Journal of Machine Learning Research 12 (2011) 2825-2830 Submitted 3/11; Revised 8/11; Published 10/11

Scikit-learn: Machine Learning in Python

Fabian Pedregosa FABIAN.PEDREGOSA@INRIA.FR
Gaël Varoquaux GAEL.VAROQUAUX@NORMALESUP.ORG
Alexandre Gramfort ALEXANDRE.GRAMFORT@INRIA.FR
Vincent Michel VINCENT.MICHEL@LOGILAB.FR
Bertrand Thirion BERTRAND.THIRION@INRIA.FR
Parietal, INRIA Saclay
Neurospin, Bât 145, CEA Saclay
91191 Gif sur Yvette – France

Olivier Grisel OLIVIER.GRISEL@ENSTA.FR
Nuxeo
20 rue Soleillet
75 020 Paris – France

Mathieu Blondel MBLONDEL@AI.CS.KOBE-U.AC.JP
Kobe University
1-1 Rokkodai, Nada
Kobe 657-8501 – Japan

Peter Prettenhofer PETER.PRETTENHOFER@GMAIL.COM
Bauhaus-Universität Weimar
Bauhausstr. 11
99421 Weimar – Germany

Ron Weiss RONWEISS@GMAIL.COM
Google Inc
76 Ninth Avenue
New York, NY 10011 – USA

Vincent Dubourg VINCENT.DUBOURG@GMAIL.COM
Clermont Université, IFMA, EA 3867, LaMI
BP 10448, 63000 Clermont-Ferrand – France

Jake Vanderplas VANDERPLAS@ASTRO.WASHINGTON.EDU
Astronomy Department
University of Washington, Box 351580
Seattle, WA 98195 – USA

Alexandre Passos ALEXANDRE.TP@GMAIL.COM
IESL Lab
UMass Amherst
Amherst MA 01002 – USA

David Cournapeau COURNAPE@GMAIL.COM
Enthought
21 J.J. Thompson Avenue
Cambridge, CB3 0FA – UK

©2011 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

Matthieu Brucher MATTHIEU.BRUCHER@GMAIL.COM
Total SA, CSTJF
avenue Larribau
64000 Pau – France

Matthieu Perrot MATTHIEU.PERROT@CEA.FR
Édouard Duchesnay EDOUARD.DUCHESNAY@CEA.FR
LNAO
Neurospin, Bât 145, CEA Saclay
91191 Gif sur Yvette – France

Editor: Mikio Braun

Abstract

Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings. Source code, binaries, and documentation can be downloaded from `http://scikit-learn.sourceforge.net`.

Keywords: Python, supervised learning, unsupervised learning, model selection

1. Introduction

The Python programming language is establishing itself as one of the most popular languages for scientific computing. Thanks to its high-level interactive nature and its maturing ecosystem of scientific libraries, it is an appealing choice for algorithmic development and exploratory data analysis (Dubois, 2007; Milmann and Avaizis, 2011). Yet, as a general-purpose language, it is increasingly used not only in academic settings but also in industry.

Scikit-learn harnesses this rich environment to provide state-of-the-art implementations of many well known machine learning algorithms, while maintaining an easy-to-use interface tightly integrated with the Python language. This answers the growing need for statistical data analysis by non-specialists in the software and web industries, as well as in fields outside of computer-science, such as biology or physics. *Scikit-learn* differs from other machine learning toolboxes in Python for various reasons: *i)* it is distributed under the BSD license *ii)* it incorporates compiled code for efficiency, unlike MDP (Zito et al., 2008) and pybrain (Schaul et al., 2010), *iii)* it depends only on numpy and scipy to facilitate easy distribution, unlike pymvpa (Hanke et al., 2009) that has optional dependencies such as R and shogun, and *iv)* it focuses on imperative programming, unlike pybrain which uses a data-flow framework. While the package is mostly written in Python, it incorporates the C++ libraries LibSVM (Chang and Lin, 2001) and LibLinear (Fan et al., 2008) that provide reference implementations of SVMs and generalized linear models with compatible licenses. Binary packages are available on a rich set of platforms including Windows and any POSIX platforms.

Furthermore, thanks to its liberal license, it has been widely distributed as part of major free software distributions such as Ubuntu, Debian, Mandriva, NetBSD and Macports and in commercial distributions such as the "Enthought Python Distribution".

2. Project Vision

Code quality. Rather than providing as many features as possible, the project's goal has been to provide solid implementations. Code quality is ensured with unit tests—as of release 0.8, test coverage is 81%—and the use of static analysis tools such as `pyflakes` and `pep8`. Finally, we strive to use consistent naming for the functions and parameters used throughout a strict adherence to the Python coding guidelines and numpy style documentation.

BSD licensing. Most of the Python ecosystem is licensed with non-copyleft licenses. While such policy is beneficial for adoption of these tools by commercial projects, it does impose some restrictions: we are unable to use some existing scientific code, such as the GSL.

Bare-bone design and API. To lower the barrier of entry, we avoid framework code and keep the number of different objects to a minimum, relying on numpy arrays for data containers.

Community-driven development. We base our development on collaborative tools such as git, github and public mailing lists. External contributions are welcome and encouraged.

Documentation. *Scikit-learn* provides a $\sim$300 page user guide including narrative documentation, class references, a tutorial, installation instructions, as well as more than 60 examples, some featuring real-world applications. We try to minimize the use of machine-learning jargon, while maintaining precision with regards to the algorithms employed.

3. Underlying Technologies

Numpy: the base data structure used for data and model parameters. Input data is presented as numpy arrays, thus integrating seamlessly with other scientific Python libraries. Numpy's view-based memory model limits copies, even when binding with compiled code (Van der Walt et al., 2011). It also provides basic arithmetic operations.

Scipy: efficient algorithms for linear algebra, sparse matrix representation, special functions and basic statistical functions. *Scipy* has bindings for many Fortran-based standard numerical packages, such as LAPACK. This is important for ease of installation and portability, as providing libraries around Fortran code can prove challenging on various platforms.

Cython: a language for combining C in Python. Cython makes it easy to reach the performance of compiled languages with Python-like syntax and high-level operations. It is also used to bind compiled libraries, eliminating the boilerplate code of Python/C extensions.

4. Code Design

Objects specified by interface, not by inheritance. To facilitate the use of external objects with *scikit-learn*, inheritance is not enforced; instead, code conventions provide a consistent interface. The central object is an `estimator`, that implements a `fit` method, accepting as arguments an input data array and, optionally, an array of labels for supervised problems. Supervised estimators, such as SVM classifiers, can implement a `predict` method. Some estimators, that we call `transformers`, for example, PCA, implement a `transform` method, returning modified input data. Estimators

	scikit-learn	mlpy	pybrain	pymvpa	mdp	shogun
Support Vector Classification	**5.2**	9.47	17.5	11.52	40.48	5.63
Lasso (LARS)	**1.17**	105.3	-	37.35	-	-
Elastic Net	**0.52**	73.7	-	1.44	-	-
k-Nearest Neighbors	0.57	1.41	-	**0.56**	0.58	1.36
PCA (9 components)	**0.18**	-	-	8.93	0.47	0.33
k-Means (9 clusters)	1.34	0.79	$\star$	-	35.75	**0.68**
License	BSD	GPL	BSD	BSD	BSD	GPL

-: Not implemented. $\star$: Does not converge within 1 hour.

Table 1: Time in seconds on the Madelon data set for various machine learning libraries exposed in Python: MLPy (Albanese et al., 2008), PyBrain (Schaul et al., 2010), pymvpa (Hanke et al., 2009), MDP (Zito et al., 2008) and Shogun (Sonnenburg et al., 2010). For more benchmarks see `http://github.com/scikit-learn`.

may also provide a `score` method, which is an increasing evaluation of goodness of fit: a log-likelihood, or a negated loss function. The other important object is the *cross-validation iterator*, which provides pairs of train and test indices to split input data, for example K-fold, leave one out, or stratified cross-validation.

Model selection. Scikit-learn can evaluate an estimator's performance or select parameters using cross-validation, optionally distributing the computation to several cores. This is accomplished by wrapping an estimator in a `GridSearchCV` object, where the "CV" stands for "cross-validated". During the call to `fit`, it selects the parameters on a specified parameter grid, maximizing a score (the `score` method of the underlying estimator). `predict`, `score`, or `transform` are then delegated to the tuned estimator. This object can therefore be used transparently as any other estimator. Cross validation can be made more efficient for certain estimators by exploiting specific properties, such as warm restarts or regularization paths (Friedman et al., 2010). This is supported through special objects, such as the `LassoCV`. Finally, a `Pipeline` object can combine several `transformers` and an estimator to create a combined estimator to, for example, apply dimension reduction before fitting. It behaves as a standard estimator, and `GridSearchCV` therefore tune the parameters of all steps.

5. High-level yet Efficient: Some Trade Offs

While *scikit-learn* focuses on ease of use, and is mostly written in a high level language, care has been taken to maximize computational efficiency. In Table 1, we compare computation time for a few algorithms implemented in the major machine learning toolkits accessible in Python. We use the Madelon data set (Guyon et al., 2004), 4400 instances and 500 attributes, The data set is quite large, but small enough for most algorithms to run.

SVM. While all of the packages compared call libsvm in the background, the performance of *scikit-learn* can be explained by two factors. First, our bindings avoid memory copies and have up to 40% less overhead than the original libsvm Python bindings. Second, we patch libsvm to improve efficiency on dense data, use a smaller memory footprint, and better use memory alignment and pipelining capabilities of modern processors. This patched version also provides unique features, such as setting weights for individual samples.

LARS. Iteratively refining the residuals instead of recomputing them gives performance gains of 2–10 times over the reference R implementation (Hastie and Efron, 2004). *Pymvpa* uses this implementation via the Rpy R bindings and pays a heavy price to memory copies.
Elastic Net. We benchmarked the *scikit-learn* coordinate descent implementations of Elastic Net. It achieves the same order of performance as the highly optimized Fortran version *glmnet* (Friedman et al., 2010) on medium-scale problems, but performance on very large problems is limited since we do not use the KKT conditions to define an active set.
kNN. The k-nearest neighbors classifier implementation constructs a ball tree (Omohundro, 1989) of the samples, but uses a more efficient brute force search in large dimensions.
PCA. For medium to large data sets, *scikit-learn* provides an implementation of a truncated PCA based on random projections (Rokhlin et al., 2009).
k-means. *scikit-learn*'s k-means algorithm is implemented in pure Python. Its performance is limited by the fact that numpy's array operations take multiple passes over data.

6. Conclusion

Scikit-learn exposes a wide variety of machine learning algorithms, both supervised and unsupervised, using a consistent, task-oriented interface, thus enabling easy comparison of methods for a given application. Since it relies on the scientific Python ecosystem, it can easily be integrated into applications outside the traditional range of statistical data analysis. Importantly, the algorithms, implemented in a high-level language, can be used as building blocks for approaches specific to a use case, for example, in medical imaging (Michel et al., 2011). Future work includes *online* learning, to scale to large data sets.

References

D. Albanese, G. Merler, S.and Jurman, and R. Visintainer. MLPy: high-performance python package for predictive modeling. In *NIPS, MLOSS Workshop*, 2008.

C.C. Chang and C.J. Lin. LIBSVM: a library for support vector machines. `http://www.csie.ntu.edu.tw/cjlin/libsvm`, 2001.

P.F. Dubois, editor. *Python: Batteries Included*, volume 9 of *Computing in Science & Engineering*. IEEE/AIP, May 2007.

R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. LIBLINEAR: a library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010.

I Guyon, S. R. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge, 2004.

M. Hanke, Y.O. Halchenko, P.B. Sederberg, S.J. Hanson, J.V. Haxby, and S. Pollmann. PyMVPA: A Python toolbox for multivariate pattern analysis of fMRI data. *Neuroinformatics*, 7(1):37–53, 2009.

T. Hastie and B. Efron. Least Angle Regression, Lasso and Forward Stagewise. `http://cran.r-project.org/web/packages/lars/lars.pdf`, 2004.

V. Michel, A. Gramfort, G. Varoquaux, E. Eger, C. Keribin, and B. Thirion. A supervised clustering approach for fMRI-based inference of brain states. *Patt Rec*, page epub ahead of print, April 2011. doi: 10.1016/j.patcog.2011.04.006.

K.J. Milmann and M. Avaizis, editors. *Scientific Python*, volume 11 of *Computing in Science & Engineering*. IEEE/AIP, March 2011.

S.M. Omohundro. Five balltree construction algorithms. ICSI Technical Report TR-89-063, 1989.

V. Rokhlin, A. Szlam, and M. Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2009.

T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *The Journal of Machine Learning Research*, 11:743–746, 2010.

S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN machine learning toolbox. *Journal of Machine Learning Research*, 11:1799–1802, 2010.

S. Van der Walt, S.C Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 11, 2011.

T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. Modular toolkit for data processing (MDP): A Python data processing framework. *Frontiers in Neuroinformatics*, 2, 2008.