66
- N. Releases: 34
- Tried with 9 Cutoffs: percentile at 10, 20, … , 90 – *referred to the #Violations of defect-proneless tendency which determine metric erasure*
- Metrics Categories:size, complexity, maintainability, object orientation
- The smaller the N. of Selected Metrics, the bigger the percentile.

Context
Experimental settings
Results
Conclusions

Comparing Models Statistics

Average	Accuracy			
	Bagging	J48	LMT	AdaBoost
CLAMI	95.47%	95.23%	96.35%	95.58%
CLAMI+	96.52%	96.14%	97.24%	97.21%

- ML techniques: classication and regression algorithms on training datasets with 10-fold cross validation to predict defectiveness on test datasets.
- Kappa statistic < 0.81 when Accuracy < 90%
- Kappa statistic in these cases was always in the range [0.82, 95]

Conclusions

- At the moment our approach uses CLAMI, its contribution is two-fold:
  - it automatically labels dataset based on the magnitude of metric values
  - it can be easily automated and used by non ML experts.
- In our testbed Bagging, LMT, J48 and AdaBoost performed well in terms of accuracy and kappa.
  - learning techniques can be complementary to existing SE tools and methodologies to address SE tasks.
- In the near future, we are going to experiment other clustering techniques and define a dictionary for code changes.

Thanks and Questions

Be curious! Have fun!

Acknowledgements:

- INFN CNAF for funds
- Imagix Corp. for Imagix4D license
- Doina Cristina Duma for VM
- Daniele Cesini for GPU-onboard resource

Contact: elisabetta.ronchieri@cnaf.infn.it

Backup slides

Metrics

https://www.imagix.com/user_guide/software-metrics.html

Testbed Description

The experimental Testbed was composed by 2 Machines:

Physical Machine	Virtual Machine
• CPU: 2xIntel(R)E5-2640v2 • @2.00GHz • Number of Cores: 32 (HT) • GPU: 2 x NVIDIA TeslaK40m • Memory: 128GB RAM. • Operating System: CentOS • Linux release 7.4.1708. • Python: 2.7.5 • Jupyter-notebook: 5.7.8	• CPU: 16 V CPU • Disk: 40 GB • Memory: 32 GB RAM • Operating System: Ubuntu • Linux release 18.04 • Python: 3.6.7 • R: 3.5.2 Jupyter-notebook: 5.7.4 hosted on an hypervisor with the following characteristics: • CPU: 2 x 12 AMD • Opteron(TM) Processor 6238 • RAM: 80GB

Preprocessing Time (VM)

N. Permutations: 500

N. Releases: 34

N. Cutoff (i.e. percentile): 10

N. Days: 8

Total Preprocessing Time: 11928 [min]

Average Time per permutation: 23 [min]

ML Techniques Tested

ML techniques	ML techniques
AdaBoost	Dl4jMlpClassifier
J48	Naïve Baise
Bagging	MultiClassClassifier
LMT	Logistic
Random Forest	SMO
LogitBoost	Multilayer Perceptron

- Frameworks: Weka, R, scikit-learn, Theano