ML2R Coding Nuggets
Reproducible Machine Learning Experiments

Lukas Pfahler
Artificial Intelligence Group
TU Dortmund University
Dortmund, Germany

Alina Timmermann
Artificial Intelligence Group
TU Dortmund University
Dortmund, Germany

Katharina Morik
Artificial Intelligence Group
TU Dortmund University
Dortmund, Germany

ABSTRACT

The scientific areas of artificial intelligence and machine learning are rapidly evolving and their scientific discoveries are drivers of scientific progress in areas ranging from physics or chemistry to life sciences and humanities. But machine learning is facing a reproducibility crisis that is clashing with the core principles of the scientific method: With the growing complexity of methods, it is becoming increasingly difficult to independently reproduce and verify published results and fairly compare methods. One possible remedy is maximal transparency with regard to the design and execution of experiments. For this purpose, best practices for handling machine learning experiments are summarized in this Coding Nugget. In addition, a convenient and simple library for tracking of experimental results, `meticulous-ml` [17], is being introduced in the final hands-on section.

1 REPRODUCIBLE MACHINE LEARNING

The release of data and code was declared a necessary condition for scientific publication [15]. Unfortunately in machine learning research, unpublished code and sensitivity to exact training conditions make many claims hard to verify [8]. Thus, most computational research results presented today at conferences and in publications cannot be verified [3]. Reproduction of ML experiments can be necessary due to biases, scientific diligence, fraud [4] or means of comparison. There is also the assumption that common demand for high publication rates encourages misleading discoveries [20]. It seems that addressing reproducibility issues requires a transformation of publication culture [12]. Therefore, reproducibility in machine learning and data science gained more importance at top conferences in the last decade [1, 8, 18, 21].

There are no defined terms for reproducibility. Some distinguish reproducibility and replicability [12]. Reproducibility focuses on recreating results by using the original code while replication generates results independently from original data [21]. Others differentiate reproducibilty of results and reproducibility of findings [1]. Reproducibility of results relates to the replication of generated numbers. The validity of experimental conclusions is covered by reproducibility of findings. Codesharing is an important practice for improving reproduciblity of results, but insufficient for obtaining reproducibility of findings. Some researchers argue that appropriate experimental design is key for achieving adequate overall model performance and high reproducibility of findings [1, 13, 14]. In this paper, the reproducibility of results will be the main topic. For this purpose, common best practices for codesharing, result-tracking and archiving, and for proper experimental design will be summarized. Furthermore, a solution in form of the python library `meticulous-ml` created by Ashwin Paranjape will be presented.

2 BEST PRACTICES

For ensuring proper reproducibility of results of machine learning experiments, there will be suggestions for some best-practices in the following.

Version control of Source Code. All the source code written to execute the experiments should be under version control. The de-facto standard tool for version control is Git [22], although alternatives exist. Code should run from clean Git repositories only, i.e. repositories with no uncommitted changes. This allows to associate the exact status of the code with the experiment run by its unique commit identifier that is provided by the version control software. Consequently, to replicate an experimental result, we can revert the code to the exact version used.

Version Control of all Dependencies. Most machine learning experiments rely heavily on software libraries such as Tensorflow or Scikit-learn. These libraries are rapidly evolving and new versions are released frequently. Hence it is important to keep track of which version of libraries was used in an experiment. It is generally recommended to use one environment for each project using tools like `Venv` or Anaconda for python environments, or Docker [2] on the operating system level, and specify which libraries should be installed using the respective config-files. However, we should not rely on these config-files for ensuring reproduciblity: They often allow vague version constraints (e.g. “newer than v1.2”), and a user can update a library without also editing the environment config-file. Hence we should also capture the software versions at runtime.

Version Control of Data. There are less established software solutions for version control of data, though research data management is becoming increasingly important in science and more protocols are established in institutions [24]. For instance, open science data is often published and associated with a unique digital object identifier (DOI). Locally, all scripts that access, filter or preprocess the data should also be under version control. Whenever possible, we should include a script for obtaining the original data, alternatively we can maintain a description of how and where to obtain it. Intermediate data files, e.g. preprocessed data, should have meta data that contains, among others, a timestamp, a reference to the git commit of the preprocessing utility script that has been used, as well as a reference to all the input files that were used to generate the output. Many file formats support attaching meta data. For instance, in `.json` files or `.hdf5` files a new field for meta data can be introduced and `.csv` files can begin with a comment section. If this is not possible, a metadata file should be stored next to the data file and copied around with the data.

Tracking of all Hyperparameters. Machine learning methods, particularly deep learning approaches, have many hyperparameters and much time is spend tuning these parameters to maximize performance, either manually or automatically. Thus, we need to track all these hyperparameters to replicate the experiment later. When we are interested in reproducing the results of randomized algorithms, it is important to also track the random seeds used for the random generator.

Tracking of all Results. We want to archive all the outcomes of an experiment. That includes any metric, including runtime, that we evaluate to judge the quality of a machine learning model, but also other outputs like the model itself or any other result files. It is important that each result or output can be associated with the corresponding experiment. Often it is also useful to capture all console outputs as well as error messages in text files.

Reproducibility of Findings. For maximum reproducibility of findings, experiments should be carried out multiple times with various initialization and different environments. These practices result in claims with sufficient statistical significance [7]. For training data, unbiased data in large quantities should be used. Negative outcomes in an experimental setup should also be published [19]. These measures potentially highlight pros and cons of a model and enhance the understanding of how or when it operates superior.

Use Experiment Tracking Software. All the above information needs to be tracked, linked and archived. For this purpose, software solutions such as Sacred [6], MLflow [10], Tensorboard, Wandb [23], Theano [11] or Gym [16] exist and should be used. They provide convenient solutions to reproducibility that do not require changing large amounts of code. When we do machine learning experiments on cloud platforms, they often provide their own tools for reproducibility [9]. In the next section, we will present another software solution, `meticulous-ml`, in greater detail.

3 THE `meticulous-ml` LIBRARY FOR PYTHON

In this section, we present the python library `meticulous-ml` [17], originally written by Ashwin Paranjape, that supports machine learning researchers by handling many of the requirements for reproducible research established above, while requiring only minimal code changes for existing experiment scripts. Perhaps most importantly, it is, as Hady Elsahar puts it, "*suitable for the messy, clueless nature of research*" [5]. We see an example for tracking a simple machine learning experiment, training and evaluating a random forest classifier, in Listing 1 and continue to discuss the highlighted, crucial changes to incorporate meticulous.

First, to install `meticulous-ml`, we recommend using pip:

```
pip install \
    "git+https://github.com/Whadup/meticulous-ml"
```

but manual installation is also possible.

To enable tracking of an experiment, we have to instanciate an `Experiment` object. One convenient way is to derive it from the argument parser, simultaneously capturing all the hyperparameters provided by the user. Alternatively, we can manually provide all arguments that we want to capture. When the experiment is initialized, a folder is created on the local filesystem. It contains all information related to this experiment run in human-readable format: All standard output and errors will be captured, the hyperparameters provided through the argument parser will be stored and the meta data discussed in the previous section, including version control information, used software libraries, execution time, etc., will be captured.

When we want to capture metrics like accuracy or f1-score, we can use the `experiment.summary()` method. Now these scores are stored in a standardized way and can easily be retrieved and compared, for instance using the CLI tool described below.

Listing 1: Capturing an experiment with meticulous-ml. We highlighted the changes necessary to ensure reproducibility.

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, f1_score)
import argparse
import pickle
from meticulous import Experiment

parser = argparse.ArgumentParser ()
parser.add_argument('--max-depth', type=int , default=8)
parser.add_argument('--dataset', type=str , default="mozilla4")
# Adds the "meticulous" argument group to your script
Experiment.add_argument_group(parser)
# Creates experiment object and extracts all hyperparameters
experiment = Experiment.from_parser(parser)
args = parser.parse_args()

#load and split training data
X, y = fetch_openml(args.dataset, return_X_y=True)
y = LabelEncoder().fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# train and validate the model
model = RandomForestClassifier(max_depth=args.max_depth)
model.fit(X_train, y_train)

# Log metrics in experiment directory
accuracy = accuracy_score(model.predict(X_test), y_test)
f1 = f1_score(model.predict(X_test), y_test)
print(f"Accuracy: {accuracy:.4f}, F1-Score: {f1:.4f}")
experiment.summary(accuracy=accuracy, f1=f1)

# Writes model file to the experiment directory
with experiment.open('model.pickle', 'wb') as f:
    pickle.dump(model , f)
```

Listing 2: Automatically generated file hierarchy of the `meticulous-ml` experiment directory (left) with description of each file or sub-directory (right).

```
$ find ./experiments_dir

    ./experiments_dir/   ← directory for all records
      1/                 ← first experiment run
        [...]
      2/                 ← second experiment run
        args.json        ← all cmd-line arguments
        some_output.txt  ← we can save custom files, too
        metadata.json    ← all meta-data
        STATUS           ← success, failure or still running?
        stdout           ← captured std-output
        stderr           ← captured std-error
        summary.json     ← all user summaries
```

Listing 3: Example for using the CLI tool `meticulous` to aggregate metrics and summarize experimental results. Meticulous automatically computes mean and standard deviation for numerical summaries if we use the `--groupby` argument.

```
$ meticulous experiment_dir
    --filter 'args_dataset=="magictelescope"' \
    --groupby 'args_max_depth' \
    --sort 'summary_accuracy_mean' \
    --columns 'args_max_depth,summary_accuracy_mean,summary_accuracy_std,summary_f1_mean,summary_f1_std'
    --export results.tex
```

	args_max_depth	summary_accuracy_mean	summary_accuracy_std	summary_f1_mean	summary_f1_std
7	10	0.8706	0.0041	0.7946	0.0080
6	9	0.8681	0.0045	0.7913	0.0068
5	8	0.8610	0.0056	0.7756	0.0080
4	7	0.8515	0.0048	0.7579	0.0079
3	6	0.8486	0.0048	0.7488	0.0070
2	5	0.8379	0.0062	0.7269	0.0101
1	4	0.8288	0.0060	0.7041	0.0124
0	3	0.8038	0.0080	0.6382	0.0165

When we want to store and archive files, replace `open()` with `experiment.open()` to obtain a file handle in the folder associated to the experiment run. Listing 2 summarizes the folder structure automatically created to archive the experiment results by `meticulous-ml`.

To ease browsing the results, we recommend using the CLI tool `meticulous` to select, filter, group and sort the runs and export the resulting tabular data into a variety of formats. In Listing 3 we see an example for analyzing the experimental results of the performance of the random forest classifier in our example. We have run this experiment for different values of `max_depth`, using multiple runs per configuration. We filter the results to only show runs relating to the `magictelescope` dataset and compute means and standard deviations over the repeated runs. We sort the results to show the runs with the best average accuracy first and select only a subset of the table columns. Exporting this table in LaTex format allows us to easily include the results in a scientific article.

This concludes the quick introduction into the `meticulous-ml` library: We have seen that minimal code changes already result in large increases in reproducibilty of results. Furthermore, it provides a convenient commandline tool for browsing, summarizing and exporting the results.

ACKNOWLEDGMENTS

This material was produced within the Competence Center for Machine Learning Rhine-Ruhr (**ML2R**) which is funded by the Federal Ministry of Education and Research of Germany (grant no. 01|S18038C). The authors gratefully acknowledge this support.

REFERENCES

[1] Xavier Bouthillier, César Laurent, and Pascal Vincent. 2019. Unreproducible Research is Reproducible. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR.

[2] Inc. Docker. 2021. Docker Website. https://www.docker.com/

[3] David Donoho, Arian Maleki, Inam Ur Rahman, and Morteza Shahram. 2009. 15 Years of Reproducible Research in Computational Harmonic Analysis. *Computing in Science & Eng.* 11, 1 (2009), 8–18. https://doi.org/10.1109/MCSE.2009.15

[4] David Eisner. 2017. Reproducibility of Science: Fraud, Impact Factors and Carelessness. *Journal of molecular and cellular cardiology* (2017). https://doi.org/10.1016/j.yjmcc.2017.10.009

[5] Hady Elsahar. 2019. How do you manage your Machine Learning Experiments? https://hadyelsahar.medium.com/how-do-you-manage-your-machine-learning-experiments-ab87508348ac

[6] Klaus Greff, Aaron Klein, Martin Chovanec, and Frank Hutter. 2017. The Sacred Infrastructure for Computational Research. In *Proceedings of the 15th Python in Science Conference (SCIPY 2017)*. 49–56.

[7] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2019. Deep Reinforcement Learning that Matters.

[8] By Matthew Hutson. 2018. Artificial intelligence faces reproducibility crisis. 359, 6377 (2018).

[9] Richard Isdahl and Odd Erik Gundersen. 2019. Out-of-the-box Reproducibility : A Survey of Machine Learning Platforms. November (2019). https://doi.org/10.1109/eScience.2019.00017

[10] Harutaka Kawamura, Arjun DCunha, Andrew Chen, Richard Zang, and Others. [n. d.]. MLflow. https://mlflow.org/

[11] LISA lab Revision. 2021. Theano Documentation. https://theano-pymc.readthedocs.io/en/latest/

[12] Randall J. LeVeque, Ian M. Mitchell, and Victoria Stodden. 2012. Tools and Strategies. *Computing in Science and Engineering* (2012), 13–17.

[13] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. 2018. Are GANs Created Equal? A Large-Scale Study.

[14] Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the State of the Art of Evaluation in Neural Language Models.

[15] National Academy of Sciences. 2021. PNAS's submission guidelines including data sharing plans. https://www.pnas.org/authors/submitting-your-manuscript#manuscript-formatting-guidelines

[16] OpenAI. 2021. Gym Website. https://gym.openai.com/

[17] Ashwin Paranjape and Lukas Pfahler. [n. d.]. meticulous-ml. https://github.com/Whadup/meticulous-ml/

[18] Edward Raff, Booz Allen Hamilton, and Baltimore County. 2019. A Step Toward Quantifying Independently Reproducible Machine Learning Research. NeurIPS (2019).

[19] D. Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. 2018. Winner's Curse? On Pace, Progress, and Empirical Rigor. https://openreview.net/forum?id=rJWF0Fywf

[20] Paul E. Smaldino and Richard McElreath. 2016. The natural selection of bad science. *Royal Society Open Science* 3, 9 (Sep 2016), 160384. https://doi.org/10.1098/rsos.160384

[21] Victoria Stodden, Friedrich Leisch, and Roger D. Peng. 2014. *Implementing reproducible research.* CRC Press.

[22] Linus Torvalds, Junio Hamano, and Others. 2005. Git - fast, scalable, distributed revision control system. https://git-scm.com/

[23] Weights & Biases. 2021. wandb. https://docs.wandb.ai/quickstart

[24] Mark D. Wilkinson and others. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* 3, 1 (2016), 160018. https://doi.org/10.1038/sdata.2016.18