Journal of Machine Learning Research 22 (2021) 1-5 Submitted 11/19; Published 5/21

# OpenML-Python: an extensible Python API for OpenML

**Matthias Feurer** FEURERM@CS.UNI-FREIBURG.DE
*University of Freiburg, Freiburg, Germany*

**Jan N. van Rijn** J.N.VAN.RIJN@LIACS.LEIDENUNIV.NL
*Leiden University, Leiden, Netherlands*

**Arlind Kadra** KADRAA@CS.UNI-FREIBURG.DE
*University of Freiburg, Freiburg, Germany*

**Pieter Gijsbers** P.GIJSBERS@TUE.NL
*Eindhoven University of Technology, Eindhoven, Netherlands*

**Neeratyoy Mallik** MALLIK@CS.UNI-FREIBURG.DE
*University of Freiburg, Freiburg, Germany*

**Sahithya Ravi** S.RAVI@TUE.NL
*Eindhoven University of Technology, Eindhoven, Netherlands*

**Andreas Müller** ANDREAS.MUELLER.ML@GMAIL.COM
*Microsoft, Sunnyvale, USA*

**Joaquin Vanschoren** J.VANSCHOREN@TUE.NL
*Eindhoven University of Technology, Eindhoven, Netherlands*

**Frank Hutter** FH@CS.UNI-FREIBURG.DE
*University of Freiburg & Bosch Center for Artificial Intelligence, Freiburg, Germany*

**Editor:** Balazs Kegl

## Abstract

OpenML is an online platform for open science collaboration in machine learning, used to share datasets and results of machine learning experiments. In this paper, we introduce *OpenML-Python*, a client API for Python, which opens up the OpenML platform for a wide range of Python-based machine learning tools. It provides easy access to all datasets, tasks and experiments on OpenML from within Python. It also provides functionality to conduct machine learning experiments, upload the results to OpenML, and reproduce results which are stored on OpenML. Furthermore, it comes with a scikit-learn extension and an extension mechanism to easily integrate other machine learning libraries written in Python into the OpenML ecosystem. Source code and documentation are available at https://github.com/openml/openml-python/.

**Keywords:** Python, Collaborative Science, Meta-Learning, Reproducible Research

## 1. Introduction

OpenML is a collaborative online machine learning (ML) platform, meant for sharing and building on prior empirical machine learning research (Vanschoren et al., 2014).

It goes beyond open data repositories, such as UCI (Dua and Graff, 2019), PMLB (Olson et al., 2017), the 'datasets' submodules in scikit-learn and tensorflow (Pedregosa et al.,

2011; Abadi et al., 2016), and the closed-source data sharing platform at Kaggle.com, since OpenML also collects millions of shared experiments on these datasets, linked to the exact ML pipelines and hyperparameter settings, and includes comprehensive logging and uploading functionalities which can be accessed programmatically via a REST API. An introduction and detailed information can be found on https://docs.openml.org.

OpenML-Python is a seamless integration of OpenML into the popular Python ML ecosystem,[1] that takes away this complexity by providing easy programmatic access to all OpenML data and by automating the sharing of new experiments.[2] In this paper, we introduce OpenML-Python's core design, showcase its extensibility to new ML libraries, and give code examples for several common research tasks.

## 2. Use cases for the OpenML-Python API

OpenML-Python allows for easy dataset and experiment sharing and reuse by handling all communication with OpenML's REST API. In this section, we briefly describe how the package can be used in several common machine learning tasks and highlight recent uses.

**Working with datasets.** OpenML-Python can retrieve the thousands of datasets on OpenML (all of them, or specific subsets) in a unified format, retrieve meta-data describing them, and search through them with filters. Datasets are converted from OpenML's internal format into *numpy*, *scipy* or *pandas* data structures, which are standard for ML in Python. To facilitate contributions from the community, it allows people to upload new datasets in only two function calls, and to define new *tasks* on them (combinations of a dataset, train/test split and target attribute).

**Publishing and retrieving results.** Sharing empirical results allows anyone to search and download them in order to reproduce and reuse them in their own research. One goal of OpenML is to simplify the comparison of new algorithms and implementations to existing approaches by comparing to the results on OpenML. To this end we also provide an interface for integrating new machine learning libraries with OpenML and we have already integrated *scikit-learn*. OpenML-Python can then be used to set up and conduct machine learning experiments for a given *task* and *flow* (an ML pipeline), and publish reproducible results (including hyperparameter settings and random states).

**Use cases in published works.** OpenML-Python has already been used to scale up studies with hundreds of consistently formatted datasets (Feurer et al., 2015; Fusi et al., 2018), supply large amounts of meta-data for meta-learning (Perrone et al., 2018), answer questions about algorithms such as hyperparameter importance (van Rijn and Hutter, 2018) and facilitate large-scale comparisons of algorithms (Strang et al., 2018).

## 3. High-level Design of OpenML-Python

The OpenML platform is organized around several entity types which describe different aspects of a machine learning study. It hosts *datasets*, *tasks* that define how models should be evaluated on them, *flows* that record the structure and other details of ML pipelines, and

1. https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/
2. Other clients already exist for R (Casalicchio et al., 2017) and Java (van Rijn, 2016).

```
import openml; import numpy as np
import matplotlib.pyplot as plt
df = openml.evaluations.list_evaluations_setups(
    'predictive_accuracy', flows=[8353], tasks=[6],
    output_format='dataframe', parameters_in_separate_columns=True,
) # Choose an SVM flow (e.g. 8353), and the dataset 'letter' (task 6).
hp_names = ['sklearn.svm.classes.SVC(16)_C','sklearn.svm.classes.SVC(16)_gamma']
df[hp_names] = df[hp_names].astype(float).apply(np.log)
C, gamma, score = df[hp_names[0]], df[hp_names[1]], df['value']
cntr = plt.tricontourf(C, gamma, score, levels=12, cmap='RdBu_r')
plt.colorbar(cntr, label='accuracy')
plt.xlim((min(C), max(C))); plt.ylim((min(gamma), max(gamma)))
plt.xlabel('C (log10)', size=16); plt.ylabel('gamma (log10)', size=16)
plt.title('SVM performance landscape', size=20)
```

Figure 1: Code for retrieving the predictive accuracy of an SVM classifier on the 'letter' dataset and creating a contour plot with the results.

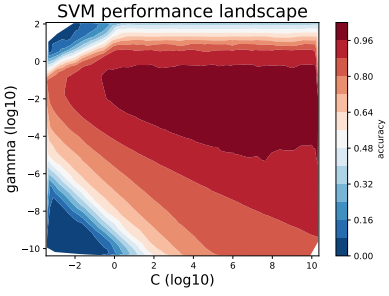
*runs* that record the experiments evaluating specific *flows* on certain *tasks*. For instance, an experiment (*run*) shared on OpenML can show how a random forest (*flow*) performs on 'Iris' (*dataset*) if evaluated with 10-fold cross-validation (*task*), and how to reproduce that result. In OpenML-Python, all these entities are represented by classes, each defined in their own submodule. This implements a natural mapping from OpenML concepts to Python objects. While OpenML is an online platform, we facilitate offline usage as well.

**Extensions.** To allow users to automatically run and share machine learning experiments with different libraries through the same OpenML-Python interface, we designed an extension interface that standardizes the interaction between machine learning library code and OpenML-Python. We also created an extension for *scikit-learn* (Pedregosa et al., 2011), as it is one of the most popular Python machine learning libraries. This extension can be used for any library which follows the *scikit-learn* API (Buitinck et al., 2013).

An extension's responsibility is to convert between the libraries' models and OpenML flows, interact with its training interface and format predictions. For example, the *scikit-learn* extension can convert an *OpenMLFlow* to an Estimator (including hyperparameter settings), train models and produce predictions for a task, and create an *OpenMLRun* object to upload the predictions to the OpenML server. The extension also handles advanced procedures, such as *scikit-learn*'s random search or grid search and uploading its traces (hyperparameters and scores of each model evaluated during search). We are working on more extensions, and anyone can contribute their own using the *scikit-learn* extension implementation as a reference.

## 4. Examples

We show two example uses of OpenML-Python to demonstrate its API's simplicity. First, we show how to retrieve results and evaluations from the OpenML server in Figure 1 (generating the plot on the right).




SVM hyperparameter contour plot generated by the code in Figure 1.

```
from openml import study, tasks, runs, extensions
from sklearn import compose, impute, pipeline, preprocessing, tree
cont, cat = extensions.sklearn.cont, extensions.sklearn.cat # feature types
clf = pipeline.make_pipeline( compose.make_column_transformer(
    (impute.SimpleImputer(), cont),
    (preprocessing.OneHotEncoder(handle_unknown='ignore'), cat)),
    tree.DecisionTreeClassifier())  # build a classification pipeline
benchmark_suite = study.get_suite('OpenML-CC18')  # task collection
for task_id in benchmark_suite.tasks:  # iterate over all tasks
  task = tasks.get_task(task_id)  # download the OpenML task
  run = runs.run_model_on_task(clf, task)  # run classifier on splits
  # run.publish()  # upload the run to the server; optional, requires API key
```

Figure 2: Training and evaluating a classification pipeline from scikit-learn on each task of the OpenML-CC18 benchmark suite (Bischl et al., 2019).

Second, in Figure 2 we show how to conduct experiments on a benchmark suite (Bischl et al., 2019). Further examples, including how to create datasets and tasks and how OpenML-Python was used in previous publications, can be found in the online documentation.[3]

## 5. Project development

The project has been set up for development through community effort from different research groups, and has received contributions from numerous individuals. The package is developed publicly through Github which also provides an issue tracker for bug reports, feature requests and usage questions. To ensure a coherent and robust code base we use continuous integration for Windows and Linux as well as automated type and style checking. Documentation is also rendered on continuous integration servers and consists of a mix of tutorials, examples and API documentation.

For ease of use and stability, we use well-known and established 3rd-party packages where needed. For instance, we build documentation using the popular *sphinx Python documentation generator*,[4] use an extension to automatically compile examples into documentation and Jupyter notebooks,[5] and employ standard open-source packages for scientific computing such as *numpy* (Harris et al., 2020), *scipy* (Virtanen et al., 2020), and *pandas* (McKinney, 2010). The package is written in Python3 and open-sourced with a 3-Clause BSD License.[3]

## 6. Conclusion

*OpenML-Python* allows easy interaction with OpenML from within Python. It makes it easy for people to share and reuse the data, meta-data, and empirical results which are generated as part of an ML study. This allows for better reproducibility, simpler benchmarking and easier collaboration on ML projects. Our software is shipped with a scikit-learn extension and has an extension mechanism to easily integrate other ML libraries written in Python.

3. We provide documentation, a list of extensions and code examples on http://openml.github.io/openml-python and host the project on http://github.com/openml/openml-python.
4. http://www.sphinx-doc.org [5]https://sphinx-gallery.github.io/

## Acknowledgments

**MF**, **NM** and **FH** acknowledge funding by the Robert Bosch GmbH. **AK**, **JvR** and **FH** acknowledge funding by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721. **JV** and **PG** acknowledge funding by the Data Driven Discovery of Models (D3M) program run by DARPA and the Air Force Research Laboratory. **The authors** also thank Bilge Celik, Victor Gal and everyone listed at https://github.com/openml/openml-python/graphs/contributors for their contributions.

## References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. In *Proc. of OSDI'16*, 2016.

B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren. OpenML Benchmarking Suites. *arXiv:1708.03731v2 [cs.LG]*, 2019.

L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Müller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD LML Workshop*, 2013.

G. Casalicchio, J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 32(3), 2017.

D. Dua and C. Graff. UCI machine learning repository, 2019. URL http://archive.ics.uci.edu/ml.

M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Efficient and Robust Automated Machine Learning. In *Proc. of NeurIPS'15*, 2015.

N. Fusi, R. Sheth, and M. Elibol. Probabilistic Matrix Factorization for Automated Machine Learning. In *Proc. of NeurIPS'18*. 2018.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with NumPy. *Nature*, 585, 2020.

W. McKinney. Data Structures for Statistical Computing in Python. In *Proc. of SciPy*, 2010.

R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(36), 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, et al. Scikit-learn: Machine Learning in Python. *JMLR*, 12, 2011.

V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. Scalable Hyperparameter Transfer Learning. In *Proc. of NeurIPS'18*. 2018.

B. Strang, P. van der Putten, J. N. van Rijn, and F. Hutter. Don't Rule Out Simple Models Prematurely: A Large Scale Benchmark Comparing Linear and Non-linear Classifiers in OpenML. In *Proc. of IDA XVII*, 2018.

J. N. van Rijn. *Massively Collaborative Machine Learning*. PhD thesis, Leiden University, 2016.

J. N. van Rijn and F. Hutter. Hyperparameter Importance Across Datasets. In *Proc. of KDD'18*, 2018.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD*, 15(2):49–60, 2014.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 2020.