

# Software Defect Prediction on Unlabelled Dataset using Machine Learning



Elisabetta Ronchieri, INFN CNAF

June 5, 2019



**Workshop della  
Commissione Calcolo e  
Reti dell'I.N.F.N.  
Hotel Hermitage  
La Biodola Isola d'Elba  
3 - 7 giugno 2019**

Background

Experimental Settings

Results

Discussion and Future Work

# Background

---



Machine Learning (ML) may help in various SE tasks, such as **software defects prediction and estimation** and **test code generation**.

To accomplish these tasks, **data** have to be **collected** and **properly preprocessed** before the application of machine learning techniques. These activities are **essential to manage missing values** and **inconsistencies amongst data**.

Datasets are composed of **instances** and **features** used to build learning models with Machine Learning techniques.

- ▶ *Instances*: modules, such as files, classes and functions;
- ▶ *Features*: software metrics.

**In SE practice, datasets may lack information**, such as defectiveness, mandatory for SL techniques.

**Labeled dataset** are related to software project whose features have been extracted over time, e.g. defect data are included.

New projects or projects with partial historical data may lack some features' data, e.g. defect data are not included.

- ▶ Their datasets are called **unlabelled datasets**.

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

- ▶ The extraction of the complete set of features (defectiveness included) implies effort and time.

Only in the last decade unlabelled datasets have been investigated for analysis and (defect) prediction.

	$Metric_1$	$Metric_2$	$Metric_{...}$	$Metric_M$	$Labels$
$Instance_1$					
$Instance_2$					
$Instance_3$					
$Instance_{...}$					?
$Instance_N$					?

$instance_{Unlabelled}$       ?

$instance_{Buggy-labelled}$

$Metric_{value}$

$instance_{Clean-labelled}$

Prediction models are **trained** with the **labelled instances** and **tested** with the **unlabelled instances**.

An **instance** can be e.g. a file, a class, a function.

**Each cell** contains a **metric value**.

# Example of Unlabelled Dataset for Defect Prediction 7

	$Metric_1$	$Metric_2$	$Metric_{...}$	$Metric_M$	Labels
$instance_1$					?
$instance_2$					?
$instance_3$					?
$instance_{...}$					?
$instance_N$					?

$instance_{Unlabelled}$ 

				?
--	--	--	--	---

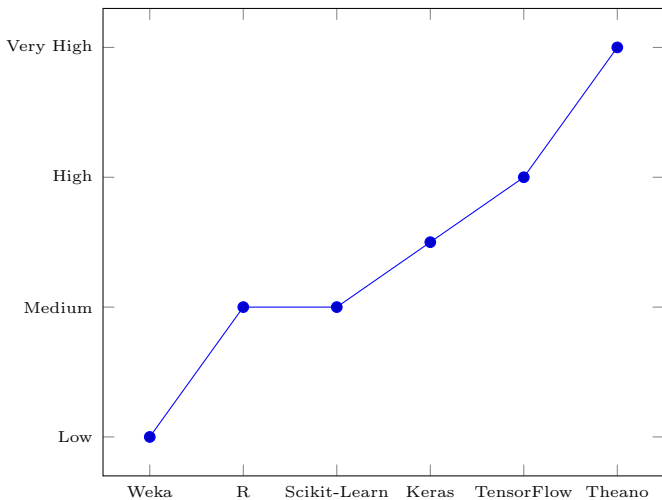
$Metric_{value}$

To build a prediction model, different approaches are available.

Approach	Limitation
Cross-project defect prediction	uses specific data from other projects
Expert-based defect prediction	always requires human experts
Threshold-based defect prediction	needs to decide metrics thresholds in advance
<b>Clustering, LAbeling, Metric selection, Instance selection</b>	<b>claims to be independent on thresholds</b>
Transfer defect learning	needs similar feature distributions of the source and target projects

- ▶ Machine learning techniques can be used as alternative methods to traditional statistics ones for the assessment of software characteristics.
- ▶ Employing these techniques on unlabelled datasets entails a high number of permutations to perform prediction analysis, and involves resource and time consumption on average systems and platforms.
- ▶ Therefore, we assess their application by using Cloud computing infrastructure, GPU-equipped resources and adequate ML framework in order to build software defect prediction model within a reasonable computation time.
- ▶ The analysis has been performed on the full software stack of Geant4 by considering its unlabelled software metrics datasets and existing documentation.





# Experimental Settings

---



## . Input:

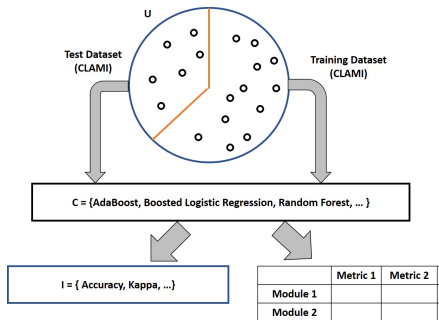
- ▶  $U$  = set of unlabelled instances
- ▶  $C$  = set of machine learning techniques

## . Process:

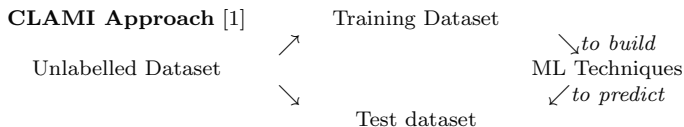
1. Repeat 2-5 N times for each  $u \in U$  to conduct M predictions
2. Randomly split dataset in training (67%) dataset (with labelled defective instances) and test (33%) dataset
3. Construct classifier by applying  $c \in C$  to training dataset
4. Assess classifier
5. Predict test dataset

## . Output:

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



- ▶ Instances predicted are then checked with existing documentation.
- ▶ Those that are still unknown are then checked with developers.



[1] J. Nam, S. Kim, CLAMI: Defect Prediction on Unlabeled Datasets, In Proc. 30th IEEE/ACM International Conference on Automated Software Engineering

To generate TRaining (TR) dataset:	To generate Test (T) dataset:
<b>TR1 clustering instances</b> <b>TR2 labelling instances in clusters</b> <b>TR3 selecting metrics</b> TR4 selecting instances (not applicable for this case) TR1,TR2 allow to label all instances TR3,TR4 allow to remove noisy metrics and instances	<b>T selecting metrics</b>  T dataset has the same set of metrics specified in the Training dataset

I = instance index, J = metric index

<i>inst.</i>	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$	$m_6$
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	?

Example of cutoff threshold is  $\text{Median}_j$   
(cutoff threshold for each  $m_j$ )

<i>Median</i>	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$
	11	10	15	8	8

Yellow cell $_{i,j}$  = j-th metric value of i-th instance greater than  $\text{Median}_j$

[2] 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.

$K$  = Number of metrics for each instance whose values are greater than the median for each metric

<i>instances</i>	$K$
A	$K = 1$
D	$K = 3$
E	$K = 2$
F	$K = 0$
G	$K = 1$
H	$K = 3$
I	$K = 3$

$\text{Cluster}_z$  = group of instances with  $K=z$  identified by different colours [2]

Clusters divided into 2 groups:

1. Clean for  $K \in \{0,1,2\}$  (a bottom half)
2. Buggy for  $K=3$  (a top half)

**The instances that have larger value on all metrics are more likely to be defective.** [2]

Gray cell $_{i,j}$  = Metric value that violates the defect-proneness tendency [2]:

- ▶ D is Buggy, but  $m_2 = 10$  is not greater than Median $_2$
- ▶ E is Clean, but  $m_1 = 15$  is greater than Median $_1$

<i>inst.</i>	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$	$m_6$
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

<i>Median</i>	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$
	11	10	15	8	8

$MVS_j$  = the ratio between the number of violation in the j-th metric and the number of metric values in the j-th metric

	$m_1$	$m_2$	$m_3$	$m_4$	$m_5$
<i>MVS</i>	$\frac{1}{7}$	$\frac{5}{7}$	$\frac{1}{7}$	$\frac{0}{7}$	$\frac{0}{7}$

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

<i>inst.</i>	$m_4$	$m_5$	$m_6$
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

Each measure can be defined on the basis of the confusion matrix below.

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

**Kappa statistic** is a metric (whose value is  $\in [0,1]$ ) that compares an Observed accuracy with an Expected Accuracy [3].

It determines how much better a classifier is performing over the performance of a classifier that simply guesses at random.

If **Kappa statistic**  $\in [0.81, 0.99]$ , then the value indicates an almost perfect agreement.

**Accuracy** is the percentage of instances correctly classified as either buggy or non-buggy (i.e. clean).

$$\frac{TP+TN}{TP+FP+TN+FN}$$

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

# Results





The experimental Testbed was composed by 2 Machines.

Physical 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

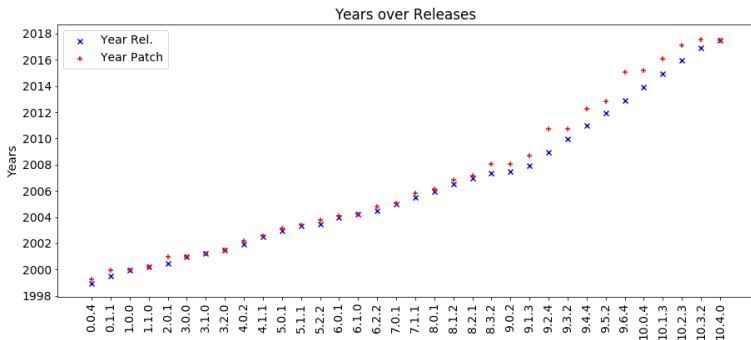
Virtual Machine:

- ◇ 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

## Years over Geant4 software releases



Considered the last patch for each release when available.

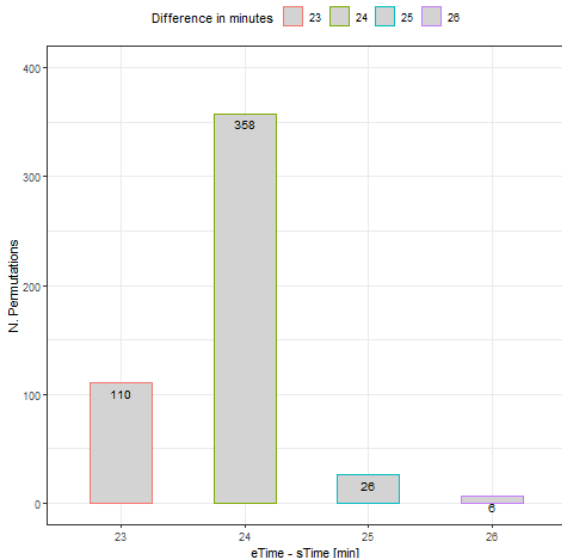
The data belongs to the **Geant4** software.

So far collected data for 34 releases by using Imagix4D tool.

Summarized some information for the major release 10 at **class** level.

For each instance the name of class is reported.

Release	#Instances	Buggy (%)	#Metrics
10.4.0	482	?	66
10.3.2	482	?	66
10.2.3	482	?	66
10.1.3	482	?	66
10.0.4	482	?	66



**N. Permutations:**  
500

N. Releases: 34

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

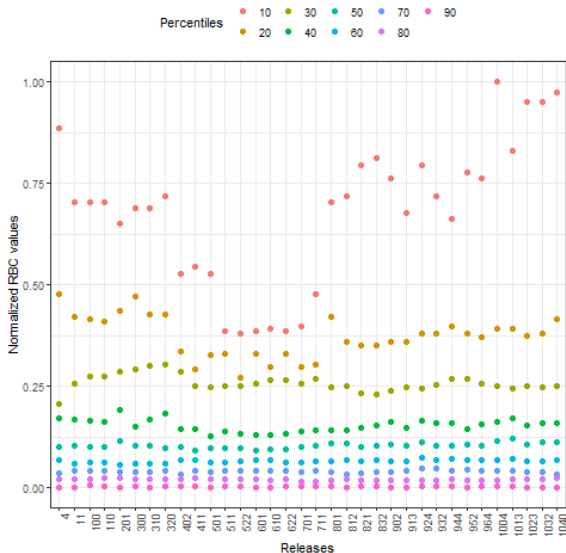
N. Days: 8

Total Preprocessing Time: 11928 [min]

Average Time per permutation: 23.856 [min]

**eTime - sTime:** time requested to build training and test sets per permutation

**Testbed:** Virtual machine on cloud infrastructure



$$RBC = \frac{N.Buggies}{N.Clean}$$

N. Releases: 34

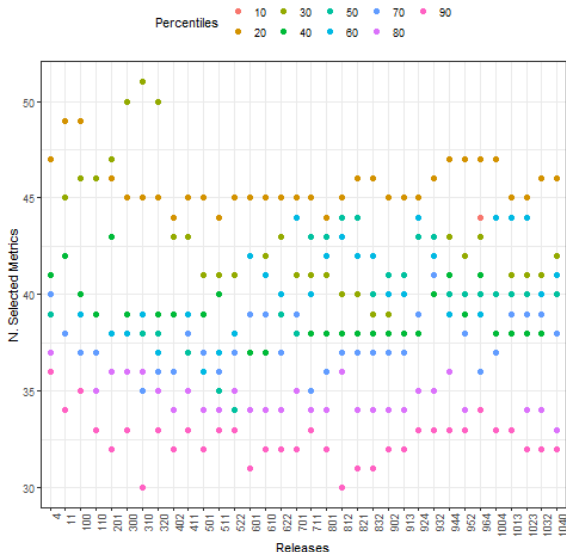
N. Permutations: 500

N. Cutoff 9: percentile at 10, 20, ... , 90

Omitted percentile at 100

N. Training datasets per release: 4500

**The greater RBC, the lower the percentile.**



**N. Selected Metrics:** [45%,77%]

**Average Selected Metrics:** 38 out of 66

N. Releases: 34

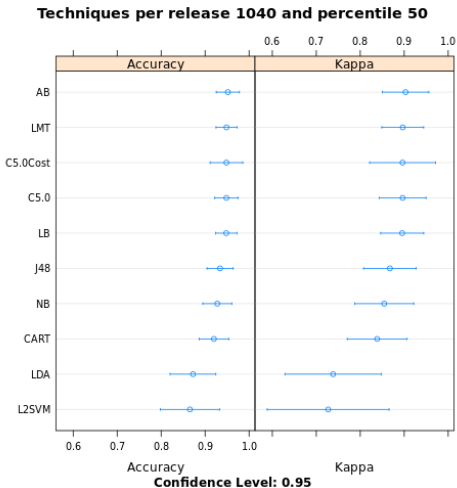
N. Cutoff 9: percentile at 10, 20, ... , 90

Omitted percentile at 100

**Metrics Categories:** size, complexity, maintainability, object orientation

Class information in the dataset.

**The smaller the N. of Selected Metrics, the bigger the percentile.**



N. Release: 10.4.0

N. Percentile: 50

**N. Classification**

**Techniques: 10**

Accuracy Processing  
Time per technique:  
36 [sec]

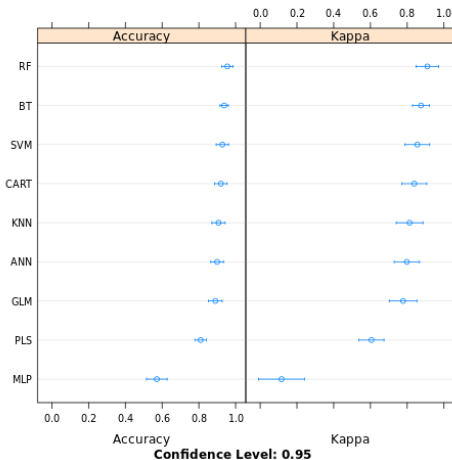
Cross validation: 10

**Best ML Technique:**  
Ada Boost

**Kappa statistic:** al-  
most perfect agreement

**Testbed:** Virtual ma-  
chine on cloud infras-  
tructure

Techniques per release 1040 and percentile 50



N. Release: 10.4.0

N. Percentile: 50

**N. Classification & Regression Techniques: 9**

Accuracy Processing  
Time per technique:  
36 [sec]

Cross validation: 10

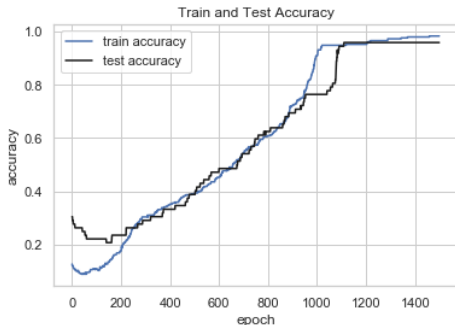
**Best ML Technique:**

Random Forest

**Kappa statistic:** almost perfect agreement

**Testbed:** Virtual machine on cloud infrastructure





N. Release: 10.4

N. Percentile: 50

**N. Classification**

**Techniques:** 10

Accuracy Processing

Time: 3 [sec]

Cross validation: 10

**Technique:** Logistic  
Regression

**Testbed:** physical  
machine with GPU

**Processing Time**  
performance improved by 10 compared to the other machine.

## Discussion and Future Work

---



AdaBoost, Logistic Regression and Random Forest techniques have achieved the best average accuracy.

The effectiveness of this procedure to detect likely defective instances depends on existing software documentation and datasets, such as release notes and software metrics.

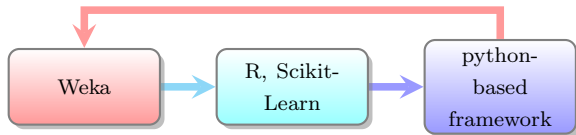
- ▶ So far it is possible to detect pieces of software that require particular attention.
- ▶ Amongst the defective modules predicted, 70% have been verified according to existing documentation.
- ▶ Learning techniques are complementary to existing SE tools and methodology to address SE tasks.

The CLAMI approach enables developers to build a prediction model on unlabelled datasets in an automated manner.

- ▶ Once obtained a labelled dataset, one can employ all the other supervised and semi-supervised techniques to detect defective instances.
- ▶ However, noise in data can make difference in the results.

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.



- ▶ Assessment of the other 30% modules predicted as defective (in collaboration with the Geant4 team)
- ▶ Investigating other statistical tests to detect the best ML techniques (in addition to the Friedman test to rank the ML techniques)
- ▶ Exploration of other software datasets (from WLCG software, coming from HEP-related experiments and middleware) with the same features to apply cross project and transfer learning approaches
- ▶ Adding other tests on GPU-equipped resources
- ▶ Adding other labels (not only clean and buggy)
- ▶ Making available software written by using the various framework (python, R and java code)

*Be curious! Have fun!*

Co-authors:

Marco Canaparo

Davide Salomoni

Acknowledgements:

Doina Cristina Duma

for VM and PM

Daniele Cesini

for GPU

UQ

for dataset

Contact:

[elisabetta.ronchieri@cnafe.infn.it](mailto:elisabetta.ronchieri@cnafe.infn.it)

E. Ronchieri, M. Canaparo, D. Salomini, "Software Defect Prediction on Unlabelled Dataset with Machine Learning Techniques," submitted at IEEE NSS MIC 2019

E. Ronchieri, M. Canaparo, D. Salomoni, "Machine Learning Techniques for Software Analysis of Unlabelled Program Modules," under pub Proc. of ISGC 2019

E. Ronchieri, M. Canaparo, D. C. Duma, A. Costantini, "Data mining techniques for software quality prediction: a comparative study," under pub Proc. of IEEE NSS MIC 2018

M. Canaparo, E. Ronchieri, "Data mining techniques for software quality prediction in open source software: an initial assessment," under pub Proc. of CHEP 2018