Pycobra: A Python Toolbox for Ensemble Learning and Visualisation

Benjamin Guedj   BENJAMIN.GUEDJ@INRIA.FR
Modal project-team, Lille - Nord Europe research center
Inria, France

Bhargav Srinivasa Desikan   BHARGAV.SRINIVASA-DESIKAN@INRIA.FR
Modal project-team, Lille - Nord Europe research center
Inria, France

Editor: Geoff Holmes

Abstract

We introduce `pycobra`, a Python library devoted to ensemble learning (regression and classification) and visualisation. Its main assets are the implementation of several ensemble learning algorithms, a flexible and generic interface to compare and blend any existing machine learning algorithm available in Python libraries (as long as a `predict` method is given), and visualisation tools such as Voronoi tessellations. `pycobra` is fully `scikit-learn` compatible and is released under the MIT open-source license. `pycobra` can be downloaded from the Python Package Index (PyPi) and Machine Learning Open Source Software (MLOSS). The current version (along with Jupyter notebooks, extensive documentation, and continuous integration tests) is available at https://github.com/bhargavvader/pycobra and official documentation website is https://modal.lille.inria.fr/pycobra.

Keywords: ensemble methods, machine learning, Voronoi tesselation, Python, open source software

1. Introduction

Combined statistical procedures, also known as ensemble or aggregation methods, are very popular in Machine Learning competitions – the celebrated Netflix Challenge (as are repeatedly Kaggle competitions) was won by an ensemble technique (see for example Bell and Koren, 2007, for a discussion). In a nutshell, ensemble methods combine different predictors (each trained on the same dataset) to improve performance. `scikit-learn` offers implementations of some ensemble methods but it does not ship with a specific analysis toolbox for aggregation. `pycobra` attempts to fill this gap by providing to the `scikit-learn` environment a toolbox to analyse and visualise the different preliminary predictors (also called machines hereafter), in addition to providing pythonic implementations of several ensemble algorithms (the COBRA algorithm introduced by Biau et al., 2016; the Exponential Weighted Aggregate (EWA) introduced by Vovk, 1990; a COBRA-flavored majority vote inspired by Mojirsheibani, 1999). All these algorithms are supported by oracle bounds which prove that the risk of the aggregated predictor is upper-bounded by the smallest risk of the initial pool of predictors, up to a remainder term which decays to zero (typically at a rate $\mathcal{O}(1/\sqrt{n})$ or faster).

COBRA (standing for COmBined Regression Alternative) is a nonlinear ensemble method designed for regression problems. To predict the response for a new data point, COBRA operates in two steps. First, it retains data points for which the prediction made by preliminary machines is close (in the Euclidean sense) to the prediction made for the new point. This is called the *consensus step*. Second, the final prediction is then formed by averaging responses over the retained points' indices. COBRA outputs a predictor which outperforms any combination of the preliminary predictors (as shown by see Theorem 2.1 in Biau et al., 2016). We describe the COBRA algorithm as implemented in `pycobra` in Algorithm 1. Exponential weights have been used for a long time in statistical learning theory (Vovk, 1990). The EWA algorithm implemented in Pycobra is inspired by the description of Dalalyan and Tsybakov (2007). EWA amounts to forming an exponentially weighted average of preliminary predictors, where the weights include a measure of each predictor's performance. COBRA has been inspired by the work of Mojirsheibani (1999) in classification, and therefore `pycobra` includes a classification version of COBRA called `ClassifierCobra`. The consensus step shares a similar philosophy to what is done in COBRA regression but the final prediction is delivered by a majority vote among the labels of the retained data points.

`pycobra` allows the user to gauge the performance of the preliminary predictors used in the aggregation, with built-in methods to easily plot boxplots and QQ-plots. A salient feature of `pycobra` is using Voronoi tessellations for generic visualisation. By implementing ensemble algorithms which were not represented in the Python machine learning community, and providing a variety of tools to visualise and analyse their behavior and performance, we present to the machine learning open source community a toolbox designed for ensemble learning and visualisation.

2. The `pycobra` library

Our toolbox is written in Python and uses `NumPy` (Walt et al., 2011) and `scikit-learn` (Pedregosa et al., 2011) for computation and linear algebra operations. `Matplotlib` (Hunter, 2007) is used for visualisation purposes, and `SciPy` (Jones et al., 2001) is used to help create Voronoi tessellations. Tutorials and examples are created using Jupyter IPython notebooks (Pérez and Granger, 2007). Currently `pycobra` supports any machine learning algorithm with a `predict` method, casting our procedure into a very flexible and generic framework. By default, `pycobra` relies on `scikit-learn` implementations of Lasso, Random Forest, Decision Trees and Ridge regression for the `EWA` and `COBRA` implementations. As for `ClassifierCobra`, `scikit-learn` implementations of SVM classifier, KNN, Decision Tree classifier, and a classifier which implements regularised linear models with SGD (SGDClassifier object). `pycobra` itself and all software it relies on is open source, with no dependence on proprietary software. Algorithm 1 presents the pseudo-code of the COBRA implementation. All the pycobra estimators are `scikit-learn` compatible and can be used as part of the existing `scikit-learn` ecosystem, such as [GridSearchCV] and [Pipeline]. While hyperparameter initialisation is systematically done using `scikit-learn`'s `GridSearchCV`, `pycobra`'s `Diagnostics` class allows us to compare between different combinations of the constituent predictors and data-splitting, among other basic parameters.

Algorithm 1: The original COBRA algorithm from Biau et al. (2016).

Data: input vector **X**, epsilon, alpha, basic-machines, training-set-responses, training-set

\# `training-set` is the set composed of all `data_point` and the responses. `training-set-responses` is the set composed of the responses.

Result: prediction **Y**

for *machine j in basic-machines* **do**

    machine_set = ;

    \# *machine_set is a dictionary mapping each machine M to a set* `machine_set[M]` ;

    pred = $r_j(\mathbf{X})$;   \# *where $r_j(x)$ denotes the prediction made by machine j at point x*;

    for *response in training-set-responses* **do**

        if $|\text{data\_point} - \text{pred}| \leq$ epsilon, collect `response` in `machine_set[M]` ;

    end

end

array = [ ] ;

for *data_point in training-set* **do**

    if at least `alpha` machines out of M have collected `data_point` in their `set`, store point in array ;

end

result = average(array) ;

The `visualisation` class allows the user to compare all the machines used to create aggregates, as well as visualise the results, for all `pycobra` estimators. `pycobra` ships with [a notebook on visualisation] to illustrate this.

QQ-plots and boxplots. Once all the basic machines are initialized and trained, the user can easily compare their performance with boxplots and QQ-plots. All the plotting details are handled by both the `diagnostics` and `visualisation` classes. An example is given in Figure 1.

Visualisation through Voronoi tessellations. Voronoi tessellations can be used to visualise the selection of optimal machines for COBRA, as well as visualising outputs from clustering algorithms, as illustrated by the [tutorial notebook]. An example is reproduced Figure 2.

3. Project Focus

Community-based development. We intend `pycobra` to be an ongoing collaborative project. To that matter, it is referenced on [Python Package Index (PyPi)] and [Machine Learning Open Source Software]. Moreover, `pycobra` is under active development and avail-
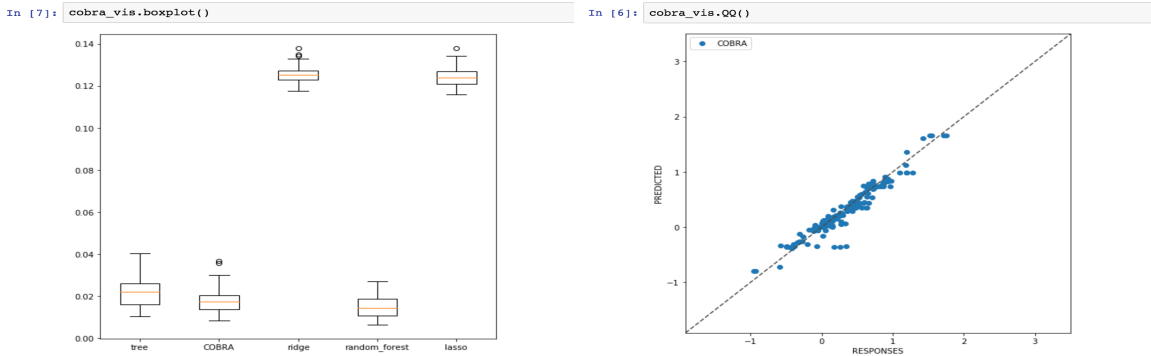

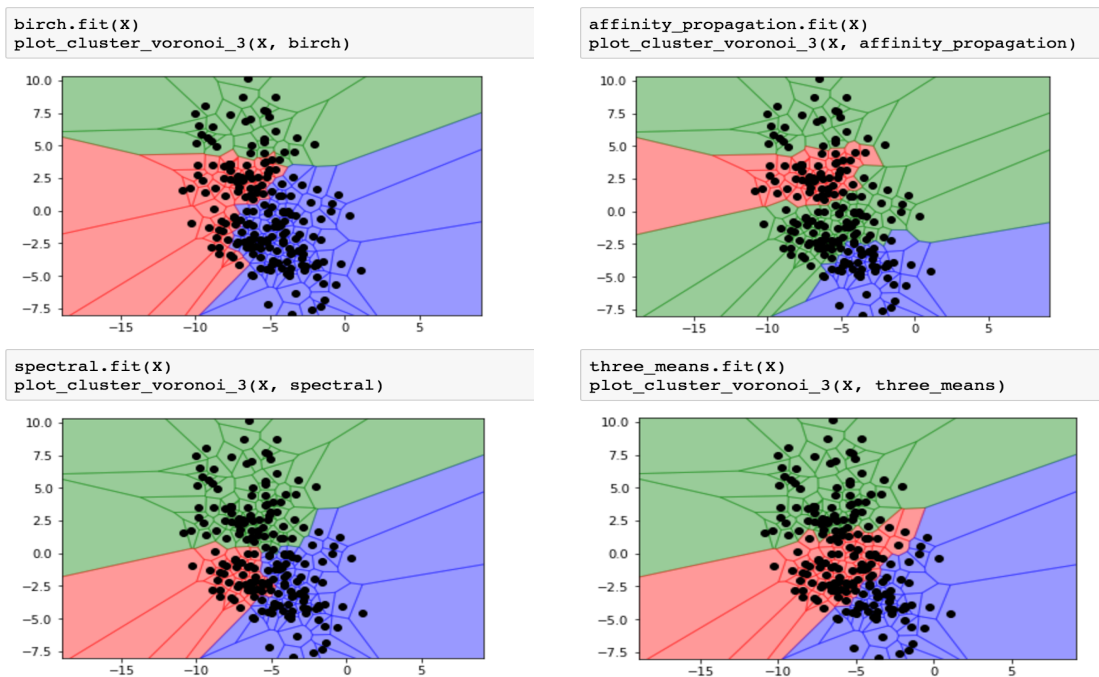Figure 1: Assessing the performance of regression machines and COBRA.

```
birch.fit(X)
plot_cluster_voronoi_3(X, birch)
```

```
affinity_propagation.fit(X)
plot_cluster_voronoi_3(X, affinity_propagation)
```

```
spectral.fit(X)
plot_cluster_voronoi_3(X, spectral)
```

```
three_means.fit(X)
plot_cluster_voronoi_3(X, three_means)
```

Figure 2: Visualising clustering through Voronoi tessellations.

able on [GitHub] to promote collaborative programming, issue tracking and idea discussions.

Documentation and Jupyter Notebooks. A consistent API documentation is provided, along with an additional installation guide and examples. The documentation website is available at https://modal.lille.inria.fr/pycobra. The [notebooks directory] contains Jupyter notebooks which serve as both a documentation tool and a tutorial resource. These notebooks cover use-cases of Pycobra, from solving regression and classification problems to using Voronoi tesselations.

Ease of use and quality assurance. Ensemble learning with `pycobra` is as simple as loading trained `scikit-learn` machines – or any machine which has a `predict` method. Visualising involves little to no parameters, and after loading machines it is straightforward to analyse their performance on a particular dataset. In order to ensure code quality, a set of unit tests is provided for all classes in `pycobra`, and continuous integration via [Travis CI] ensures all commits are tested. The package follows the PEP8 convention, keeping the code easy to read and contribute to.

4. Conclusion and Future Work

The future of `pycobra` would be to grow its user base by adding new ways to add predictors, as well as further implement ensemble learning techniques and visualisation tools. Statistical aggregation and ensemble learning are an important part of the machine learning literature and are widely used by practitioners, yet it seems under-represented in the machine learning open source community. By creating `pycobra` and releasing it to the community, we intend to enrich the existing ecosystem with ensemble algorithms, a generic toolbox for ensemble learning on-the-go, along with analysis and visualisation tools.

References

Robert M. Bell and Yehuda Koren. Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

Gérard Biau, Aurélie Fischer, Benjamin Guedj, and James D. Malley. COBRA: A combined regression strategy. *Journal of Multivariate Analysis*, 146:18–28, 2016.

Arnak Dalalyan and Alexandre Tsybakov. Aggregation by exponential weighting and sharp oracle inequalities. In *Computational Learning theory (COLT)*, pages 97–111. Springer, 2007.

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

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL http://www.scipy.org/.

Majid Mojirsheibani. Combining classifiers via discretization. *Journal of the American Statistical Association*, 94(446):600–609, 1999.

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.

Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.53. URL http://ipython.org.

Volodimir G. Vovk. Aggregating strategies. In *Computational Learning theory (COLT)*, pages 371–386. Morgan Kaufmann Publishers Inc., 1990.

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.