Machine Learning Automation Toolbox (MLAUT)

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

In this paper we present MLAUT (Machine Learning AUtomation Toolbox) for the python data science ecosystem. MLAUT automates large-scale evaluation and benchmarking of machine learning algorithms on a large number of datasets. MLAUT provides a high-level workflow interface to machine algorithm algorithms, implements a local back-end to a database of dataset collections, trained algorithms, and experimental results, and provides easy-to-use interfaces to the scikit-learn and keras modelling libraries. Experiments are easy to set up with default settings in a few lines of code, while remaining fully customizable to the level of hyper-parameter tuning, pipeline composition, or deep learning architecture.

This is a short ("extended abstract") version, abridged for the NIPS submission prior to the public release of MLAUT, of a longer manuscript which also includes: full mathematical background and description of implemented post-hoc analyses, a detailed overview of the package design, and results of a large-scale benchmarking study conducted with MLAUT.

1 Introduction to MLAUT

MLAUT is a modelling and workflow toolbox in python, written with the aim of simplifying large scale benchmarking of machine learning strategies, e.g., validation, evaluation and comparison with respect to predictive/task-specific performance or runtime. Key features are:

(i) automation of the most common workflows for benchmarking modelling strategies on multiple datasets including statistical post-hoc analyses, with user-friendly default settings
(ii) unified interface with support for scikit-learn strategies, keras deep neural network architectures, including easy user extensibility to (partially or completely) custom strategies
(iii) higher-level meta-data interface for strategies, allowing easy specification of scikit-learn pipelines and keras deep network architectures, with user-friendly (sensible) default configurations
(iv) easy setting up and loading of data set collections for local use (e.g., data frames from local memory, UCI repository, openML, Delgado study, PMLB)
(v) back-end agnostic, automated local file system management of datasets, fitted models, predictions, and results, with the ability to easily resume crashed benchmark experiments with long running times

1.1 State-of-art: modelling toolbox and workflow design

A hierarchy of modelling designs may tentatively be identified in contemporary machine learning and modelling ecosystems, such as the python data science environment and the R language:

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

Level 1. implementation of specific methodology or a family of machine learning strategies, e.g., the most popular packages for deep learning, Tensorflow [1], MXNet [2], Caffe [3] and CNTK [4].

Level 2. provision of a unified interface for methodology solving the same "task", e.g., supervised learning aka predictive modelling. This is one core feature of the Weka [5], scikit-learn [6] and Shogun [7] projects which both also implement level 1 functionality, and main feature of the caret [8] and mlr [9] packages in R which provides level 2 functionality by external interfacing of level 1 packages.

Level 3. composition and meta-learning interfaces such as tuning and pipeline building, more generally, first-order operations on modelling strategies. Packages implementing level 2 functionality usually (but not always) also implement this, such as the general hyper-parameter tuning and pipeline composition operations found in scikit-learn and mlr or its mlrCPO extension. Keras [10] has abstract level 3 functionality specific to deep learning, Shogun possesses such functionality specific to kernel methods.

Level 4. workflow automation of higher-order tasks performed with level 3 interfaces, e.g., diagnostics, evaluation and comparison of pipeline strategies. Mlr is, to our knowledge, the only existing modelling toolbox with a modular, class-based level 4 design that supports and automates re-sampling based model evaluation workflows. The Weka GUI and module design also provides some level 4 functionality.

A different type of level 4 functionality is automated model building, closely linked to but not identical with benchmarking and automated evaluation - similarly to how, mathematically, model selection is not identical with model evaluation. Level 4 interfaces for automated model building also tie into level 3 interfaces, examples of automated model building are implemented in auto-Weka [11], auto-sklearn [12], or extensions to mlrCPO [13].

In the Python data science environment, to our knowledge, there is currently no widely adopted solution with level 4 functionality for evaluation, comparison, and benchmarking workflows. The reasonably well-known skll [14] package provides automation functionality in python for scikit-learn based experiments but follows an unencapsulated scripting design which limits extensibility and usability, especially since it is difficult to use with level 3 functionality from scikit-learn or state-of-art deep learning packages.

Prior studies conducting experiments which are level 4 use cases, i.e., large-scale benchmarking experiments of modelling strategies, exist for supervised classification, such as [15, 16]. Smaller studies, focusing on a couple of estimators trained on a small number of datasets have also been published [17]. However, to the best of our knowledge: none of the authors released a toolbox for carrying out the experiments; code used in these studies cannot be directly applied to conduct other machine learning experiments; and, deep neural networks were not included as part of the benchmark exercises.

At the current state-of-art, hence, there is a distinct need for level 4 functionality in the scikit-learn and keras ecosystems. Instead of re-creating the mlr interface or following a GUI-based philosophy such as Weka, we have decided to create a modular workflow environment which builds on the particular strengths of python as an object oriented programming language, the notebook-style user interaction philosophy of the python data science ecosystem, and the contemporary mathematical-statistical state-of-art with best practice recommendations for conducting formal benchmarking experiments - while attempting to learn from what we believe works well (or not so well) in mlr and Weka.

1.2 Scientific contributions

MLAUT is more than a mere implementation of readily existing scientific ideas or methods. We argue that the following contributions, outlined in the manuscript, are scientific contributions closely linked to its creation:

(1) design of a modular "level 4" software interface which supports the predictive model validation/comparison workflow, a data/model file input/output back-end, and an abstraction of post-hoc evaluation analyses, at the same time.

(2) a comprehensive overview of the state-of-art in statistical strategy evaluation, comparison and comparative hypothesis testing on a collection of data sets. We further close gaps in

said literature by formalizing and explicitly stating the kinds of guarantees the different analyses provide, and detailing computations of related confidence intervals.

(3) as a principal test case for MLAUT, we conducted a large-scale supervised classification study in order to benchmark the performance of a number of machine learning algorithms, with a key sub-question being whether more complex and/or costly algorithms tend to perform better on real-world datasets. On the representative collection of UCI benchmark datasets, kernel methods and random forests perform best.

(4) as a specific but quite important sub-question we investigated whether common off-shelf deep learning strategies would be worth considering as a default choice on the "average" (non-image, non-text) supervised learning dataset. The answer, somewhat surprising in its clarity, appears to be that they are not - in the sense that alternatives (and sometimes even naive baselines) usually perform better. However, on the smaller tabular datasets, the computational cost of off-shelf deep learning architectures is also not as high as one might naively assume.

Literature relevant to these contribution will be discussed in the respective sections.

1.3 Ease of Use

We present a short demo of core MLAUT functionality and user interaction, designed to be convenient in combination with jupyter notebook or scripting command line working style.

The first step is setting up a database for the dataset collection, which has to happen only once per computer and dataset collection, and which we assume has been already stored in a local MLAUT HDF5 database. The first step in the core benchmarking workflow is to define hooks to the database input and output files:

```
input_io = data.open_hdf5(...) #path to input HDF5 file
out_io = data.open_hdf5(...) #path to output HDF5 file
```

After the hooks are created we can proceed to preparing fixed re-sampling splits (training/test) on which all strategies are evaluated. By default MLAUT creates a single evaluation split with a uniformly sampled $\frac{2}{3}$ of the data for training and $\frac{1}{3}$ for testing.

```
data.split_datasets(hdf5_in=..., hdf5_out=..., dataset_paths=...)
```

For a simple set-up, a standard set of estimators that come with sensible parameter defaults can be initialized. Advanced commands allow to specify hyper-parameters, tuning strategies, keras deep learning architectures, scikit-learn pipelines, or even fully custom estimators.

```
est = ['RandomForestClassifier','BaggingClassifier']
estimators = instantiate_default_estimators(estimators=est)
>>> estimators
    <mlaut.estimators.ensemble_estimators.Random_Forest_Classifier>
    <mlaut.estimators.ensemble_estimators.Bagging_Classifier>
```

The user can now proceed to running the experiments. Training, prediction and evaluation are separate; partial results, including fitted models and predictions, are stored and retrieved through database hooks. This allows intermediate analyses, and for the experiment to easily resume in case of a crash or interruption. If this happens, the user would simply need to re-run the code above and the experiment will continue from the last checkpoint, without re-executing prior costly computation.

```
orchest.run(modelling_strategies=estimators)
>>> RandomForestClassifier trained on dataset 1
    RandomForestClassifier trained on dataset 2
    ...
orchest.predict_all(trained_models_dir='data/trained_models', estimators=
    estimators, verbose=False)
>>> Predictions of RandomForestClassifier on dataset 1 saved in database
    Predictions of RandomForestClassifier on dataset 2 saved in database
    ...
```

The last step in the pipeline is executing post-hoc analyses for the benchmarking experiments. The `AnalyseResults` class allows to specify performance quantifiers to be computed and comparison tests to be carried out, based on the intermediate computation data, e.g., predictions from all the strategies.

```
analyze.prediction_errors(score_accuracy, estimators)
```

The `prediction_errors()` method returns two sets of results: `errors_per_estimator` dictionary which is used subsequently in further statistical tests and `errors_per_dataset_per_estimator_df` which is a dataframe with the loss of each estimator on each dataset that can be examined directly by the user.

We can also use the produced errors in order to perform the statistical tests for method comparison. The code below shows an example of running a t-test.

```
_, t_test_df = analyze.t_test(errors_per_estimator)
>>> t_test_df
                 Estimator 1       Estimator 2
                 t_stat  p_val     t_stat  p_val
     Estimator 1 ...     ...       ...     ...
     Estimator 2 ...     ...       ...     ...
     ...
```

Data frames or graphs resulting from the analyses can then be exported, e.g., for presentation in a scientific report.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[2] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv:1512.01274 [cs]*, December 2015. arXiv: 1512.01274.

[3] Yangqing Jia. Caffe | Deep Learning Framework. `http://caffe.berkeleyvision.org`.

[4] Frank Seide and Amit Agarwal. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2135–2135, New York, NY, USA, 2016. ACM.

[5] Sudhir B. Jagtap and Kodge B. G. Census Data Mining and Data Analysis using WEKA. *arXiv:1310.4647 [cs]*, October 2013. arXiv: 1310.4647.

[6] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[7] SĆ Sonnenburg, Sebastian Henschel, Christian Widmer, Jonas Behr, Alexander Zien, Fabio de Bona, Alexander Binder, Christian Gehl, VojtÄ Franc, et al. The shogun machine learning toolbox. *Journal of Machine Learning Research*, 11(Jun):1799–1802, 2010.

[8] Max Kuhn Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R. Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and and Tyler Hunt. caret: Classification and Regression Training, May 2018.

[9] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine Learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

[10] François Chollet. Keras. `https://keras.io`, 2015.

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[12] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[13] Janek Thomas, Stefan Coors, and Bernd Bischl. Automatic gradient boosting. *arXiv preprint arXiv:1807.03873*, 2018.

[14] scikit-learn laboratory. `https://skll.readthedocs.io`.

[15] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[16] Jacques Wainer. Comparison of 14 different families of classification algorithms on 115 binary datasets. *arXiv:1606.00930 [cs]*, June 2016. arXiv: 1606.00930.

[17] J. Huang, J. Lu, and C.X. Ling. Comparing naive Bayes, decision trees, and SVM with AUC and accuracy. pages 553–556. IEEE Comput. Soc, 2003.