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

1. Examiner: Prof. Dr. Michael Granitzer
Supervisor: Mehdi Ben Amor
Date: March 9, 2022

Contents

Abstract

Please write a short abstract summarizing your work.

Acknowledgments

I would first like to thank ...

List of Figures

List of Tables

1 Introduction

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

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

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

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

In this thesis, we are interested in the identification and classification of reproducibility factors which influence machine learning experiments. Firstly, a precise definition of the term reproducibility [KS15] is needed. In an article from 2014 McNutt states that "[...] just because a result is reproducible does not necessarily make it right, and just because it is not reproducible does not necessarily make it wrong." [McN14]. Although this statement is correct, experiments can only be verified if they can be reproduced. The results from an academic

paper must be verified by third parties to be recognized by the general public. Therefore, the reproducibility of experiments is not only desirable but necessary [IT18a]. A 2016 Nature survey, where 1,576 researchers from different scientific fields participated, found that "More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments." [Bak16]. This indicates that we are in a reproducibility crisis. We want to address this problem by creating a basis for software-based analysis of reproduction factors. Hence, we must first determine which factors influence reproducibility.

The identification of factors that affect reproducibility is an important subject of current research [Ban+21; Eis18; GGA18; IT18b; Liu+21; MK17; McD+19; OBA17; SK21]. NeurIPS, a machine learning and computational neuroscience conference, has even included reproducibility as part of its submission policy [Pin+19]. In a report from the NeurIPS 2019 reproducibility program, processes and guidelines were listed that are intended to improve reproducibility. Software tools that would support researchers were not introduced in this article, but the authors noted that the "Standardization of such tools would [...] improve ease of reproducibility" [Pin+20]. Therefore, we will collect the findings on this from other scientific papers and process them systematically. In addition, we will delimit which of the identified reproducibility factors also influence replicability.

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

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

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

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

2 Background

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

2.1 Reproducibility

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

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

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

2.1.1 Reproducibility vs. Replicability

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

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

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

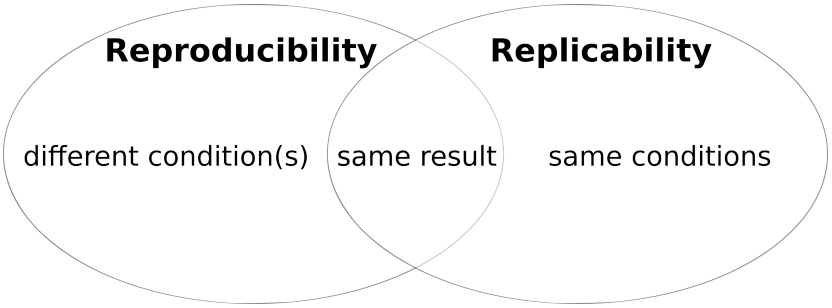


Figure 2.1: Reproducibility versus Replicability.

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

2.1.2 Challenges & best practices

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

Olorisade et al. state that "[...] it may sometimes be hard or even impossible to reproduce computational studies [...].[OBA17]". Furthermore, they point out that "[...] the minimum standard expected of any computational study is for it to be replicable.". Note: We have changed the terminologies for reproducibility and replicability in the citations to match our definitions. This again shows the need to create at least replicable work. In the following sections we present some key technologies that simplify this. In addition, we give an overview of the current state of affairs in chapter **??**. Also, we show which steps can be taken to increase the reproducibility probability without having to do a lot of additional work.

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

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

2.2 Containerization

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

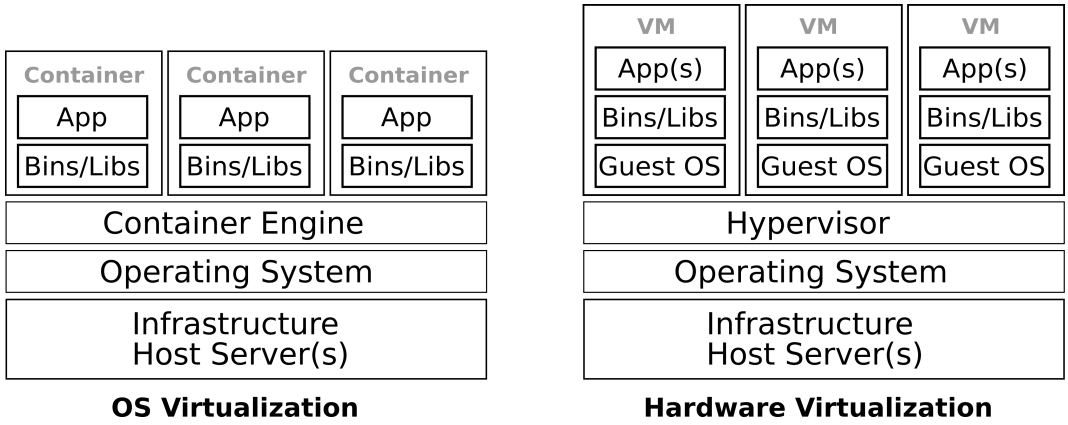


Figure 2.2: OS Virtualization versus Hardware Virtualization.

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

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

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

2.2.1 Docker

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

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

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

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

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

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

2.2.2 Kubernetes

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

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

2.3 Jupyter Notebooks

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

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

One possible use case is lab notebooks, which are a log of scientific activity. These should contain every detail (e.g. hypothesis, experiments, and interpretation of results) which later

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

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

2.4 Binder

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

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

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

2.4.1 BinderHub

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

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

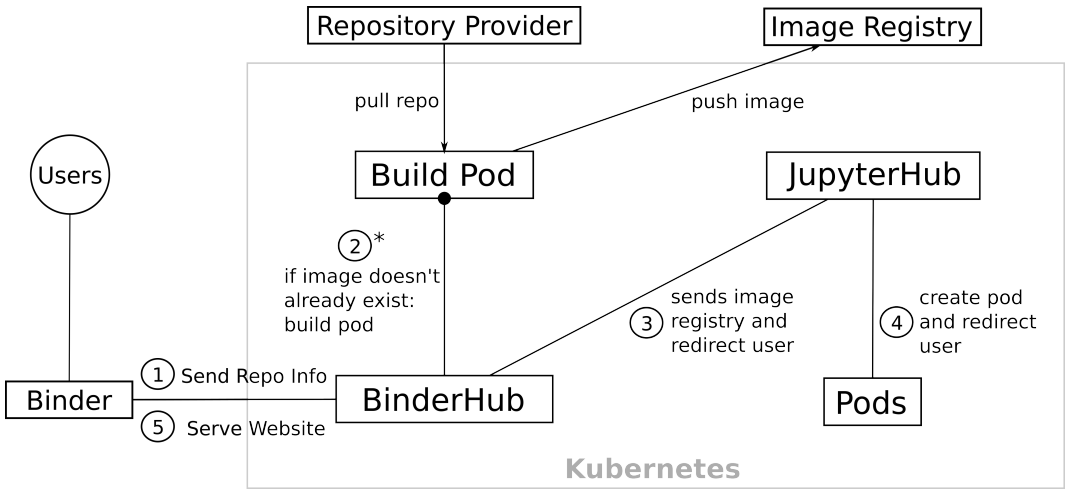


Figure 2.3: Adapted figure from the official BinderHub documentation.[4]

2.4.2 repo2docker

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

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

The following configuration files[5] are currently supported:

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

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

2.4.3 JupyterHub

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

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

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. Here the BinderHub switches on beforehand to ensure that an image for the repository is stored in the registry.

2.5 Machine learning workflow

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

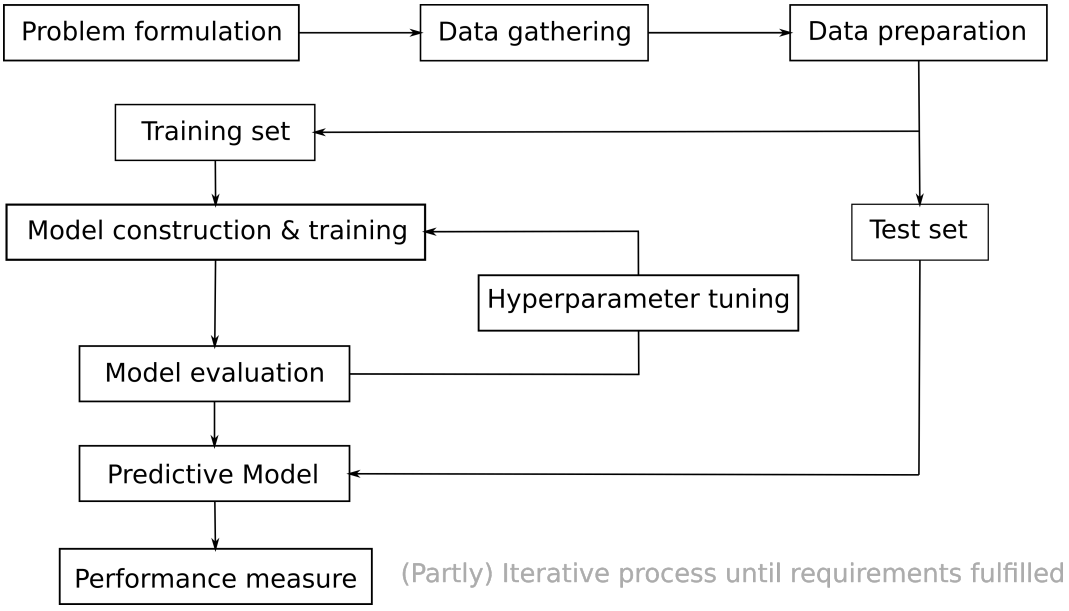


Figure 2.4: Abstracted machine learning workflow.

2.5.1 Data gathering & preparation

Matching the problem to be solved, a suitable data set is first selected as input. The content of the data record is of no further importance from the point of view of reproducibility. We

are more interested in whether this data is publicly available.

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

2.5.2 Train-test split

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

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

2.5.3 Model construction & training

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

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

2.5.4 Model evaluation

After a model has been trained it can be evaluated using the test data set. This set consists of data that the model has never seen before. Based on specified requirements, such as a baseline or previous iterations, a decision is then made as to whether the model is good enough. If

it does not meet the expectations, one can, for example, adjust hyperparameters in a new iteration or change the general learning strategy to achieve a better result.

2.5.5 Performance measure

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

3 Related work

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

Identification of reproducibility factors: Since our goal is to classify reproducibility factors that are relevant for machine learning experiments, we have to find them beforehand. To do this, we use existing knowledge and summarize the findings. However, influencing factors are ML-specific to different degrees. Every author of a scientific paper should be aware of general reproducibility hurdles [Eis18; SK21]. On the other hand, some factors only appear when software is part of the work [IT18b; MK17]. In addition, there are also factors that occur specifically in the ML area [Ban+21; GGA18; Liu+21; McD+19; OBA17].

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

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

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

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

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

4 Identification of reproducibility factors

FOR EACH FACTOR: why/how influencing repro/repli? source? TABLES below are just reminders, they will be moved to the classification/implementation part

4.1 Source code availability & documentation

SCAD

TABLE
Level 0 not available and not open-source
Level 1 available but not open open-source
Level 2 Level 1 and open-source but not documented
Level 3 Level 2 and with sufficient readme
Level 4 Level 3 and with sufficient code-documentation
Level 5 Level 4 and clean code (pep8)

4.2 Hardware environment

HE

SPECIFIED OR NOT SPECIFIED

Is the used hardware described? Hardware is changing over time so in 20 years it might be difficult to reproduce the hardware env anyway. Extincted hardware

4.3 Software environment

SE

Is the used software specified? To which level of detail (library versions? os version? etc.)?

TABLE
Level 0 no config
Level 1 config but not all libraries from config file specified
Level 2 Level 1 and all libraries in config file specified
Level 3 Level 3 and all used libraries in source-code specified in config file

4.4 Out-of-the-box buildability

OOTBB

Can the repo builded with binder out of the box? if yes -¿ fast setup to test/check the relevant stuff

BUILDABLE OR NOT BUILDABLE

4.5 Dataset availability & preprocessing

DA

Is the used dataset accessible? Needed for replication

Is it clear how the dataset was preprocessed? Needed for replication but also for reproducibility

TABLE
Level 0 not available
Level 1 available but not documented how preprocessed
Level 2 Level 1 + preprocessing documented

4.6 Random seed control

RSC

FIXED SEED OR NOT FIXED SEED

4.7 Hyperparameter declaration

HPD

DECLARATED OR NOT

4.8 Model serialization

MS

SERIALIZATION ARTEFACT OR NOT

4.9 Research practices & experimental design

RPED

Paper explanation, How well are the methods described? etc.

Can be estimated see paper from related work. But we only check if paper is accessible.

SPECIFIED OR NOT SPECIFIED

4.10 Overfitting & Underfitting

OF

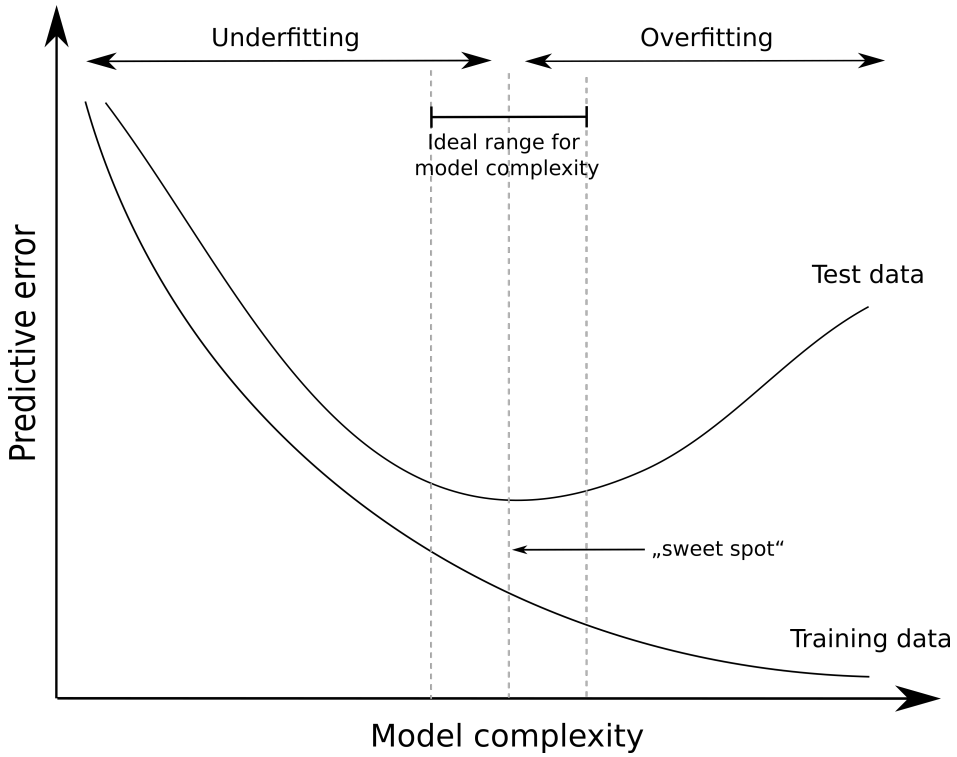


Figure 4.1: Adapted figure explaining Overfitting & Underfitting [Hea+15].

explain how they may influence repo/repli

VERY COMPLEX!

FIGURE

4.11 Knowledge gap

KG

4.12 Probability hacking

PH

4.13 Bias

BI

Different types of biases

5 Classification of reproducibility factors

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

Def. Factor

Def. Indicator

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

Connection of factors and indicators? (figure)

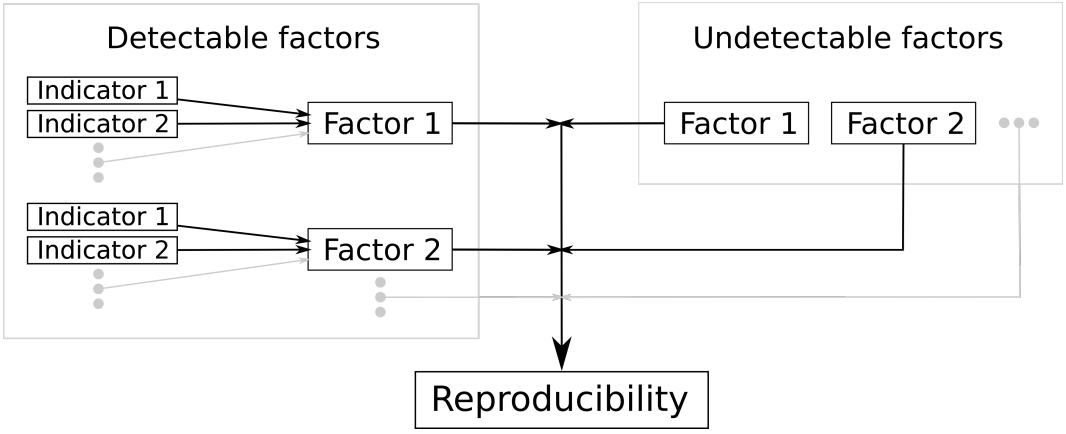


Figure 5.1: Connection of factors, indicators and reproducibility.

5.1 Detectable factors

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

5.1.1 Assignment of indicators

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

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

Table 5.1: Reproducibility indicators with their corresponding indicator

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

This figure shows the relationship indicator -¿ factor

5.1.2 Complexity scale

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

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

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

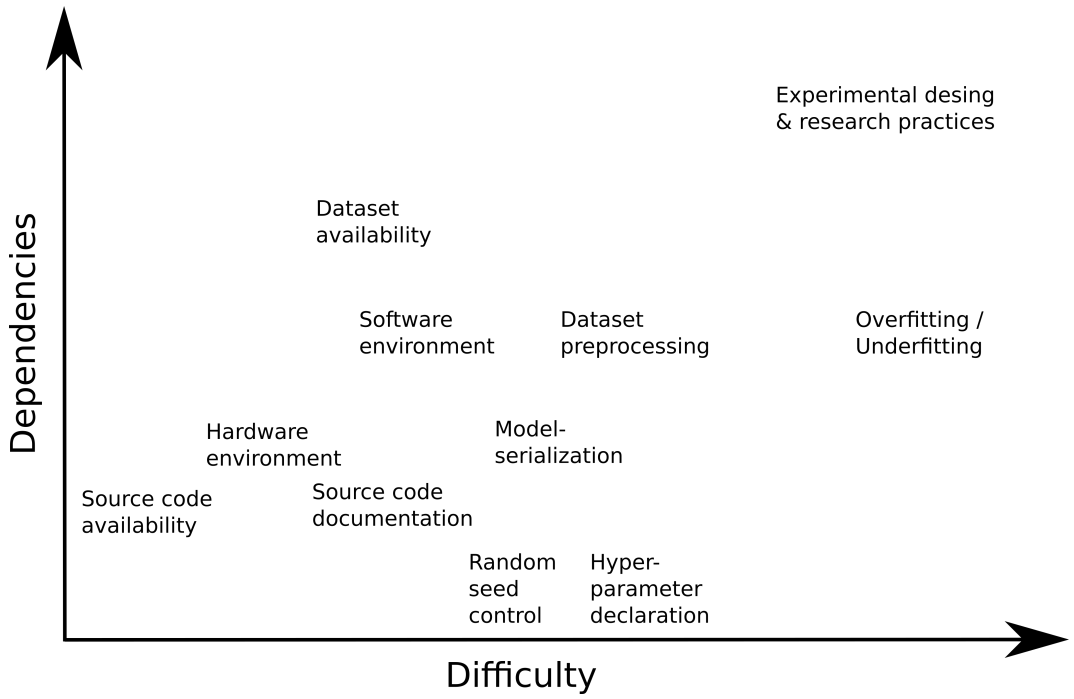


Figure 5.2: Complexity scale of the detectable reproducibility factors

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

5.2 Undetectable factors

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

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

6 Implementation

This chapter explains how to deal with the identified and classified factors in practice.

NOTE: PROBABLY NO IMPLEMENTATION FOR UNDETECTABLE FACTORS (TIME)

IMPLEMENTATION IS PROOF OF CONCEPT AND CAN BE IMPROVED IN A LOT OF WAYS (SEE LIMITATIONS & FUTURE WORK)

6.1 Analysis tool for software-based measurable data

Important note: The tool analyzes a repository whether indicators are present or not. There are indicators (Chapter + Section xy) for each detectable factor. The goal is to be able to give a user fast, automated feedback as to whether an ML repository is reproducible or not.

6.1.1 Assumptions

Restrictions? (JupyterNotebook, Pyhton Repos) but extendable (HOW). Proof of concept

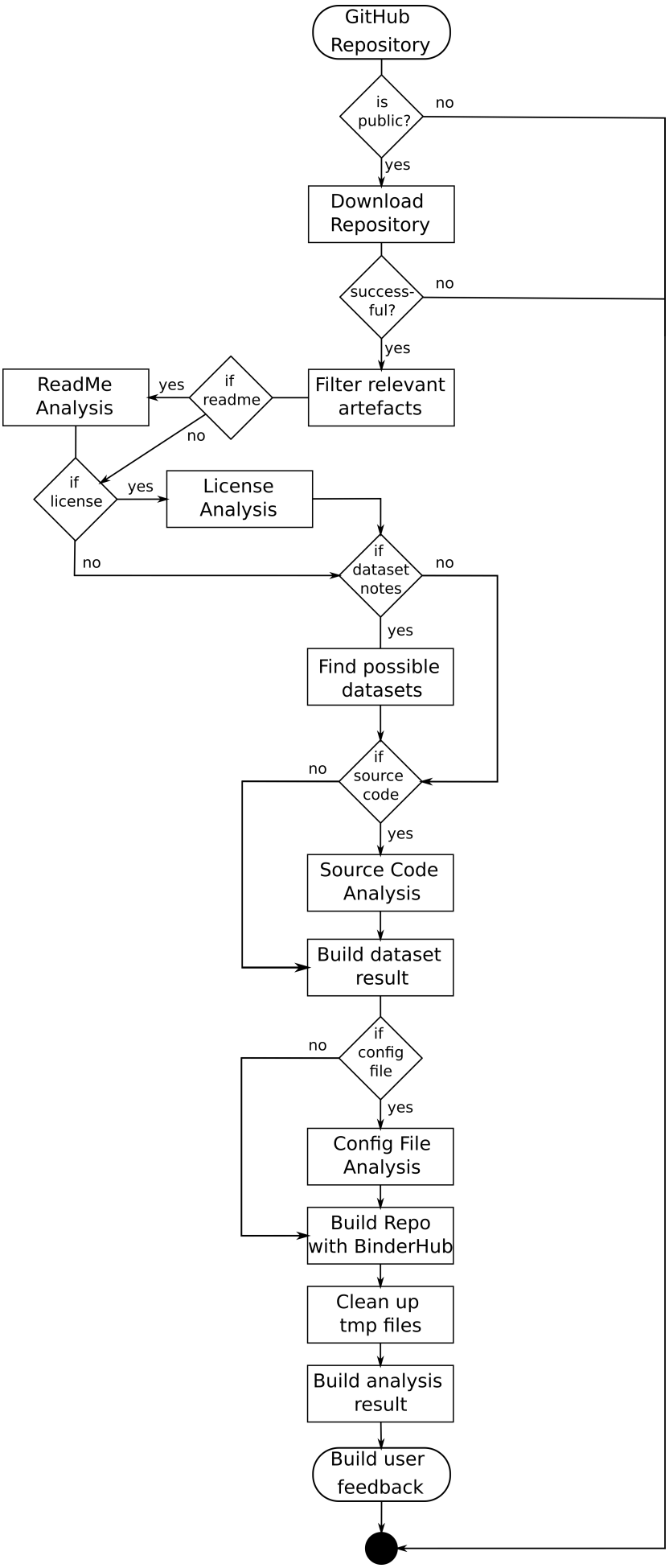
Git repositories can be very different in terms of structure. The goal of this thesis is not to develop an analysis tool for all types of repositories. Rather, it is about showing a way to measure all detectable factors. The only assumption we have made about the chosen repositories is that they have been written in python. Of course, they also have to be ML-related.

6.1.2 Implementation details

How is the tool structured? FIGURE OF ARCHITECTURE + TABLE

How do individual parts of the tool work exactly. EXPLAIN THE ALGORITHM AND THOUGHT-PROCESS.

The individual steps of the analysis tool are shown on a higher level in the figure below.


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

As input, the analysis tool expects a git repository that meets the requirements mentioned in **??**.

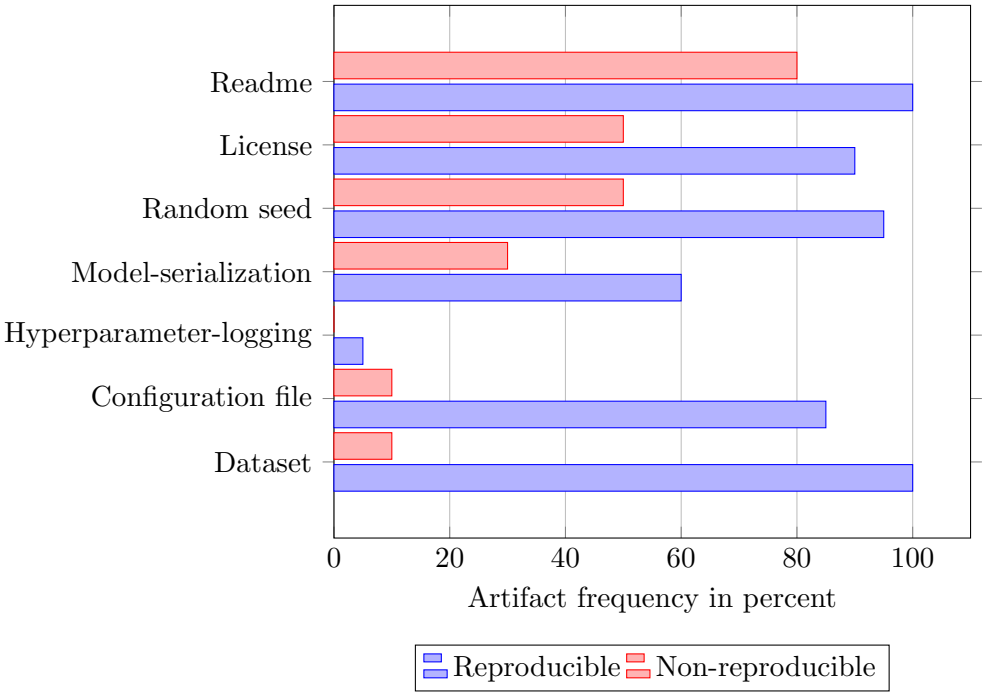
6.1.3 Representative indicator values & weights

What output do we expect? -¿ We check against indikators but want factors. Connction based on base line. We retrieve the base line by analysis ground truth reproducible ML repos and analyze what values are statistically good.

Based on this baseline we implement the algorithm wether the tool assumes good reproducibility.

Twenty reproducible and ten non-reproducible repositories were analyzed using the tool to retrieve representive values for each indicator. More repositories for each set would be better and more representive. But due to the fact, that we had to analyze them manually, more was not possible in the given time span.

Figure 6.2: Artifact frequency comparison



We observe, that the set of reproducible repositories perform across all categories significantly better. The exception is the existence of a readme artifact. This is not surprising, since this file can also be created during repository creation by setting a check mark (which is often

already preselected).

TABLE(S) OF BASELINE VALUES FOR EVERY FACTOR + INTERPRETATION

SOURCE CODE AVAILABILIITY AND DOCUMENTATION:

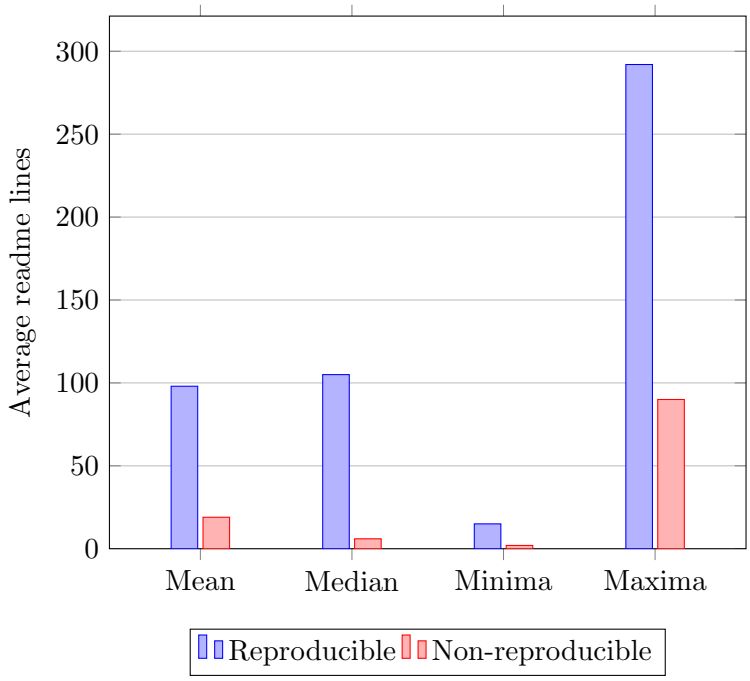
Since the existence of the readme file has apparantly no meaningfulness, we analyze which of the following properties allow conclusions to be drawn about the reproducibility probability:

Readme length
Number of readme links
% not accessible links

We found that readme links + % not accessible links don't differ that much between reproducible and non-reproducible ones. -¿ we don't mind how much links there are, but we still want them to be usable if they are there. The overall impact on reproducibility seems not that much.

readme links: Reproducible median: 8; Non-reproducible median: 1.5 (difference not that surprising since the reproducible set has on average longer readmes)
% not accessible links: Reproducible median: 0%; Non-reproducible median: 0% (no difference, but still important)

Therefore, the weight of the readme length should be the highest (by far), number of readme links should not have any impact at all, % not accessible links should have at least some impact (it's usually bad if links used to describe something aren't accessible).



Additionally, we analyze the average code-comment-ratio and pylint score of the source code files. While the median regarding the code-comment-ratio was about the same (around 5:1) for both sets, differences were noticed regarding the pylint score. The mean for the reproducible set is 4.71. For the non-reproducible set we got a mean of 1.73. Since the code-comment-ratio didn't differ much, we argue that no conclusions can be drawn about the reproducibility probability.

SOFTWARE ENVIRONMENT:

As figure 6.2 shows, the existence of a configuration file indicates reproducibility by itself. But we also want to check if the config file is meaningful. For this we analyze the following properties:
Number of declarations in configuration file
IS THERE A CORRELATION BETWEEN IT AND GOOD/BAD? Percentage of strict (==) declarations
SHOULD ALWAYS BE 100% Percentage of declarations to all used
BETTER: Percentage of used and declarated SHOULD ALWAYS BE 100% Percentage of public availabe libraries SHOULD ALWAYS BE 100%

DATASET AVAILABILITY & PREPROCESSING:

no preprocessing functionality (POC Implementation)

As figure 6.2 shows, the availability of the dataset indicates reproducibility by itself. But we also want to check if other measurements are meaningful. For this we analyze the following properties:

Number of dataset candidates
IS THERE A CORRELATION BETWEEN IT AND GOOD/BAD? Number of mentioned

datasets in source-code
IS THERE A CORRELATION BETWEEN IT AND GOOD/BAD? Percentage mentioned in relation to all found
MAYBE BETTER: If we found at least 1 match should be good to go. For everything else we need more context.

ONLY IMPORTANT ONE: FOUND DATASET FILE CANDIDATE + ALSO MENTIONED VALUE ¿= 1

AVAILABLE (mentioned ¿= 1) OR NOT AVAILABLE (no dataset file candidates) INBETWEEN (NOTE) found file candidates but nothing in code matches

RANDOM SEED:

IF THERE 100% should be fixed

no differences found between reproducible and non-reproducible repositories (fixing seed seems to be standard) if one is declared

MODEL-SERIALIZATION:

double the usage of model serialization in reproducible repos in comparison to non-reproducible ones

BUT VALUE IS USED OR NOT USED

HYPERPARAMETER-LOGGING:

only found one occurance of HP logging in all reproducible and non-reproducible repos

It was one reproducible that had indicators (18). Also that repository didnt use MS.

COMBINE Model-serialization + HP Logging?

Goal is the same for both. MS is generally more powerful. But both document the steps taken.

Combination: MODEL DOCUMENTATION

OUT-OF-THE-BOX BUILDABILITY:

Didn't evaluate this property. Because: Can either be buildable or not. No indirect indicator.

6.1.4 Factor-Indicator-Connection using range normalization

We then normalize the results and build a baseline for each indicator as a reference for the tool. Based on this baseline and the retrieved indicator weights we can then connect the indicators with the factors. We extracted the weights using heuristic and range normalized the baseline values. The weights for each factor sum up to 1.

algorithm to make identificator values comparable to others (using same range) RANGE NORMALIZATION ALGORITHM (mention source or equation useable anyway?):

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

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

7 Evaluation

Why important?

7.1 Evaluation setup

Uni-Ressources

Input (APPENDIX)

7.2 Statistical evaluation

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

8 Discussion

FINDING, EXPECTATIONS, etc.

8.1 Limitations

Ground truth base line input size small

Evaluation input small

python language only

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

8.2 Future work

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

9 Conclusion

summary of findings etc.

Bibliography

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

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

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

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

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

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

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

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

[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.

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

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

[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.

[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.

[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.

[OBA17] 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.

[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.

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

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

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

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

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

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

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

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

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

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

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

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

A Code

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

B Dataset

C Content of the CD

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

Declaration of Academic Integrity / Eidesstattliche Erklärung

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

Passau, 9. März 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, 9. März 2022

Martin Johannes Loos