PyChemFlow: an automated pre-processing pipeline in Python for reproducible machine learning on chemical data

Mario Lovrić *,[1,2,3], Tomislav Duričić [4], Hussain Hussain [4], Bono Lučić [5], Roman Kern [4]
[1] Mario Lovrić, Centre for Applied Bioanthropology, Institute for Anthropological Research, 10000 Zagreb, Croatia
[2] Mario Lovrić, Faculty of Electrical Engineering, University of Osijek, Kneza Trpimira 2b, HR-31000 Osijek, Croatia
[3] Mario Lovrić, Copenhagen Prospective Studies on Asthma in Childhood, Herlev and Gentofte Hospital, University of Copenhagen, Copenhagen, Denmark
[4] Tomislav Duričić, Hussain Hussain, Roman Kern, Know-Center, Sandgasse 36, AT-8010 Graz
[5] Bono Lučić, Ruđer Bošković Institute, Bijenička Cesta 54, HR-10000 Zagreb
*Corresponding author: mario.lovric@ferit.hr

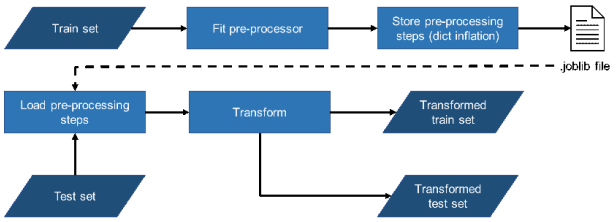
Abstract: *PyChemFlow* is a Python library for automated and reproducible data pre-processing. Based on open-source code, *PyChemFlow* has simple requirements that rely on pandas, scikit-learn and joblib. The library's backbone is built up of transformer objects, which are fully constructed during the PyChemFlow fitting process using training data and can be conveniently stored using joblib. The user can run the library with a one-line command after splitting data into train and validation sets or while working with additional data. This is especially useful when reproducibility is critical. *PyChemFlow* also offers the ability to persistently store metadata, in addition to providing customizable and configurable data manipulation steps.

1 Introduction

With ever rising computational resources, good quality data and developments in chemical informatics there is also a growing need for reproducibility, as already addressed by numerous researchers [1]–[3]. While there are currently available software (e.g. www.github.com) and data repositories (e.g. https://zenodo.org/) supporting reproducible research, there is still a lot of clutter since many researchers publish the same data set with their individual processing methods but also already pre-processed data. Therefore, one might find it difficult to reproduce the data transformation pipeline. In settings which are beyond conducting research, such as industrial application, one also wants to ensure a model generalizes well on unknown data, i.e., in the extrapolation regime or being out of applicability domain [4] sometimes referred to as out-of-distribution generalisation.
Data can stem from different laboratories or be measured by different instruments. Hence one needs to ensure models and data transformation pipelines are reproducible and applicable regardless of the data source while being convenient to use. Furthermore, storing the same data in multiple locations can lead to unnecessary energy consumption. Worth mentioning are also websites like https://paperswithcode.com/ which support reproducibility by providing the code along with the published papers. Reproducibility by means of data processing or manipulation was previously discussed in the literature. A metanalysis by Kapoor and Narayanan [5] from 2022 shows that still many authors do not follow the conventions for the correct use of data pre-processing. The reproducibility crisis is well described in [6]. Among the topics of this study are the *Pre-processing of training and test set* which is described as "using the entire dataset for any pre-processing steps such as imputation or over/under sampling" and *Feature selection on training and test set*, being "Feature selection on the entire dataset results in using information about which feature performs well on the test set", which are the issues in the focus of this work. The driver for such incomplete research is easy-to-use machine learning (ML) algorithms without expertise in ML/chemometrics. Another work by Krstajić et al [7] presented the importance of processing data prior to validating and prior to cross-validation during model training in cheminformatics. Beyond the importance in supervised ML models, correct processing of data is also crucial in transfer learning [8]. There are several prior works presenting automated data pre-processing pipelines. Torniainen et al. created an open-source python module for pre-processing of near infrared spectroscopic data [9] which constitutes of multiple steps such as normalization, smoothing and filtering. Another Python based initiative was published by Bilal et al [10]. Their tool was developed for use in ML and includes the following automated components: data type detection, missing values imputation, qualitative data encoding features scaling, feature selection and extraction. However, no code was published alongside the paper. Incorrect data pre-processing, as discussed, can cause information leakage leading to inflated results and overly optimistic model generalization [11].
The motivation for *PyChemFlow* is to create an open-source flexible automated data pre-processing pipeline to ensure reproducible machine learning model creation. Besides data manipulation, pipeline storing, or persistence is in the spotlight as well. Therefore, the code in the given repository given creates persistent pre-processing transformers, which can be stored and re-used. The following sections describe the pipeline, program code and how to use the library.

2. Computational Methods

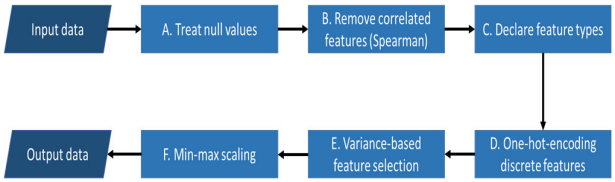
Transformer object
PyChemFlow is written in the programming language Python, while also utilizing the libraries *joblib* (https://joblib.readthedocs.io/en/latest/), *pandas* [12] (https://pandas.pydata.org/) and *scikit-learn* [13] (https://scikit-learn.org). PyChemFlow is forked from the repository published previously with [14]. The pre-processing steps are depicted in Scheme 1.



Scheme 1. A schematics representation of the transformer object

The key steps in *PyChemFlow* development were: 1) creating a *Pipeline* object from scikit-learn which can ingest multiple processing steps which gives a layer of flexibility to it; 2) a custom transformer object based on the scikit-learn *TransformerMixin* class (https://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html) which creates empty dictionary objects and inflates them with meta-information of the data during pipeline fitting based on 3) a custom pre-processing class.
The *Pipeline* and *TransformerMixin* objects are key to creating persistent transformers for later use of PyChemFlow. Once PyChemFlow is fit on training data, the *joblib* library is used to save PyChemFlow object to persistent storage as a .joblib file which can easily be loaded and applied (transform) to another data set given the same variable names.

Pre-processing procedure
Data manipulation functions are packing the repositories' *CustomPreprocessor* class. The class consists of a multitude of steps as depicted in Scheme 2 (in the first version of the procedura, the steps are not optional).



Scheme 2. A schematics representation of the pre-processing class

A total of 7 pre-processing steps are applied to the input data and we shortly describe each of these steps. **A.** Handling null values involves identifying and replacing missing or undefined data points in the dataset. This can be done by methods such as data imputation or deletion. **B.** Removing correlated features using the Spearman method involves calculating the correlation coefficient between features in a pairwise fashion and identifying features with a high correlation that can be removed to improve model performance. **C.** The declaration of feature types includes the identification of continuous and discrete features, as this information is important for proper data preprocessing and modeling. **D.** One-hot coding of discrete features involves creating a new binary representation for each discrete feature category. This allows models to better handle categorical data. **E.** Variance-based feature selection allows the identification and removal of features with low variance because they are unlikely to contain useful information for the model. **F.** Min-max scaling is a technique in which the values of a feature are scaled to a specific range, usually between 0 and 1. This ensures that all features are on the same scale which could prevent bias due to varying feature ranges.

Missing Data Imputation
In practice, missing data is a frequent occurrence because of manual data entry systems, incorrect measurements, equipment malfunctions, intentional omissions etc. A few missing values in some features (if the number of instances were reduced) can reduce the sample size, hence here an imputation step is done.

Qualitative Data Encoding
Many ML algorithms expect all input and output attributes to be numeric. This means if a dataset contains categorical data, first encode it in a numeric format before using an ML algorithm. Encoding is a mandatory pre-processing stage when working with qualitative data for ML models and there is a spectrum of methods for categorical data encoding.

Feature Scaling
It is common for real-world datasets to contain features that vary in units, size, and scale. As a result, feature scaling is required for ML models to comprehend these variables on the same scale. Some machine learning algorithms are sensitive to feature scaling while others are completely insensitive to it. Scaling data is required for machine learning methods such as logistic regression, linear regression and neural networks that use gradient descent as an optimization technique.

Utilization of the library
The GitHub repository has a readme.md file which presents/describes the main steps. The data set should be split into a training set and a validation/test set prior to preprocessing. The *PyChemFlow* pipeline must be imported from the *core* directory.

```
import joblib
import pandas as pd
from core.transformer import preproc_pipe
```

The *preproc_pipe* must then be applied via the *fit_transform* function to the *train* set loaded as a *pandas DataFrame*, which can be stored in persistent storage as a *.joblib file*.

```
processed_train = preproc_pipe.fit(train)
joblib.dump(preproc_pipe, "file.joblib")
```

The pipeline can then be reloaded and applied to the test or other data sets by using the *transform* function.

```
preproc_pipe_load = joblib.load("file.joblib")
processed_test = preproc_pipe_load.transform(test)
```

In this example, the train and *test* are *pandas* data frames which were pre-split.

3. Limitations and future work

As with many pre-processing and data manipulation tools and libraries there is no one-fits-all and a user might run into need for further customization. The *PyChemFlow* codes in open source are well documented and can easily be further modified by those with a solid foundation in Python programming. However, the authors tried to cover the basic steps of data processing so that the library can be used as is. Future work of this library is an increased customization, addition of optional steps and flexibility by providing additional arguments into the functions and classes. The

authors see this library as a dynamic one which will be used and developed in the future. Even though the authors suggest using such libraries for increasing transparency and reproducibility, the usability of such will also on data distributions.

Supporting information

The open-source python code is available at the following repository https://github.com/mariolovric/pychemflow .
The structure of the repository is described in the readme file

Funding and acknowledgements

M.L. is funded by the EU-Commission Grant Nr-101057497-EDIAQI. The Know-Center is funded within the Austrian COMET Program – Competence Centers for Excellent Technologies – under the auspices of the Austrian Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry of Economy, Family and Youth and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

References

[1] P. Gramatica, "Principles of QSAR models validation: Internal and external," *QSAR and Combinatorial Science*, vol. 26, no. 5, pp. 694–701, May 2007, doi: 10.1002/qsar.200610151.

[2] W. P. Walters, "Modeling, informatics, and the quest for reproducibility," *Journal of Chemical Information and Modeling*, vol. 53, no. 7, pp. 1529–1530, Jul. 2013, doi: 10.1021/CI400197W/ASSET/IMAGES/LARGE/CI-2013-00197W_0002.JPEG.

[3] R. D. Clark, "A path to next-generation reproducibility in cheminformatics," *Journal of Cheminformatics*, vol. 11, no. 1, pp. 1–3, Oct. 2019, doi: 10.1186/S13321-019-0385-0/METRICS.

[4] A. Tropsha, P. Gramatica, and V. K. Gombar, *The Importance of Being Earnest: Validation is the Absolute Essential for Successful Application and Interpretation of QSPR Models*, vol. 22, no. 1. Wiley-VCH Verlag, 2003, pp. 69–77. doi: 10.1002/qsar.200390007.

[5] S. Kapoor and A. Narayanan, "Leakage and the Reproducibility Crisis in ML-based Science," 2020.

[6] M. F. Dacrema, P. Cremonesi, and D. Jannach, "Are we really making much progress? A worrying analysis of recent neural recommendation approaches," *RecSys 2019 - 13th ACM Conference on Recommender Systems*, pp. 101–109, Sep. 2019, doi: 10.1145/3298689.3347058.

[7] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas, "Cross-validation pitfalls when selecting and assessing regression and classification models," *Journal of Cheminformatics*, vol. 6, no. 1, pp. 1–15, Mar. 2014, doi: 10.1186/1758-2946-6-10/FIGURES/16.

[8] M. Lovrić *et al.*, "Should We Embed in Chemistry? A Comparison of Unsupervised Transfer Learning with PCA, UMAP, and VAE on Molecular Fingerprints," *Pharmaceuticals*, vol. 14, no. 8, 2021, doi: 10.3390/ph14080758.

[9] J. Torniainen, I. O. Afara, M. Prakash, J. K. Sarin, L. Stenroth, and J. Töyräs, "Open-source python module for automated preprocessing of near infrared spectroscopic data," *Analytica Chimica Acta*, vol. 1108, pp. 1–9, Apr. 2020, doi: 10.1016/J.ACA.2020.02.030.

[10] M. Bilal, G. Ali, M. W. Iqbal, M. Anwar, M. S. A. Malik, and R. A. Kadir, "Auto-Prep: Efficient and Automated Data Preprocessing Pipeline," *IEEE Access*, vol. 10, no. October, pp. 107764–107784, 2022, doi: 10.1109/ACCESS.2022.3198662.

[11] A. Elangovan, J. He, and K. Verspoor, "Memorization vs. Generalization: Quantifying data leakage in NLP performance evaluation," *EACL 2021 - 16th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, vol. 2, pp. 1325–1335, 2021, doi: 10.18653/v1/2021.eacl-main.113.

[12] W. Mckinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51–56. [Online]. Available: http://conference.scipy.org/proceedings/scipy2010/mckinney.html

[13] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, doi: 10.1007/s13398-014-0173-7.2.

[14] M. Lovrić *et al.*, "Machine learning in prediction of intrinsic aqueous solubility of drug-like compounds: Generalization, complexity, or predictive ability?," *Journal of Chemometrics*, vol. 35, no. 7–8, p. e3349, Jul. 2021, doi: 10.1002/cem.3349.