GigaScience

Experimenting with Reproducibility: a case study of Robustness in Bioinformatics

--Manuscript Draft--

Manuscript Number:	GIGA-D-17-00317R2
Full Title:	Experimenting with Reproducibility: a case study of Robustness in Bioinformatics
Article Type:	Review
Funding Information:	Institut Pasteur (FR) — Not applicable H2020 Health — Not applicable Institut National des Sciences de l'Univers, Centre National de la Recherche Scientifique (FR) — Not applicable Université Paris Diderot (FR) — Not applicable Conny-Maeva Charitable Foundation — Not applicable Cognacq-Jay Foundation — Not applicable Orange — Not applicable Fondation pour la Recherche Médicale — Not applicable GenMed Labex — Not applicable BioPsy Labex — Not applicable
Abstract:	Reproducibility has been shown to be limited in many scientific fields. This question is a fundamental tenet of scientific activity, but the related issues of reusability of scientific data are poorly documented. Here, we present a case study of our difficulties to reproduce a published bioinformatics method even though code and data were available. First, we tried to re-run the analysis with the code and data provided by the authors. Second, we reimplemented the whole method in a Python package to avoid dependency on a MATLAB license and ease the execution of the code on a HPCC (High-Performance Computing Cluster). Third, we assessed reusability of our reimplementation and the quality of our documentation, testing how easy it would be to start from our implementation to reproduce the results. In a second section, we propose solutions from this case study and other observations to improve reproducibility and research efficiency at the individual and collective level. While finalizing our code, we created case specific documentation and tutorials for the associated Python package StratiPy. Readers are thus invited to experiment with our reproducibility case study by generating the two confusion matrices of Fig 3 (see more in 2.2.2). Here we decided to propose two options: 1) a step-by-step process to follow in a Jupyter/IPython notebook; or 2) a Docker container ready to be built and run. Availability: the latest version of StratiPy (Python) with two examples of reproducibility and dataset are available via GitHub https://github.com/GHFC/Stratipy and archived in Zenodo.
Corresponding Author:	Yang-Min KIM Institut Pasteur Paris, Île-de-France FRANCE
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Institut Pasteur
Corresponding Author's Secondary Institution:	
First Author:	Yang-Min KIM, M.Sc.
First Author Secondary Information:	

Order of Authors:	Yang-Min KIM, M.Sc.
	Jean-Baptiste Poline, PhD
	Guillaume Dumas, PhD
Order of Authors Secondary Information:	
Response to Reviewers:	Dear Editor, We answered point-by-point all the insightful reviewer comments, with a specific focus on clarification in the text. We hope the revised manuscript conforms to the journal standards. Best regards, Yang-Min KIM, on the behalf of all the authors ************** [Reviewer comments] ************** !! Conversion from word to PDF has changed some section numbers in the old version but we keep the correct numbers in this response. -- -- Reviewer #1 ************** (1)"2.2 Robustness: from MATLAB to Python" and "2.3 Reproducibility of Robustness: from Python to Python" seem have a little overlap, because in Background they were parallel concept. Please reorganize the contents in these two parts. ************** Since 2.2 is about robustness (change in code) and 2.3 is about re-running the python code on different platforms, it seems to us that this two sections can be kept separate, but the description of figure 2 has been moved from the end of paragraph 2.2.1 (Metadata and File formats, page 4, line 86) to the end of paragraph 2.2.3 (Documentation and examples, page 5, line 115) to conclude the robustness section. We also agree that the subheadings were not entirely consistent and reorganize some of the text (see below). ************** (2) The subtitles need to be more logical, for example, 2.2.1 Metadata and File formats, and 2.2.2 Codes and parameters, but 2.2.3 Jupyter/IPython. The first two subtitles are describing the effect of data, format and code, parameter on the robustness, but the "Jupyter/IPython" is not a parallel concept with the first two subtitles on the robustness, they are only platform or shell environment. Please well design the subtitles or this subsection "Jupyter/IPython" can be integrated into the 2.2.2, in brief, please make it more logical for reading. In fact, in this paper, similar problems exist several places. ************** We worked to make the logic of the text easier to follow and more consistent. In particular, we renamed several subsections like the “jupyter” subsection 2.2.3 into “Documentation and examples” (page 5, line 115), "environment" subsection 3.1.1 into "Publish software and their environment" (page 7, line 175) and "metadata" subsection 3.1.2 into "Document with appropriate Metadata" (page 7, line 187). We also split subsection “2.3 Reproducibility of Robustness: from Python to Python” into two parts: “2.3 Collaborative coding and best practices” (page 5, line 132) and “2.4 Reproducibility of Robustness: from Python to Python” (page 6, line 143). ************** (3) Please pay attention to the first sentence of a paragraph, it should give the main spirit of the paragraph instead of just starting a new talking. For example,

"Once the environment, file format and data issues were resolved, the code was finally executed"… For another example, "Given the observed difficulties, in this section we draw some conclusions on this reproducibility case study experiment and suggest some tools and best practices.", why always "some conclusion"? why cannot directly summarize the conclusion here? Another example, "3.1.1 Environment In 1995, Buckheit and Donoho were already thinking about reproducible research in computer science", this is a composition or fiction genre instead of a scientific paper.
(4) After rewriting all first sentences for each paragraphs, please reorganize the content of their following sentences referring to other published scientific articles.
****

We rewrote the first sentences of the paragraph across the paper.

Page 4, line 100, paragraph 2.2.2 Codes and parameters: we changed the sentence "Once the environment, file format and data issues were resolved, the code was finally executed" into "Beyond documentation and file formats, code initialization and parameters settings are also key for reproducibility."

Page 5, line 133, paragraph 2.3 Collaborative code development and best practices: “Throughout the project we used the version control system (VCS) Git to document the development of our Python package.”

Page 6, line 144, paragraph 2.4 Reproducibility of Robustness: from Python to Python: “Knowing how difficult it can be to re-run someone else’s code, we then attempted to start the analysis from scratch and to reproduce the results on another platform from our newly developed python package.”

Page 7, line 172, paragraph 3.1 Act locally: simple practices and available tools: we replaced the sentence "Given the observed difficulties, in this section we draw some conclusions on this reproducibility case study experiment and suggest some tools and best practices." by “We conclude from this reproducibility case study experiment by suggesting tools and best practices following the programming best practices”.

Page 7, line 176, paragraph 3.1.1 Publish software and their environment: regarding the sentence about Buckheit and Donoho, we totally rewrote it as follow: "Increased reproducibility and replicability can be obtained by following Buckheit and Donoho’s long standing motto: “When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures” by offering a complete and free package (WaveLab) to reproduce the published output [30]."

Page 7, line 195, paragraph 3.1.3 Write readable code: we changed the sentence “Anyone who has spent time to understand someone else’s code would advise some simple basic rules to help make the code readable and understandable.” into “We draw some conclusion from our experience in working with others code.”. We then follow the reviewer advice to directly summarize the conclusion.

************** (5) Based on the size of core content of this article, please cut it down.
****

We removed some of the text to make it more dense.. Especially in the conclusion, we adopt a straightforward bullet-point list of key messages and recommandations:

Page 11, line 297: "To summarize, our experiment at reproducing initial results led to the following conclusions and recommendations:
- Improve life scientists software development skills
- Use online repositories and tools to help other scientists in their exploration of the method [26,27,31]
- Enhance the cooperation between academic education and industry [40,41,47]
- Develop an open source continuous testing ecosystem with community standards, well-identified datasets to validate tools across versions and datasets, and go beyond the publication of a PDF file”

	In total we have reduced 217 words. -- -- Reviewer #2 ************** The authors have successfully responded to my comments. I congratulate the authors for a simple and nice paper. ************** We thank reviewer 2 for all his helpful comments and interest in our paper.
Additional Information:	
Question	**Response**
Are you submitting this manuscript to a special series or article collection?	No
Experimental design and statistics Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our Minimum Standards Reporting Checklist. Information essential to interpreting the data presented should be made available in the figure legends. Have you included all the information requested in your manuscript?	No
If not, please give reasons for any omissions below. as follow-up to "**Experimental design and statistics** Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our Minimum Standards Reporting Checklist. Information essential to interpreting the data presented should be made available in the figure legends. Have you included all the information requested in your manuscript? "	This review is especially based on the reproducibility issues. The illustrated method is directly based on the original paper by Hofree et al, 2013. We nevertheless described any specific variations.
Resources A description of all resources used, including antibodies, cell lines, animals and software tools, with enough	Yes

information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite Research Resource Identifiers (RRIDs) for antibodies, model organisms and tools, where possible. Have you included the information requested as detailed in our Minimum Standards Reporting Checklist?	
Availability of data and materials All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using a unique identifier in the references and in the "Availability of Data and Materials" section of your manuscript. Have you have met the above requirement as detailed in our Minimum Standards Reporting Checklist?	Yes

Experimenting with Reproducibility: a case study of Robustness in Bioinformatics

Authors

Yang-Min KIM [1,2,3,4,*] , Jean-Baptiste Poline [5,6] , Guillaume Dumas [1,2,3,4]

[1]Institut Pasteur, Human Genetics and Cognitive Functions Unit, Paris, France, [2]CNRS UMR 3571 Genes, Synapses and Cognition, Institut Pasteur, Paris, France, [3]University Paris Diderot, Sorbonne Paris Cité, Paris, France, [4]Centre de Bioinformatique, Biostatistique et Biologie Intégrative (C3BI, USR 3756 Institut Pasteur and CNRS), Paris, France, [5]Montreal Neurological Institute, Brain Imaging Center, Ludmer Center, McGill University, [6]Henry H. Wheeler Jr. Brain Imaging Center, Helen Wills Neuroscience Institute, University of California, Berkeley, California, USA

*To whom correspondence should be addressed

Correspondence: yang-min.kim@pasteur.fr, ORCID: 0000-0002-1583-3297; jbpoline@gmail.com; ORCID: 0000-0002-9794-749X; guillaume.dumas@pasteur.fr, ORCID: 0000-0002-2253-1844

Abstract

Reproducibility has been shown to be limited in many scientific fields. This question is a fundamental tenet of scientific activity, but the related issues of reusability of scientific data are poorly documented. Here, we present a case study of our difficulties to reproduce a published bioinformatics method even though code and data were available. First, we tried to re-run the analysis with the code and data provided by the authors. Second, we reimplemented the whole method in a Python package to avoid dependency on a MATLAB license and ease the execution of the code on a HPCC (High-Performance Computing Cluster). Third, we assessed reusability of our reimplementation and the quality of our documentation, testing how easy it would be to start from our implementation to reproduce the results. In a second section, we propose solutions from this case study and other observations to improve reproducibility and research efficiency at the individual and collective level.

While finalizing our code, we created case specific documentation and tutorials for the associated Python package *StratiPy*. Readers are thus invited to experiment with our reproducibility case study by generating the two confusion matrices of Fig 3 (see more in 2.2.2).

Here we decided to propose two options: 1) a step-by-step process to follow in a Jupyter/IPython notebook; or 2) a Docker container ready to be built and run.

Availability: the latest version of StratiPy (Python) with two examples of reproducibility and dataset are available via GitHub https://github.com/GHFC/Stratipy and archived in Zenodo.

Keywords

- Reproducibility; Robustness; Reusability; Network Based Stratification (NBS); Standard consensus dataset; Cancer

1 Background

The collective endeavor of science depends on researchers being able to replicate the work of others. In a recent survey of 1,576 researchers, 70% of them admitted having difficulty in reproducing experiments proposed by other scientists [1]. For 50%, this reproducibility issue even concerns their own experiments. Despite the growing attention on the replication crisis in science [2,3], this controversial subject is far from being new: already in the 17th century, scientists criticized the air pump invented by physicist Robert Boyle because it was too complicated and expensive to build [4].

Several concepts for reproducibility in computational science are closely associated [5,6]. Here we define them as mentioned by K. Whitaker [6]: obtaining the same results using same data and same code is ***Reproducibility***; if code is different, it is ***Robustness***. If we used different data but with the same code, it is ***Replicability***. Lastly, using different data and different code is referred as ***Generalizability***. Here we will primarily elaborate on ***Reproducibility*** and ***Robustness***, and acknowledge that new datasets or hardware environment introduce additional hurdles [7]. Reproducibility is a key first step, for instance, among the 400 algorithms published during the major artificial intelligence conferences, only 6% offered the code [8]. Even when authors provide data and code, the outcome can vary either marginally or fundamentally [9]. Tackling irreproducibility in bioinformatics thus requires considerable effort beyond code and data availability, an effort that is still poorly recognized in the current publication based research community. In most cases, there is a significant gap between apparent executable work (Fig 1 - i.e. above water portion of iceberg) and necessary effort in practice (Fig 1 - i.e. full iceberg). Such effort is necessary to increase the consistency of the literature and *efficiency* of the scientific research process by making research products easily reusable.

2 Reproducibility and Robustness in bioinformatics: a case study

2.1 Reproducibility: from MATLAB to MATLAB, OS and environment

Our team studies Autism Spectrum Disorders (ASD), a group of neurodevelopmental disorders well known for its heterogeneity. One of the current challenges of our research is to uncover homogeneous subgroups of patients (i.e. stratification) with more precise clinical outcomes, improving their prognosis and treatment [10,11]. An interesting stratification method was recently proposed in the field of cancer research [12], where the authors proposed to combine genetic profiles of patients' tumors with protein-protein interaction networks to uncover meaningful homogeneous subgroups, a method called Network Based Stratification (NBS).

Before using NBS method on our data, we studied the method by reproducing results from the original study. We are very grateful to the main authors who kindly provided online all the related data and code, and gave us invaluable input upon request. The authors of this study did much more to help reproduce their results than is generally done. Despite their help we experienced a number of difficulties that we document here, hoping that this report will help future researchers to improve the reproducibility of results and reusability of research products.

The first step of our project was to execute the original method code with the given data: reproducibility (Table 1). To improve execution speed, the original authors used a library for MATLAB on a Linux platform, using executable compiled code MEX file [13]: MTIMESX 14], a library allowing acceleration of large matrix multiplication. MEX files however are specific to the architecture and have to be recompiled for each Operating System (OS). Since our lab was using Mac OS X Sierra, the compilation of this MEX file into a mac64 binary required a new version of MTIMESX. It was also necessary to install and to configure properly OpenMP [15], a development library for parallel computing. After this, the original MATLAB code was successfully run in our environment.

	Code	Data	Technical issues	Other issues
Reproducibility	Same: MATLAB	Same	**OS**: MEX file specificity linked to OS (e.g. Linux → OSX)	
Robustness	MATLAB → Python	Same	**File format**: we can load sparse matrices from *.mat* file but cannot save them into HDF5 using h5py package **Default parameters**: linkage method use for the hierarchical clustering • MATLAB (MathWorks): UPGMA (average) • Python (SciPy): single	• Metadata structure • Important parameter value not explained in the original paper • Remaining discarded work ('code ruins') and traces of debugging
Reproducibility of Robustness	Same: Python	Same	**OS:** Numpy package and BLAS library compiled for specific OS (e.g. OSX → Linux)	Documentation

Table 1: Technical problems encountered during our reproducibility and robustness case study.

These issues are classic but may not be overcome by researchers with little experience in compilation or installation issues. For these reasons alone, many individuals may turn down the opportunity of reusing code and therefore the method.

The next part will focus on code re-implementation, a procedure, which can help understanding the method, but is even more time consuming.

2.2 Robustness: from MATLAB to Python, language and organization

To fully master the method, we developed a complete open source toolkit of genomic stratification in Python [16]. Python is also an interpreted programming language, but contrary to MATLAB is free of use and has a GPL-compatible license [17], which fosters both robustness and generalizability. Recoding in another language in a different environment will lead to be some unavoidable problems such as variation in low level libraries (e.g. glibc): it is likely that the outcomes will vary even if the same algorithm is implemented [18]. In addition, we rely on Python packages to perform visualization or linear algebra computations (e.g. Matplotlib, SciPy, NumPy [19–21]), and results may depend on these packages versions. Python is currently in a transitional period between two major versions 2 and 3. We chose to write the code in Python 3, which is the current recommendation.

2.2.1 Metadata and file formats

Even if the original code could be run, we had to handle several file formats to check and understand the structure of the original data. For instance the data were provided by The Cancer Genome Atlas (TCGA) [22] and made available in a MATLAB *.mat* file format v7.2 as compressed data (sparse matrices). Thanks to SciPy, Python can load those MATLAB files version. We wanted to use the open format HDF5 for saving the results, however Scipy's sparse matrices could not be stored in HDF5 format (Table 1). We thus decided to continue saving in *.mat* format. Moreover, the original authors had denoted download dates of patients' data of TCGA, thereby clarifying the data provenance. But in the absence of structural metadata, we had to explore the hierarchical structure of the variables (e.g. patient ID, gene ID, phenotype).

2.2.2 Codes and parameters

Beyond documentation and file formats, code initialization and parameters settings are also keys for reproducibility. Upon execution of the code, "unexpected" results were obtained. One cause was the application of the hierarchical clustering step for which we used the clustering tools of SciPy. Both SciPy and MATLAB (MathWorks) functions offer seven linkage methods, however, SciPy's default option (single method) [23] differs from MATLAB's default option (UPGMA or average method) [24], which was used in the original study (Table 1). Another cause for the variation in results is the value of one of the most important parameters of the method, the graph regulator factor, which was not clarified in the original paper. From the article, we believed that this factor had a constant value of 1.0 until we found in the original code that its value varies across iterations and converges to an optimal value around 1800. Therefore, we initially obtained very different results from the original NBS (Fig 3 a) with heterogeneous subgroups. Once the optimal value was set up, we finally observed homogenous clusters (Fig 3 b). Moreover, during our attempts to run the original code to understand the causes of the errors, we realized that some parts of the code were not run anymore (e.g. discarded work, remaining traces of debugging) which made understanding the implementation harder. To allow others to reproduce our results, we wrote some documentation and tutorials for the Python package *StratiPy* [16]. Readers are thus invited to experiment with reproducibility by generating the two confusion matrices of Fig 3. This is described by the following different tools: GitHub, Docker, and Jupyter/IPython notebook.

2.2.3 Documentation and examples

During the re-coding process, we used an enhanced Python interpreter to debug: IPython, an interactive shell supporting both Python 2 and 3. Since the dataset is large and the execution takes a significant amount of time, we used IPython to re-run interactively some sub-sections of the script, which is one of the most helpful features. IPython can be integrated in the web interface Jupyter Notebook, offering an advanced structure for mixing code and documentation. While the Jupyter/IPython notebook was therefore initially convenient, it does not scale well to large programs and is not well adapted to versioning. However, ability of mixing code with document text is very useful for tutorials: a user of the code can read documentation (docstring), text explanations, and see how to run the code, explore parameters and visualize results in the browser. Our work on NBS, as related here, can be reproduced with a Jupyter/IPython notebook available via our GitHub repository [16]. One can find more examples and several helpful links via this “gallery of interesting Jupyter Notebooks” [25], which contains a section about “Reproducible academic publications”.

To conclude, we were able to test the robustness of the method with our python implementation, but this took approximately two months of a senior researcher and six months of a master student. Fig 2 illustrate this work through an analogy between robustness issues and road transport: driving in a different environment (e.g. OS), we attempt to obtain identical results (i.e. to reach the same location) using the same input data (i.e. gasoline), but with different computational environment (i.e. cars), different implementation of the method (i.e. engine) and different programming languages (i.e. MATLAB and Python roads).

2.3 Collaborative code development and best practices

Throughout the project we used the version control system (VCS) Git to document the development of our Python package. Git is arguably one of the most powerful VCS, allowing easy development of branches and the distributed team (Paris, Berkeley, Montreal) to work collaboratively on the project. This project, *StratiPy*, is hosted on GitHub, a web-based Git repository hosting service [16]. While the original code was not available on GitHub, the main authors shared their code on a website. This should be sufficient for reproducibility and replicability our purposes but makes it less easy to collaborate on the code. While working on our GitHub repository, researchers from USA, India, China, and Europe contacted us about our robustness experiment. Not only GitHub supports a better organization of projects, it also facilitates the collaboration on open-source software projects through, thanks to its social network functions [26]. We adopted open source coding standards and learnt how to efficiently use Git and GitHub. Both required considerable training efforts on the short-term but brought clear benefits on the long-term, especially regarding collaboration and debugging.

2.4 Reproducibility of Robustness: from Python to Python

Knowing how difficult it can be to re-run someone else’s code, we then attempted to start the analysis from scratch and to reproduce the results on another platform from our newly developed python package. While the Python code was developed under Mac OS X Sierra (10.12), we used an Ubuntu 16.04.1 (Xenial) computer to test the Python implementation. Some additional issues emerged (Table 1). First, our initial documentation did not include the list of the required packages and instructions to launch the code. Second, the code was very slow to the extent that it was impractical to run it on a laptop because the Numpy package had not been compiled with BLAS (Basic Linear Algebra Subprograms) that speeds up low-level routines performing basic vector and matrix operations. Last, there was (initially) no easy way to check whether the results obtained on a different architecture were the expected ones. We added documentation and tests on the results files md5sum to solve this. To summarize, although the reuse and reproducibility of the results of the developed package were improved, these were far from being optimal in the first attempt.

3 Potential solutions: from local to global

3.1 Act locally: simple practices and available tools

We conclude this reproducibility case study experiment by suggesting tools and best practices following the programming best practices of Wilson et al., such as modularizing and re-using code, unit testing, document design, data management, and project organization [2,27], as well as keeping data provenance and recording all intermediate results [28].

3.1.1 Publish software and their environment

Increased reproducibility and replicability can be obtained by following Buckheit and Donoho's long standing motto: "When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures" by offering a complete and free package (WaveLab) to reproduce the published output [29]. Container and virtual machines technologies such as Docker [30], Vagrant [31], Singularity [32,33] (easily works in cluster environments) are becoming a standard solution to mitigate installation issues. These rely however on competencies that we think too few biologists possess today. While a container might encapsulate everything needed for a software execution, it can be hard to develop in a container. For instance, running Jupyter/IPython notebooks in Docker containers requires knowledge on advanced port forwarding, which may be discouraging for many biologists. Therefore, we decided to propose two options in our example implementation of reproducibility: 1) a step-by-step process to follow in a Jupyter/IPython notebook; or 2) a Docker container ready to be built and run. Mastering Docker –or other container tools– is increasingly becoming an important skill for biologists who use computational tools.

3.1.2 Document with appropriate Metadata

Standard metadata are vital for an efficient documentation of both data and software. In our example, we still lack the standard lexicon to document the data as well as documenting the software. We however aim to follow the recommendations by Stodden *et al.* [34]: *"Software metadata should include, at a minimum, the title, authors, version, language, license, Uniform Resource Identifier/DOI, software description (including purpose, inputs, outputs, dependencies), and execution requirements".* The more comprehensive is the metadata description, the more likely the reuse will be both efficient and appropriate [35].

3.1.3 Write readable code

We draw some conclusion from our experience in working with others code.

First, the structure of the program should be clear and easily accessible. Second, good concise code documentation and naming convention will help readability. Third, the code should not contain left-overs of previously tested solutions. When a solution takes a long time to compute, an option to store it locally can be proposed. Using standard coding and documentation conventions (e.g. PEP 8 and PEP 257 in Python [36,37]) with detailed comments and references of papers makes the code more accessible. When an algorithm is used, any modification from the original reference should be explained and discussed in the article as well as in the code. We advocate for researchers to write code "for their colleagues", hence, ask for the opinion and review of co-working or partner laboratories. Furthermore, the collaboration between researchers working on different environments can more easily isolate reproducibility problems. In the future, journals may consider review of code as part of the standard review process [38].

3.1.4 Test the code

To check if the code is yielding a correct answer, software developers associate test suites (unit tests or integration tests) with their software. While we developed only a few tests in this project, we realize that this practice has a number of advantages, such as checking if the software installation seems correct, check if updates in the code or in the operating system impact the results, etc. In our case, we propose to check for the integrity of the data and for the results of some key processing.

3.2 Think globally: from education to community standards

3.2.1 Training the new generation of scientists to digital tools and practices

The training in coding and software development is still too limited for biologists. Often, it is limited to self-training from searching answers on Stack Overflow or equivalent. Despite efforts by organizations such as Software [39] or Data Carpentry [40] and the growing demand for 'data scientists' in life sciences, university training and assessment on coding practices is still not generalized. The difficulty to access and understand code may lead to applying code blindly without checking the validity of the results: often, scientists may prefer to believe that results are correct because checking the validity of the results may require significant time. Mastering a package such that results are truly understood can take a long time, as it was the case in our experiment.

Academia could - and we argue should - instruct young scientists in best practices for reproducibility. For instance, Hothorn and Leisch organized a reproducibility workshop gathering mostly PhD students and young postdocs specialized in bioinformatics and biostatistics. Then they evaluated 100 random sample papers from *Bioinformatics* [3]. Their study revealed how such a workshop can raise young scientists awareness about *"what makes reproduction easy or hard at first hand"*. Indeed, they found out that only a third of the original papers and two-thirds for applications notes had given access to the source code of software used.

3.2.2 Standard consensus dataset and testing ecosystem

We propose here that bioinformatics methods publications are systematically accompanied with a test dataset, code source and some basic tests (given ethical and legal constraints). As the method is tested on new datasets, the number of tests and range of applications would expand. We give a first example with our NBS re-implementation.

A schematic overview of a possible testing ecosystem generalizing our test study is shown in Fig 4. The core of this system would be a set of standard consensus datasets used to validate methods. For instance in the field of machine learning, standard image databases are widely used for training and testing (e.g. MNIST for handwritten digits [41]). In the case of our proposal, data could be from different categories such as binary, text, image (shown as folders in different colors, Fig 4 b), and sub-categories to introduce criteria such as size, quantitative/qualitative, discrete/continuous using a tagging system. Datasets could be issued from simulations or from acquisition, and would validate a method on a particular component. This testing ecosystem will help scientists that cannot release their data because of privacy issues (Fig 4 a.1) although these can often be overcome, but also give access to data and tests to a wide community including establishments with little financial means.

We divide those who interact with scientific software or analysis code in two broad categories. First, the authors ("A") who propose a method and need to verify its validity and usefulness with open and/or private – data. Second, the users ("U", e.g. developers, engineers, bioinformaticians) who need to test and evaluate the proposed methods with other data.

When authors propose a new method, we propose that authors and users progressively build its reproducibility profile (Fig 4 b.3, b.4) to document which method best work with which data. During the optimization of a project, the software code and associated documentation should be accessible to foster collaboration on additional use cases and data. When the work achieves some level of maturity, a full fledge article can be posted on a preprint servers such as bioRxiv [42,43] and be associated with a GitHub/GitLab repository with a digital object identifier (DOI). With considerable effort, Stodden *et al.* conducted a reproducibility study on 204 random articles of Science: despite some availability of the code, it had often been changed after publication, causing difficulties in replication [44]. In our proposed testing ecosystem, users will be able to launch reproducibility projects more easily thanks to code and article versioning.

Users who test and approve reproducibility on original or new data could be accredited and recognized by the scientific and developer communities (i.e. Stack Overflow, GitHub). This testing ecosystem could thus facilitate collaborations between methodology development and biological research communities.

4 Conclusion and perspective

In the 19th century, Pasteur introduced a detailed "Methods" section in his report: this advanced approach was necessary to reproduce his experiments and became the norm in the practice of science [45]. Today with the advent of computational science, the reproducibility issue is seen as a growing concern. To summarize, our experiment at reproducing initial results led to the following recommendations:

- Improve life scientists software development skills
- Use online repositories and tools to help other scientists in their exploration of the method [25,26,30]
- Enhance the cooperation between academic education and industry [39,40,46]
- Develop an open source continuous testing ecosystem with community standards, well-identified datasets to validate tools across versions and datasets, and go beyond the publication of a PDF file

Verifying a previously published method can be a very time consuming, and is often poorly acknowledged. Some top-down initiatives already provide some incentives for such a process i.e. Horizon 2020 (H2020) [47] project of the European Commission (EC) mandates open access of research data, while respecting security and liability. H2020 supports OpenAIRE [48] a technical infrastructure of the open access, which allows the interconnection between projects, publications, datasets, and author information across Europe. Thanks to common guidelines, OpenAIRE interoperates with other web-based *generalist* scientific data repositories such as Zenodo, hosted by CERN, which allows the combination of data and GitHub repository via DOIs. The Open Science Framework also hosts data and software for a given project [49]. Respecting standard guidelines to transparently communicate the scientific work is a key step towards tackling irreproducibility and insures a robust scientific endeavor.

Key points

- Main barrier for reproducibility is in the lack of compatibility between environments, programming languages, software versions, etc.
- At the individual level, the key is in research practices such as well written, tested and documented code, and well curated data and the use of online repositories and collaborative tools.
- At the community level, we propose a testing ecosystem where standard consensus datasets are used to validate new methods and foster their generalizability.

Declarations

• Abbreviations

ASD: Autism Spectrum Disorders; BLAS: Basic Linear Algebra Subprograms; DOI: Digital Object Identifier; H2020: Horizon 2020; OS: Operating System; TCGA: The Cancer Genome Atlas; VCS: version control system.

• Ethics approval and consent to participate

We used the uterine endometrial carcinoma dataset downloaded on January 1st, 2013 from the TCGA portal as used by Hofree and colleagues in their previous paper [12].

• Consent for publication

Not applicable

• Availability of data and materials

The latest version of StratiPy (Python) with two examples of reproducibility and dataset are available at GitHub 16], and archived via a Zenodo DOI [50]

• Competing interests

The authors declare that they have no competing interests.

● Funding

This work was supported by:

- Institut Pasteur (http://dx.doi.org/10.13039/501100003762)
- H2020 Societal Challenges (http://dx.doi.org/10.13039/100010676)
- Centre National de la Recherche Scientifique (http://dx.doi.org/10.13039/501100004794)
- Université Paris Diderot (http://dx.doi.org/10.13039/501100005736)
- Conny-Maeva Charitable Foundation
- Cognacq-Jay Foundation
- Orange (http://dx.doi.org/10.13039/501100003951)
- Fondation pour la Recherche Médicale (http://dx.doi.org/10.13039/501100002915)
- GenMed Labex
- BioPsy Labex.

● Authors' contributions

Y-M. K., J-B. P., and G.D. wrote the manuscript, Y-M. K. and G.D. developed the StratiPy module.

YMK: Conceptualization, Software, Validation, Writing – original draft, Writing – review & editing

JBP: Validation; Writing – original draft, Writing – review & editing

GD: Conceptualization, Software, Supervision, Validation, Writing – original draft, Writing – review & editing.

All authors read and approved the final manuscript.

● Acknowledgements

We thank Thomas Rolland and Freddy Cliquet for sharing their advice and comments.

● Authors' information

Yang-Min KIM [1,2,3,4,*] is a PhD student at Human Genetics and Cognitive Functions unit at the neuroscience department of the Institut Pasteur in Paris. Her research on next-generation sequencing data and biological networks is focused on stratification of patients with Autism.

Keywords: Autism Spectrum Disorder; Network; Protein-Protein Interaction; Personalize Medicine; Network-Based Stratification; Computational Biology.

Jean-Baptiste Poline [5,6] is a researcher at Henry H. Wheeler Jr. Brain Imaging Center, Helen Wills Neuroscience Institute, University of California, Berkeley, California, USA. His research focuses on neuroimaging methods, imaging-genetic biostatistics and neuroinformatics.

Keywords: Neuroinformatics; Statistical methods; Brain imaging; Imaging genomics.

Guillaume Dumas [1,2,3,4] is research fellow of the Human Genetics and Cognitive Functions unit at the neuroscience department of the Institut Pasteur in Paris. His interdisciplinary work is at the cross-road of social psychology, cognitive neuroscience, and system biology.

Keywords: Open Science; Complex Systems; Computational Biology; Cognitive Science; Social Neuroscience; Autism Spectrum Disorder.

References

1. Baker M. 1,500 scientists lift the lid on reproducibility. Nat News. 2016;533:452.

2. Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK. Good enough practices in scientific computing. PLOS Comput Biol. 2017;13:e1005510.

3. Hothorn T, Leisch F. Case studies in reproducibility. Brief Bioinform. 2011;12:288–300.

4. Shapin S, Schaffer S. Leviathan and the Air-Pump: Hobbes, Boyle, and the Experimental Life (New in Paper). Princeton University Press; 2011.

5. Peng RD. Reproducible research in computational science. Science. 2011;334:1226–7.

6. Whitaker K. Showing your working: a how to guide to reproducible research. Figshare. 2017. https://doi.org/10.6084/m9.figshare.5443201.v1

7. Nekrutenko A, Taylor J. Next-generation sequencing data interpretation: enhancing reproducibility and accessibility. Nat Rev Genet. 2012;13:667–72.

8. Hutson M. Missing data hinder replication of artificial intelligence studies. Science. 2018. doi: 10.1126/science.aat3298

9. Herndon T, Ash M, Pollin R. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. Camb J Econ. 2014;38:257–79.

10. Bourgeron T. From the genetic architecture to synaptic plasticity in autism spectrum disorder. Nat Rev Neurosci. 2015;16:551–63.

11. Loth E, Spooren W, Ham LM, Isaac MB, Auriche-Benichou C, Banaschewski T, et al. Identification and validation of biomarkers for autism spectrum disorders. Nat Rev Drug Discov. 2016;15:70–73.

12. Hofree M, Shen JP, Carter H, Gross A, Ideker T. Network-based stratification of tumor mutations. Nat Methods. 2013 Nov;10(11):1108-15. doi: 10.1038/nmeth.2651.

13. Introducing MEX Files - MATLAB & Simulink - MathWorks France https://fr.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html?requestedDomain=www.mathworks.com Accessed 1st June 2018

14. Tursa. MTIMESX - Fast Matrix Multiply with Multi-Dimensional Support - File Exchange - MATLAB Central.. 2009 http://fr.mathworks.com/matlabcentral/fileexchange/25977-mtimesx-fast-matrix-multiply-with-multi-dimensional-support Accessed 1st June 2018

15. tim.lewis. OpenMP Specifications. http://www.openmp.org/specifications/ Accessed 1st June 2018

16. Stratipy: Graph regularized nonnegative matrix factorization (GNMF) in Python. GHFC; 2017. https://github.com/GHFC/Stratipy

17. Python Software Foundation. History and License — Python 3.6.1 documentation. 2017 https://docs.python.org/3/license.html#licenses-and-acknowledgements-for-incorporated-software Accessed 1st June 2018

18. Glatard T, Lewis LB, Ferreira da Silva R, Adalat R, Beck N, Lepage C, et al. Reproducibility of neuroimaging analyses across operating systems. Front Neuroinformatics. 2015;9:12.

19. Michael Droettboom, Thomas A Caswell, John Hunter, Eric Firing, Jens Hedegaard Nielsen, Antony Lee, … Peter Würtz. (2018, March 17). matplotlib/matplotlib v2.2.2 (Version v2.2.2). Zenodo. http://doi.org/10.5281/zenodo.1202077

20. Pauli Virtanen, Ralf Gommers, Evgeni Burovski, Travis E. Oliphant, David Cournapeau, Warren Weckesser, … Tyler Reddy. (2018, March 24). scipy/scipy: SciPy 1.0.1 (Version v1.0.1). Zenodo. http://doi.org/10.5281/zenodo.1206941

21. NumPy homepage http://www.numpy.org/ Accessed 1st June 2018

22. TCGA. Cancer Genome Atlas - Natl. Cancer Inst. https://cancergenome.nih.gov/ Accessed 1st June 2018

23. Eads. Hierarchical clustering (scipy.cluster.hierarchy) — SciPy v0.19.0 Reference Guide. 2007 https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html Accessed 1st June 2018

24. Hierarchical Clustering - MATLAB & Simulink - MathWorks France. https://fr.mathworks.com/help/stats/hierarchical-clustering-12.html Accessed 1st June 2018

25. A gallery of interesting Jupyter Notebooks · jupyter/jupyter Wiki. https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks Accessed 1st June 2018

26. Blischak JD, Davenport ER, Wilson G. A Quick Introduction to Version Control with Git and GitHub. PLoS Comput Biol. 2016;12:e1004668.

27. Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. Best Practices for Scientific Computing. Eisen JA, editor. PLoS Biol. 2014;12:e1001745.

28. Sandve GK, Nekrutenko A, Taylor J, Hovig E. Ten Simple Rules for Reproducible Computational Research. Bourne PE, editor. PLoS Comput Biol. 2013;9:e1003285-4.

29. Buckheit JB, Donoho DL. WaveLab and Reproducible Research. Wavelets Stat.. Springer, New York, NY; 1995. p. 55–81. https://link.springer.com/chapter/10.1007/978-1-4612-2544-7_5

30. Boettiger C. An introduction to Docker for reproducible research, with examples from the R environment. ACM SIGOPS Oper Syst Rev. 2015;49:71–9.

31. Introduction. Vagrant HashiCorp.. https://www.vagrantup.com/intro/index.html Accessed 1st June 2018

32. Singularity homepage.. http://singularity.lbl.gov/ Accessed 1st June 2018

33. Kurtzer GM, Sochat V, Bauer MW. Singularity: Scientific containers for mobility of compute. PLOS ONE. 2017;12:e0177459.

34. Stodden V, McNutt M, Bailey DH, Deelman E, Gil Y, Hanson B, et al. Enhancing reproducibility for computational methods. Science. 2016;354:1240–1.

35. Hill SL. How do we know what we know? Discovering neuroscience data sets through minimal metadata. Nat Rev Neurosci. 2016;17:735–6.

36. PEP 8 -- Style Guide for Python Code. Python.org. [cited 2017 Aug 21]. https://www.python.org/dev/peps/pep-0008/

37. PEP 257 -- Docstring Conventions. Python.org. https://www.python.org/dev/peps/pep-0257/

38. Eglen SJ, Marwick B, Halchenko YO, Hanke M, Sufi S, Gleeson P, et al. Toward standard practices for sharing computer code and programs in neuroscience. Nat Neurosci. 2017;20:770–3.

39. Software Carpentry website. http://software-carpentry.org//index.html Accessed 1st June 2018

40. Data Carpentry website. http://www.datacarpentry.org/ Accessed 1st June 2018

41. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. http://yann.lecun.com/exdb/mnist/

42. Bourne PE, Polka JK, Vale RD, Kiley R. Ten simple rules to consider regarding preprint submission. PLOS Comput Biol. 2017;13:e1005473.

43. Preprints in biology. Nat Methods. 2016;13:277–277.

44. Stodden V, Seiler J, Ma Z. An empirical analysis of journal policy effectiveness for computational reproducibility. Proc Natl Acad Sci. 2018;115:2584–9.

45. Day RA, Gastel B. Historical Perspectives. Write Publ Sci Pap Seventh Ed. ABC-CLIO; 2011. p. 6–8.

46. Academia – Industry Software Quality & Testing summit - ISTQB® International Software Testing Qualifications Board. http://www.istqb.org/special-initiatives/istqb-conference-network-2istqb-conference-network-academia/academia-%E2%80%93-industry-software-quality-testing-summit.html

47. Open Research Data in Horizon 2020. https://ec.europa.eu/research/press/2016/pdf/opendata-infographic_072016.pdf

48. Open Access in Horizon 2020 - EC funded projects https://www.openaire.eu/edocman?id=749&task=document.viewdoc

49. Foster ED, Deardorff A. Open Science Framework (OSF). J Med Libr Assoc JMLA. 2017;105:203–6.

50. Kim Yang-Min, Poline Jean-Baptiste, Dumas Guillaume. StratiPy. Zenodo; 2017. https://doi.org/10.5281/zenodo.1042546

Figure legends

Figure 1: Hidden reproducibility issues are like an underwater iceberg. Scientific journals readers have the impression that they can almost see the full work of method. But in reality, articles do not take into account adjustment and configuration for significant replication in most cases. Therefore, there is a significant gap between apparent executable work (i.e. above water portion of iceberg) and necessary effort in practice (i.e. the full iceberg).

Figure 2: Analogy between robustness issues and road transport. The aim is to achieve same output (i.e. to reach the same location) using published methods (i.e. engine). Despite the same input data (i.e. gasoline), we obtained different results due to different programming languages —e.g. MATLAB and Python— (i.e. different roadways) and environments (i.e. different vehicles).

Figure 3: Normalized confusion matrices between original and replicated results. Before (a) and after (b) applying appropriate value of graph regularization factor on NBS method. Each row or column corresponds to a subgroup of patients (here three subgroups). The diagonal elements show the frequency of correct classifications for each subgroup: a high value indicates a correct prediction.

Figure 4: Working principles of testing ecosystem with private data. Figure 4a shows a classical case: (a.1) Authors take private data (e.g. blue data) then publish their method and corresponding results; (a.2) Users having their own data (e.g. orange data) find a relevant paper but will be lost in the labyrinth of reproducibility. Figure 4b shows testing ecosystem with standard consensus dataset: (b.1) If authors work with their own data, they must identify corresponding standard data tag(s) (e.g. blue data); (b.2) Authors initiate to develop their method with corresponding standard data and reproducibility profile will be progressively built. Bar length on iceberg corresponds to progression of replication test; (b.3) Users can test proposed method with other

standard data (e.g. orange and green data) and thus participate to enhancement of the reproducibility profile; (b.4) Thanks to the collective work on testing, the method could be optimized and authors can upgrade their initial paper (versioning).

Fig 1

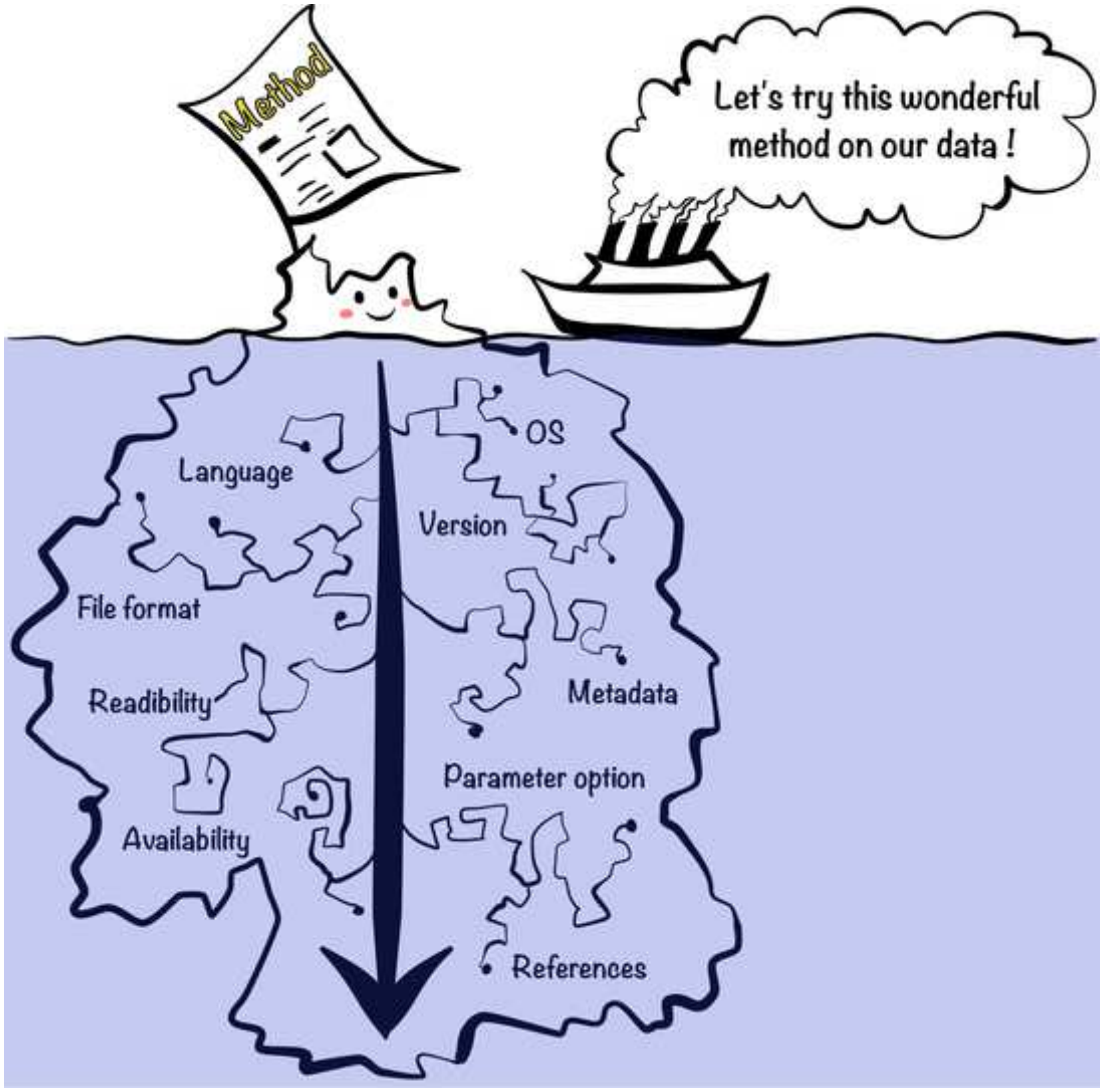
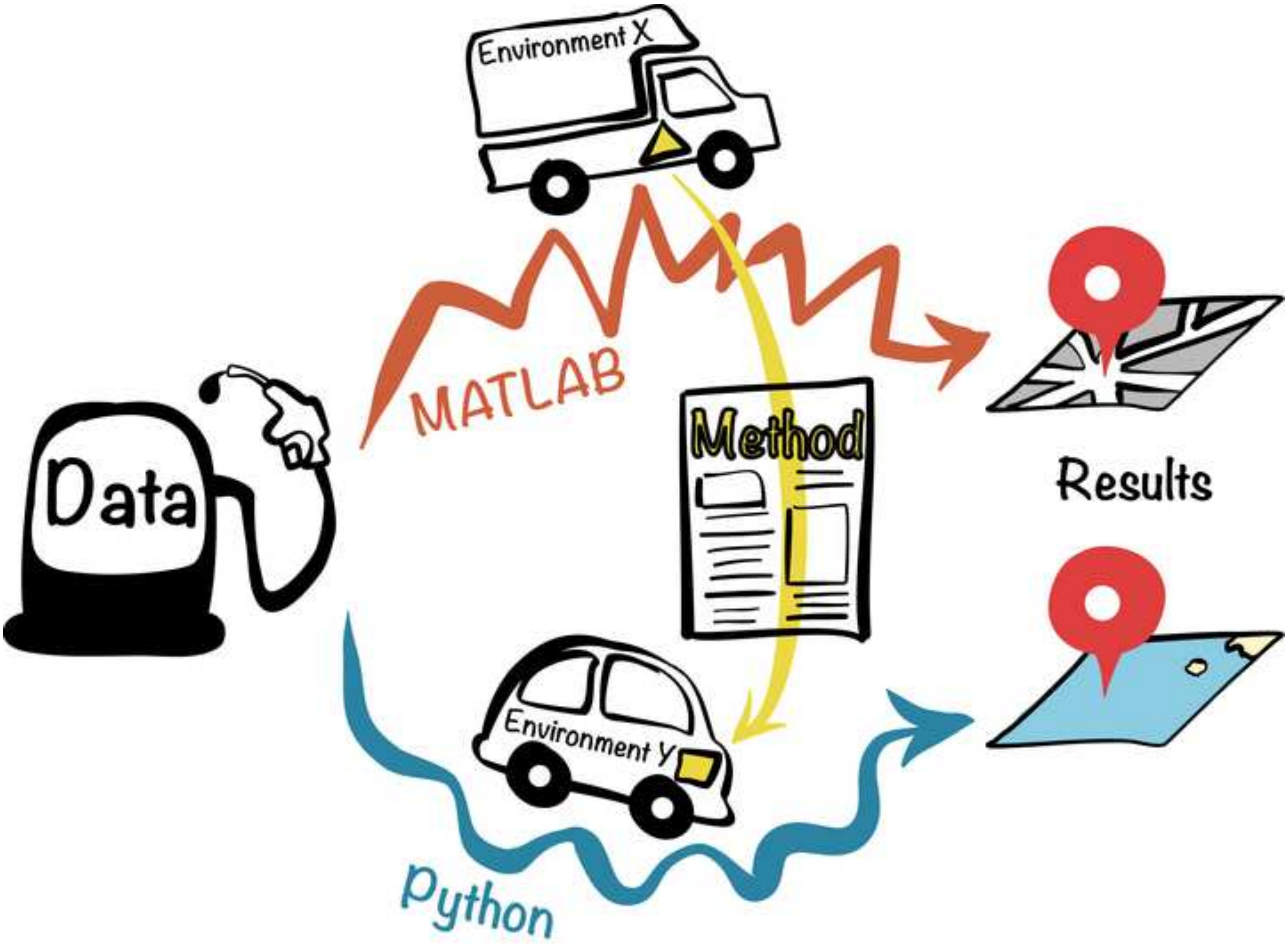
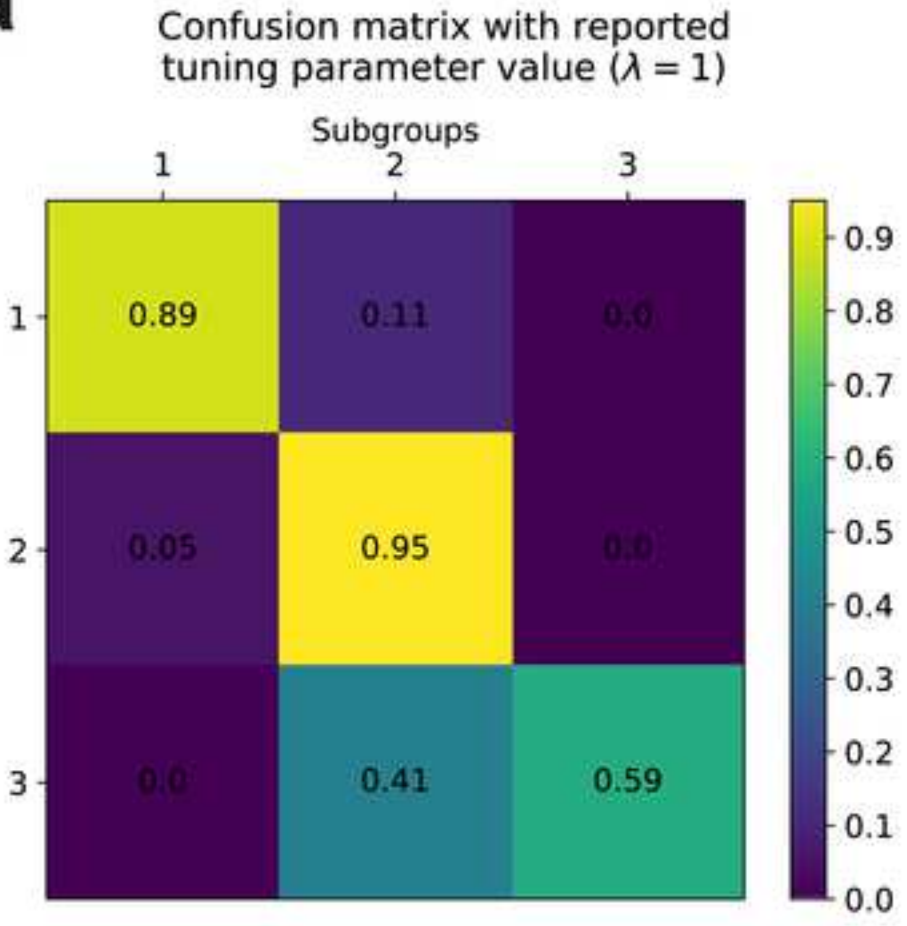

Method
Let's try this wonderful method on our data !
OS
Language
Version
File format
Metadata
Readibility
Parameter option
Availability
References

Fig 2


Environment X
Data
MATLAB
Method
Results
Environment Y
Python

a


Confusion matrix with reported tuning parameter value ($\lambda = 1$)
Subgroups
1
2
3
0.89
0.11
0.0
0.05
0.95
0.0
0.0
0.41
0.59
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

b

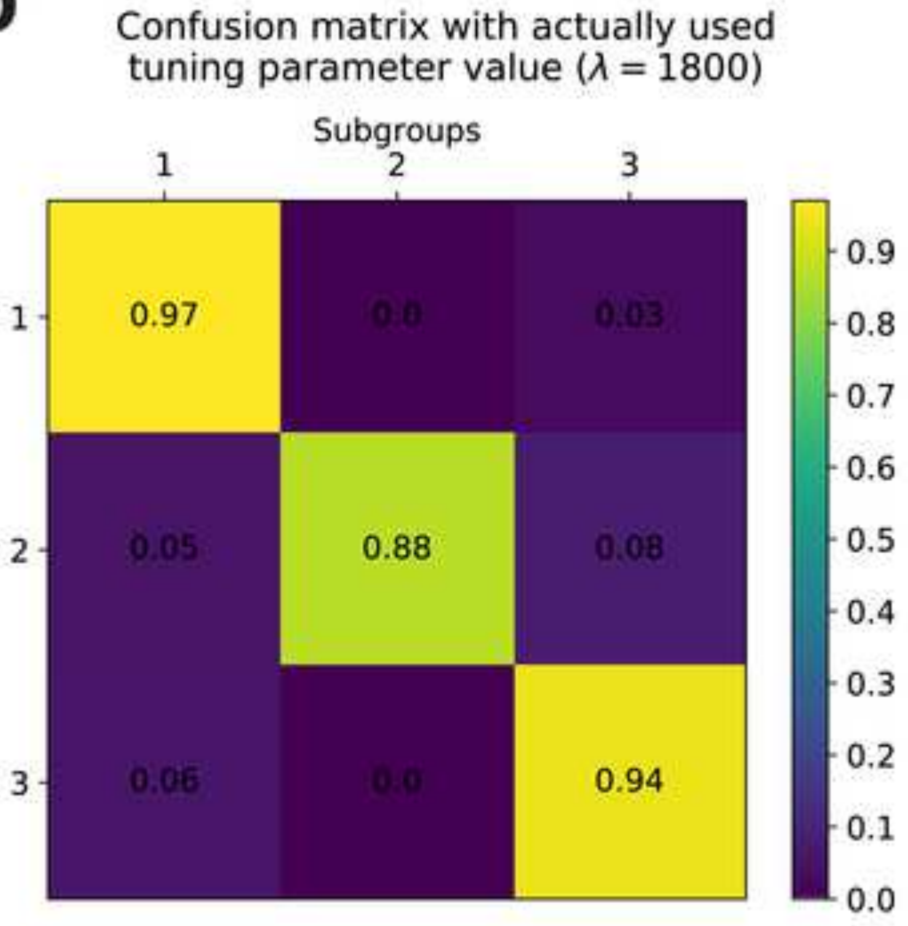
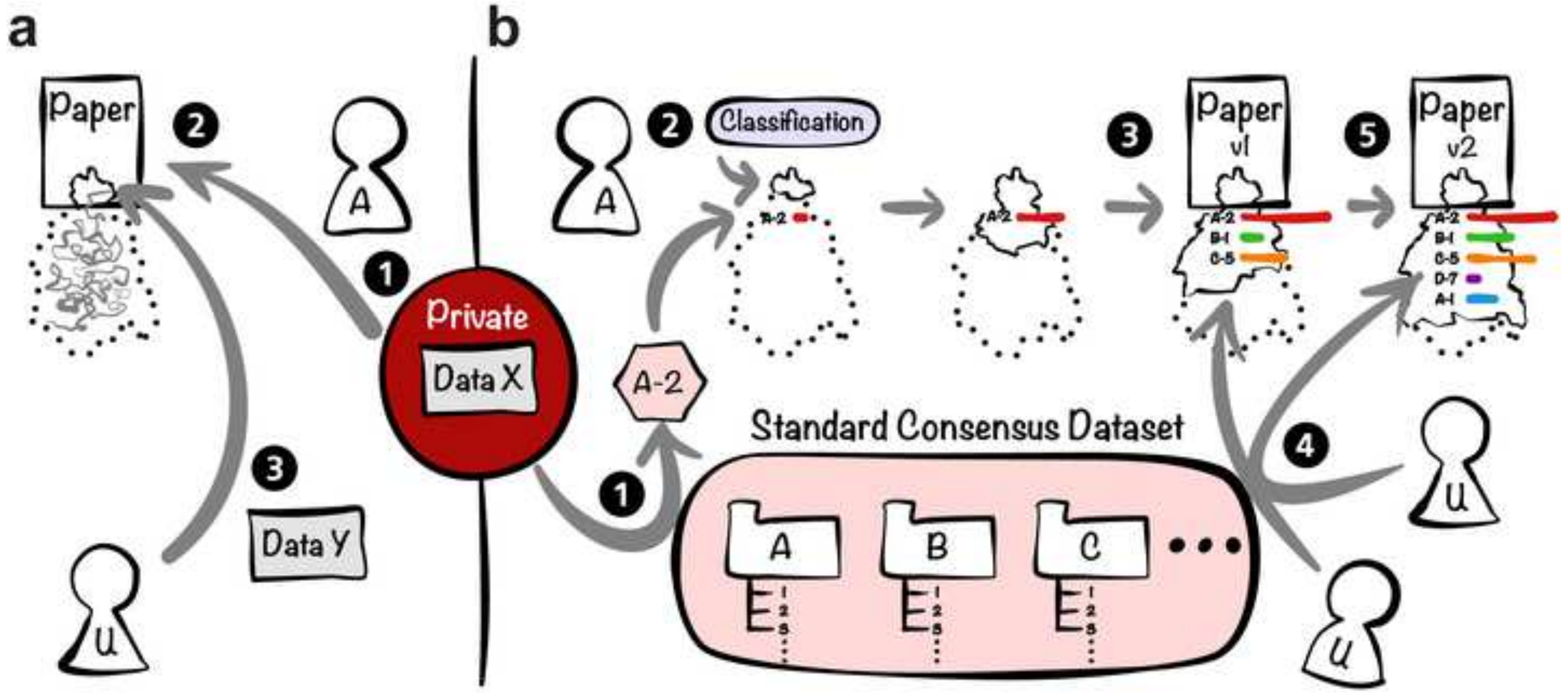

Confusion matrix with actually used tuning parameter value ($\lambda = 1800$)
Subgroups
1
2
3
0.97
0.0
0.03
0.05
0.88
0.08
0.06
0.0
0.94
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

Fig 4

Click here to access/download;Figure;fig4_workflow_lzw.tif



a
b
Paper
2
A
1
Private
Data X
3
Data Y
U
A
2
Classification
A-2
A-2
3
Paper v1
A-2
B-1
C-5
5
Paper v2
A-2
B-1
C-5
D-7
A-1
1
A-2
Standard Consensus Dataset
A
B
C
1
2
3
4
U
U