pycobra: A Python toolkit for ensemble regression analysis and visualisation

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

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

Abstract

We introduce `pycobra`, a Python library devoted to ensemble regression and visualisation. Its main assets are the implementation of the COBRA algorithm (an ensemble method introduced by Biau et al., 2016), a flexible and generic interface to compare most regression methods available in Python libraries (including, but not limited to, `scikit-learn`), and visualisation tools based on Voronoi tessellations. `pycobra` is released under the MIT open-source license and can be downloaded from the Python Package Index (PyPi) and Machine Learning Open Source Software. The current development (along with Jupyter notebooks) is available at https://github.com/bhargavvader/pycobra.

Keywords: Regression and Ensemble methods, Machine learning, Voronoi tesselation, Python, Open Source Software.

1. Introduction

Combined statistical procedures, also known as ensemble 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). Ensemble methods combine predictors which are previously trained on a different dataset. `scikit-learn` offers implementations of most of the current ensemble methods – but does not serve as a way to analyse the predictors used. It rather focuses on improving their accuracy. `pycobra` attempts to fill this gap by providing a toolkit to analyse the different basic predictors, in addition to providing a pythonic implementation of the COBRA algorithm introduced by Biau et al. (2016).

COBRA is a nonlinear ensemble method which outputs a nonparametric combined predictor. Given a sample $(X_i, Y_i)_{i=1}^n$ and a collection of basic predictors $\mathbf{r} = (r_1, \ldots, r_M)$, COBRA is defined as the weighted average $T_n(x) = \sum_{i=1}^n W_{n,i}(x) Y_i$ for any new query point $x \in \mathbb{R}^d$, where the weights

$$W_{n,i}(x) \propto \mathbf{1}\left[\bigcap_{m=1}^{M} \{|r_m(x) - r_m(X_i)| \leq \varepsilon\}\right]$$

sum up to one. The COBRA algorithm is supported by oracle inequalities showing that it outperforms any of the basic predictors in terms of expected quadratic loss, up a to a remainder term of magnitude $n^{-\frac{2}{2+M}}$ (see Theorem 2.1 in Biau et al., 2016).

`pycobra` allows the user to gauge the performance of the basic predictors used in the collective, with built-in methods to easily plot boxplots and QQ-plots. A unique feature of `pycobra` is using Voronoi tessellations for generic visualisation. This feature was initially included to better understand the aforementioned weights $W_{n,i}$ and it has been further developed to visualise clustering algorithms. By implementing a novel nonlinear ensemble method and providing a variety of tools to visualise and analyse the basic predictors used in the ensemble, we present to the Machine Learning open source community a useful ensemble regression analysis and visualisation toolkit.

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 regression predictor which has 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. `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. While `pycobra` provides default values for `epsilon`, `alpha` and `basic-machines`, the `diagnostics` class allows the user to tune the `pycobra` object and find optimal hyperparameters for a particular dataset. The `epsilon` parameter acts as a bandwidth and induces a closeness notion. The higher `epsilon`, the more points will be retained in the collective. The performance of ensemble methods may suffer from the initial split of the data, between a set used to train basic predictors and another used to blend them altogether. `pycobra` includes a way of selecting the optimal `split`. Requiring a consensus over all basic predictors might be uneffective in some settings, hence we introduce

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

```
Data: input Vector $\mathbf{X}$, epsilon, alpha, basic-machines, training-set
Result: prediction $\mathbf{Y}$
for machine $j$ in basic-machines do
    set = [ ] ;
    pred = $r_j(\mathbf{X})$;
    for point in training-set do
        if |point − pred| $\leq$ epsilon, collect point in machine $M$'s set ;
    end
end
array = [ ] ;
for point in training-set do
    if at least alpha machines out of $M$ have collected point in their set, store point in array ;
end
result = average(array) ;
```

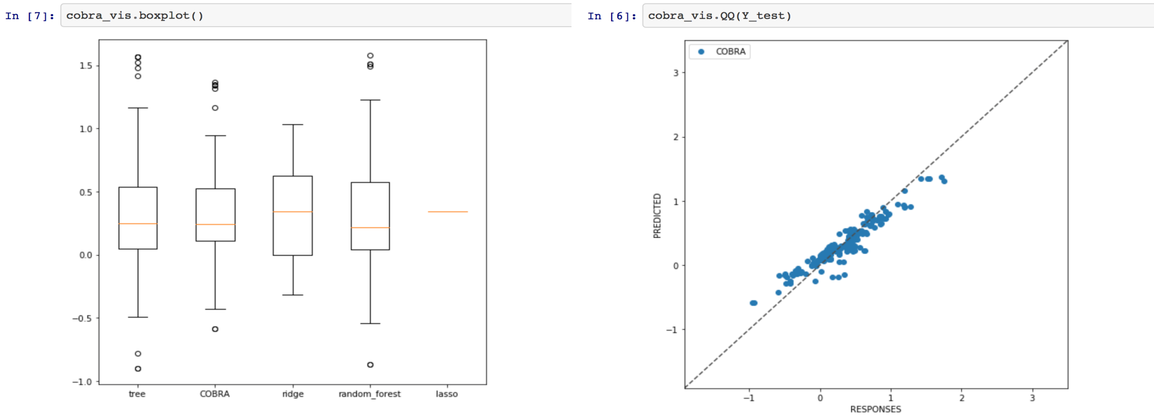
the relaxing parameter `alpha`. This is the proportion of basic predictors which are required to reach a consensus. Finally, `pycobra` also offers the possibility to obtain the subset of basic predictors which achieve the best performance (see the `optimal-machines` function).

The `visualisation` class allows the user to compare all the machines used to create the COBRA aggregate, as well as visualise the results. `pycobra` ships with a notebook on visualisation to illustrate this.

QQ plots and boxplots. Once all the basic machines are set up, 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.

Pointwise best machines. An interesting way to analyse how machines work is to identify which points use which basic machines to get an optimal prediction. The user can visualise this by either coloring the points or viewing it as a Voronoi tessellation.

Visualise Clustering through Voronoi tessellations. Voronoi tessellations are an interesting way to visualise data, and in another tutorial notebook we describe how to use Voronoi tessellations to visualise `scikit-learn` clustering algorithms, as shown in Figure 2.



```
In [7]: cobra_vis.boxplot()
```

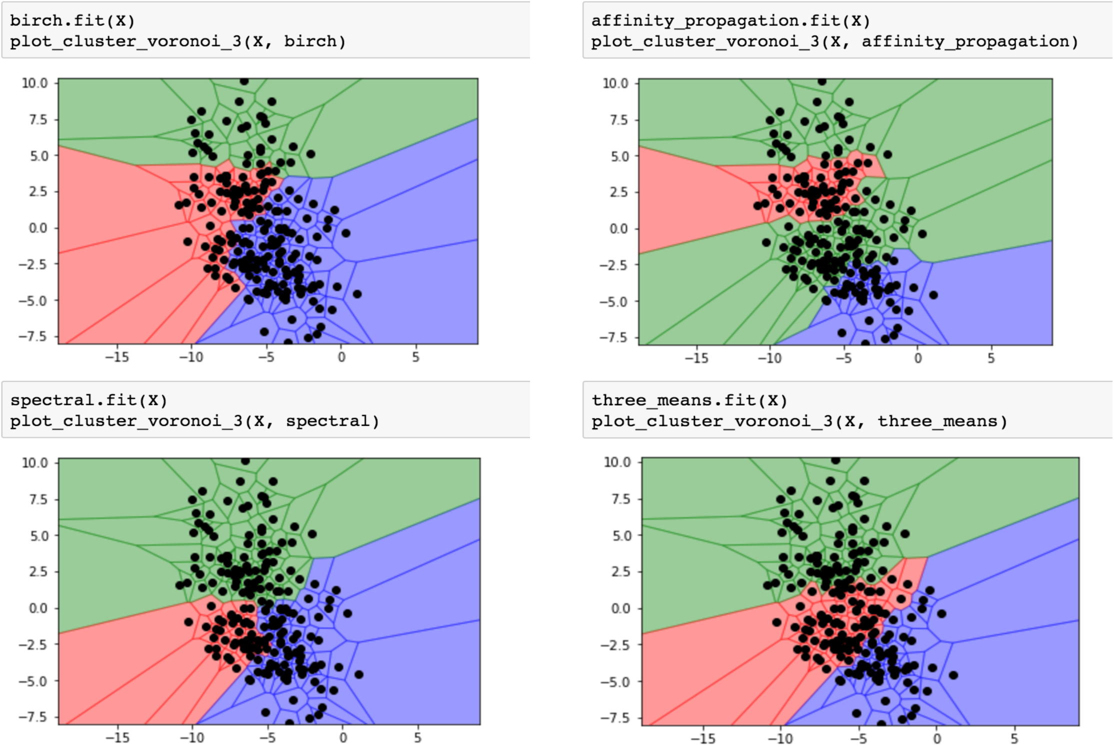
```
In [6]: cobra_vis.QQ(Y_test)
```

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

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 available on GitHub to enforce collaborative programming, issue tracking, code integration, and idea discussions.

Documentation and Jupyter Notebooks. A consistent API documentation is provided, along with an additional installation guide and examples. The notebooks directory



```
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.

contains 3 Jupyter notebooks which serve as both a documentation tool and a tutorial resource. They cover how to set up and use pycobra, basic regression analysis, and how to use Voronoi tessellations to visualise clustering.

Ease of use and quality assurance. Ensemble regression 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.

4. Conclusion and Future Work

The future of pycobra would be to grow its user base by adding more regression comparison tools, new ways to add regression machines, and on implementing more novel nonlinear ensemble techniques - all of which are on the GitHub issue list. Predictor aggregation and ensemble learning are an important part of the machine learning literature and is 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 provide an interesting alternative to the current ensemble algorithms, while also including useful analysis and visualisation options.

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.

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/.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 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.

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.