Machine learning and reproducibility impact of random numbers

David Hill, Benjamin A. Antunes, Anthony Bertrand, Engelbert Mephu Nguifo, Loïc Yon, Jeanne Nautré-Domanski, Antoine Violaine

MACHINE LEARNING AND REPRODUCIBILITY IMPACT OF RANDOM NUMBERS

David R.C. Hill, Benjamin Antunes, Anthony Bertrand, Engelbert Mephu Nguifo, Loic Yon, Jeanne Nautré-Domanski, Violaine Antoine

Université Clermont Auvergne, Clermont Auvergne INP, ENSM St Etienne, CNRS, LIMOS, F-63000 Clermont–Ferrand, France

david.hill@uca.fr [1]

Abstract :

Reproducibility stands as a pivotal pillar of the scientific method, bolstering confidence in research outcomes. When the findings of a study cannot be consistently reproduced, it gives rise to concerns regarding the validity of the drawn conclusions. This paper delves into an exploration of the Deep Embedded Clustering algorithm. After a first observation of non-reproducibility, we tried to reconstruct the algorithm and we faced repeatability issues, not being able to obtain identical results from run-to-run with the same hardware and exactly the same environment. We have then studied a number of avenues, which could lead to such problems. We were finally able to achieve run-to-run repeatability on an identical machine, thanks to a particular insight into stochastic parameters and a proper use of hidden pseudorandom number generators. However, we observed differences from one machine to another, indicating that portability is not guaranteed leading to more investigation. Nevertheless, with repeatability on the same machine, we can initiate a reproducibility study.

Keywords: Machine learning, Reproducibility, Repeatability, Random numbers.

[1] Contacting author.

1 Introduction

Reproducibility is a crucial element of the scientific method, as Karl Popper, the renowned philosopher of science, points out. Since the initial work of Clarebout and his colleagues in 1992, a paper published in Plos Medicine in 2005 by Ioannidis explaining "how most research findings are false", there is a rise in realizing that there is a scientific reproducibility crisis. A more recent paper published by Ioannidis and his colleagues explains what it means particularly in medicine (Goodman et al. 2016). Gundersen and Kjensmo (2018) give a state of the art dealing with reproducibility issues in artificial intelligence, and Gundersen proposes a broader approach stepping back to the fundamental principles of reproducibility coming from epistemology (Philosophy of Science) in 2021. With the later, we have evidence that the polysemy of the term 'reproducibility' can be confusing as stated in (Plesser 2018). In computer Science, the terms were recently redefined by the ACM in 2020 to match the usage in other Scientific Disciplines. In this paper, we use the definitions of reproducibility and replicability presented in the National Academies of Science Engineering and Medicine report (NASEM 2019). They are consistent with the new ACM definitions (version 1.1) (ARB 2020) which also precise the definition of repeatability which is essential in Computer Science for debugging! Computer Science is also impacting the majority of other research field, indeed, since more than a decade, a reproducibility crisis is identified in many research domains. In the context of research relying on software, researchers are now facing more problems when they try to reproduce the results of computational studies. This crisis has become a significant concern in various scientific disciplines, including computer science (Krishnamurthi and Vitek 2015), and data science (Madduri et al. 2019) which makes an intensive use of machine learning tools and frameworks. A recent survey dealing with the policies of Journals for Software and Data Management in Scientific Publications is proposed by Hernández and Colom (2023).

Reproducibility reinforces confidence in research findings, because if the results of a study cannot be reliably reproduced, this rises concerns about the validity of the drawn conclusions. Scientists rely on reproducibility to establish the credibility of findings and ensure the robustness of scientific knowledge. For computer programmer, we first need repeatability, which is mandatory for software development. To setup a reliable program, we all have a debugging phase and this implies that the developing team is able to obtain the same numerical results from run to run in the same environment. For computing applications, this may require bitwise identical results on the same hardware, this is the required level of repeatability for debugging. A complete and up to date survey dealing with reproducibility and repeatability, with up to date definitions and applications is now available in (Antunes and Hill 2024).

In this paper, we will present how the reproducibility crisis impacts machine learning and what are the sources of non-reproducibility. We will outline a catalog of potential issues commonly encountered

in machine learning studies. We will then focus particularly on mastering random number sources and we will show a small case study where we solved a repeatability and reproducibility problem. This problem occurred with an unsupervised and nonlinear neural network analysis (Deep Embedded Clustering or DEC) that had been carried out in our laboratory during an internship. We will highlight the identified problems with the initial results and present our proposed solution to rectify these issues, ultimately showcasing the obtaining of repeatable results on the same machine enabling to start a reproducibility study.

2 Sources of non-reproducibility in machine learning

2.1 Marching learning and reproducibility issues

Many factors can cause reproducibility issues in machine learning, Henderson et al. (2018) and Gundersen et al. (2022) give for instance overviews of deep reinforcement learning and machine learning issues, respectively. Kapoor and Narayanan recently discussed the reproducibility crisis in machine learning-based science (2023). In the above references, many other papers will cover issues related to reproducibility and machine learning. For example, Pham et al. (2020) and Zhuang et al. (2022) cover sources of variability in deep learning methods, the initialization of pseudorandom numbers generators (PRNGs) is one of them. Multi-threading is another, but there are several others. Only setting what is commonly named 'seeds' and thread parameters will not be enough to make the result of a neural network deterministic. A study by Nagarajan et al. (2019) found that the portability on different kinds of Graphical Processing Units (GPUs) is also influencing the outputs. Though setting the initial states of generators can make the computations deterministic when there is no hidden randomness in functions of the machine-learning framework, the output generated by different GPUs can still differ, and this can be linked to the portability of PRNGs.

2.2 So where should we look?

Reproducibility in machine learning hinges on three fundamental elements inherent to any learning model: data, code and environment. The amalgamation of these essential components constitutes your learning model (not only the code and the selected algorithm). The inclusion of new datasets or the alterations of the data, for instance by applying a preprocessing step (a normalization, etc.) can significantly impact model outcomes. Thus, it is crucial to record dataset versioning and changes to maintain reproducibility.

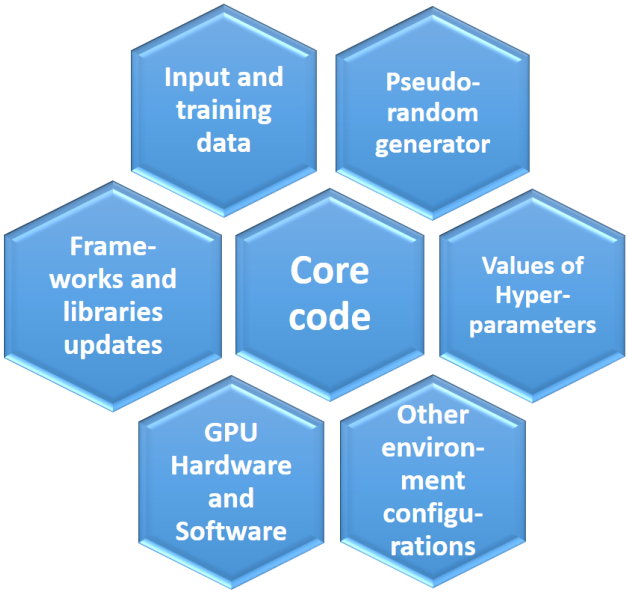

Figure 1: Sources of non-repeatability of run-to-run machine learning experiments

To ensure reproducibility, it is also important to have a meticulous logging and with a proper documentation of the underlying code and algorithm enabling the tracking of modifications with a versioning tool. Moreover, capturing the characteristics of the project environment during the development is essential. This entails documenting framework dependencies, versions, hardware specifications, and all other components of the environment, ensuring that they are well-logged and easily reproducible. Throughout experimentation, there are many critical parameters like the values of hyperparameters, which play a pivotal role and could lead to inconsistencies. The input data is also obviously essential, if there is a change in the training data, reproducing identical results will become impossible. The addition of new training data to the dataset after the publication of the initial results will preclude achieving the same outcome. Furthermore, inaccuracies in data transformations (such as cleaning processes) or changes in data distribution will significantly impact the likelihood of achieving reproducibility. Updates of libraries or frameworks can also introduce alterations to the outcome. The evolving landscape of GPU architectures further complicates reproducibility and needs a tracking of hardware settings, software configurations, or compilers options to keep the research results reproducible. The pseudorandom parameters and generators, the stochastic choices, are also essential elements and are shown below. Figure 1 presents the major sources of non-repeatability for run-to-run machine learning experiments.

2.3 A focus on stochastic aspects

2.3.1 A stochastic context

Machine learning is replete with stochasticity, a modelling approach using randomness. In this context, the mastering of random initializations, the introduction of random noise, random augmentations and

bootstrapping techniques, the selection of hidden layers, data shuffling and dropout are different aspects were scientists need a reasonable level of mastering of random number generation. For instance, dropout prevents the overfitting in neural networks. This occurs when a model learns the training data too well, capturing noise or specific patterns that do not generalize well to new, unseen data. The dropout technique addresses this issue by randomly setting to zero ("dropping out") a proportion of the neurons or units in a neural network during each training iteration, meaning that during training, certain neurons are temporarily removed. This makes the network more robust and with better capacities when facing new data. The dropout technique is often applied to neural network hidden layers and the dropout rate determines the proportion of neurons that are removed during each training iteration.

2.3.2 When non-determinism is in fact deterministic

When using a Quantum computer, we expect true stochasticity and reproducible results (Hill et al. 2023). With classical computers, determinism is the rule and the stochastic aspects are simulated with deterministic algorithms! Pseudorandom numbers are deterministic by design in order to produce stochastic programs with deterministic models of randomness (the pseudorandom number generators). The design enables debugging. Deep learning algorithms often use efficiently the Stochastic Gradient Descent (SGD) or Monte Carlo methods. Non-determinism is commonly obtained with independent replications with the same PRNG with a sound parallelization technique. Each kind of generator comes with some initialization constraints and also has preferable parallelization techniques (Hill et al. 2013). This forms a tuple which enters in resonance with the application (PRNG, Initilalization, Parallelization Technique, Application). A fine setup will help to obtain a better exploration of the hyperspace of hyperparameters. The mastering the underlying technique is precious for a thorough training phase. The case of deep reinforcement learning is also prone to nondeterminism as agents learn from a somewhat nonstationary distribution of experiences influenced by non-deterministic environments and policies. Other elements that can impact the results are the random network initialization or the size used for mini-batch sampling. We will use our expertise in parallel random numbers to identify the problems. When the output varies for the same input across different runs for a nondeterministic algorithms, this presents a significant obstacle to repeatability and debugging, and then to reproducibility. Deep learning algorithms are using such algorithms and their stochastic aspects have to be mastered. When scientists are partially aware that their results depend on the initialization of one or many pseudorandom number generators, and that they do not know the generators they are using, then come the troubles...

3 Reaching software repeatability

3.1 Algorithm and initial context

We study the Deep Embedded Clustering (DEC) algorithm presented in article (Xie et al. 2016). It is an unsupervised learning technique that combines a neural network and a clustering algorithm. The aim of this method is to learn a latent representation (i.e. the innermost hidden representation) from the input data in order to perform a linear clustering algorithm on this latent space. This involved processing medical data from a medical research group, with the aim of grouping patients suffering from chronic pain into coherent groups, across their various symptoms, to enable personalized treatment for each group. While the results obtained initially appeared plausible, they lacked both reproducibility and repeatability.

The training set, which was used in the article and on the software implementation we checked, is the MNIST dataset, containing a set of 70,000 images of handwritten digits. This MNIST dataset is fairly well known, labelled and easy to cluster, allowing us to control the obtained results. The algorithm was implemented in Python using the NumPy library, the frameworks scikit-learn and Keras as a high-level coming with an industry strength Application Programmer Interface (API) of the TensorFlow platform. In order to assess the relevance of the partitions provided by the developed models, two metrics were used. First, a silhouette coefficient measuring the quality of a partition of a dataset based on separability inter-clusters and the compactness of each cluster. The second metric is the Adjusted Rand Index (ARI) which evaluates the similarity between two partitions of the same dataset by measuring the rate of agreement between them.

3.2 Absence of repeatability and reproducibility problems

With the initial code produced during last year internship, we first observed that the partitions obtained using the different models obtained with DEC presented most of the time a silhouette coefficient very close to 1, this was indicating good data separation. However, when we evaluate the ARI of these different partitions against each other, we systematically observe results very close to 0. The partitions were therefore very different from one generated model to the other, without a clear reason. In addition, numerical results were different from run to run in the same conditions. This lack of repeatability was problematic and it was impossible to build on a shifting base. The other aspect we encountered is linked to a lack of reproducibility. The resources of the initial internship study did not allow us to generate models close to what was obtained. Some of the Python libraries used, such as Keras, had been updated, making some lines of code obsolete. In addition, getting back to grips with the project was complex, not least because of a lack of documentation. Here is the link to our Gitlab repository: https://gitlab.isima.fr/jedomanski/machine-learning-repeatability-quest .

3.3 Trying harder

In our try to reproduce reliably the DEC results, various methods were applied to identify whether the problem lay in the parameter settings. Firstly, according to the DEC article, we set: SGD as optimizer, network dimensions d-500-500-2000-10, with 'd' the dimension of the starting base, each layer was pre-trained during 50000 iterations, with a dropout rate of 20%. The autoencoding is then refined over 100,000 iterations without dropout. The batch size was 256 and the learning rate was set to 0.1, then divided by 10 every 20,000 iterations, and the weight decay method was set to 0. We experimented with different mini-batch sizes, going from the original 256 down to just 32, then we tried loading weights from previously trained models, in order to identify whether the inconsistency between the resulting partitions was due to a learning problem.

Another approach was to use the SGD minimization function. On the one hand, we studied the results of this function using box plots, and on the other, we tried training with a non-stochastic gradient descent (the plain Gradient Descent), in an attempt to limit the sources of randomness introduced by the SGD. A versioning problem identified earlier has also been studied. Due to the Keras library update, lines of code using the optimizer had become obsolete. To keep the algorithm running, replacements had to be made, with as few modifications as possible. A neural network model analysis tool, "netron.app" was used to identify the initial version used in our project, it was version v2.11.0, while the latest version we are now using is v2.13.1. All the tries were still producing repeatability issues.

3.4 Targeting random sources

The code was not making explicit calls to random sources but we were aware of at least 3 potentially hidden sources. The generator used in python is the Mersenne Twister, the default in NumPy is PCG (Permuted Congruential Generator) and the one used in TensorFlow is by default Philox (a crypto secure generator introduced in 2011 at the Supercompting conference). The original code mixes all three generators in its 'black box' and did not specify initial states for the 3 pseudorandom number generators with common seeding Application Programmer Interface. We then compared models generated with the same seed on the same machine for the 3 different generators, and then we made the test on a different server, always keeping the same seed (though the generators do not have the same internal structures). We studied the 8 possible combinations, with or without seed initialization for the 3 generators and observed the repeatability results (Table 1).

With our "run-to-run" comparisons, we identified two combinations leading to repeatable results on the same machine, enabling us to start a reproducibility study even if we don't find the same results on different machines. This point will be investigated in a further study as well as the impact of seed initialization.

Table 1: Exploring different initializations with the 3 pseudorandom number generators found in the software

Mersenne Twister	PCG	Philox	Results
Not initialized	Not initialized	Not initialized	Non repeatable results
Not initialized	Not initialized	Initialized	Non repeatable results
Not initialized	Initialized	Not initialized	Non repeatable results
Not initialized	Initialized	Initialized	Non repeatable results
Initialized	Not initialized	Not initialized	Non repeatable results
Initialized	Not initialized	Initialized	**Repeatable results**
Initialized	Initialized	Not initialized	Non repeatable results
Initialized	Initialized	Initialized	**Repeatable results**

3.5 When seeds are not seeds

The term 'seed' is found confusing by experts. It comes from old generators such as Linear Congruential Generators (LCGs). A simple integer number served as the initial state of a generator, and it was called a seed. Such generators are statistically weak and cannot be considered seriously for scientific applications. Modern generators with strong statistical properties for scientific applications have initial states, sometimes called statuses, much larger that a single integer (be it 64 bits). For instance the MRG32k3a pseudorandom number generator from Pierre L'Ecuyer has an initial status with 6 double precision numbers (6 x 8 bytes = 6 x 8 x 8 bits = 384 bits). Statistically it is a very sound generator, designed to be successful at the most stringent battery of statistical tests, TestU01 from L'Ecuyer and Simard (2006). The famous Mersenne Twister (MT) from Makoto Matsumoto and Nishimura Takuji (1998) needs an initialization state (or status) around of 2 kilobytes (more than 2048 bits). It presents some statistical weaknesses (not crypto secure) but it is 20 times faster than MRG32k3a on modern processors and it is equidistributed in 623 dimensions. The latter property is very important for space filling problems such as exploring the impact of models with a large number of parameters. Recent generators like PCG (the default Numpy generator) is recognized as weak for parallel codes. An extension has been proposed: PCG64DXSM. Vigna has recently shown that both are weak. There are slower than other older known PRNGs (URL: https://pcg.di.unimi.it/pcg.php). Finally, when a seeding API (Application Programming Interface) proposes a seeding function with

an integer parameter, scientist have to be aware that this integer is not the PRNG state, there is no bijection between a 64 bits integer and the different states that the initial MT can propose (2^{19937}). Many scientists who did not receive training with pseudorandom numbers are not aware of this. In addition, many calls to random sources are hidden in machine learning frameworks, some calls implying parallel executions. The techniques used to propose proper parallel stochastic codes are not well known and this increases the challenge to obtain repeatable machine learning experiences. A last point will show at which extremity we can go in scientific computing: it is not uncommon in many source codes and even in reputable Parallel Computing literature, to find the misguided advice to initialize your PRNG with "time(NULL)" for "true randomness". Many generators are sensitive to their initialization states, and it should be learned in computer science lectures that this practice is unsuitable for scientific purposes. To ensure the repeatability PRNGs, it is crucial to meticulously manage and save the initial states and to use the parallelization technique adapted to your generator structure. Choosing an appropriate of parallelization technique before running parallel code with pseudorandom numbers should be strongly considered (Hill et al., 2013). In the previously cited papers, the techniques explained are not complex but are not so commonly known to many scientists who run parallel stochastic codes without being aware of it. The usage of complex framework is hiding this aspect. This could be a very good point if such parallelization is done properly, but it is mostly not the case since the framework software developers are not trained to produce independent parallel stochastic computing. Proposed solutions are opaque and do not help to make progress in artificial intelligence explainability.

4 Conclusion

Scientists rely on reproducibility to establish the credibility of findings and ensure the robustness of scientific knowledge. In the realm of machine learning programs, we found intricate challenges that often render the reproduction of results from scientific papers seemingly almost impossible. It was our case when we tried to study an unsupervised and nonlinear neural network analysis known as Deep Embedded Clustering. We then started to learn about the different sources of non-reproducibility in machine learning and we gave an overview of the potential sources. With our first experiments with the MNIST dataset, we even identified a repeatability problem linked to the sources of randomness hidden in the language, the libraries and the framework used. We reached run-to-run repeatability with a careful initialization of the Python and Keras default pseudorandom generators. Machine learning frameworks come with a seeding API, which often causes confusion because the seeds provided by programmers are not the states of the generators but merely an index to some of them. Additionally, calls to random sources are sometimes hidden within API functions, and the parallelization of the default generators (which are sometimes statistically weak) is also unclear. Even if we could reach the

repeatability "grail" on the same machine, portability from one server to another remains an issue, prompting further investigation. However, with repeatability on the same server, we can at least begin to focus on achieving reproducibility.

References

Antunes B., Hill D.R.C. 2024. "Reproducibility, Replicability and Repeatability: A survey of reproducible research with a focus on high performance computing", Computer Science Review, 53, 28 p.

Claerbout, J. F., & Karrenbach, M. 1992. "Electronic documents give reproducible research a new meaning". In SEG technical program expanded abstracts, Society of Exploration Geophysicists, pp. 601-604.

Goodman S. N., Fanelli D. & Ioannidis J. P. 2016. "What does research reproducibility mean?" Science translational medicine, 8(341), 341ps12-341ps12.

Gundersen, O.E., Kjensmo, S. 2018. "State of the art: Reproducibility in artificial intelligence". In : Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32. n°1, pp. 1644-1651.

Gundersen, O. E. 2021. "The fundamental principles of reproducibility". Philosophical Transactions of the Royal Society A, 379(2197), 20200210.

Gundersen, O. E., Coakley, K., Kirkpatrick, C., & Gil, Y. 2022. "Sources of irreproducibility in machine learning: A review". arXiv preprint arXiv:2204.07610.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. 2018. "Deep reinforcement learning that matters". In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1).

Hill D.R.C., Passerat-Palmbach J., Mazel C. and Traore, M.K. 2013. "Distribution of Random Streams for Simulation Practitioners", Concurrency and Computation: Practice and Experience, Vol. 25, Issue 10, pp. 1427-1442.

David R.C. Hill, Benjamin A. Antunes, Thomas Cluzel, Claude Mazel. A few words about quantum computing, epistemology, repeatability and reproducibility. EU/MEeting 2023, Apr 2023, Troyes (France), France. <hal-04089148>, 5 p.

Kapoor, S., & Narayanan, A. 2023. "Leakage and the reproducibility crisis in machine-learning-based science". Patterns, 4(9).

Nagarajan P., Warnell G., and Stone Peter. 2019. "The Impact of Nondeterminism on Reproducibility in Deep Reinforcement Learning". Presented at the AAAI 2019 Workshop on Reproducible AI, Honolulu, Hawaii (2019).

Plesser, H. E. 2018. "Reproducibility vs. replicability: a brief history of a confused terminology". Frontiers in neuroinformatics, vol. 11, p. 76.

Shriram Krishnamurthi and Jan Vitek. 2015. "The real software crisis: Repeatability as a core value". Communications of the ACM, vol. 58, n°3, p. 34-36.

Ravi Madduri, Kyle Chard, Mike d'Arcy, *et al.* 2019. "Reproducible big data science: A case study in continuous FAIRness". *PloS one*, vol. 14, no 4, p. e0213013.

Xie Junyuan, Ross Girshick And Ali Farhadi, 2016, "Unsupervised deep embedding for clustering analysis". In: International conference on machine learning. PMLR. pp. 478-487.

Zhuang, D., Zhang, X., Song, S., & Hooker, S. 2022. "Randomness in neural network training: Characterizing the impact of tooling". Proceedings of Machine Learning and Systems, vol 4, pp. 316-336.

Web references

ARB. 2020. "Artifact Review and Badging Version 1.1, August 24, 2020", Retrieved July 8th, 2024, from: https://www.acm.org/publications/ policies/artifact-review-and-badging-current

NASEM. 2019. "Reproducibility and Replicability in Science, National Academies of Science Engineering and Medicine". Retrieved January 24rd, 2024, from: https://nap.nationalacademies.org/catalog/25303/reproducibility-and-replicability-in-science

Neptune AI. 2023. "How to Solve Reproducibility in ML". MLOps blog. Retrieved January 23rd, 2024 from: https://neptune.ai/blog/how-to-solve-reproducibility-in-ml

BIOGRAPHIES

Violaine Antoine is an associate professor at Clermont Auvergne University, in the LIMOS laboratory (UMR CNRS 6158). She earned her PdD in 2011 in the University of Compiègne and her Research Director Habilitation in 2023 in the Clermont Auvergne University. Her research interest is focused on data mining and machine learning.

Benjamin Antunes is a Phd Student at Clermont Auvergne University (UCA) in the LIMOS laboratory (UMR CNRS 6158). He holds a Master in Computer Science (head of the list). His thesis subject is about the reproducibility of numerical results in the context of high performance computing. He is especially working on reproducibility issues in parallel stochastic computing. His email address is benjamin.antunes@uca.fr and his homepage is https://perso.isima.fr/~beantunes/

Anthony Bertrand is a Phd Student at Clermont Auvergne University (UCA) in the LIMOS laboratory (UMR CNRS 6158). He holds a Master in Computer Science (second head of the list). His thesis subject is about the software-based measurement of energy consumption of Machine Learning programs in High Performance Computing (HPC). He also adresses the reproducibility challenges in the domain of Machine Learning. His email address is anthony.bertrand@uca.fr.

David R. C. Hill is a full professor of Computer Science at University Clermont Auvergne (UCA) doing his research at the French Centre for National Research (CNRS) in the LIMOS laboratory (UMR 6158). He earned his Ph.D. in 1993 and Research Director Habilitation in 2000 both from Blaise Pascal University and later became Vice President of this University (2008-2012). He is also past director of a French Regional Computing Center (CRRI) (2008-2010) and was appointed two times deputy director of the ISIMA Engineering Institute of Computer Science – part of Clermont Auvergne INP, #1 Technology Hub in Central France (2005-2007 ; 2018-2021). He is now Director of an international

graduate track at Clermont Auvergne INP. Prof Hill has authored or co-authored more than 280 papers and has also published several scientific books. He recently supervised research at CERN in High Performance Computing (https://isima.fr/~hill/)

Jeanne Nautré-Domanski is an engineering student at ISIMA, a French "Grande École d'Ingénieur". During an internship at the LIMOS laboratory (UMR CNRS 6158), she dealt with the reproducibility issues of a deep embedded clustering application. Her email address is: jeanne.domanski@etu.uca.fr

Engelbert M. Nguifo is a full professor of computer science at University Clermont Auvergne (UCA), France, where he is the director of Master Degree Program in Computer Science. He is leading research on machine learning and data mining for complex data in the joined University-CNRS laboratory LIMOS where he is co-chair of the Information and Communication Systems research group. His research interests also include formal concept analysis, artificial intelligence, pattern recognition, bioinformatics, big data, and knowledge representation. He was Board member of the French Association on Artificial Intelligence. He is member of the executive board of the French CNRS research group on Artificial Intelligence (GDR RADIA).

Loïc Yon is an associate professor at the ISIMA Engineering Institute of Computer Science – part of Clermont Auvergne INP and is in charge of professional training? He is head of « SOFTWARE ENGINEERING AND COMPUTER SYSTEMS » department at ISIMA and does his research at LIMOS laboratory (CNRS UMR 6158). His prime research interest concerned Operation Research and slides to high performance computing (https://perso.isima.fr/loic)

Roles:

Antoine Violaine: Project Management, Formal analysis, Writing – Review & Editing, Supervision.
Antunes Benjamin: Software Investigation, Writing – Review & Editing.
Bertrand Anthony: Writing – Review & Editing, Submission administration.
Hill David R. C.: Writing – Original Draft, Formal Analysis, Software Investigation, Supervision
Nautré-Domanski Jeanne: Software, Formal analysis, Writing – Review & Editing.
Nguifo Engelbert M.: Writing – Review & Editing, Supervision.
Yon Loïc: Writing – Review & Editing.