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 27, 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. We define the terms reproducibility and replicability in 2.1 and distinguish them from each other. This is of particular importance as different definitions of reproducibility are used in different works. In the following quotations we mark with * if the definition differs from ours. In these cases, replicability is meant.

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 replicated. Otherwise, the results from an academic paper can not be recognized by the general public. Therefore, the replicability 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.

1.1 Aims and Objectives of the Thesis

Chapter 3 provides an overview of the contributions that others have already made to address this problem. We want to use these findings to propose a new approach based on them. When creating a scientific work, the time available is one of the limiting factors. Therefore, we want to use a software tool to reduce the time required to check whether the reproducibility guidelines have been complied with. By compliance with the guidelines, we mean that the factors identified in Chapter 4 have been taken into account.

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]. Hence, our goal is to create a supporting tool for researchers and reviewers that checks an ML repository whether the detectable reproducibility factors are within the guidelines or not.

Hypothesis

To test the functionality of the tool, we *hypothesize* that, in general, *a repository belonging to an ML experiment published by academic publishers can be expected to adhere to the reproducibility guidelines statistically significantly better than an ML repository on GitHub* (on average). In order to check this, the following research questions must first be answered.

`Research Question 1:` Which factors influence the reproduciblity of ML experiments? This identification step is carried out in Chapter 4 based on a literature search.

`Research Question 2:` Which influencing factors can be detected with the help of a software tool and which can not? Our classification approach is presented in Chapter 5.

`Research Question 3:` How can the detectable factors be analyzed using a software tool? To answer this question, we describe the implementation of our tool in Chapter 6.

After these questions have been dealt with, we examine our hypothesis in Chapter 7. For this purpose, we use the tool presented to evaluate two different sets, each containing 100 ML repositories. We then discuss the knowledge gained in Chapter 8.

2 Background

This Chapter contains basic knowledge on basic knowledge on the main concepts of this work. 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 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]"

The term machine learning (ML) is explained in Section 2.5. The presented workflow makes it clear that an experiment in the ML 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 within the open-science guidelines. 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.

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.

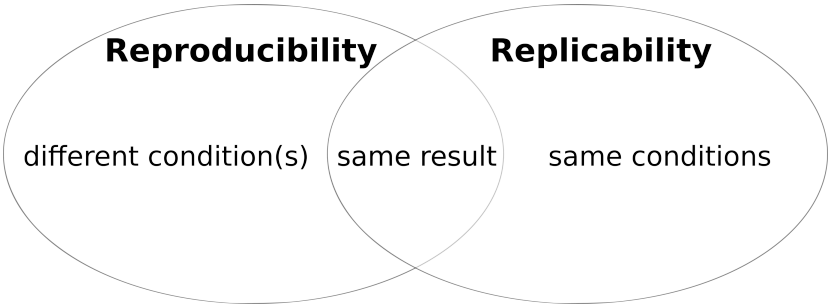


Figure 2.1: Reproducibility versus replicability. Since reproducibility and replicability are confused terminologies, a precise distinction is required. An experiment is said to be replicable if repeating the experiment under exactly the same conditions produces the same result. Reproducibility requires that the result remains the same even though some conditions have been (slightly) changed.

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.

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.". It should be noted that 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 3. 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 additional 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 is important to this thesis for two reasons. First, this technique is useful from a reproducibility point of view. Second, BinderHub utilizes Docker and Kubernetes, among other technologies. For these reasons, we explain the basic concepts, starting with the definition of the term.

"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 (OS) virtualization.

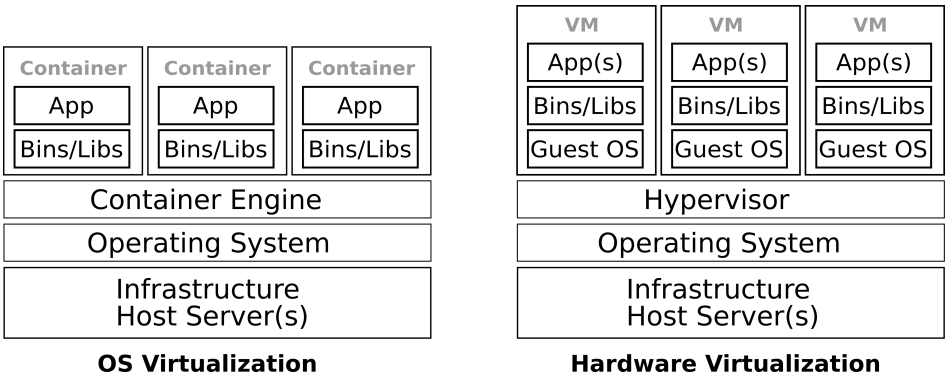


Figure 2.2: OS virtualization versus hardware virtualization. Both types of virtualization pursue a common goal, namely encapsulation. The difference lies in the layer on which this happens. OS virtualization occurs at the application layer, while hardware virtualization occurs at the hardware layer.

In Figure 2.2 we compare two different types of virtualization. Both have a common 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.

Docker

Without going into details about the inner workings of Docker, we want to explain some terms that should be clear, as they are of great importance from a reproducibility point of view.

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.

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.

A *Docker image* is a snapshot of a virtual machine at a specific point in time. One 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].

Firstly, 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.

Kubernetes

Container orchestration systems are primarily required when large numbers of containers have to be managed. Kubernetes is a component of BinderHub, which is why we want to briefly explain this technology below.

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.

It 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

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

applications [Vay+19]. But numerous other services such as BinderHub, which will also be discussed in this thesis, use the combination of Kubernetes and Docker.

2.3 Literate Programming

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. Literate programming technologies, such as jupyter notebooks, provide all the functionality required for this. The pyhilosophy behind it 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 notebooks provide the functionality needed to implement this philosophy. It can not only execute the contained source code. In addition to the code, its output is saved. Also, it is 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. 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 addition, Binder also supports this technology. But 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, environment.yaml or conda.env) and the creation of a Jupyter notebook. Binder provides the functionality to combine both approaches.

"Binder is a free, open source, and massively publicly available tool for easily creating sharable, interactive, replicable environments in the cloud [RW18]." 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.

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. Figure 2.3 shows the architecture of this technology.

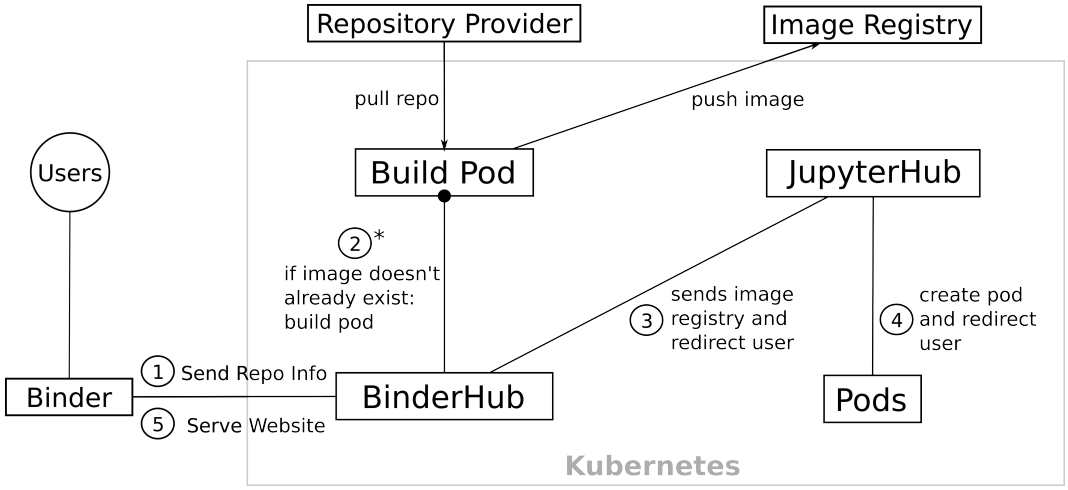


Figure 2.3: Simplified overview of the BinderHub architecture[4]. Shown here is the flow of information when a user sends a request to start a pod to BinderHub. Step 2 is omitted if an image is already stored in the registry for the requested repository.

repo2docker

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

"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]"

[3] https://mybinder.org accessed: 24.01.2022
[4] `https://binderhub.readthedocs.io/en/latest/overview.html` accessed: 22.01.2022

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.

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 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. 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 use the ML development workflow shown in Figure 2.4 to explain the general procedure. This is a simplified, modified version of the ML lifecycle that Ashmore et al. presented in their paper [ACP19]. After one has a general overview of this process, the ML-specific influencing factors identified in Chapter 4 can be better understood.

[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

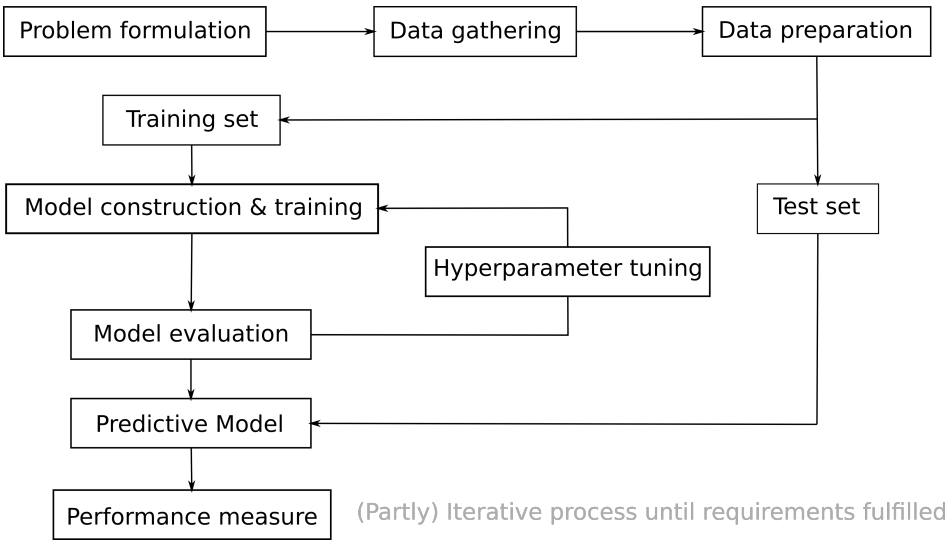


Figure 2.4: Abstracted, simplified machine learning workflow. A possible workflow for the development of a machine learning model is shown. Since a model is generally developed iteratively, some steps are run through several times. For this reason, it is important to document the changes made in order to be able to understand them afterwards.

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.

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.

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.

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.

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, we provide an overview of various scientific papers that have dealt with issues that are relevant to us.

`Identification of Reproducibilty Factors:`

Since our goal is to classify reproducibility factors that are relevant for machine learning experiments, we have to identify them beforehand. To do this, we use existing knowledge and summarize the findings. 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]. 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 Reproducibilty 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.

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

`Estimating the Replicability Probability of a Paper:`

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.

`State of the Art Reviewing Process:`

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. The downside to this is that this review is done manually. Hence, our goal is to reduce the use of manual review activities to the bare minimum. Nevertheless, checklists are a good standardized approach that we continue to recommend, especially for factors that we have classified as undetectable.

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 want to take a different perspective with our classification. Previous approaches to assess reproducibility are based on manual reviews. We want people only to be involved in dealing with software-based undetectable factors in order to minimize the time required for verification and thus save resources. There is already an approach to estimate the probability of replicating an ML paper, but none that does this based on a repository analysis. Hence, we base our classification approach on whether a factor can be detected using a software tool or not. For this, we use an indicator-based argumentation which is presented in Chapter 5.

By increasing the likelihood of successfully reproducing an ML experiment, its scientific relevance and informative value could 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 2.4. 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, we have already found that replicability 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 data set 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 data sets 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, 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.

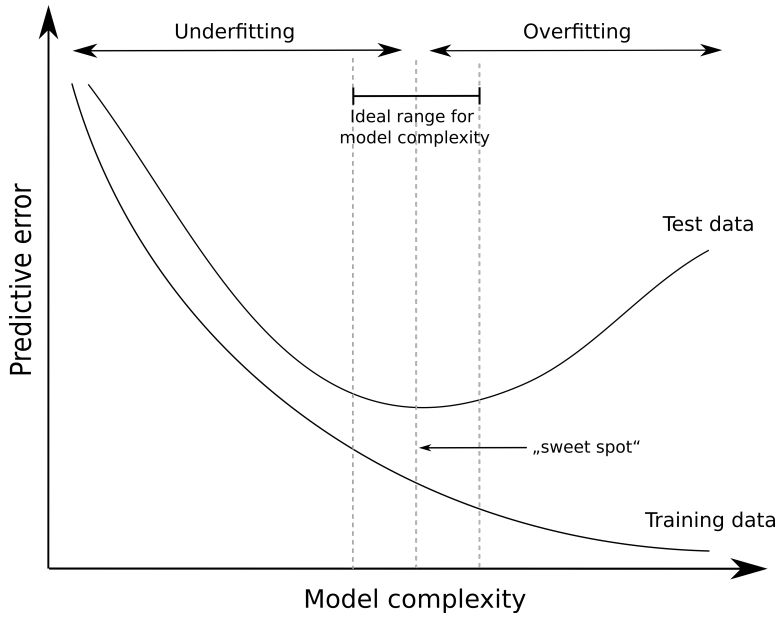


Figure 4.1: Adapted figure explaining overfitting & underfitting [Hea+15a]. While neither the test nor the training data are well represented in an underfitting model due to its low complexity, it adapts too much to the training data in an overfitting model. A model complexity within the represented range close to the so-called "sweet spot" is ideal.

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

After the influencing factors on the reproducibility were identified and explained in Chapter 4, we want to classify them in this Chapter. Our goal is to present a classification approach that divides the factors into two different groups based on whether they can be detected by software or not.

We are particularly interested in the factors that a tool can evaluate. Building on this, in Chapter 6 we present a proof of concept (PoC) implementation of an analysis tool that provides this functionality. In order to be able to understand the concepts discussed below, we first differentiate between the terms factor and indicator.

Factors have a direct correlation with reproducibility. By indicators, on the other hand, we understand software-based detectable properties of a factor. This relationship is shown in Figure 5.1.

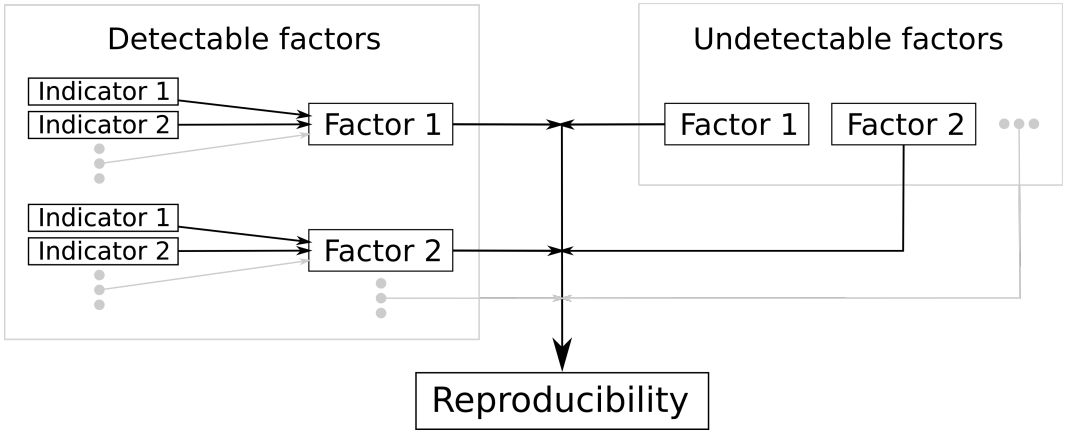


Figure 5.1: Connection of factors, indicators, and reproducibility. In order to be able to decide to which class a factor belongs, we use indicator-based reasoning. If at least one indicator that can be measured with a software tool can be assigned to a factor, then this is a detectable factor.

From this definition of the term, the basic concept of our classification becomes clear. We use indicator-based reasoning to classify the factors into one of the two groups. If at least one indicator can be identified for a factor, this can also be evaluated using software. Otherwise this is an undetectable factor. These will be covered in Section 5.2.

5.1 Detectable Factors

In the following, we deal with the factors that we classify as detectable. If at least one indicator can be found for a factor, we assign it to this group. In this Section, we provide an overview of all identified indicators with their associated factors. In addition, we want to use a scale to clarify for each factor the degree of complexity to be expected with regard to the implementation. Using this assessment, we then determine which of the factors we can include in our PoC implementation (due to time constraints).

However, we are not yet concerned here with how the indicators belonging to a factor can be connected to it. This approach is presented in 6.2.

Assignment of Indicators

For each factor, we considered whether and which indicators it has. In the following we give an overview of the results. This is the basis for later merging the analyzed data on the factors.

Table 5.1 contains all identified indicators and their respective factors.

Table 5.1: List of All Identified Reproducibility Indicators and Their Corresponding Factors.

Reproducibility indicator	**Reproducibility factor**
GitHub URL	Source code availability
Open-source license	Source code documentation
Readme length	Source code documentation
Readme hyperlinks	Source code documentation
Source-code-comment-ratio	Source code documentation
Pylint rating	Source code documentation
Notes on hardware environment	Hardware environment
Configuration file	Software environment
Strict declarations in the configuration file	Software environment
Imported libraries in the configuration file	Software environment
BinderHub-API build response	Out-of-the-box buildability
Binder badge in readme	Out-of-the-box buildability
Data set in the repository	Dataset availability
Data set reference in the readme	Dataset availability
Notes on data set preprocessing	Dataset preprocessing
RS declaration (fixed) in the source code	Random seed control
HP-logging declaration in the source code	Hyperparameter declaration
Artifacts of model-serialization	Model-serialization
Model-serialization used in the source code	Model-serialization
Paper link in the readme	Research practices & experimental design

Complexity Scale

This scale estimates the complexity associated with the implementation of a factor's indicators. From this we derive the amount of time that can be expected for this. This is helpful to be able to present a comprehensible approach, which of the detectable factors are implemented in our PoC implementation and which are not. Due to the overall complexity, not all factors can be covered with our tool.

To determine this complexity for each factor, we consider the associated prerequisites for computation and the expected level of implementation difficulty. The resulting scale is shown in Figure 5.2.

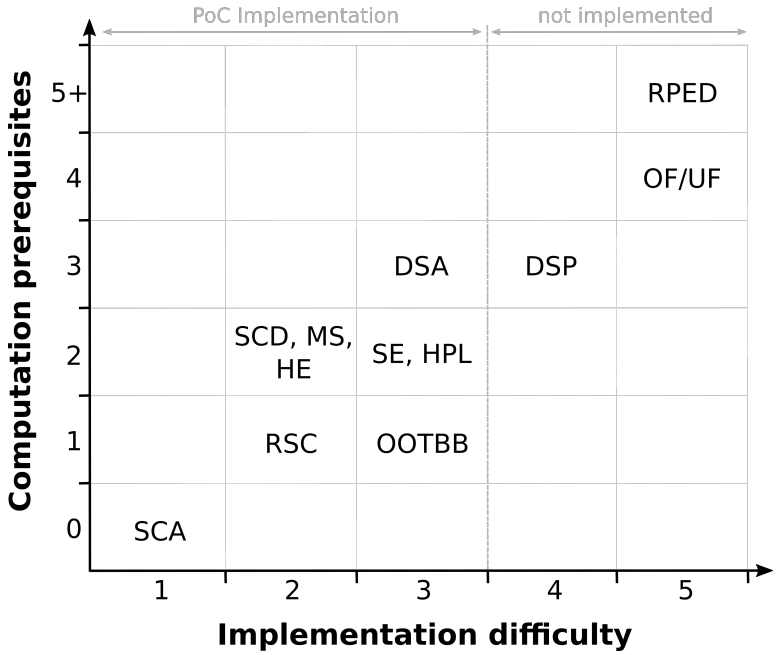


Figure 5.2: Classification of the detectable reproducibility factors in a complexity scale with regard to the number of their computation prerequisites and the implementation difficulty. Computation prerequisites refer to the number of required artifacts that have to be availabe, detected or analyzed in order to examine all indicators of a factor. Implementation difficulty is a subjective assessment based on the authors skills and knowledge. Factors classified with a implementation difficulty above three are not included in the PoC implementation. Also not included is the hardware environment factor, as we have found that if specified, it is mostly stated in the paper.

By computation prerequisites we mean the number of required artifacts that have to be available, detected or analyzed in order to be able to examine the indicators of a factor in each case. Implementation difficulty is a subjective assessment. We break this down into 5 different categories, with 1 being the lowest level and 5 being the highest level. In addition to our own programming knowledge, we also took into account the theoretical concepts of the respective factors. Factors with a difficulty level of 4 or 5 were not integrated into our tool. This affects the factors DSP, OF/UF and RPED.

In the following, we give a brief justification for the respective assessment for each factor. Since all factors can theoretically also be dealt with in the paper, we do not see this as an prerequisite for computation but assume access to it.

`Source Code Availability (SCA):`

SCA has no prerequisites, since a link in the paper is not counted as such. An implementation is superfluous but could be done by testing the hyperlink.

`Hardware Environment (HE):`

HE has two prerequisites. This can be specified in a separate file or within the readme. The difficulty with the implementation lies in the fact that there is no standard case as far as the documentation of this factor is concerned. Hence, we have decided not to include this factor as the only one with a difficulty level of less than 4 in our PoC tool.

`Source Code Documentation (SCD):`

Source code is usually explained in the respective source code file through comments. Additionally, we consider the operation of all code here. This is usually described in the readme. Therefore we assign 2 prerequisites to this factor. In order to analyze SCD in practice, all source code files and readmes must be identified. It should be noted here that we do not directly check the content of the readme in our implementation, but try to use quantitative evaluations to assess whether it may have any meaningfulness in this regard.

`Random Seed Control (RSC):`

In order to analyze RSC, all source code files must be identified. Within these, one can then search for random seed declarations. For the implementation, this of course assumes that code lines are recognized and these declarations methods are known (for different frameworks).

`Hyperparameter-Logging (HPL):`

In addition to analyzing all source code files, it may also be necessary to identify separate log files to identify HPL. We have not considered this log file detection in our tool, since there is no standardized methodology for this, which would make the implementation too time-consuming (compared to the benefit). For the implementation it requires knowledge of the possible logging methods (for different frameworks) and in addition to code line also import line detection.

`Software Environment (SE):`

The SE is usually defined in a configuration file. But sometimes the relevant libraries are also listed in the readme. In our implementation, however, we only want to cover the standardized case of the declaration in a separate configuration file. To do this, they must be parsed and the import lines identified within the source code.

`Model-Serialization (MS):`

MS detection requires that all source code files have been identified and relevant artifacts can be detected in the repository. The implementation requires knowledge of possible MS file extensions and libraries that support this.

`Dataset Availability (DSA):`

In order to be able to track down data sets, we require the detection of a readme, the source code and relevant artifacts in the repository. During implementation, it is then necessary to develop an algorithm which can use these files to estimate whether an external data set is referenced or stored locally in the repository.

`Dataset Preprocessing (DSP):`

DSP can be documented in the readme or in the source code. Source code files or scripts could also exist which can automatically execute the steps taken. Because of the many ways in which preprocessing can be specified, we have divided this factor into difficulty level 4. Therefore our PoC implementation does not include the detection of DSP.

`Overfitting/Underfitting (OF/UF):`

In order to be able to assess overfitting/underfitting, we need access to a similar or the same data set and the preprocessing steps. In addition, the code must be executable and the measured results must be documented in the paper. In our opinion, the implementation of an algorithm that can estimate the presence of OF or UF is generaly possible. However, this is very complex and was not pursued further by us.

`Research Practices & Experimental Design (RPED):`

For the examination of RPED it is assumed that the paper and repository can be examined with one or more suitable tools. With this we mainly refer to the extraction of the required information from the paper. The software-based comparison of whether specific guidelines have been complied with is then also necessary. From our point of view, this is the most complex of all identified and detectable factors.

5.2 Undetectable Factors

Since no indicator was found for these factors, they are not detectable with a software tool. This means that a manual review process is unavoidable for assessing whether and to what extent any of these factors influence an ML experiment. In the following we briefly explain how to deal with the undetectable factors.

`Knowledge Gap:`

From the point of view of a researcher conducting an ML experiment, it is important to take this issue into account when creating it. Hence, everything should be documented as precisely and understandably as possible. In this way, a possible knowledge gap as described in 4.11 can possibly be reduced preventively or, in the best case, be closed. From a reviewer's point of view, due to the interdisciplinary nature of the ML field, it may still be unavoidable to work with reviewers from other disciplines to replicate or reproduce the experiment.

`Bias & Probability Hacking:`

The main issue here is that even with a manual review, these factors are very difficult, if not impossible, to detect. However, it is the ethical duty of a researcher to avoid these reproducibility problems. There are however types of bias that can arise subconsciously or cannot be avoided. Hence, it is important to document all steps as precisely as possible, explain the decisions and interpretations made, and publish the data used [SN14].

In general, these influencing factors are often difficult to avoid or arise subconsciously (apart from p-hacking). Therefore, researchers should always be aware that these factors can cause reproducibility and replicability difficulties.

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) implementation.

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 check a repository with regard to the reproducibility guidelines and to be able to assess whether there are any problem areas. This means that compliance with the recommendations can be checked during development or before publication with as less effort as possible. 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 5.1 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").

Flowchart 6.1 gives a visual overview of the implemented modules and how they interact with each other when the tool is executed.

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

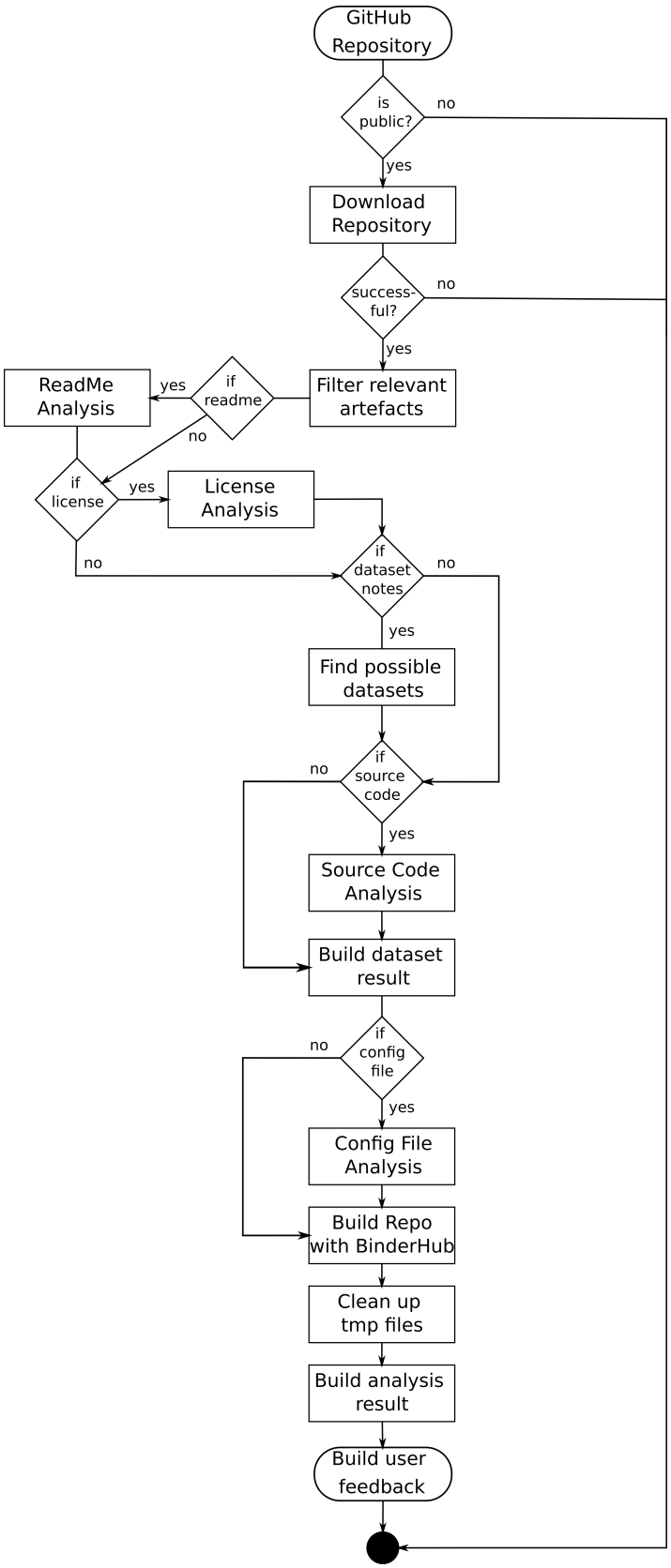


Figure 6.1: The simplified flowchart of the repository analysis tool shows the execution order of the python modules. Additionally, the conditions under which the tool terminates and when the execution of individual modules is skipped are displayed.

Repository Download & Preprocessing

Firstly, 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.

Readme & License Analysis

To identify 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 for each file. All hyperlinks are also tested for functionality. Large, memory-intensive data sets are often not stored locally in the repository but are referenced in the readme. In order to detect them, we saved the names of about 350 known data sets in a file. If one of these is mentioned, we save it as a dataset reference. This also applies if a section containing "data" in the name and at least one hyperlink is present. In addition, it is checked whether a paper is linked or a binder badge has been specified. This is done for all files and the numerical results are averaged.

For the detection of licenses, *filter_repository_artifacts.py* also uses the same approach as explained in the readme part above. The contents of the recognized files are then checked by *license_analysis.py* using keyword matching to see whether the text mentions open source licenses defined by the Free Software Foundation[2] (FSF). Both open-source and non-open-source licenses are then stored in separate lists.

Retrieving Dataset File Candidates (DSFC)

Next, the search for included data sets in the repository is started. All folders that match a specified regex, along with files that have "data" in their name, are passed from FRA to the *dataset_analysis* module for further processing. Files found in the detected folders which do not end with ".py", ".ipynb" or ".md" are saved together with the files containing "data" in the name as so-called data set file candidates (DSFC). 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 data sets.

Source Code Analysis

For the analysis of source code files, FRA collects all files with the extension ".py" or ".ipynb". The *source_code_analysis.py* file calls the appropriate module for each file according to their 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

[2] https://en.wikipedia.org/wiki/Comparison_of_free_and_open -source_software_licences accessed: 26.03.2022

into a ".py" file for further processing using nbconvert[3]. We are particularly interested in the code-comment-ratio (CCR), 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 their respective method calls. These were taken from the documentation of the listed libraries. Finally, we also look for indicators of model serialization. For this we check the code lines for calls of *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", that they have a file name equal to "model" or to have one of the following file extensions: ".dvc", ".h5", ".pkl" or ".model" (these artifacts are all collected by FRA if found). Also, we filter out the Python Standard Libraries[4] and locally defined modules from all found imports and eliminate duplicates. We refer to the remaining libraries in the following as the relevant libraries. Finally, it is tested whether these are publicly accessible (using pip-search). The average of the numerical analysis results is then recorded and saved together with the other measured values in an overall result.

Software Environment Analysis

The *config_analysis.py* module receives all from FRA detected configuration files (which match a specified regex) as well as a preprocessed list of all used relevant 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.

Out-of-the-box Buildability Analysis

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 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" if the build was sucessful or otherwise "failed".

Output

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. This is also persisted in a ".csv" file (e.g. for debugging purposes). The output of the tool is discussed in more detail in Section 6.3.

[3] https://nbconvert.readthedocs.io/en/latest/
[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 (without taking into account the feedback). 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 connect 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 scoring? 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 measured 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.

As already mentioned, for each indicator we need comparative values for both reproducible and non-reproducible repositories. Therefore, twenty reproducible and twenty non-reproducible repositories were collected. Hence, we had to check them manually and depending on whether we were able to reproduce them successfully they were assigned to one of the two sets. More repositories for each set would have been more representive, but due to the fact that we had to check them manually, more was not possible in the given time span. The two sets were then analyzed with our tool and the representative values and weights were determined based on the results obtained. In the Appendix we have included the relevant data points of these sets in Figure B.1.

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 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 scorings, 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

Y = Indicator value in new range
x = Original indicator value
$min(d)$ = Lowest value in original range
$max(d)$ = Highest value in original range
$min(n)$ = Lowest value in new range
$max(n)$ = Highest value in new range

Most important for our approach to connecting factors with indicators are *min(d)* and *max(d)*. In Section 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 twenty 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. Also, 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. For each of the artifact types, we observe that the set of reproducible repositories contains it significantly more often. 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).

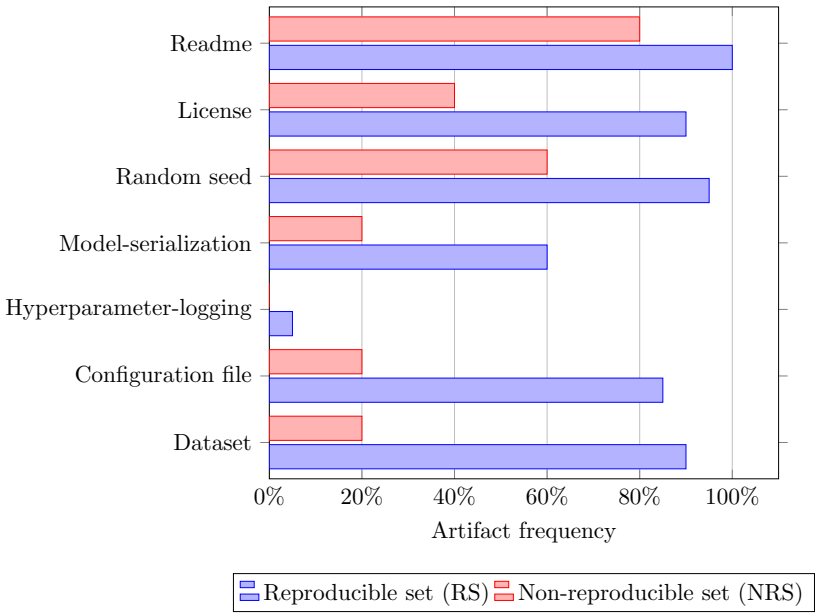


Figure 6.2: Statistical evaluation of the frequency with which relevant artifacts are present in the reproducible and non-reproducible ML repository set. From this figure it can be observed that the reproducible set contained all measured artifact types significantly more frequently on average. An exception is the readme artifact, where a similar frequency is given. This is probably due to the fact that this is usually generated when a repository is created by setting a check mark.

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

1. Assessment of whether there is a connection with reproducibility based on the observations made.
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). If there are multiple ones their weight sum up to 1.

Source Code Availability & Documentation

To assess this factor, we examine a total of six indicators extracted from the following three artifact types:

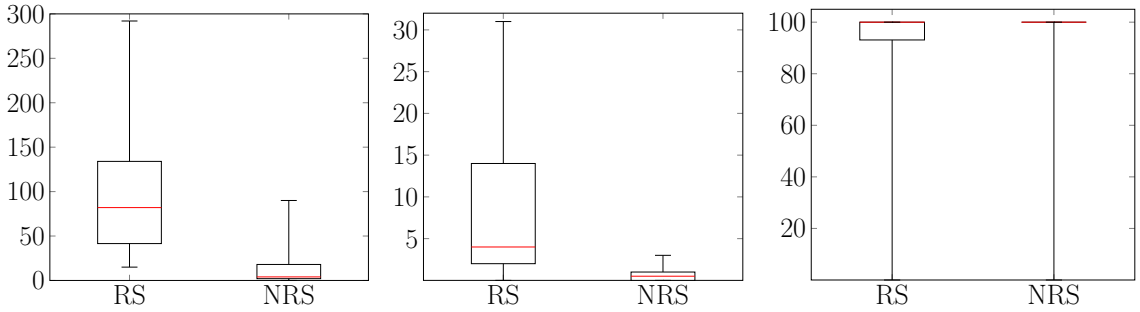
- *Readme:* The relevant indicators are the average length, the average number of hyperlinks as well as the percentage of accessible links.
- *License:* We want to check the presence of an (open-source) license.
- *Source code:* The average code-comment-ratio and the average pylint rating will be measured.

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

`Readme Scoring:`

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 shows that readmes in the RS are significantly longer than those in the NRS. This observation also applies 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.



(a) Length (mean) **(b)** Links (mean) **(c)** Accessible links (%)

Figure 6.3: Readme indicators measured to determine the representative values and weights. To obtain these values, we used our tool for all entities of the reproducible (RS) and non-reproducible (NRS) set and analyzed their readme files. Both *(a) Length* and *(b) Links* indicate a significant difference. From *(c) accessible links* it can be deduced that this indicator is probably not meaningful in terms of reproducibility.

`Determination of the Representative Values:`

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 3rd quantile of the NRS as *min(d)* for both the length and the number of accessible hyperlinks. The reason for this is that for all values below *min(d)* the score of the indicator will be 0 after normalization. It follows that most repositories from the NRS will get the appropriate result. The median of the RS is chosen for *max(d)*. Our reasoning for this is that the assessment of an indicator should promote compliance with reproducibility guidelines. Therefore, the 1st quantile was not chosen because high standards are desirable. Since all entities are reproducible within the RS, the 3rd quantile would not be a suitable value either, as most would then be below it. Therefore, we chose the median as a compromise.

`Determination of the Weights:`

When determining the weights, 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. Our assumption is 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 should be given significantly more weight.

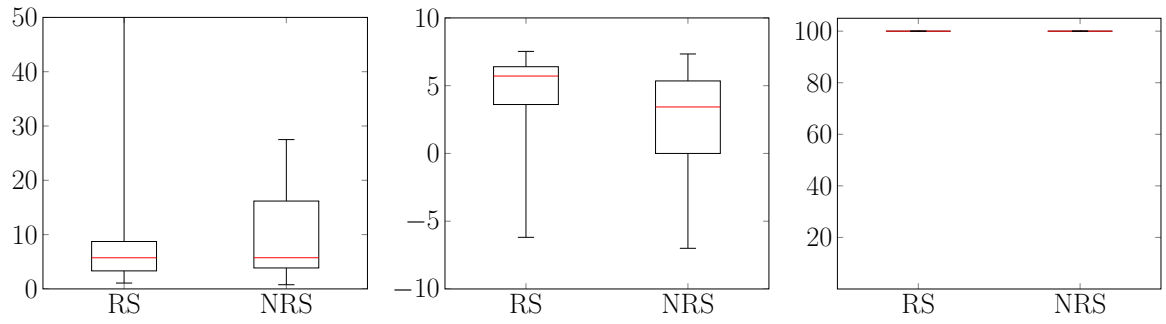
Table 6.1 shows the classification that results from these observations:

Table 6.1: Readme indicators with their assigned representative values and weights.

Indicator	min(d)	max(d)	Weight
Length	18	82	0.8
Accessible hyperlinks	1	4	0.2

`Source Code Availability & Documentation Scoring:`

Now let's turn our attention to the indicators of the other artifact types. Figure 6.4 shows that the median of the code-comment-ratio (CCR) is almost identical for both sets. However, the ratio is significantly different as far as the 3rd quantile is concerned. It can therefore be assumed that this indicator has some significance. Since 100% of the licenses for the RS and NRS were open-source (if they contained a license), no significance can be attributed to this indicator alone. In connection with the findings from Figure 6.2 we argue 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 scoring. Differences can also be observed in the pylint rating (PR), especially with regard to the respective 1st quantile.



(a) Code-comment-ratio (mean)

(b) Pylint rating (mean)

(c) Open source license (%)

Figure 6.4: Source code availability & documentation indicators measured to determine the representative values and weights. In order to obtain these values, we used our tool for all entities of the reproducible (RS) and non-reproducible (NRS) sets of ML repositories and analyzed their source code files and all licenses found. From *(a) code-comment-ratio* it can be read that on average entities of the reproducible and non-reproducible set showed similar values. However, the values of the 3rd quantile show a significant difference. Equally clear is the difference of the 1st quantile as far as *(b) pylint rating* is concerned. Based on *(c) open source license*, it becomes obvious that if this artifact type was found, it could be assigned to the open source category in all cases, which is why this indicator alone is probably not meaningful.

`Determination of the Representative Values:`

If no or a non-open-source license is detected, the score of this indicator is 0, otherwise 1. We choose these values because we only check the existence of this indicator.

While the median of the measured code-comment-ratios is almost the same for both sets, there is a significant difference when it comes to the 3rd quantile. A higher CCR means there are fewer comment lines per line of code. We argue that a high ratio has a negative effect on the understandability of source code. Therefore, we assign a scoring of 1 if it is lower than the 3rd quantile of the RS. For larger values we calculate the scoring with the formula:

1 - normalized value (with *min(d)* = 3rd quantile of RS and *max(d)* = 3rd quantile of NRS).

Looking at the pylint evaluation, it is noticeable that the median of the sets is similar, but the 1st quantile differs significantly. For *min(d)* we choose the value of the 1st quantile of the NRS, as this represents the minimum rating we expect. To encourage a standardized coding style, we chose the median of the RS as *max(d)* as this seems to be a high but achievable value.

`Determination of the Weights:`

When determining the readme weights, we already mentioned that the source code availability & documentation factor primarily refers to the documentation. For this reason, the readme score is given the highest weight for the overall factor score. It is also important that the content used is publicly available. Hence, we assigned a corresponding weight to the license score. For the remaining indicators CCR and PR, the median of both sets was very similar. Therefore, in most cases, they are less meaningful than the indicators discussed previously. This was taken into account when determining the weights.

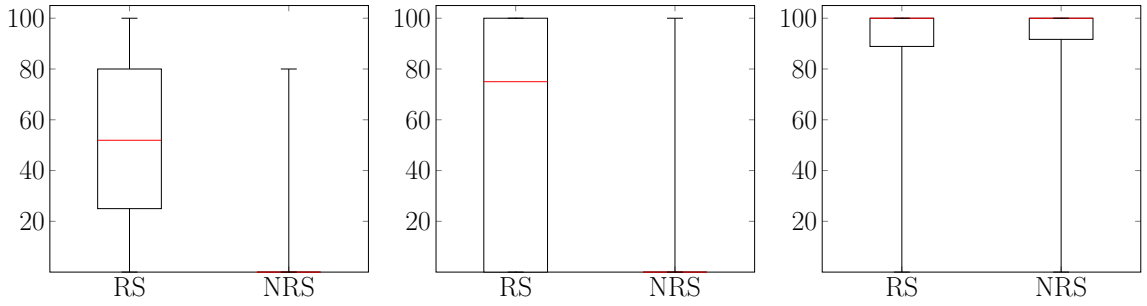
The following classification results from the observations made:

Table 6.2: Source code availability & documentation indicators with their assigned representative values and weights.

Indicator	**min(d)**	**max(d)**	**Weight**
Readme	-	-	0.5
License	0	Number of licenses	0.3
Code-comment-ratio	8.73	16.18	0.1
Pylint rating	0	5.71	0.1

Software Environment

As one can see in Figure 6.2, a repository from the RS is much more likely to contain a configuration file. 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 6.5.



(a) Source code imports in the configuration file (%)

(b) Strict dependency declarations (%)

(c) Publicly accessible libraries in the source code (%)

Figure 6.5: Software environment indicators measured to determine the representative values and weights. To obtain these values, we used our tool for all entities of the reproducible (RS) and non-reproducible (NRS) sets of ML repositories and analyzed their configuration and source code files. An important factor influencing this evaluation was that the majority of the NRS entities did not have a configuration file. Significant differences can be observed based on *(a) source code imports in the configuration file* and *(b) strict dependency declarations*. The evaluation of *(c) publicly accessible libraries in the source code* probably does not allow any conclusions to be drawn about the reproducibility property based on the measured values.

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. The median for the RS is around 50% coverage. Although this is not yet an optimal value, it is a significant deviation compared to the NRS. It can also be observed that dependency declarations of reproducible repositories are usually version strict. The metrics of both sets of repositories in terms of how many of the libraries used in the source code are publicly available are very similar. Hence, this indicator is not suitable for estimating whether an analyzed repository is more likely to be reproducible or non-reproducible. 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.

`Determination of the Representative Values & Weights:`

Since all specified key indicators 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 weights. This results in the following classification:

Table 6.3: Software environment indicators with their assigned representative values and 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 almost all repositories in the RS. In contrast, this was included significantly less frequently in the NRS. From this, it can be deduced that the non-existence of a data set tends to indicate a non-reproducible repository. It is important to note at this point that this is a PoC implementation. For this reason, the functions related to the preprocessing of the data set are not yet supported.

In the *dataset_analysis.py* module we identify possible data set files. We are referring to them as data set file candidates (DSFC). The *source_code_analysis.py* module checks which of these candidates is used in the code. We assume that matches are relevant files containing the data set. Additionally, during readme analysis, we look for a data set reference (based on a list of known datasets and hyperlinks within a section that has "data" in its name). These are the identified meaningful indicators for this factor. Therefore, it is only examined whether at least one of the identified DSFC is mentioned in the source code or if a data set reference is found in the readme. If so, we score this factor with the maximum value of 1. Otherwise, we assume that no data set is included or referenced in the repository and we assign the minimum value of 0.

Random Seed Control

Almost all repositories from the RS have taken this influencing factor into account (see Figure 6.2). For the NRS, this only applies to 60%. A statistical correlation can thus be observed. In addition to the declaration, we also measured how many of them 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. However, from a reproducibility perspective, we want to ensure that a declared seed always has a fixed value assigned. Therefore we want to keep this indicator although no statistical deviations could be determined. Hence, the input for the min-max normalization is only random seed declarations with a fixed seed.

To determine a factor scoring we choose: $min(d) = 0$ and $max(d) =$ number of all random seed declarations. If no seed was used, the score is 0, since then there is no control of the starting value of the random number generator.

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) was not used in almost any repository of the two sets. 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 score 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. In 2.4 we discussed why the use is recommended from a reproducibility point of view. The inclusion in the feedback file is primarily intended to encourage the use of binder or similar technologies.

6.2.3 Reproducibility Factor Thresholds

Threshold values are used for some factor ratings within the *feedback_builder.py* module to assess whether they correspond to the guidelines or not. This allows us to give context to these scores. Therefore, we calculated the average measured factor scores for both sets (RS & NRS), shown in Figure 6.6, and use them as thresholds. This approach is not necessary for the factors DSAP, MS, HPL and OOTBB, since these can only receive scorings that are either 0 or 1.

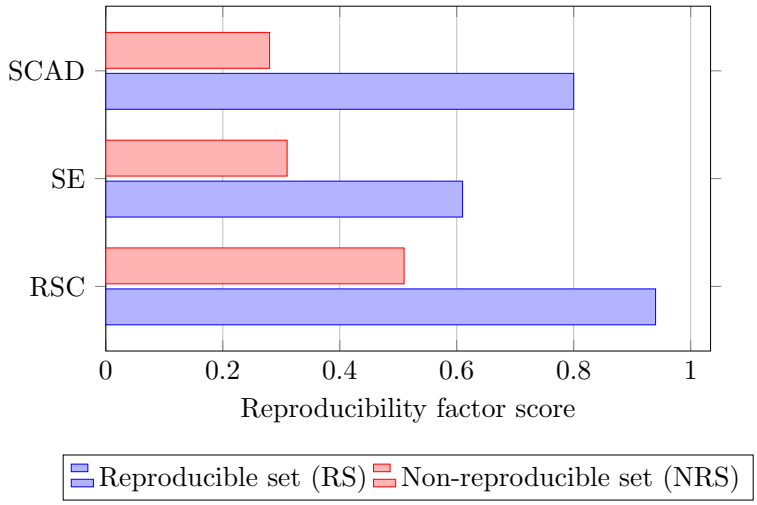


Figure 6.6: Statistical evaluation of the measured factor scores for source code availability & documentation (SCAD), software environment (SE), and random seed control (RSC) for the reproducible and non-reproducible ML repository set. The scores shown are average values of the respective factors in a set. It can be observed that all scorings of the reproducible set are significantly better. Based on these measurements, we set thresholds, which give context to measured values as to whether they are good or bad.

For the factors source code availability & documentation, software environment and random seed control we define the following thresholds:

- Top threshold (T): Scores that are above or equal to this threshold indicate compliance with the reproducibility guidelines, as these are at least as good as the average scores measured by repositories from the RS.

- Lower threshold (L): Scores that are below or equal to this threshold indicate non-compliance with the reproducibility guidelines, since these are at best as good as the average scores measured by repositories from the NRS.
- Average threshold (A): This threshold is calculated by the formula $\frac{T+L}{2}$. Values lower than T but above this threshold indicate a rather good factor score. Values that are at most as good as this threshold but are above L indicate a less desirable factor score. Therefore, if a value is between T and L, the average threshold is used to estimate whether the factor conforms more or less well to the guidelines.

Table 6.4 shows an overview of the determined threshold values which we assigned to their corresponding factor.

Table 6.4: Threshold Values of the Factors Source Code Availability & Documentation, Software Environment and Random Seed Control.

Factor	***T***	***A***	***L***
Source-code availability & documentation	0.8	0.54	0.28
Software environment	0.61	0.46	0.31
Random seed control	0.94	0.73	0.51

6.3 Tool Output

This tool produces output in three different places. Each output serves a different purpose. Most important is the *feedback.md* file which assigns a score and context to the values measured by the tool and provides recommendations.

`Command Line:`

The intermediate results of each analysis step are displayed in the command line. This information can be used for debugging purposes in case something goes wrong.

`Result File (.csv):`

The measured values for all examined indicators as well as additional data points are saved in a ".csv" file. Hence, these values can be saved permanently if one needs them again at a later point in time.

`Feedback File (.md):`

A feedback file is created based on the measured indicator values, the representative values and weights. This includes textual feedback as well as scoring values for both factors and their indicators. The factor scorings can then be interpreted using their respective thresholds. Figure A.1 in the Appendix shows an example file for illustration. Here you can also see that not only the scorings of the factors and indicators are displayed, but also that recommendations are given.

By default, the two output files are saved in the \tmp folder. This can be configured differently in *main.py* if required.

7 Evaluation

Why important?

Evaluate our hypothesis

Evaluate our chosen weights using feature importance ranking

Uni-Ressources

7.1 Evaluation of the Hypothesis

How were the two sets gathered?
Input (APPENDIX)

Method

Result

7.2 Evaluation of the Indicator Weights

Method

Result

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.

[SN14] Joanna Smith and Helen Noble. "Bias in research". In: *Evidence-based nursing* 17.4 (2014), pp. 100–101.

[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

FEEDBACK SCREENSHOT

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

explain stuff whats in here:

Figure A.1: Exemplary feedback .md file.

REPRODUCIBILITY FACTOR SCORING (from 0 to 1):

Reproducibility factor	Score	Feedback	T*	A*	L*
Source-code availability and documentation	0.93	Score higher than top threshold (T). Very good.	0.8	0.54	0.28
Software environment	0.93	Score higher than top threshold (T). Very good.	0.61	0.46	0.31
Dataset availability and preprocessing	1	Score equal to top threshold (T). Very good.	1	-	0
Random seed control	1.0	Score higher than top threshold (T). Very good.	0.94	0.73	0.51
Model serialization	1	Score equal to top threshold (T). Very good.	1	-	0
Hyperparameter logging	0	Score equal to lower threshold (L). Major improvements should be made.	1	-	0
Out-of-the-box buildability	0	Score equal to lower threshold (L). Major improvements should be made.	1	-	0

> *: Thresholds computed from respective reproducible and non-reproducible sets of repositories. More information on this in the associated thesis in chapter 6.

SOURCE CODE AVAILABILITY AND DOCUMENTATION FEEDBACK

1. License scoring (score: 1.0 out of 1.0): All found licenses are open-source.
2. README overall scoring (score: 0.93 out of 1.0): See sub-ratings for more information.

- Average length scoring (score: 1.0 out of 1.0): We determined that 82 or more lines are best. We found 222.0 lines (in average) in the README file(s)
- Average accessible links scoring (score: 0.65 out of 1.0): We determined that 4 or more links are best. We found 3.0 accessible links (in average) in the README file(s)

3. Source-code pylint scoring (score: 0.63 out of 1.0): Not normalized pylint rating is 3.61 where 10.0 is best. We found that readable code is best for reproducibility and consider ratings over 5.71 as desirable.
4. Source-code-comment-ratio scoring (score: 1.0 out of 1.0): We measured a average ratio of 6.44. We consider a ratio below 8.73 as best. But more comments are fine too (which would make the ratio smaller). Well documented code is important.

> Source-code availability and documentation is calculated from the above identifiers. Since these have a different influence on the overall result, they are weighted as follows:

- License weight: 0.3
- Readme weight: 0.5
- Sub-readme: Length weight: 0.8
- Sub-readme: Average accessible links weight: 0.2
- Pylint rating weight: 0.1
- Code-comment-ratio weight: 0.1

SOFTWARE ENVIRONMENT FEEDBACK

1. Libraries used in the source-code and mentioned in config file scoring (score: 0.53 out of 0.6): We found that 87.5% of the used libraries are defined in the config file(s). We expect that 100% of the in the source-code used relevant libraries are defined in the config file(s).
2. Strictly defined libraries in config file(s) scoring (score: 0.2 out of 0.2): We expect that 100% of the defined libraries in the config file(s) are strictly (==) defined. We found 31 libraries in the config file(s). From these 100.0% were strictly specified.

- Public available libraries in source code file(s) scoring (score: 0.2 out of 0.2: We expect all of the not local modules or standard python libraries to be publicly available. We found that 100.0% are publicly available. We tested if the used library imports are accessible on https://pypi.org. If the score is not 0.2 (=100%): Please try avoiding the use of not public libraries as third parties may not be able to use your repository. Please note: It is also possible that, if the score is not the maxima, all libraries are publicly available, but we could not find a match. If you are unsure please recheck manually.

> Software environment is calculated from the above identifiers. Since these have a different influence on the overall result, they are weighted as follows:

- Source-code imports in config file weight: 0.6
- Strict dependency declarations in config file weight: 0.2
- Public libs in source code weight: 0.2

> Important note: For our analysis we exclude python standard libraries (see: https://docs.python.org/3/library/) as well as local file imports. We also eliminate duplicates (if one import occurs in multiple files we count it as one).

DATASET AVAILABILITY AND PREPROCESSING FEEDBACK

Dataset availability and preprocessing scoring (score: 1.0 out of 1.0): Dataset file candidate(s) were mentioned in the source code. This is best practice, because one should always provide a dataset (if possible) in the repository in order for others to reproduce your repository with the same input data.

> Important note: Dataset preprocessing detection is currently not implemented. But: If you preprocessed the dataset in any way please make sure to include either the final dataset or files to reproduce the steps taken.

RANDOM SEED FEEDBACK

Random seed lines with fixed seed scoring (score: 1.0 out of 1.0): We found that 100.0% of the 42 found random seed declaration lines had a fixed seed. If the value is not 100% make sure to fix the random seeds.

MODEL SERIALIZATION FEEDBACK

Model serialization scoring (score: 1.0 out of 1.0): Model serialization artifacts found. Model serialization helps in the field of machine learning, where incremental improvements should be documented in order for others to understand the steps taken. We look for one of the following: a) folders named ".dvc", b) files with the extensions ".dvc", ".h5", ".pkl" or ".model" c) one of the following keywords in the source code: "torch.save()", "pickle.dump" or "joblib".

HYPERPARAMETER LOGGING FEEDBACK

Hyperparameter logging scoring (score: 0 out of 1.0): No hyperparameter logging indicators found. We look out for imports of the following libraries in the source code: "wandb", "neptune", "mlflow" and "sacred". These libraries enable the logging of these parameters in order to document them. Also, we are looking for method calls of these libraries regarding logging. Logging changes of the hyper-parameters helps in the field of machine learning, where incremental improvements should be documented in order for others to understand the steps taken.

OUT-OF-THE-BOX BUILDABILITY FEEDBACK

Out-of-the-box buildability scoring (score: 0 out of 1.0): Tested building the repository with BinderHub resulted in an error. Repository is not buildable with BinderHub. Please try fixing this, as it greatly helps reproducing the found results. You can use the free and publicly available infrastructure accessible under https://www.mybinder.org to test. If you have included a Dockerfile in your repository it may not be compatible with BinderHub. Check: https://mybinder.readthedocs.io/en/latest/tutorials/dockerfile.html

B Dataset

These are the collected datasets. Also available in repo. Anonymized.

XY shows RS and NRS used for the implementation...

XY and CYZ are respective the datasets for evaluation part XY

Figure B.1: Relevant data points of the reproducible and non-reproducible data set.

Non-reproducible set (NRS)

id	avg readme length	# readme links	% not acc. links	readme ds reference	# licences	# os licenses	avg CCR	avg pylint score	# rdm seed lines	% fixed rdm seed	MS used?	# HP-logging indicators	% pub avail. libs	# config files	% strict decl. In conf	% decl. In conf of used in sc	# DSFC in sc	OOTB
1	90.0	1	100.0	0	1	1	1.07	0.0	3	100.0	No	0	100.0	1	25.0	80.0	0	Not tested
2	0	0	0	0	0	0	0.77	-7.0	0	0	No	0	100.0	0	0	0.0	0	Not tested
3	2.0	0	0	0	0	0	5.86	5.49	1	0.0	No	0	100.0	0	0	0.0	0	Not tested
4	2.0	1	0.0	0	1	1	2.82	3.67	0	0	Yes	0	100.0	0	0	0.0	0	Not tested
5	7.0	2	0.0	0	1	1	4.41	4.84	0	0	No	0	100.0	0	0	0.0	0	Not tested
6	4.0	1	0.0	1	0	0	6.02	4.58	0	0	No	0	100.0	0	0	0.0	0	Not tested
7	32.0	1	0.0	1	1	1	4.25	0.99	2	50.0	Yes	0	100.0	0	0	0.0	0	Not tested
8	0	0	0	0	1	1	27.5	3.18	4	100.0	Yes	0	83.33	0	0	0.0	0	Not tested
9	2.0	0	0	0	0	0	5.61	1.5	8	100.0	No	0	100.0	0	0	0.0	0	Not tested
10	11.0	0	0	0	0	0	7.84	0.0	0	0	No	0	100.0	0	0	0.0	0	Not tested
11	9.0	1	0.0	0	1	1	1.5	3.75	1	100.0	No	0	100.0	0	0	0.0	0	Not tested
12	2.0	0	0	0	0	0	19.8	5.66	0	0	No	0	100.0	0	0	0.0	0	Not tested
13	4.0	1	0.0	0	0	0	6.08	-1.14	0	0	No	0	100.0	1	93.75	66.67	0	Not tested
14	0	0	0	0	0	0	4.77	0.03	1	100.0	No	0	70.0	0	0	0.0	0	Not tested
15	21.0	3	0.0	1	0	0	3.86	-1.49	3	33.33	No	0	100.0	0	0	0.0	0	Not tested
16	18.0	0	0	0	1	1	11.71	6.22	12	83.33	No	0	91.67	1	100.0	75.0	0	Not tested
17	6.0	1	0.0	0	1	1	25.56	7.34	13	76.92	No	0	88.89	1	100.0	77.78	0	Not tested
18	3.0	0	0	0	0	0	16.18	-2.74	10	100.0	Yes	0	92.86	0	0	0.0	0	Not tested
19	18.0	2	0.0	1	0	0	4.33	4.66	4	75.0	No	0	60.0	0	0	0.0	0	Not tested
20	0	0	0	0	0	0	22.6	5.35	0	0	No	0	100.0	0	0	0.0	0	Not tested

Reproducible set (RS)

id	avg readme length	# readme links	% not acc. links	readme ds reference	# licences	# os licenses	avg CCR	avg pylint score	# rdm seed lines	% fixed rdm seed	MS used?	# HP-logging indicators	% pub avail. libs	# config files	% strict decl. In conf	% decl. In conf of used in sc	# DSFC in sc	OOTB
1	222.0	3	0.0	1	1	1	6.44	3.61	42	100.0	Yes	0	100.0	1	100.0	87.5	2	Not tested
2	83.0	4	0.0	0	0	0	4.4	7.53	33	100.0	No	0	100.0	1	100.0	53.85	3	Not tested
3	141.0	14	7.14	0	1	1	1.36	4.66	11	100.0	Yes	0	100.0	1	100.0	100.0	1	Not tested
4	79.0	4	0.0	0	1	1	1.59	5.24	4	75.0	No	0	88.89	1	0.0	44.44	1	Not tested
5	68.0	29	6.9	1	1	1	8.73	5.62	13	100.0	No	0	100.0	2	40.0	50.0	0	Not tested
6	41.5	5	0.0	1	1	1	3.32	5.8	6	100.0	No	18	80.0	1	100.0	26.67	7	Not tested
7	46.0	2	0.0	1	1	1	7.72	3.64	5	100.0	Yes	0	100.0	1	100.0	18.18	1	Not tested
8	103.0	24	0.0	1	1	1	5.27	5.94	4	100.0	Yes	0	90.91	1	10.0	63.64	1	Not tested
9	131.0	4	0.0	1	1	1	8.01	-6.19	3	100.0	Yes	0	100.0	1	33.33	60.0	0	Not tested
10	23.33	10	0.0	0	2	2	3.29	6.45	0	0	Yes	0	87.5	1	100.0	25.0	6	Not tested
11	26.0	4	0.0	1	1	1	7.98	2.97	16	100.0	Yes	0	88.89	1	100.0	66.67	3	Not tested
12	126.0	1	100.0	0	1	1	1.08	7.17	3	100.0	No	0	100.0	1	50.0	100.0	1	Not tested
13	161.0	31	0.0	1	1	1	6.05	6.32	8	100.0	Yes	0	94.74	1	0.0	36.84	2	Not tested
14	83.0	1	0.0	0	1	1	8.86	-1.4	3	100.0	Yes	0	100.0	1	0.0	80.0	3	Not tested
15	134.0	2	0.0	1	1	1	507	5.42	9	100.0	No	0	100.0	1	100.0	100.0	1	Not tested
16	292.0	19	94.74	1	1	1	5.41	6.01	9	100.0	Yes	0	83.33	1	100.0	50.0	0	Not tested
17	80.5	10	10.0	1	1	1	52.5	6.4	16	100.0	Yes	0	100.0	1	100.0	66.67	9	Not tested
18	15.0	1	0.0	0	1	1	4.18	3.37	41	100.0	Yes	0	100.0	0	0	0.0	0	Not tested
19	26.0	0	0	0	1	1	4.75	5.89	2	100.0	No	0	100.0	0	0	0.0	0	Not tested
20	81.0	12	0.0	1	0	0	13.4	6.78	7	100.0	No	0	100.0	0	0	0.0	0	Not tested

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, 27. 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, 27. April 2022

Martin Johannes Loos