Journal of Machine Learning Research 14 (2013) 2349-2353 Submitted 3/13; Published 8/13

Orange: Data Mining Toolbox in Python

Janez Demšar JANEZ.DEMSAR@FRI.UNI-LJ.SI
Tomaž Curk TOMAZ.CURK@FRI.UNI-LJ.SI
Aleš Erjavec ALES.ERJAVE@FRI.UNI-LJ.SI
Črt Gorup CRT.GORUP@FRI.UNI-LJ.SI
Tomaž Hočevar TOMAZ.HOCEVAR@FRI.UNI-LJ.SI
Mitar Milutinovič MITAR.MILUTINOVIC@FRI.UNI-LJ.SI
Martin Možina MARTIN.MOZINA@FRI.UNI-LJ.SI
Matija Polajnar MATIJA.POLAJNAR@FRI.UNI-LJ.SI
Marko Toplak MARKO.TOPLAK@FRI.UNI-LJ.SI
Anže Starič ANZE.STARIC@FRI.UNI-LJ.SI
Miha Štajdohar MIHA.STAJDOHAR@FRI.UNI-LJ.SI
Lan Umek LAN.UMEK@FRI.UNI-LJ.SI
Lan Žagar LAN.ZAGAR@FRI.UNI-LJ.SI
Jure Žbontar JURE.ZBONTAR@FRI.UNI-LJ.SI
Marinka Žitnik MARINKA.ZITNIK@FRI.UNI-LJ.SI
Blaž Zupan BLAZ.ZUPAN@FRI.UNI-LJ.SI
Faculty of Computer and Information Science
University of Ljubljana
Tržaška 25, SI-1000 Ljubljana, Slovenia

Abstract

Orange is a machine learning and data mining suite for data analysis through Python scripting and visual programming. Here we report on the scripting part, which features interactive data analysis and component-based assembly of data mining procedures. In the selection and design of components, we focus on the flexibility of their reuse: our principal intention is to let the user write simple and clear scripts in Python, which build upon C++ implementations of computationally-intensive tasks. Orange is intended both for experienced users and programmers, as well as for students of data mining.

Keywords: Python, data mining, machine learning, toolbox, scripting

1. Introduction

Scripting languages have recently risen in popularity in all fields of computer science. Within the context of explorative data analysis, they offer advantages like interactivity and fast prototyping by gluing together existing components or adapting them for new tasks. Python is a scripting language with clear and simple syntax, which also made it popular in education. Its relatively slow execution can be circumvented by using libraries that implement the computationally intensive tasks in low-level languages.

Python offers a huge number of extension libraries. Many are related to machine learning, including several general packages like scikit-learn (Pedregosa et al., 2011), PyBrain (Schaul et al., 2010) and mlpy (Albanese et al., 2012). Orange was conceived in late 1990s and is among the oldest of such tools. It focuses on simplicity, interactivity through scripting, and component-based design.

2. Toolbox Overview

Orange library is a hierarchically-organized toolbox of data mining components. The low-level procedures at the bottom of the hierarchy, like data filtering, probability assessment and feature scoring, are assembled into higher-level algorithms, such as classification tree learning. This allows developers to easily add new functionality at any level and fuse it with the existing code. The main branches of the component hierarchy are:

data management and preprocessing for data input and output, data filtering and sampling, imputation, feature manipulation (discretization, continuization, normalization, scaling and scoring), and feature selection,

classification with implementations of various supervised machine learning algorithms (trees, forests, instance-based and Bayesian approaches, rule induction), borrowing from some well-known external libraries such as LIBSVM (Chang and Lin, 2011),

regression including linear and lasso regression, partial least square regression, regression trees and forests, and multivariate regression splines,

association for association rules and frequent itemsets mining,

ensembles implemented as wrappers for bagging, boosting, forest trees, and stacking,

clustering, which includes k-means and hierarchical clustering approaches,

evaluation with cross-validation and other sampling-based procedures, functions for scoring the quality of prediction methods, and procedures for reliability estimation,

projections with implementations of principal component analysis, multi-dimensional scaling and self-organizing maps.

The library is designed to simplify the assembly of data analysis workflows and crafting of data mining approaches from a combination of existing components. Besides broader range of features, Orange differs from most other Python-based machine learning libraries by its maturity (over 15 years of active development and use), a large user community supported through an active forum, and extensive documentation that includes tutorials, scripting examples, data set repository, and documentation for developers. Orange scripting library is also a foundation for its visual programming platform with graphical user interface components for interactive data visualization.

The two major packages that are similar to Orange and are still actively developed are scikit-learn (Pedregosa et al., 2011) and mlpy (Albanese et al., 2012). Both are more tightly integrated with numpy and at present better blend into Python's numerical computing habitat. Orange was on the other hand inspired by classical machine learning that focuses on symbolic methods. Rather than supporting only numerical arrays, Orange data structures combine symbolic, string and numerical attributes and meta data information. User can for instance refer to variables and values by their names. Variables store mapping functions, a mechanism which for instance allows classifiers to define transformations on training data that are then automatically applied when making predictions. These features also make Orange more suitable for interactive, explorative data analysis.

3. Scripting Examples

Let us illustrate the utility of Orange through an example of data analysis in Python shell:

```
>>> import Orange
>>> data = Orange.data.Table("titanic")
>>> len(data)
2201
>>> nbc = Orange.classification.bayes.NaiveLearner()
>>> svm = Orange.classification.svm.SVMLearner()
>>> stack = Orange.ensemble.stacking.StackedClassificationLearner([nbc,svm])
>>> res = Orange.evaluation.testing.cross_validation([nbc, svm, stack], data)
>>> Orange.evaluation.scoring.AUC(res)
[0.7148500435874006, 0.731873352343742, 0.7635593576372478]
```

We first read the data on survival of 2,201 passengers from HMS Titanic and construct a set of learning algorithms: a naive Bayesian and SVM learner, and a stacked combination of the two (Wolpert, 1992). We then cross-validate the learners and report the area under ROC curves.

Running stacking on the subset of about 470 female passengers improves AUC score:

```
>>> females = Orange.data.Table([d for d in data if d["sex"]=="female"])
>>> len(females)
470
>>> res = Orange.evaluation.testing.cross_validation([stack], females)
>>> Orange.evaluation.scoring.AUC(res)
[0.8124014221073045]
```

We can use existing machine learning components to craft new ones. For instance, learning algorithms must implement a `__call__` operator that accepts the training data and, optionally, data instance weights, and has to return a model. The following example defines a new learner that encloses another learner into a feature selection wrapper: it sorts the features by their information gain (as implemented in `Orange.feature.scoring.InfoGain`), constructs a new data set with only the `m` best features and calls the `base_learner`.

```
class FSSLearner(Orange.classification.PyLearner):
    def __init__(self, base_learner, m=5):
        self.m = m
        self.base_learner = base_learner

    def __call__(self, data, weights=None):
        gain = Orange.feature.scoring.InfoGain()
        best = sorted(data.domain.features, key=lambda x: -gain(x, data))[:self.m]
        domain = Orange.data.Domain(best + [data.domain.class_var])
        new_data = Orange.data.Table(domain, data)
        model = self.base_learner(new_data, weights)
        return Orange.classification.PyClassifier(classifier=model)
```

Below we compare the original and wrapped naive Bayesian classifier on a data set with 106 instances and 57 features:

```
>>> data = Orange.data.Table("promoters")
>>> len(data), len(data.domain.features)
(106, 57)
```

```
>>> bayes = Orange.classification.bayes.NaiveLearner()
>>> res = Orange.evaluation.testing.cross_validation([bayes, FSSLearner(bayes)], data)
>>> Orange.evaluation.scoring.AUC(res)
[0.9329999999999998, 0.945]
```

4. Code Design

Orange's core is a collection of nearly 200 C++ classes that cover the basic data structures and majority of preprocessing and modeling algorithms. The C++ part is self-contained, without any calls to Python that would induce unnecessary overhead. The core includes several open source libraries, including LIBSVM (Chang and Lin, 2011), LIBLINEAR (Fan et al., 2008), Earth (see `http://www.milbo.users.sonic.net/earth`), QHull (Barber et al., 1996) and a subset of BLAS (Blackford et al., 2002). The Python layer also uses popular Python libraries numpy for linear algebra, networkx (Hagberg et al., 2008) for working with networks and matplotlib (Hunter, 2007) for basic visualization.

The upper layer of Orange is written in Python and includes procedures that are not time-critical. This is also the place at which users outside the core development group most easily contribute to the project.

Automated testing of the system relies on over 1,500 regression tests that are mostly based on code snippets from extensive documentation. A part of the code is also covered with stricter unit tests.

5. Availability, Requirements and Plans for the Future

Orange is free software released under GPL. The code is hosted on Bitbucket repository (`https://bitbucket.org/biolab/orange`). Orange runs on Windows, Mac OS X and Linux, and can also be installed from the Python Package Index repository (`pip install Orange`). Binary installer for Windows and application bundle for Mac OS X are available on project's web site (`http://orange.biolab.si`).

Orange currently runs on Python 2.6 and 2.7. A version for Python 3 and higher is under development. There, we will switch to numpy-based data structures and scrap the C++ core in favor of using routines from numpy and scipy (Jones et al., 2001–), scikit-learn (Pedregosa et al., 2011) and similar libraries that did not exist when Orange was first conceived. Despite planned changes in the core, we will maintain backward compatibility. For existing users, the changes of the Python interface will be minor.

Acknowledgments

We would like to acknowledge support for this project from the Slovenian Research Agency (P2-0209, J2-9699, L2-1112), National Institute of Health (P01-HD39691), and Astra Zeneca. We thank the anonymous reviewers for their constructive comments.

References

D. Albanese, R. Visintainer, S. Merler, S. Riccadonna, G. Jurman, and C. Furlanello. mlpy: Machine learning Python. *CoRR*, abs/1202.6548, 2012.

C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software*, 22(4), 1996.

L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, and G. Henry. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

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

A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, 2008.

J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3): 90–95, 2007.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg. scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.