University of Passau

Faculty of Computer Science and Mathematics

Chair for Digital Libraries & Web Information Systems

Bachelor Thesis in Internet Computing

Classification of Machine Learning Reproducibility Factors

submitted by

Martin Johannes Loos

Examiner: Prof. Dr. Michael Granitzer
Supervisor: Mehdi Ben Amor
Date: April 6, 2022

Contents

Abstract

Please write a short abstract summarizing your work.

Acknowledgments

I would first like to thank . . .

List of Figures

List of Tables

1 Introduction

Machine learning is a key technology for computationally solving complex problems in various fields [ANK18]. Therefore, reproducing and verifying the results of ML experiments is of great importance. Thus, it must be clearly defined which factors influence reproducibility in this area.

In traditional programming [Zel78] software is created on the basis of a requirements analysis with the associated specification. In this step, it is defined which input the program expects. The software architecture is then worked out in a design process and implemented in a coding phase. The goal of a program is to map the input to the expected output. Finally, the output is verified in a testing phase to verify that it operates the way it is intended to. In theory, you can test software to the point where you can guarantee correct mapping for all input data. In practice, this will not be feasible for large software programs due to many influencing factors [Pan99].

In contrast, machine learning takes a different approach [BD17]. For a machine, it is possible to build a model for a given dataset, which maps the data for us. First, the dataset is preprocessed in order to filter out superfluous information and reduce it to the required data. Several factors are then selected, including the algorithm, which is to be used or the ratio in which the dataset is divided into training and test data (train-test split). In addition, relevant features of the dataset are specified. Finally, the machine-made model is then validated to check how well the created model maps the test data of the selected data set. The advantage compared to traditional programming is that a machine can recognize relationships from the data that cannot be recognized by a human. However, there is a high probability that the machine-generated model will not be able to assign the input to the expected output in all cases. Therefore, one evaluates the quality of the representation of the model in this process and tries improving it incrementally until it meets the expectations.

Without the help of machine learning, many modern software solutions would not even be able to be implemented [ACP19]. In almost all areas such as healthcare [SSJ18], big data[Lhe+17], or intrusion detection [SP10], ML can improve software solutions or even enable them in the first place. Due to this variety of problems that are to be solved by ML, there is also the need for different, problem-specific algorithms [Dey16]. However, there are still numerous research challenges, such as the hardware design [Sze+17], ethical concerns [GHS19] or model management [Sch+18].

In this thesis, we are interested in the identification and classification of reproducibility factors which influence machine learning experiments. Firstly, a precise definition of the term reproducibility [KS15] is needed. In an article from 2014 McNutt states that "[...] just because a result is reproducible does not necessarily make it right, and just because it is not reproducible does not necessarily make it wrong." [McN14]. Although this statement is correct, experiments can only be verified if they can be reproduced. The results from an academic paper must be verified by third parties to be recognized by the general public. Therefore, the reproducibility of experiments is not only desirable but necessary [IT18a]. A 2016 Nature survey, where 1,576 researchers from different scientific fields participated, found that "More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments." [Bak16]. This indicates that we are in a reproducibility crisis. We want to address this problem by creating a basis for software-based analysis of reproduction factors. Hence, we must first determine which factors influence reproducibility.

The identification of factors that affect reproducibility is an important subject of current research [Ban+21; Eis18; GGA18; IT18b; Liu+21; MK17; McD+19; OBA17a; SK21]. NeurIPS, a machine learning and computational neuroscience conference, has even included reproducibility as part of its submission policy [Pin+19]. In a report from the NeurIPS 2019 reproducibility program, processes and guidelines were listed that are intended to improve reproducibility. Software tools that would support researchers were not introduced in this article, but the authors noted that the "Standardization of such tools would [...] improve ease of reproducibility" [Pin+20]. Therefore, we will collect the findings on this from other scientific papers and process them systematically. In addition, we will delimit which of the identified reproducibility factors also influence replicability.

Based on this identification step, these factors can then be classified. The goal of this classification is to identify factors that can be detected and analyzed using a software tool. Therefore, we decide on a division into detectable and undetectable factors. The core of this thesis is the implementation of an analysis tool for detectable reproducibility factors, which will present the result to users, including helpful information. We will then use our tool to analyze two different groups, each containing 100 repositories. Our *hypothesis* is that the group containing code published by scientific publishers will give statistically better results compared to the other. For undetectable reproducibility factors (e.g. p-Hacking [Hea+15a]) we will develop a complementary approach.

The information extracted from the repository analysis could also be used by other tools in the future. Based on the repository analysis of our tool, one could automatically generate missing artifacts and provide information on critical points of the ML experiment. Another use case could be a rating tool that works based on our analysis, similar to what the Department of Defense wants to achieve with their SCORE program [1] in the field of social and behavioral science. It could indicate the probability of successful reproduction which would facilitate the review and verification of a machine learning experiment by external researchers.

This work is the basis for a software-automated tool that aims to improve the reproducibility of machine learning experiments in the future.

[1] https://www.darpa.mil/program/systematizing-confidence-in-open-research-and-evidence

2 Background

This chapter contains basic knowledge on which the main part of this thesis draws. First, we want to define the concept of reproducibility precisely. In the literature, this term is often used with different meanings. Therefore, we will clearly distinguish it from the term replicability. We also provide an overview of other important concepts and technologies. Where necessary, we will place topics in the overall context to explain their influence on this thesis. In addition, at the end of this chapter, we give a brief overview of a typical machine learning workflow.

2.1 Reproducibility

In this section we will precisely define the term reproducibility and distinguish it from replicability. We also want to draw attention to the fact that different definitions are used in other works. This goes from slight deviations to the exact opposite meaning of the terms. One therefore speaks of a so-called confused terminology[Ple18]. We understand by reproducibility what Goodman et al. defined as results reproducibility.

Reproducibility: "Obtain the same results from an independent study with procedures as closely matched to the original study as possible.[GFI16a]"

We will explain the term machine learning later in section 2.5. The presented workflow makes it clear that an experiment in this area can only be repeated exactly under very specific conditions. However, results from scientific work must be able to be verified by third parties in order to confirm their validity. This is of paramount importance to science and a cornerstone of this way of working. Reproducibility is desirable to increase the validity and impact of a work, since results are still the same under slightly different conditions. Replicability, on the other hand, is indispensable in order to be able to verify the results of a work.

2.1.1 Reproducibility vs. Replicability

In order to keep this thesis consistent in terms of terminology, we adapt the definition of methods reproducibility from Goodman et al. and understand this to mean replicability.

Replicability: "Provide sufficient detail about procedures and data so that the same procedures could be exactly repeated.[GFI16a]"

For replicability, all information, methods and data must be accessible in order to be able to repeat the experiment exactly. If the results match those presented, the work can be replicated. For reproducibility, as much information, methods and data as possible must be accessible in order to be able to repeat the experiment under slightly different conditions. If the results then match, they are reproducible.

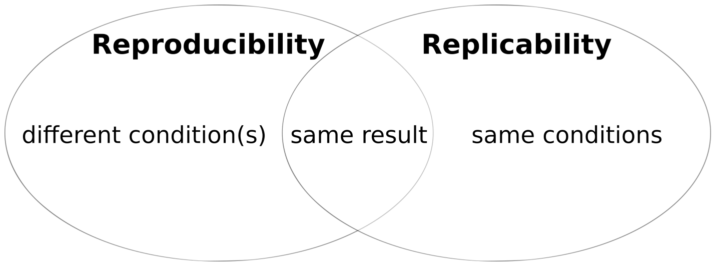


Figure 2.1: Reproducibility versus Replicability.

Drummond et al. states, that "Reproducibility requires changes; replicability avoids them. Although reproducibility is desirable [...] replicability, is one not worth having.[Dru09]". We only partially agree, because reproducibility also requires as much information as possible about the original work and the results that were achieved. If an author works transparently as far as possible, both replicability and reproducibility should ideally be achieved.

2.1.2 Challenges & best practices

Reproducibility challenges: There are various reasons why reproducibility and replicability are often not achieved. These criteria are usually not specified in the submission policies of most publishers. Furthermore, in order to achieve this, additional effort is required. It is also true that the more abstract the research field is, the more difficult it is to document the methods and processes in a comprehensible manner. In the area of machine learning, a model acquires the ability to assign the appropriate output for an input. Since this learning process is not necessarily subject to deterministic behavior, replicating an ML model is not trivial. This results in many influencing factors that need to be taken into account. These are identified and described in chapter 4.

Olorisade et al. state that "[...] it may sometimes be hard or even impossible to reproduce computational studies [...].[OBA17a]". Furthermore, they point out that "[...] the minimum standard expected of any computational study is for it to be replicable.". Note: We have changed the terminologies for reproducibility and replicability in the citations to match our definitions. This again shows the need to create at least replicable work. In the following sections we present some key technologies that simplify this. In addition, we give an overview of the current state of affairs in chapter **??**. Also, we show which steps can be taken to increase the reproducibility probability without having to do a lot of additional work.

Reproducibility best practices: Naturally, there are already some best practices that have prevailed. Regardless of ML-specific factors, we consider complete and traceable documentation (both in terms of code and methodologies), encapsulation to eliminate dependencies, and avoidance of non-open-source technologies to be most important. We aim to point out these best practices, together with ML-specific features, and want to support improving the quality of a work in this regard. In order to avoid overhead, we want to analyze the reproducibility probability as automatically as possible and give recommendations for action based on this.

For this purpose, we first identify in chapter 4 which influencing factors there are and in chapter 5 which of them can be detected automatically.

2.2 Containerization

Containerization: "Containerization is the process of creating, packaging, distributing, deploying, and executing applications in a lightweight and standardized process execution environment known as a container.[BSC17]". Another common synonym for this is operating system virtualization.

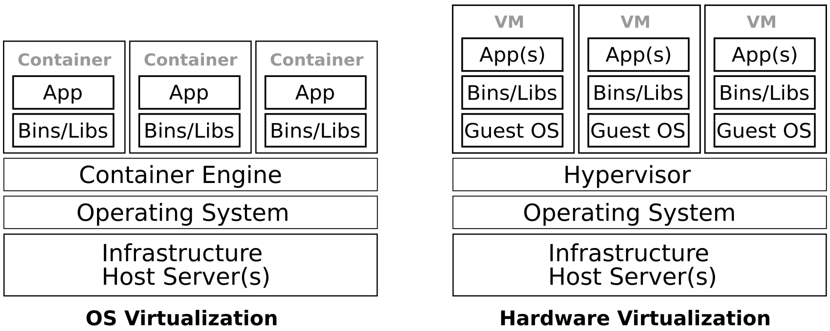


Figure 2.2: OS Virtualization versus Hardware Virtualization.

Both pursue the same goal, namely encapsulation, but on different layers. Containerization takes place on the application layer and therefore has less overhead since only the application and its dependencies are simulated. With hardware virtualization, a complete computer environment is simulated.

Above all, containerization has enjoyed great popularity for years, especially because it "[...] provides efficient, scalable and cost-effective resource management solutions in cloud infrastructures [...].[Wat+19]". This applies particularly to infrastructures that universities provide to their researchers and students. Thanks to open-source solutions such as Docker, this technology has also been established in the field of scientific work. Containerization can help eliminate the "works on my machine problem" and simplify the setup process. Since the dependencies associated with the application are also stored in the container, this concept has a positive contribution to reproducibility.

The terms Docker and Kubernetes are explained below. These technologies are leveraged by Binder, which combines the benefits of containerization with other helpful features.

2.2.1 Docker

A detailed description of how Docker works is not of great importance for our purposes. However, some terms should be clear, as they are of great importance from a reproducibility point of view.

Docker: Docker is a container engine used to create and manage containers on top of an operating system. Linux Kernel features like namespaces and control groups enable it to do so. It is an implementation of the concept of OS Virtualization.

Dockerfile: A Dockerfile is a text document that describes the steps required to create a Docker image. This is the first step in creating a Docker container.

Dockerimage: A Dockerimage is a snapshot of a virtual machine at a specific point in time. You can think of it as a digital image of a certain state. This image is immutable and can be duplicated and shared. It contains all the information and data needed to run as a container.

We have already listed possible areas of application for containerization. However, we would like to go into some properties that support reproducible research[Boe15].

On the one hand, a Docker image provides other researchers with all the data they need to replicate the development environment. This avoids the dependency hell and simplifies the setup of an experiment. In addition, containers are lightweight and portable. Docker takes care of the packaging and running of a container, so it can run on different machines without any issues. Also, one can look up all the necessary dependencies and variables to create this image centrally in the Dockerfile. There are of course many other container engines such as LXC[1] or podman[2]. However, Docker is currently the de facto standard.

2.2.2 Kubernetes

Kubernetes: Kubernetes is an open source platform which enables the automated operation of linux containers (e.g. docker containers). Groups of hosts can be combined into so-called clusters, which are then managed by Kubernetes.

Kubernetes allows computing-intensive applications to be distributed across multiple hosts, which is advantageous in different areas such as deep learning[Mao+20]. The self-healing ability is from an availability perspective also a reason for using Kubernetes, especially for microservice applications[Vay+19]. But numerous other services such as Binder, which will also be discussed in this thesis, use the combination of Kubernetes and Docker.

2.3 Jupyter Notebooks

Literate programming: The pyhilosophy behind literate programming was formulated by Donald E. Knuth in 1984. He stated that the *"time is ripe for significantly better documentation of programs, and that we can achieve this best by considering programs to be works of literature"*[Knu84]. His idea was that instead of focusing on programming what the computer should do, one should shift onto textually explaining to humans what we expect the computer should do.

Jupyter Notebook: A Jupyter Notebook can not only execute the contained source code. In addition to the code, its output is saved. It is also possible to use markdown elements (e.g. paragraph, figures and links) to add human-readable documentation. So it combines the idea of being able to execute code quickly and easily with the concept of literate programming.

One possible use case is lab notebooks, which are a log of scientific activity. These should contain every detail (e.g. hypothesis, experiments, and interpretation of results) which later can be used to replicate the work[Ker+18]. This is also known as an executable paper. Jupyter notebooks offer a great opportunity not only for researchers, but also for students.

[1] https://linuxcontainers.org accessed: 23.01.2022
[2] https://podman.io accessed: 23.01.2022

Due to their interactivity, complicated content can be conveyed well. This is particularly advantageous in the area of data science, artificial intelligence and machine learning[OBM15].

In our context, we have already explained that reproducibility requires that the methodologies, the code and the results are documented in a clear and understandable way. Jupyter notebooks provide all the necessary functionality for this. In addition, Binder also supports this technology. On the other hand, researchers are deterred by the additional effort that has to be expended to create a notebook and document it properly. They create no direct added value for the creator of a work, which is why it is often dispensed. However, we argue that the use of this technology makes sense for researchers as it can increase the probability of successful replication.

2.4 Binder

We assume that a researcher wants to pay attention to reproducibility and therefore undertakes both the specification of a configuration file (e.g. Dockerfile, requirements.txt or conda.env) and the creation of a Jupyter notebook. Binder provides the functionality to combine both approaches and create an interactive, replicable environment.

Binder: "Binder is a free, open source, and massively publicly available tool for easily creating sharable, interactive, replicable environments in the cloud [RW18]." Note: In order to keep our definitions consistent, we modified the definition slightly.

It can thus combine the concepts of encapsulation that containerization offers with the possibilities of Jupyter notebooks. Since reproducing an experiment requires the change to some conditions, the interactivity that Binder offers is also perfectly suited. This also applies to the general workflow when developing an ML model, which is based on incremental improvements. Our tool will suggest using Binder to encourage the use of this technology. Of course, reproducibility does not require the use of these technologies, but they greatly simplify the verification process. In addition, some influencing factors can be eliminated in this way.

2.4.1 BinderHub

BinderHub: BinderHub is a cloud service based on kubernetes that enables the sharing of replicable, interactive environments. These environments are generated from a repository using repo2docker. JupyterHub provides a scalable system with which users can authenticate themselves and interact with the created environment. It is not necessary to set up a BinderHub yourself, as a free infrastructure is provided at mybinder.org[3]. The generated binder badge can then be integrated into the ReadMe file so that third parties can use your repository quickly and easily.

[3] https://mybinder.org accessed: 24.01.2022

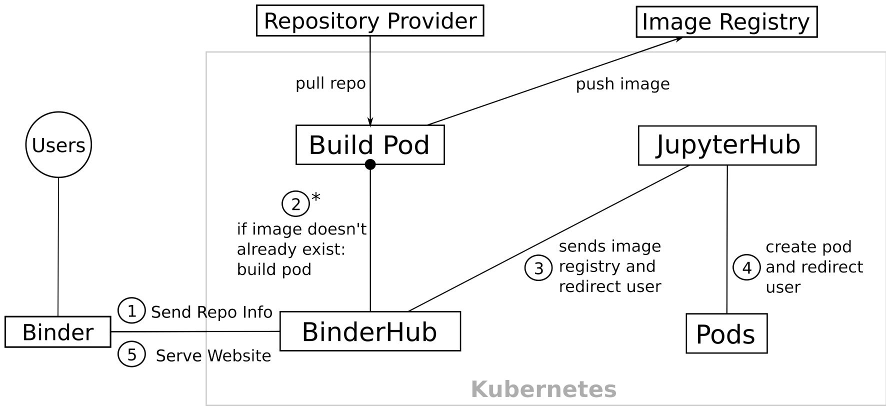


Figure 2.3: Adapted figure from the official BinderHub documentation.[4]

2.4.2 repo2docker

repo2docker: "The core feature of repo2docker is to fetch a repository at an arbitrary URL, inspect the repository for configuration files that define the environment needed to run its content, and build a container image based on the files in the repository.[For+18]"

BinderHub uses repo2docker to generate the Docker image if none is already present in the image registry.

The following configuration files[5] are currently supported:

- environment.yml
- Pipfile
- requirements.txt
- setup.py
- Project.toml
- REQUIRE
- install.R
- apt.txt
- DESCRIPTION
- postBuild
- start
- Dockerfile

The Dockerfile has the highest priority. Other configuration files will be ignored if one is present. If no Dockerfile is present, but mulitple others, they will be combined.

2.4.3 JupyterHub

JupyterHub: "JupyterHub is the best way to serve Jupyter Notebook for multiple users. It can be used in a class of students, a corporate data science group or scientific research group. It is a multi-user Hub that spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server.[6]" It not only works with Jupyter Notebooks but also with Python code in general.

A JupyterHub consists of 4 subsystems. On the one hand the hub which is the core of the system and connects the other subsystems with each other. An http proxy forwards requests

[4] `https://binderhub.readthedocs.io/en/latest/overview.html` accessed: 22.01.2022
[5] https://repo2docker.readthedocs.io/en/2021.08.0/config_files.html accessed: 22.01.2022
[6] https://jupyterhub.readthedocs.io/en/2.1.1 accessed: 22.01.2022

from the client to the hub. After a user successfully logs into the authentication service, the hub spawns a single-user Jupyter notebook server. This server provides the functionality to run the container. The hub then tells the proxy to forward user requests to this server. When using JupyterHub within a BinderHub, the proxy does not receive the requests directly from the user. Here the BinderHub switches on beforehand to ensure that an image for the repository is stored in the registry.

2.5 Machine learning workflow

The goal of an artifical intelligence (AI) is to be capable of imitating intelligent human behavior. Machine learning is part of the more general field of AI. Rather than teaching a computer human behavior directly, the concept here is to allow the computer to learn from data through experience and thus continually improve itself. ML is already being used today for things like chatbots, auto-correction or automatic translations. We want to use a possible ML model development workflow to explain the general procedure. After we have given a high-level overview of this process, the influencing factors identified in chapter 4 can be assigned to a process step.

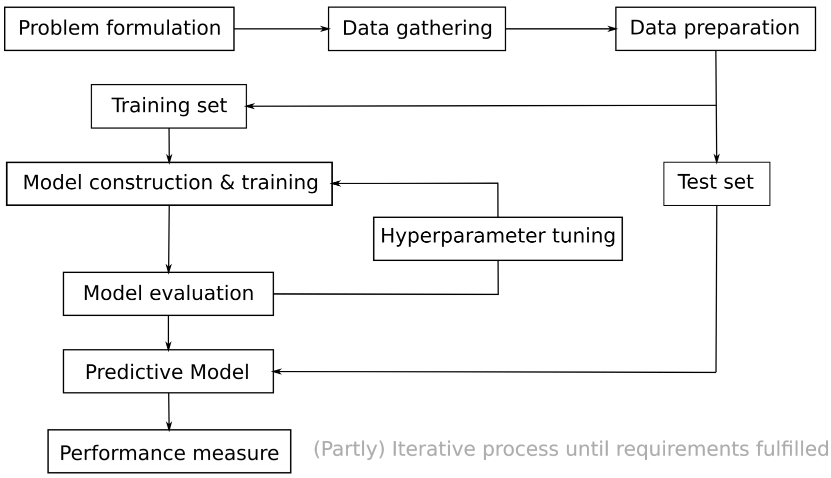


Figure 2.4: Abstracted machine learning workflow.

2.5.1 Data gathering & preparation

Matching the problem to be solved, a suitable data set is first selected as input. The content of the data record is of no further importance from the point of view of reproducibility. We are more interested in whether this data is publicly available.

Data preparation then ensures in several sub-steps that the data set is suitable as an input, as this significantly influences the ML model. As examples, we list the checking of the data quality, the comparison for uniform formatting, the substitution of missing values or the elimination of superfluous attributes as possible intermediate steps. It is therefore not sufficient to give third parties access to the original data set. It is important to document which preparation steps have been taken.

2.5.2 Train-test split

There are different ways to split the prepared data into different sets. The most common one is the train-test split. With this technique the data is splitted into two sub sets (e.g. 80/20 split). The training set is used for the model training process. After the model is trained it can then be evaluated using the test set. This split ensures that the model has never seen the data when evaluating. The train-test split is appropriate provided the dataset is not too small. In general, this technique is used for regression or classification problems and is suitable for any supervised learning algorithm.

An adaptation of the train test split is the n-fold cross validation method. The same technique is used here, but the process is repeated n times. For each run, a different block is selected as the test data set and an average is formed at the end. This reduces the risk of selecting non-representative test data. The result becomes more robust and the variance decreases.

2.5.3 Model construction & training

First, a machine learning algorithm is selected. In general, there are 3 different main categories of algorithms: supervised, unsupervised and reinforcement learning[Wan+17]. In supervised learning, a mathematical relationship is established between an input x and an output y. These relationships are then used to create a model which can also predict the appropriate output from an input. In the case of unsupervised learning, the input X is not labeled. This means that the algorithm must match the input to a suitable output without obtaining any previously known patterns or relationships. In reinforcement learning, a reward system decides how the algorithm should proceed. The reward can be positive for good matching or negative for bad matching. The model continues to improve from past mistakes.

Different ML algorithms have different hyperparameters. By setting the hyperparameters one can control the learning process. By setting a specific random seed one ensures, that when rerunning the model with a different algorithm or changed hyperparameters, every time the same training and test data set will be used. In this way, we can eliminate the influence of the used data split on the model performance.

2.5.4 Model evaluation

After a model has been trained it can be evaluated using the test data set. This set consists of data that the model has never seen before. Based on specified requirements, such as a baseline or previous iterations, a decision is then made as to whether the model is good enough. If it does not meet the expectations, one can, for example, adjust hyperparameters in a new iteration or change the general learning strategy to achieve a better result.

2.5.5 Performance measure

Based on the finished predictive model, new data can be evaluated. Typically, one notices the problem of underfitting in model evaluation, overfitting when running the model on new data (see figure 4.1). There are various performance metrics that can be measured. For example the F1 Score, Classification Accuracy or Logarithmic Loss to name a few. While the model is deployed, these can be measured continuously. Based on these metrics, a future version of this ML model can be improved.

3 Related work

We have been unable to find other scientific papers that classify factors influencing the reproducibility of machine learning experiments in terms of software-based detectability. In addition, our search for a tool that could detect such factors and then provide a report on the results did not yield anything. However, there are already several tools [Mor+21] that are supposed to improve the reproducibility probability in the area of machine learning such as Binder[1], CodeOcean[2], or Whole Tale[3]. In the following, you will get an overview of various scientific papers that have dealt with issues that are relevant to us.

Identification of reproducibility factors: Since our goal is to classify reproducibility factors that are relevant for machine learning experiments, we have to find them beforehand. To do this, we use existing knowledge and summarize the findings. However, influencing factors are ML-specific to different degrees. Every author of a scientific paper should be aware of general reproducibility hurdles [Eis18; SK21]. On the other hand, some factors only appear when software is part of the work [IT18b; MK17]. In addition, there are also factors that occur specifically in the ML area [Ban+21; GGA18; Liu+21; McD+19; OBA17a].

Classification of reproducibility factors: Tatman et al. [TVD18] defined three different levels of reproducibility which differ in the amount of information available. However, they have used an alternate definition of reproducibility which corresponds to our definition of replicability. Goodman et al. [GFI16b] proposed using a different definition in 2016. Because of the inconsistent use of the term reproducibility, they divided it into three different types: Methods, Results, and Inferential reproducibility. Methods Reproducibility can be equated with replicability. Results and inferential reproducibility are understood to mean reproducibility. Based on this work, Isdahl et al. [IG19] investigated how well ML platforms support out-of-the-box reproducibility, and Gundersen et al. [GK18] manually surveyed 400 research papers using these metrics. While all of these classifications focus on conceptual delimitation, we want to achieve something different. The classification in software-based detectable and undetectable factors should enable the implementation of a software tool. From our point of view, this is important because it should be as straightforward as possible to check an ML experiment for reproducibility.

Reviewing reproducibility manually: The use of checklists was recommended as part of the NeurIPS 2019 reproducibility program [Pin+20] and by Artrith et al. [Art+21]. This should enable the quality to be checked in a standardized manner concerning reproducibility before submitting a scientific paper. We also want to use some kind of checklist in order to document software-based undetectable factors. However, our goal is to reduce the use of manual review activities to the bare minimum. The main reason for this is that manual review processes do not scale.

[1] https://mybinder.org/
[2] https://codeocean.com/
[3] https://wholetale.org/

Estimating paper replicability: Yang et al. [YYU20] examined how machine learning can be used to estimate the probability with which a study can be replicated. Their motivation was the same as ours. The automation of processes saves time and money and at the same time offers the possibility of scaling. Researchers could use this to prioritize work that is more likely than others to be replicated. They developed a machine learning model that can conclude the probability of replicability from the narrative content of a paper. Their result was that papers that are less likely to be replicated have a higher frequency of unusual n-grams and a lower frequency of common n-grams. N-grams are n connected words with which you can check the writing style. According to the authors, this ML model could provide results that are comparable to those of the best manual methods currently available.

Summary: To identify reproducibility factors, we can fall back on a solid scientific basis from various disciplines. As far as the classification of these factors is concerned, things look different. The focus of the found works is the conceptual delimitation of reproducibility. However, we will still be able to scientifically justify the decision of whether a factor can be detected with the help of a software tool. Previous holistic approaches are based on manual reviews. We want people only to be involved in dealing with software-based undetectable factors. Our repository analysis tool for detectable factors, together with the solution for undetectable factors, should form a simple, reliable, and standardized holistic approach. By increasing the likelihood of successfully reproducing an ML experiment, its scientific relevance and informative value will be improved.

4 Identification of reproducibility factors

In this chapter, we identify factors affecting the reproducibility of machine learning experiments. To do this, we use relevant literature and collect this knowledge. For each of these factors, we explain what is meant by it and how it affects reproducibility.

This authentication step forms the basis of this thesis. The knowledge gained is used in Chapter 5 to classify the identified factors.

4.1 Source code availability & documentation

Public access to the source code of a machine learning experiment is the cornerstone for many analysis options based on it. Without its publication, only the methods described in the work are available, which is generally not enough in terms of reproducibility. Source code can be shared with in various ways. However, the most common way is to store relevant project files in a repository. In addition to the mere provision, it is also important that third parties can also understand and operate this code. Meaningful and helpful documentation is important for this [SK21]. Basically, code can be documented in different ways. In a *README* file, for example, the basic repository structure, setup and installation instructions or a brief description of the areas of application can be described. In addition, a *LICENSE* helps to clarify the conditions for using and expanding the software [GGA18]. Additionally, detailed documentation in source code files and adherence to a standardized coding style can help make it more human-readable. This in turn leads to the fact that ambiguities are eliminated and the contents of the repository can be more easily understood, which increases the chances of successful reproduction.

4.2 Hardware environment

The influence of the hardware used on reproducibility is manifold. Available resources can be very different, especially in the machine learning area. It can happen that one has no way of reproducing an experiment because he does not have access to enough computing power [IT18b]. In addition, there is also the long-term risk of wareouts in connection with extincted hardware. In the future, the components used may no longer be available, preventing an exact replication of the experiment. However, the description of the hardware used is still advisable, since the numerical accuracy varies between different CPUs, GPUs and TPUs [Jan20]. But it is more advisable to use techniques from the field of virtualization (see 2.2) instead of a textual description of the components used, since these minimize the effort to create the environment.

4.3 Software environment

Specifying the software dependencies in a separate configuration file is recommended as it reduces the effort of replicating the environment used. Here, it is important to ensure that the dependencies used can also be accessed by third parties. They must therefore be publicly available. To avoid compatibility problems, a strict declaration of the version used is recommended [IT18b]. In most cases, the creation of such a file does not represent a great deal of additional work. On the other hand, this simplifies the setup process and eliminates possible sources of error in this context, which could impede reproduction.

4.4 Out-of-the-box buildability

For us, if a repository is buildable out of the box, that means it can be used in a Binder environment without errors. We have shown the resulting benefits in XY. However, the use of this technology not only offers added value for third parties who want to examine the repository. The authors of an ML experiment are recommended to use it before or during the documentation and release phase [SK21]. Piccolo et al. have concluded that this technology can bring together the documentation capabilities of notebooks for literal programming with software encapsulation [PF16]. This allows quick and easy access to the software functionality to be replicated. The interactivity of Binder means that changes can be made directly to the code and the reproducibility can thus be tested.

4.5 Dataset availability & preprocessing

In 2.1.1, we have already found that reproducibility has a direct impact on reproducibility. For this reason, it is important that the original data set used is included. Where this data is stored only plays a minor role. It is important that it is permanently and publicly available. For this reason we advocate inclusion in the repository. However, in the ML space, it is not uncommon for datasets to take up a large amount of storage space. In these cases it is better to describe and link to this in the *README* within a separate dataset section. The disadvantage here is that the link may no longer be accessible at some point in the future. A well-known problem when dealing with data in this context are authorization issues, which prevent publication for security or other reasons [IT18b]. If possible, these datasets should be substituted with a publishable one.

If the data set was preprocessed, at least the methodology used must be explained, because an independent reviewer should be able to repeat this. However, it is better to either save the preprocessing mechanics in a file or even provide the modified data set. Robles et. al found out that there is still a need for action here, because although publicly available data was mostly used, the data set after the preprocessing step was only rarely included [Rob10].

4.6 Random seed control

At first glance, the elimination of any randomness seems desirable. But for some challenges in the machine learning area, this assumption is not correct. Randomness counteracts inherent biases and is useful for developing a generalized machine learning model. In order to use these advantages and still be able to guarantee reproducibility, it is necessary to use so-called random seeds [OBA17b]. These seeds initialize a pseudo random number generator (PRNG) with a starting value. If one uses this value again, the generated number is the same. Randomness can appear in many different places. In 2.5.3, we briefly described a possible use case in connection with a seed declaration. In summary, it is not desirable to avoid randomness, but to make it controllable and repeatable using fixed starting values (seeds).

4.7 Hyperparameter logging

Figure 2.4 shows an example machine learning workflow. Here it is important to understand that the development of an ML model is generally an iterative process. Before each round one can tweak some knobs to cause changes. Parameters that are manually chosen based on what has been learned from previous runs are called hyperparameters. The value of these parameters cannot simply be read from the data. Instead, these are chosen intuitively, so experience makes a well-chosen value more likely [Jan20]. After each iteration, it is measured whether the changes made have improved the accuracy of the predictions made by the new model compared to the previous configurations. Since usually only the final model remains at the end of the development process, the selected parameters are lost for all intermediate steps. But documenting each step in this iterative process may be necessary, especially for reproducibility [OBA17b]. It is therefore advisable to record these parameters in log files, for example.

4.8 Model serialization

Since we want to discuss Model serialization (MS), we must first clarify what serialization means. The goal of serialization is to convert an object or data structure, in our case an ML model, into a specific format. This artifact can then be stored anywhere and transformed back into the original object or data structure via deserialization. In the context of machine learning, for example, this can be useful when training a model is very resource-intensive. This means that this already trained model can be ported and used on a less powerful machine using serialization and deserialization. There are numerous ways in which this can be implemented. Pickle[1], joblib[2] or framework specific functionalities like *torch.save()* from pytorch[3] are available for this, just to name a few. Based on this technology, there are also data version control systems (e.g. DVC[4]), which can document the evolution of changing ML

[1] https://docs.python.org/3/library/pickle.html accessed: 04.04.2022
[2] https://joblib.readthedocs.io/en/latest/auto_examples/serialization_and_wrappers.html accessed: 04.04.2022
[3] https://pytorch.org/tutorials/beginner/saving_loading_models.html accessed: 04.04.2022
[4] https://dvc.org/doc accessed: 04.04.2022

models [Tsi+18]. This is particularly helpful from a reproducibility point of view, since the final models, and in the best case also the intermediate models, can be made accessible with it. This not only avoids possible sources of error during replication, but also minimizes the time required for this.

4.9 Research practices & experimental design

Successful reproducibility cannot be expected without comprehensible documentation of the research methods used and the basic experimental design. In the absence of a standardized reporting process for the results achieved, including meaningful metrics, it is difficult to assess the significance of the improvements indicated [Hen+18]. In addition, Gundersen et al. showed that in many cases the explanation of the research methods used, the specification of the problem and the associated hypothesis, the publication of the data set and source code, the results obtained and the prediction made were not discussed or specified in the paper [GGA18]. Based on these observations, they have defined the following recommendations as to what content should be included in a scientific paper:

Recommendations for Experiments:

- Hypotheses and Predictions
- Experimental Design
- Measure and Metrics
- Evaluation Protocol
- Results, Result Description & Analysis
- Workflow & Workflow Citation
- Executions
- Hardware Specification

Recommendations for AI Methods:

- Problem Description
- Conceptual Method
- Pseudocode

Compliance with these standards not only increases the likelihood that a work can be reproduced. These can also be used as a checklist when creating a paper, which leads to a higher quality. This also increases the scientific relevance and informative value in general.

4.10 Underfitting & Overfitting

Underfitting and overfitting are two problems that can arise in the field of ML that can be used to clarify the difference between replicability and reproducibility once again. Underfitting occurs when an ML model can neither model the training data nor generalize new data. This is a particular problem for replicability, because even if the data set used is available, there is a high probability that the results cannot be produced.

Overfitting describes the problem that a machine learning model has adapted too much to the test data used. The consequence of this is that the performance on new data deteriorates significantly. This is a problem from a reproducibility point of view because the results obtained will most likely change when the model has to process new unseen data. Therefore, the experimental findings of a paper should be based on performance results from different test data sets [Liu+21]. This ensures that the possible overfitting of the model is noticed before publication.

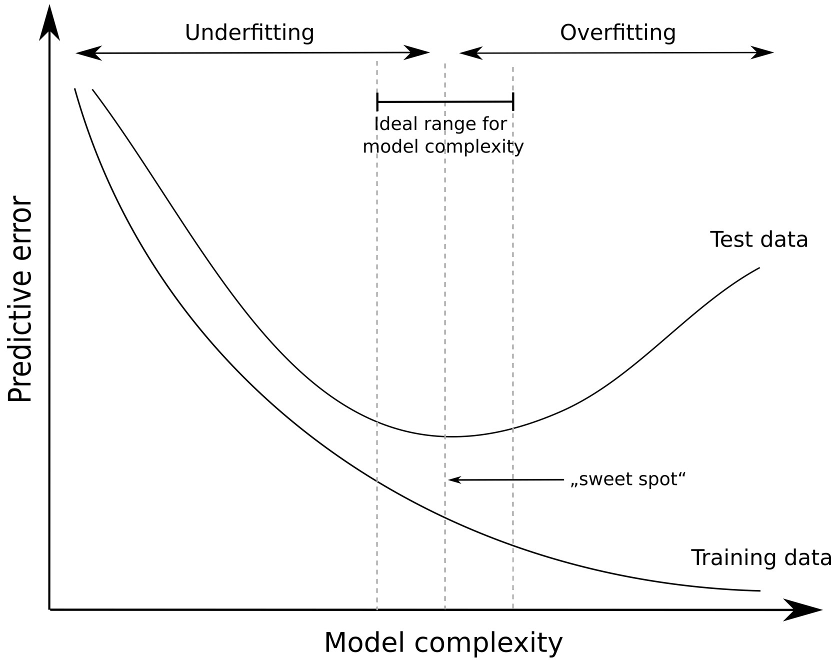


Figure 4.1: Adapted figure explaining Overfitting & Underfitting [Hea+15a].

These problems can be viewed graphically in Figure 4.1. The optimal case is also shown here. An ML model should perform equally well on both the test data and new data. Hence, one is looking for the so-called "sweet spot" between underfitting and overfitting. There are some techniques that can be used in practice to find this "sweet spot", such as using a validation dataset or resampling methods.

4.11 Knowledge gap

In order to reproduce an experiment, it must first be understood. Of course, this assumes that a third party has the relevant expertise. In the field of machine learning, however, this trivial statement can become a problem. ML is often a very interdisciplinary field. Particularly in the case of very complex research projects, many people from different areas or disciplines are usually involved.

The term "knowledge gap" means that different people have different levels of knowledge. This can become an issue given the discussed imbalance between an individual reviewer and a research group. When creating a paper and documenting the experiments, care should be taken to ensure that an individual reviewer can understand the observations made as easily as possible. Otherwise, replication or reproduction can be very difficult or even impossible [Eps+18].

4.12 Probability hacking

Probability hacking (p-hacking) means that prior to forming a hypothesis that one wishes to test, statistical tests are already performed on a data set until significant results are observed [Hea+15b]. It does not matter whether and which underlying effects are present since the hypothesis can be adapted to the measured values. The main reason why this is done is that some scientific publishers like to publish significant results, as these are often particularly effective in the media. P-hacking is a problem mainly because it is so difficult to detect.

Whether this is done intentionally is not relevant to us, as the consequences are the same in both cases. Probabilty hacking has a negative impact in the scientific environment because it allows third parties to draw false conclusions. In addition, these papers are inherently non-reproducible as the use of other underlying data will most likely not result in these reported significant results being observed [Cra17]. Significant evidence for the possible existence of p-hacking are results that differ greatly from previous work in a research area. Because it is so difficult to recognize, awareness is already an important first step.

4.13 Bias

Firstly, we want to point out that there is a difference between bias and chance variability [Ioa05]. Although the scientific approach is perfectly applied, chance variability can cause measurement results to be incorrect. Bias, on the other hand, refers to manipulations (intentional or unintentional) that can be made in both reporting and analyzing results.

There are many different types that can occur. In addition to recall bias, there is publication bias, confirmation bias or selection bias, just to name a few. Hudson et al. argues that unless any type of bias can be reliably identified through an assessment process, the reproducibility crisis is inevitable [Hud21]. The only way to counteract bias is to be aware of possible sources and try to eliminate them or at least reduce their influence.

5 Classification of reproducibility factors

Why do we need/do a classification in the first place? (basis of the tool) Not all identified factors can be detected.

Def. Factor

Def. Indicator

How to decide if a factor is detectable or not? (Indicator based arguementation; if we find an indicator then it is measurable)

Connection of factors and indicators? (figure)

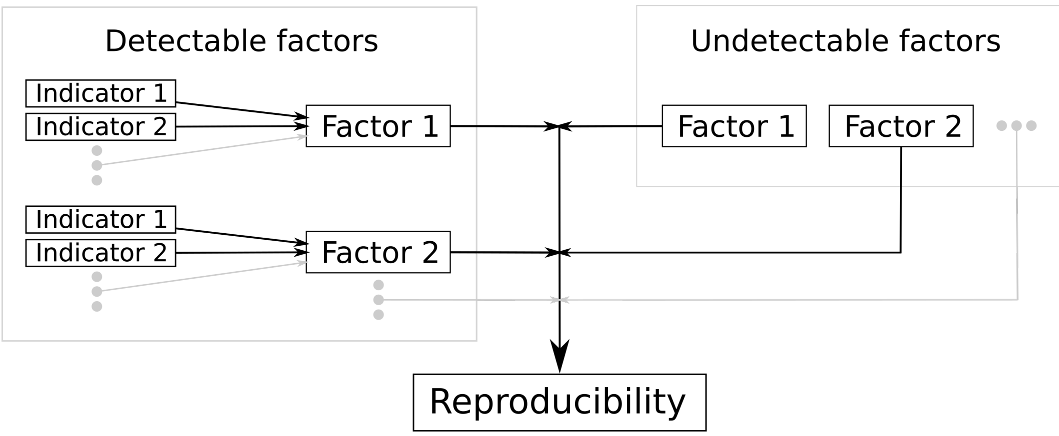


Figure 5.1: Connection of factors, indicators and reproducibility.

5.1 Detectable factors

In the following we deal with the factors that we classify as detectable. If an indicator can be found for a factor, it is basically detectable. In addition, we want to use a scale to illustrate how difficult it is to be able to measure a factor using software. Software Development Principle: Simple first

5.1.1 Assignment of indicators

The aim is to assign one or more possible indicators to each factor. (linking factor ¡-¿ indicator) This is the basis for the later traceability of the analyzed data to the factors. We will determine

Reproducibility indicator	**Reproducibility factor**
GitHub URL	Source code availability
License	Source code availability
Readme	Source code documentation
Readme - Length	Source code documentation
Source-code-comment-ratio	Source code documentation
Jupyter notebook header comment	Source code documentation
Jupyter notebook closing comment	Source code documentation
Source code quality (pep8 FOOTNOTE)	Source code documentation
Notes on hardware environment	Hardware environment
Config file	Software environment
All used library versions specified?	Software environment
BinderHub API Call Response	Out-of-the-box buildability
Readme - Binder Badge	Out-of-the-box buildability
Dataset folder candidates	Dataset availability
Dataset file candidates	Dataset availability
Notes on found file candidates	Dataset availability
Notes on dataset preprocessing	Dataset preprocessing
Notes on random seed control	Random seed control
Notes on hyperparameter declaration	Hyperparameter declaration
Artefacts of model-serialization	Model-serialization
Notes on model-serialization	Model-serialization
Readme - Paper link	Research practices & experimental design

Table 5.1: Reproducibility indicators with their corresponding indicator

which values of an indicator of a factor are good using a base line (Chapter cx Section xy). In addition, we will use a table to determine the weighting of individual indicators for some factors.

This figure shows the relationship indicator -¿ factor

5.1.2 Complexity scale

Based on the assigned indicators, we can create a scale that reflects how difficult (time investment + know-how etc.) it is to evaluate a factor.

This is helpful for the order in which we do the implementation. As the level of difficulty increases, this also implies increasing inaccuracy in our proof-of-concept implementation.

To create this scale, you not only have to identify the indicators, but also estimate how much effort it is to extract them from a repository. For this it is necessary to analyze the general structure of ML repos and to consider dependencies.

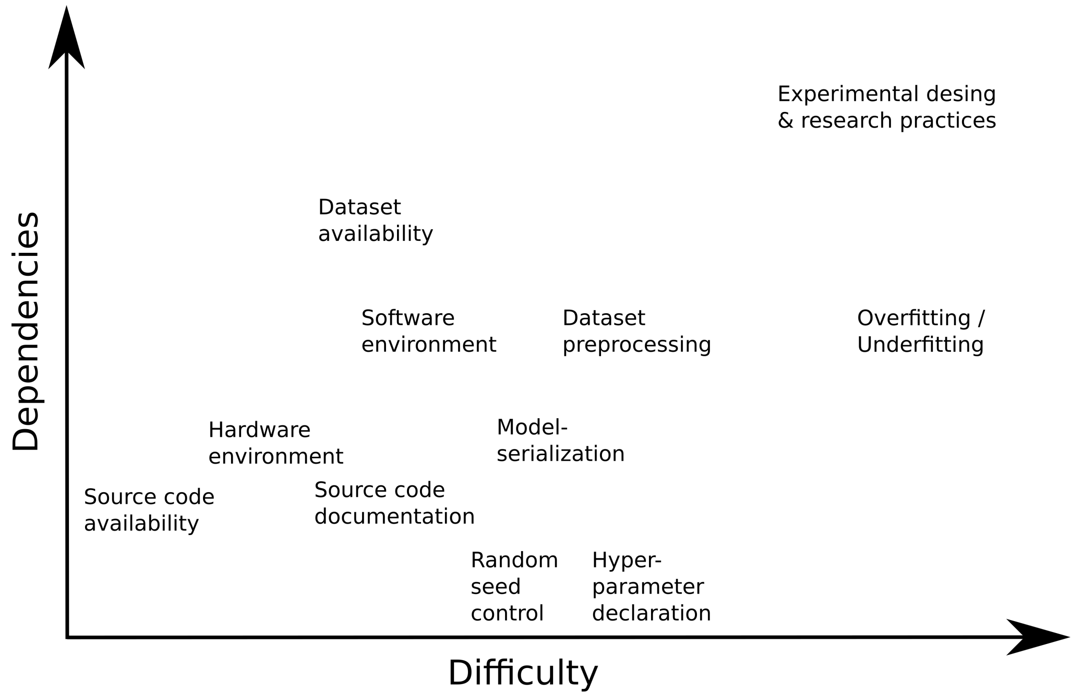


Figure 5.2: Complexity scale of the detectable reproducibility factors

SC Availability (given only license analysis), Hardware env (only keyword search / no standardized format on how to note specifications), Source code documentation (lots of properties, complex but all other things depend on analysis or use same functionality), Random seed control (keyword search in source code), Hyperparameter declaration (Keyword search in source code), Model-serialization (Artefacts search + keywords in source code), Software env (Config files analysis + source code analysis but all mostly standadized formats), Dataset availability (search for folders, files, links and notes), Dataset preprocessing (Notes + knowledge from dataset availability), Overfitting/Underfitting, Research practices & experimental design

5.2 Undetectable factors

Since no indicator was found for these factors, they are undetectable.

For each factor: What are the implications? What solutions are there?

6 Implementation

This chapter explains how we can use a software solution to automatically analyze and evaluate the factors identified in Chapter 4 and classified as detectable in Chapter 5. For this we present our proof of concept (PoC) reference implementation. We will go into more detail on possible limitations or future improvements in Chapter 8.

The tool described here will be available in a public repository[1]. The necessary requirements and installation steps are also documented there together with other helpful information in the provided readme file. In the following, we explain the conceptual considerations and thought processes that led to the creation of this implementation.

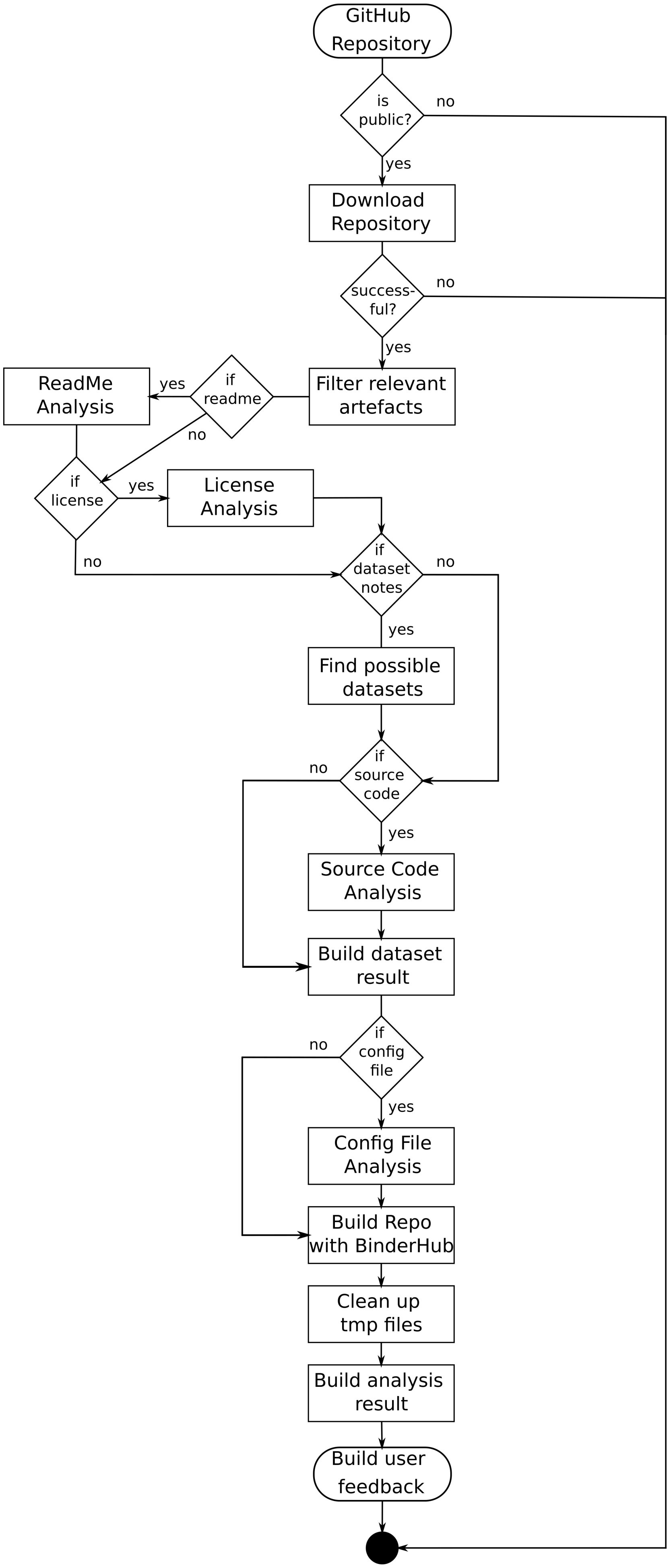
It should help authors and developers of machine learning experiments as well as reviewers of third-party work. A GUI is not required for this user group. We focused on the desired core functionality and opted for command line operation. Since Python is the programming language mainly used in the field of machine learning, we decided to use it as well. The aim is to increase the probability of reproducibility or to be able to estimate where possible problem areas are. As a result, high quality with regard to reproducibility can be ensured during development or before publication without great additional effort. In addition, a reviewer can quickly and easily examine properties of a machine learning repository in relation to reproducibility. This is made possible by the approach described in 6.2.

6.1 Architecture & Components

This section explains the chosen tool structure and the procedure to extract the values of the indicators listed in Table X from an ML repository. These measurements are required for the approach proposed in 6.2 in order to assign a representative evaluation value to each influencing factor. As mentioned, this is a PoC implementation, which is why we currently only support public repositories hosted on GitHub and programmed in Python (file extensions .py or .ipynb).

The following flowchart 6.1 provides a visual overview of the implemented modules and their execution order. The *main.py* file is not listed there, but it serves as an entry point and runs the individual analysis modules in the sequence described.

[1] https://git.fim.uni-passau.de/loosmartin/ml_repository_reproducibility_analysis_tool


GitHub Repository
is public?
no
yes
Download Repository
success-ful?
no
yes
Filter relevant artefacts
if readme
yes
ReadMe Analysis
no
if license
yes
License Analysis
no
if dataset notes
no
yes
Find possible datasets
if source code
no
yes
Source Code Analysis
Build dataset result
if config file
no
yes
Config File Analysis
Build Repo with BinderHub
Clean up tmp files
Build analysis result
Build user feedback

First, the URL of the GitHub repository to be downloaded locally is passed to the *repository_cloner* module. If it is already stored in the */tmp* folder, downloading it again is superfluous. Otherwise, the download will be started if the repository is publicly available and also has either a *master* or *main* branch.

If this is successful, the *filter_repository_artifacts (FRA)* module starts searching for relevant folders and files. These are then used by the respective analysis modules using get methods. At the end of each analysis file, the result is saved by the module so that it can be collected by the *result_builder* module at the end.

To find readmes, FRA uses string matching to search for files that contain "readme" in the name. These are then analyzed by *readme_analysis.py*. The length and the links used are saved in each case. All hyperlinks are also tested for functionality. In addition, it is checked whether a paper has been linked or if a binder badge is present. This is done for all files and the numerical results are averaged.

To detect license files, *filter_repository_artifacts.py* also uses the same approach as explained in the readme part above. The contents of the detected artifacts are then checked by the *license_analysis.py* whether in the text from the Free Software Foundation[2] (FSF) defined open-source licenses are mentioned (using keyword matching). Both open-source and non-open-source licenses are then stored in separate lists.

Next, the search for included datasets in the repository is started. From FRA, all folders that match a specified regex, along with files that have "data" in their name, are passed to the *dataset_analysis* module for further processing. All files found in the folders which do not end with .py, .ipynb or .md are saved together with the files with "data" in the name as so-called dataset candidates. For these candidates, it is then checked during the source code analysis whether they occur in the code. All candidates for which a match is found are assumed to be datasets.

For the analysis of source-code files, FRA collects all files with the file extension ".py" or ".ipynb". The *source_code_analysis.py* file calls the appropriate module for each file according to the file extension. Since we currently only support Python, at this stage of development it is always *python_source_code_analysis.py*. If a Python notebook is to be analyzed, it is converted into a .py file for further processing using nbconvert[3]. We are particularly interested in the code-comment-ratio, the pylint rating, the number of random seed declarations (and how many of them have a fixed seed) and a list of the library imports required by *config_analysis.py*. We also check the import statements and lines of code to see whether they contain hyperparameter-logging-relevant content. For imports, this affects the following libraries: "wandb", "neptune", "sacred" and "mlflow". In the code lines we look for relevant method calls. These were taken from the respective documentation of the libraries listed. Finally, we also look for indicators of model-serialization. For this we check the lines of code for calls to pickle.dump() or torch.save(). In addition, *source_code_analysis.py* also searches for model-serialization artifacts. We expect these to be either in a folder named ".dvc", the file name is "model" or have one of the following file extensions: ".dvc", ".h5", ".pkl" or ".model". (these artifacts are collected by FRA). In addition, we filter out the Python Stan-

[2] https://en.wikipedia.org/wiki/Comparison_of_free_and_open -source_software_licences accessed: 26.03.2022
[3] https://nbconvert.readthedocs.io/en/latest/

dard Libraries[4] and locally defined modules from all found imports, eliminate duplicates and test whether these are publicly accessible (using the pip_search functionality). The average of the numerical analysis results is then recorded and saved together with the other measured values in an overall result.

The *config_analysis* module receives from FRA all found configuration files (which match a specified regex) and a preprocessed list of all used library imports from *source_code_analysis.py*. The following configuration file types are currently supported: requirements.txt, environment.yml, conda.yml, and dockerfile. From the files found, all library declarations are saved in a list. In addition, it is checked how many of them have been specified strictly (==). Then it is analyzed how many of the relevant imports used in the source code were specified in the configuration file and the results are saved.

In the *binderhub_call* module, the user must specify a BinderHub IP or URL at the appropriate place in the Python file (if the associated out-of-the-box buildability factor should be checked) before the repository analysis is started. For this specification, it is first checked whether the BinderHub can be reached. If it is not available, further analysis regarding the buildability is terminated and the result recorded. If successful, a build request is made via the API interface (*<BinderHub URL>/gh/build/<repository-owner>/<repository-name>*). The result of this request can be either ready or failed.

Finally, *result_builder* collects the results of the analysis modules and forms an overall result from them, which is then saved in a list for creating the feedback and persistently in a CSV file (e.g. for debugging purposes).

The approach for creating user feedback based on the overall analysis result is explained in section 6.2.

[4] https://docs.python.org/3/library/ accessed: 26.03.2022

6.2 Factor-Indicator-Connection

In the last section, we explained how the tool works and how the individual Python modules interact with each other (leaving aside the Feedback builder module). At this stage of development, the tool can analyze a repository and store the measured values in a CSV file for manual analysis. However, our goal is to provide helpful feedback on each identified reproducibility factor based on these measurements. Hence, it is necessary to link the detectable factors theoretically defined in Chapter 5 with the associated indicators measured by the analysis tool. In order to achieve this goal, the following questions must be answered:

1. If a factor is determined by several indicators, how can the measured values be combined into an overall rating? Each indicator has its own range of values. Therefore, each value must be converted to the same value range before weighting.
2. Which of the indicators examined are statistically significant? If the representative value of an indicator is very similar for both reproducible and non-reproducible repositories, no statement can be made about the probability of reproducibility with this indicator.
3. Does a measured indicator value indicate a reproducible or non-reproducible repository? In order to be able to answer this, we need comparative values.
4. If a factor is determined by several indicators, how much influence do they each have on the factor? Therefore, these indicators must be weighted. In order to determine these weights, we look at the comparative values determined for reproducible and non-reproducible repositories and look at the distances between the values. Indicators that show a higher deviation should have a higher weight.

The first problem is dealt with in 6.2.1. Then, the questions 2 - 4 are answered in subsection 6.2.2.

As already mentioned, for each indicator we need comparative values for both reproducible and non-reproducible repositories. Therefore, twenty reproducible and ten non-reproducible repositories (TODO: LINK TO PART B DATASET) were gathered and analyzed using the tool to retrieve the representive values. More repositories for each set would be better and more representive. But due to the fact, that we had to analyze them manually, more was not possible in the given time span.

The *feedback_builder* module is the implementation of the concepts presented in the following subsections, which assigns a range normalized and weighted value to each indicator. Based on the determined comparative values, it can also be estimated for each indicator whether this indicates a reproducible or non-reproducible repository.

Using this assessment, a feedback file is created as an output. In addition to all factor and indicator ratings, this also contains the determined weightings, if applicable. In addition, textual feedback is given for each indicator, which enables users to assess the probability of successful reproducibility, suggest improvements, identify problem areas or explain the limits of the analysis tool.

6.2.1 Min-Max normalization

Using normalization, we want to make it possible to merge all measured indicator values of a factor. The Min-Max normalization is particularly suitable for our problem since both the minimum and the maximum acceptable values of different indicators can be different. We adapt the formula used by Gajera et al. [Gaj+16] to calculate the value within a new range:

$$Y = \frac{(x - min(d)) * (max(n) - min(n))}{max(d) - min(d)} + min(n)$$

where

$$\begin{aligned} Y &= \text{Indicator value in new range} \\ x &= \text{Original indicator value} \\ min(d) &= \text{Lowest value in original range} \\ max(d) &= \text{Highest value in original range} \\ min(n) &= \text{Lowest value in new range} \\ max(n) &= \text{Highest value in new range} \end{aligned}$$

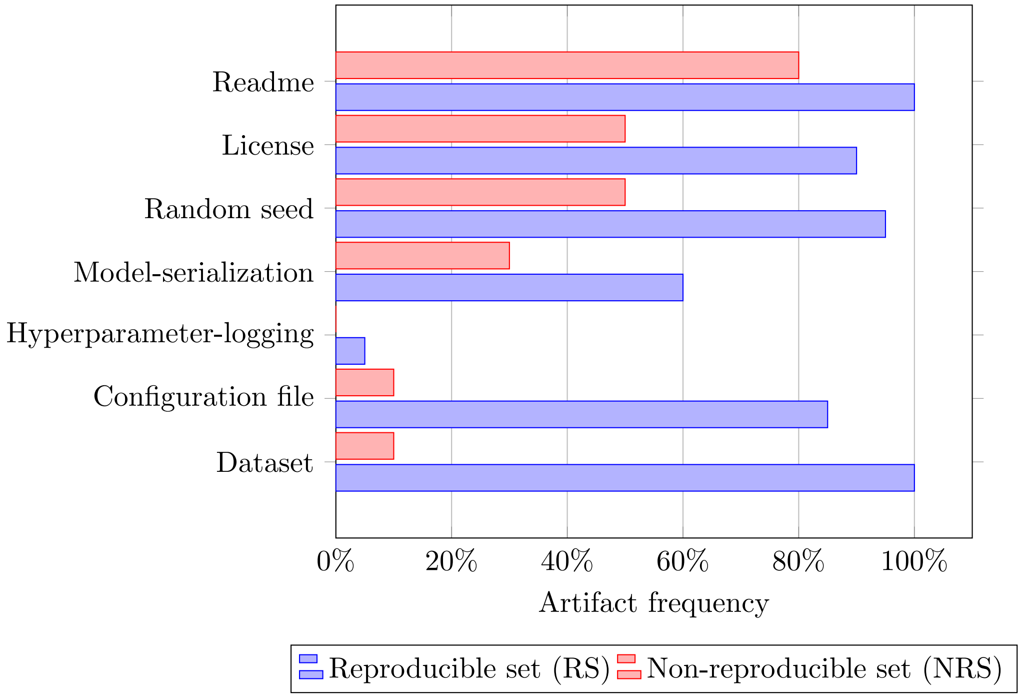
Most important for our approach to connecting factors with indicators are *min(d)* and *max(d)*. In subsection 6.2.2, we extract representative values for some of the indicators for both reproducible and non-reproducible repositories. We then use these values for the Min-Max normalization and assign the respective value *min(d)* or *max(d)*. In addition, we determine for *min(n) = 0* and *max(n) = 1*. Every measured value is within this new, uniform range after normalization.

6.2.2 Representative indicator values & weights

In this section, we compare two sets each containing ML repositories. One of the sets consists of exclusively reproducible (RS) the other non-reproducible (NRS) entities. We want to eliminate indicators that do not allow any conclusions about reproducibility and additionally determine representative values for all relevant indicators for both sets. In addition, the difference between these values is used to determine a suitable weighting in each case.

Using figure 6.2, we first consider which analyzable artifacts a repository of the respective set contains on average. We observe, that the set of reproducible repositories perform across all categories significantly better. The exception is the existence of a readme artifact. This is not surprising, since this file can also be created during repository creation by setting a check mark (which is often already preselected).

Figure 6.2: Comparison between two sets of ML repositories regarding the artifact frequency



The following steps are considered for each indicator of a factor:

1. Assessment of whether conclusions about reproducibility are possible.
2. Determining whether representative values are necessary.
3. Assignment of *min(d)* and *max(d)*.
4. After steps 1 - 3 have been carried out for all indicators of a factor, we determine their respective weighting. If the factor is determined by only one indicator, its weight is 1 (maximum weight).

All weightings of the indicators of a factor together have the value 1. In connection with normalized values, this has the advantage that, multiplied by the weights, they can simply be added up. An intermediate step in which the added weighted and normalized indicator values are divided by the added indicator weights is therefore superfluous.

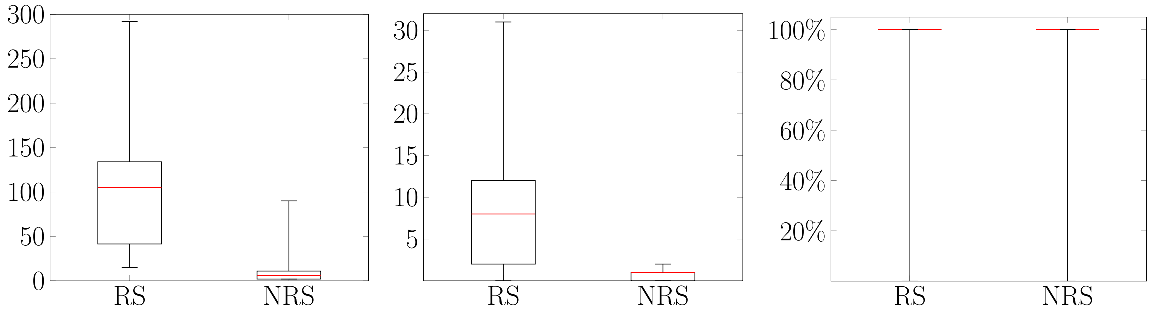
`Source code availability & documentation:` To assess this factor, we examine a total of six indicators extracted from three different artifact types:

- *Readme Artifacts:* Average readme length, the number of average readme links as well as the percentage of accessible readme links.
- *License Artifacts:* Presence of an (open source) license.
- *Source Code Artifacts:* Average code comment ratio and average pylint rating

In order to simplify the weighting, we first summarize the indicators from the readme analysis and form a partial rating from them.

Readme rating: From Figure 6.2 we can see that the mere existence of a readme file does not mean anything. This artifact type was contained in almost all cases in both the RS and the NRS. However, differences become apparent as soon as we compare the measured values of both sets for the indicators.

Figure 6.3: Comparison of the measured Readme relevant indicator values



(a) Length (AVG) **(b)** Links (AVG) **(c)** Accessible links (PCT)

From figure 6.3 it is clear that readme-files in the RS are significantly longer than those in the NRS. This observation applies analogously to the number of hyperlinks used, although the difference there is somewhat smaller. All found links of both sets were reachable. Since no differences could be determined here, this indicator probably cannot be used to draw any conclusions about reproducibility. However, since a hyperlink only makes sense if it works as intended, we combine these two indicators.

Representative comparison values are required for the two remaining indicators in order to be able to make a statement about the measured values. We choose the value of the median of the NRS as *min(d)* for both the length and the number of accessible hyperlinks. The median of the RS is chosen for *max(d)*.

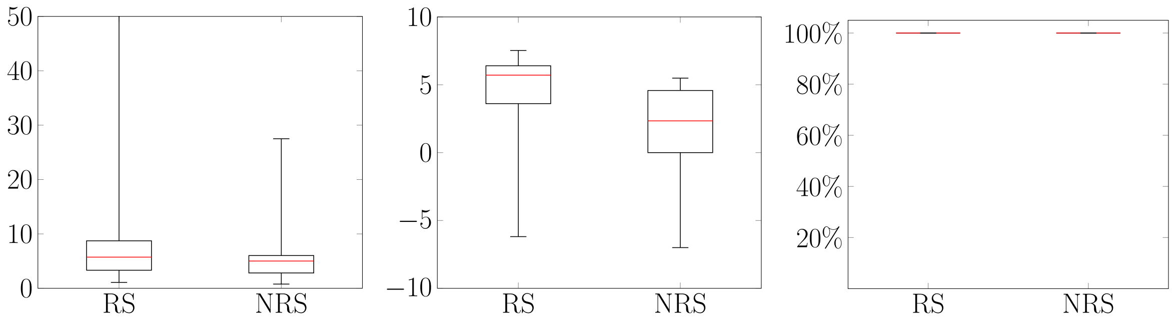
When determining the weighting, we could rely purely on the respective differences in the indicator values in the comparison of the two sets. However, since these are very similar, we add a conceptual consideration. In this context, the source code availability & documentation factor primarily refers to the documentation. It can be assumed that the contents of a readme serve precisely this purpose. Since a hyperlink is not necessarily included for documentation purposes, we argue that the length of the readme should be given significantly more weight. The following classification results from this observation:

Table 6.1: Readme: Representative values & weights

Indicator	min(d)	max(d)	Weight
Length	6	105	0.8
Accessible hyperlinks	1	8	0.2

Overall rating: Now let's turn our attention to the indicators of the other artifact types. The figure 6.5 shows that the median of the code comment ratio (CCR) is almost identical for both sets. Furthermore, the ratio for the RS is even more unfavorable from a reproducibility point of view, since fewer comment lines were found per line of code. It can therefore be assumed that this indicator does not allow any conclusions to be drawn. The Min-Max normalization cannot be used for the code-comment ratio because, based on these observations, no meaningful values can be chosen for *min(d)* and *max(d)*. Looking at the differences of the pylint rating, one comes to similar conclusions. While the value of the $^1/_4$ quartile of the NRS can be chosen for *min(d)* and the median of RS for *max(d)*, the value difference is not particularly significant. However, we argue that a high (greater than the $^3/_4$ quartile of the RS) CCR has a negative effect on the understandability of source code. Therefore, we assign a rating of 1 to the code-comment-ratio if it is lower than this threshold. For larger values we calculate the rating with *1 - normalized value* (with *min(d)* = $^3/_4$ *quartile value of RS* and *max(d)* = *2 * min(d)*). A standardized coding style should also be encouraged. For these reasons, we want to include these indicators when determining the overall rating. However, this is taken into account when the weightings are distributed.

Figure 6.5: Comparison of the remaining measured SCAAD indicator values



(a) Code-comment-ratio (AVG) **(b)** Pylint rating (AVG) **(c)** Open-source license (PCT)

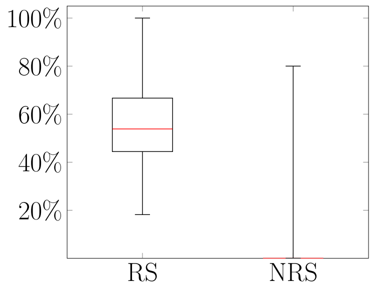
Since 100% of the licenses were open-source for both the RS and the NRS, this indicator cannot have a direct impact. In connection with the findings from figure 6.2, however, the conclusion can be drawn that a reproducible repository has a significantly higher probability of including an open-source license. Since not specifying a license is bad practice and specifying a non-open source license can lead to reproducibility problems, we want to include this indicator in the overall rating. If no or a non-open-source license is detected, the rating of this indicator is 0, otherwise 1. The following classification results from the observations made:

Table 6.2: SCAAD: Representative values & weights

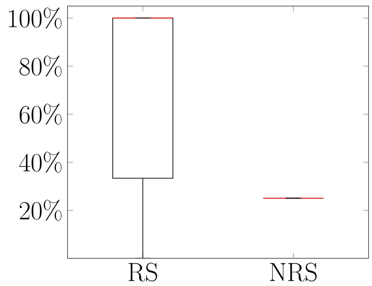
Indicator	min(d)	max(d)	Weight
Readme	-	-	0.5
License	-	-	0.3
Code-comment-ratio	8.73	17.46	0.1
Pylint rating	0	5.71	0.1

`Software environment:` As figure 6.2 shows, the existence of a configuration file indicates reproducibility by itself. But we also want to check whether and what influence the content and quality of these files have. In this context, relevant information from the source code files must also be used for the assessment. Therefore, we analyze the properties shown in figure 6.7.

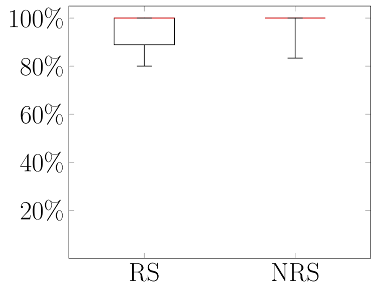
Figure 6.7: Comparison of the measured software environment indicator values



(a) Source code imports in configuration file (PCT)



(b) Strict dependency declarations (PCT)



(c) Publicly accessible libraries in source code (PCT)

When comparing how many of the imports used in the source code (excluding standard python libraries and local modules) were defined in the configuration file, a clear difference can be seen. It can also be observed that dependency declarations of reproducible repositories are usually version strict. However, the measured values of both repository sets in relation to how many of the libraries used in the source code are publicly accessible do not allow any conclusions to be drawn about the reproducibility property. Nevertheless, the use of libraries that are not open to the public is a hindrance and should therefore be reflected in the assessment of this factor. Since all specified key figures are calculated as percentage values, this results in: *min(d) = 0* and *max(d) = 100*. We argue that it is most important to include all relevant libraries used in the source code in the configuration file. In addition, a third party can only reliably access the functionality of these imports if they are publicly accessible. Finally, strict declarations avoid possible compatibility problems. These arguments are the basis of the chosen weightings. This results in the following classification:

Table 6.3: Software environment: Representative values & weights

Indicator	**min(d)**	**max(d)**	**Weight**
Source code imports in configuration file	0	100	0.6
Strict dependency declarations	0	100	0.2
Publicly accessible libraries in source code	0	100	0.2

`Dataset availability & preprocessing:` Firstly, we look at figure 6.2 again. We found a data set for all repositories in the RS. In contrast, this was observed in only every tenth in the NRS. From this, it can be deduced that the existence of a data set indicates reproducibility. It is important to note at this point that this is a PoC implementation. For this reason, the functions for preprocessing the data set are not yet supported.

In the dataset analysis module we identify possible data set files. We are referring to them as data set file candidates (DSFC). The source code analysis module then checks which of these candidates is used in the code. We assume that matches are relevant files containing the data set. This is our only meaningful indicator for this factor. It is therefore only examined whether at least one of the identified DSFC is mentioned in the source code. If so, we rate this factor with the maximum value of 1. If no match is found, we assume that no data set is included in the repository and we assign the minimum value of 0.

`Random seed:` Almost all repositories from the RS have taken this influencing factor into account (see figure 6.2). For the NRS, this only applies to 50%. A statistical connection can therefore be established. In addition to the declaration, we also measured how many of these contain a fixed random seed. Our analysis has shown that there are no differences in this respect when comparing these sets. If a random seed was defined, it was defined with a fixed value in all cases. Nevertheless, from a reproducibility point of view, we want to make sure that no seed is random. To determine a factor rating we choose: *min(d) = 0* and *max(d) = number of random seed declarations*. If no seed was used, the rating is 1, since then there is no randomness in this regard.

`Model-serialization & Hyperparameter-logging:`

As figure 6.2 shows, most reproducible repositories used model-serialization (MS). On the other hand, one can also see that hyperparameter logging (HPL) does not play a major role. We still want to keep this factor, because the tool should not only consider popular technologies or technologies that are often used by reproducible repositories, but all helpful possibilities.

We summarize these two factors under this point, as they both use the same feedback generation mechanics. If at least one relevant artifact was detected, the rating is 1, otherwise 0.

`Out-of-the-box buildability:` We left this factor out of the analysis because a repository can either be buildable or not buildable, regardless of the measured values. In any case, conclusions about the reproducibility property can only be made to a limited extent with this factor. The main benefit is the simplification of the reviewing process if a binder build is available or at least possible. Why the use is recommended in this context was discussed in 2.4. The inclusion in the feedback file is primarily intended to encourage the use of binder or similar technologies.

7 Evaluation

Why important?

Evaluate our hypothesis

7.1 Evaluation setup

Uni-Ressources

How were the two sets gathered?

Input (APPENDIX)

7.2 Statistical evaluation

For each of the two evaluation sets: for each factor rating: calculate (median, 1/4 quartile, 3/4 quartile, min, max) like in implementation part and create boxplots to compare the results for each factor between the two sets

Output (FOR EVERY FACTOR -¿ TABLE WITH RETRIEVED VALUES + INTERPRETATION)

8 Discussion

FINDING, EXPECTATIONS, etc.

8.1 Limitations

Ground truth base line input size small

Evaluation input small

python language only

only if we assign indicators correctly can they be evaluated correctly. bugs?

8.2 Future work

list of possible future work that can build on this work (or profit from it).

9 Conclusion

summary of findings etc.

Bibliography

[ACP19] Rob Ashmore, Radu Calinescu, and Colin Paterson. "Assuring the machine learning lifecycle: Desiderata, methods, and challenges". In: *arXiv preprint arXiv:1905.04223* (2019).

[ANK18] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. "Machine learning from theory to algorithms: an overview". In: *Journal of physics: conference series*. Vol. 1142. 1. IOP Publishing. 2018, p. 012012.

[Art+21] Nongnuch Artrith et al. "Best practices in machine learning for chemistry". In: *Nature Chemistry* 13.6 (2021), pp. 505–508.

[Bak16] Monya Baker. "Reproducibility crisis". In: *Nature* 533.26 (2016), pp. 353–66.

[Ban+21] Vishnu Banna et al. "An Experience Report on Machine Learning Reproducibility: Guidance for Practitioners and TensorFlow Model Garden Contributors". In: *arXiv preprint arXiv:2107.00821* (2021).

[BD17] Rabi Behera and Kajaree Das. "A Survey on Machine Learning: Concept, Algorithms and Applications". In: *International Journal of Innovative Research in Computer and Communication Engineering* 2 (Feb. 2017).

[Boe15] Carl Boettiger. "An introduction to Docker for reproducible research". In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 71–79.

[BSC17] Tom van den Berg, Barry Siegel, and Anthony Cramp. "Containerization of high level architecture-based simulations: A case study". In: *The Journal of Defense Modeling and Simulation* 14.2 (2017), pp. 115–138.

[Cra17] Harry Crane. "Why'Redefining statistical significance'will not improve reproducibility and could make the replication crisis worse". In: *Available at SSRN 3074083* (2017).

[Dey16] Ayon Dey. "Machine learning algorithms: a review". In: *International Journal of Computer Science and Information Technologies* 7.3 (2016), pp. 1174–1179.

[Dru09] Chris Drummond. "Replicability Is Not Reproducibility: Nor Is It Good Science". In: *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML* (2009).

[Eis18] DA Eisner. "Reproducibility of science: Fraud, impact factors and carelessness". In: *Journal of molecular and cellular cardiology* 114 (2018), pp. 364–368.

[Eps+18] Ziv Epstein et al. "Closing the AI knowledge gap". In: *arXiv preprint arXiv:1803.07233* (2018).

[For+18] Jessica Forde et al. "Reproducible research environments with repo2docker". In: (2018).

[Gaj+16] Vatsal Gajera et al. “An effective Multi-Objective task scheduling algorithm using Min-Max normalization in cloud computing”. In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. 2016, pp. 812–816. DOI: 10.1109/ICATCCT.2016.7912111.

[GFI16a] Steven N. Goodman, Daniele Fanelli, and John P. A. Ioannidis. “What does research reproducibility mean?” In: *Science Translational Medicine* 8.341 (2016), 341ps12–341ps12. DOI: 10.1126/scitranslmed.aaf5027. URL: https://www.science.org/doi/abs/10.1126/scitranslmed.aaf5027.

[GFI16b] Steven N Goodman, Daniele Fanelli, and John PA Ioannidis. “What does research reproducibility mean?” In: *Science translational medicine* 8.341 (2016), 341ps12–341ps12.

[GGA18] Odd Erik Gundersen, Yolanda Gil, and David W Aha. “On reproducible AI: Towards reproducible research, open science, and digital scholarship in AI publications”. In: *AI magazine* 39.3 (2018), pp. 56–68.

[GHS19] Daniel Greene, Anna Lauren Hoffmann, and Luke Stark. “Better, nicer, clearer, fairer: A critical assessment of the movement for ethical artificial intelligence and machine learning”. In: *Proceedings of the 52nd Hawaii international conference on system sciences*. 2019.

[GK18] Odd Erik Gundersen and Sigbjørn Kjensmo. “State of the art: Reproducibility in artificial intelligence”. In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[Hea+15a] Megan L Head et al. “The extent and consequences of p-hacking in science”. In: *PLoS Biol* 13.3 (2015), e1002106.

[Hea+15b] Megan L Head et al. “The extent and consequences of p-hacking in science”. In: *PLoS biology* 13.3 (2015), e1002106.

[Hen+18] Peter Henderson et al. “Deep reinforcement learning that matters”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[Hud21] Robert Hudson. “Should we strive to make science bias-free? A philosophical assessment of the reproducibility crisis”. In: *Journal for General Philosophy of Science* 52.3 (2021), pp. 389–405.

[IG19] Richard Isdahl and Odd Erik Gundersen. “Out-of-the-box reproducibility: A survey of machine learning platforms”. In: *2019 15th international conference on eScience (eScience)*. IEEE. 2019, pp. 86–95.

[Ioa05] John PA Ioannidis. “Why most published research findings are false”. In: *PLoS medicine* 2.8 (2005), e124.

[IT18a] Peter Ivie and Douglas Thain. “Reproducibility in scientific computing”. In: *ACM Computing Surveys (CSUR)* 51.3 (2018), pp. 1–36.

[IT18b] Peter Ivie and Douglas Thain. “Reproducibility in scientific computing”. In: *ACM Computing Surveys (CSUR)* 51.3 (2018), pp. 1–36.

[Jan20] PS Janardhanan. “Project repositories for machine learning with TensorFlow”. In: *Procedia Computer Science* 171 (2020), pp. 188–196.

[Ker+18] Mary Beth Kery et al. "The story in the notebook: Exploratory data science using a literate programming tool". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–11.

[Knu84] Donald Ervin Knuth. "Literate programming". In: *The computer journal* 27.2 (1984), pp. 97–111.

[KS15] Ron S Kenett and Galit Shmueli. "Clarifying the terminology that describes scientific reproducibility". In: *Nature methods* 12.8 (2015), pp. 699–699.

[Lhe+17] Alexandra L'heureux et al. "Machine learning with big data: Challenges and approaches". In: *Ieee Access* 5 (2017), pp. 7776–7797.

[Liu+21] Chao Liu et al. "On the Reproducibility and Replicability of Deep Learning in Software Engineering". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.1 (2021), pp. 1–46.

[Mao+20] Ying Mao et al. "Speculative container scheduling for deep learning applications in a kubernetes cluster". In: *arXiv preprint arXiv:2010.11307* (2020).

[McD+19] Matthew McDermott et al. "Reproducibility in machine learning for health". In: *arXiv preprint arXiv:1907.01463* (2019).

[McN14] Marcia McNutt. *Journals unite for reproducibility.* 2014.

[MK17] Lech Madeyski and Barbara Kitchenham. "Would wider adoption of reproducible research be beneficial for empirical software engineering research?" In: *Journal of Intelligent & Fuzzy Systems* 32.2 (2017), pp. 1509–1521.

[Mor+21] Marçal Mora-Cantallops et al. "Traceability for Trustworthy AI: A Review of Models and Tools". In: *Big Data and Cognitive Computing* 5.2 (2021), p. 20.

[OBA17a] Babatunde K Olorisade, Pearl Brereton, and Peter Andras. "Reproducibility in machine Learning-Based studies: An example of text mining". In: (2017).

[OBA17b] Babatunde K Olorisade, Pearl Brereton, and Peter Andras. "Reproducibility in machine Learning-Based studies: An example of text mining". In: (2017).

[OBM15] Keith O'Hara, Douglas Blank, and James Marshall. "Computational notebooks for AI education". In: *The Twenty-Eighth International Flairs Conference*. 2015.

[Pan99] Jiantao Pan. "Software testing". In: *Dependable Embedded Systems* 5 (1999), p. 2006.

[PF16] Stephen R Piccolo and Michael B Frampton. "Tools and techniques for computational reproducibility". In: *Gigascience* 5.1 (2016), s13742–016.

[Pin+19] Joelle Pineau et al. "ICLR Reproducibility Challenge 2019". In: *ReScience C* 5.2 (2019), p. 5.

[Pin+20] Joelle Pineau et al. "Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program)". In: *arXiv preprint arXiv:2003.12206* (2020).

[Ple18] Hans E. Plesser. "Reproducibility vs. Replicability: A Brief History of a Confused Terminology". In: *Frontiers in Neuroinformatics* 11 (2018). ISSN: 1662-5196. DOI: 10.3389/fninf.2017.00076. URL: https://www.frontiersin.org/article/10.3389/fninf.2017.00076.

[Rob10] Gregorio Robles. "Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings". In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE. 2010, pp. 171–180.

[RW18] Benjamin Ragan-Kelley and Carol Willing. "Binder 2.0-Reproducible, interactive, sharable environments for science at scale". In: *Proceedings of the 17th Python in Science Conference (F. Akici, D. Lippa, D. Niederhut, and M. Pacer, eds.)* 2018, pp. 113–120.

[Sch+18] Sebastian Schelter et al. "On challenges in machine learning model management". In: (2018).

[SK21] Sheeba Samuel and Birgitta König-Ries. "Understanding experiments and research practices for reproducibility: an exploratory study". In: *PeerJ* 9 (2021), e11140.

[SP10] Robin Sommer and Vern Paxson. "Outside the closed world: On using machine learning for network intrusion detection". In: *2010 IEEE symposium on security and privacy*. IEEE. 2010, pp. 305–316.

[SSJ18] K. Shailaja, B. Seetharamulu, and M. A. Jabbar. "Machine Learning in Healthcare: A Review". In: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2018, pp. 910–914. DOI: 10.1109/ICECA.2018.8474918.

[Sze+17] Vivienne Sze et al. "Hardware for machine learning: Challenges and opportunities". In: *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2017, pp. 1–8.

[Tsi+18] Christos Tsirigotis et al. "Orion: Experiment version control for efficient hyperparameter optimization". In: (2018).

[TVD18] Rachael Tatman, Jake VanderPlas, and Sohier Dane. "A practical taxonomy of reproducibility for machine learning research". In: (2018).

[Vay+19] Leila Abdollahi Vayghan et al. "Kubernetes as an availability manager for microservice applications". In: *arXiv preprint arXiv:1901.04946* (2019).

[Wan+17] Mowei Wang et al. "Machine learning for networking: Workflow, advances and opportunities". In: *Ieee Network* 32.2 (2017), pp. 92–99.

[Wat+19] Junzo Watada et al. "Emerging trends, techniques and open issues of containerization: a review". In: *IEEE Access* 7 (2019), pp. 152443–152472.

[YYU20] Yang Yang, Wu Youyou, and Brian Uzzi. "Estimating the deep replicability of scientific findings using human and artificial intelligence". In: *Proceedings of the National Academy of Sciences* 117.20 (2020), pp. 10762–10768.

[Zel78] Marvin V. Zelkowitz. "Perspectives in Software Engineering". In: *ACM Comput. Surv.* 10.2 (June 1978), pp. 197–216. ISSN: 0360-0300. DOI: 10.1145/356725.356731. URL: https://doi.org/10.1145/356725.356731.

A Code

Things, which have no place in the main content should be in the Appendix.

B Dataset

C Content of the CD

- This work as PDF file – in the folder *PDF*
- The source code of the implementation – in the folder *SRC*
- The implementation as a runnable .jar file – in the folder *JAR*
- The LaTeXsource code – in the folder *LATEX*

Declaration of Academic Integrity / Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind. Mit der aktuell geltenden Fassung der Satzung der Universität Passau zur Sicherung guter wissenschaftlicher Praxis und für den Umgang mit wissenschaftlichem Fehlverhalten vom 31. Juli 2008 (vABlUP Seite 283) bin ich vertraut. Ich erkläre mich einverstanden mit einer Überprüfung der Arbeit unter Zuhilfenahme von Dienstleistungen Dritter (z.B. Anti-Plagiatssoftware) zur Gewährleistung der einwandfreien Kennzeichnung übernommener Ausführungen ohne Verletzung geistigen Eigentums an einem von anderen geschaffenen urheberrechtlich geschützten Werk oder von anderen stammenden wesentlichen wissenschaftlichen Erkenntnissen, Hypothesen, Lehren oder Forschungsansätzen.

Passau, 6. April 2022

Martin Johannes Loos

I hereby confirm that I have composed this scientific work independently without anybody else's assistance and utilising no sources or resources other than those specified. I certify that any content adopted literally or in substance has been properly identified. I have familiarised myself with the University of Passau's most recent Guidelines for Good Scientific Practice and Scientific Misconduct Ramifications from 31 July 2008 (vABlUP Seite 283). I declare my consent to the use of third-party services (e.g., anti-plagiarism software) for the examination of my work to verify the absence of impermissible representation of adopted content without adequate designation violating the intellectual property rights of others by claiming ownership of somebody else's work, scientific findings, hypotheses, teachings or research approaches.

Passau, 6. April 2022

Martin Johannes Loos