PyUoI: The Union of Intersections Framework in Python

Pratik S. Sachdeva[1, 2, 3], **Jesse A. Livezey**[1, 3], **Andrew J. Tritt**[4], and **Kristofer E. Bouchard**[1, 3, 4, 5]

1 Redwood Center for Theoretical Neuroscience, University of California, Berkeley, Berkeley, California, USA **2** Department of Physics, University of California, Berkeley, Berkeley, California, USA **3** Biological Systems and Engineering Division, Lawrence Berkeley National Laboratory, Berkeley, California, USA **4** Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, California, USA **5** Helen Wills Neuroscience Institute, University of California, Berkeley, Berkeley, California, USA

DOI: 10.21105/joss.01799

Software

- Review
- Repository
- Archive

Editor: Yuan Tang

Reviewers:

- @puolival
- @yzhao062

Submitted: 04 October 2019
Published: 06 December 2019

Summary

The increasing size and complexity of scientific data requires statistical analysis methods that scale and produce models that are both interpretable and predictive. Interpretability implies one can interpret the output of the model in terms of processes generating the data (Murdoch, Singh, Kumbier, Abbasi-Asl, & Yu, 2019). This typically requires identification of a small number of features in the actual data and accurate estimation of their contributions (Bickel et al., 2006). Meanwhile, achieving predictive power requires optimizing the performance of some statistical measure such as precision, mean squared error, etc. Across inference procedures, there is often a trade-off between interpretability and predictive power. The impact of this trade-off is particularly acute for scientific applications, where the output of the model is used to provide insight into the underlying physical processes that generated the data.

We recently introduced Union of Intersections (UoI), a flexible, modular, and scalable framework designed to enhance both the identification of features (model selection) as well as the estimation of the contributions of these features (model estimation) (Bouchard et al., 2017). UoI-based methods leverage stochastic data resampling and a range of sparsity-inducing regularization parameters to build families of potential feature sets robust to perturbations of the data, and then average nearly unbiased parameter estimates of selected features to maximize predictive accuracy. Models inferred through the UoI framework are characterized by their usage of fewer parameters with little or no loss in predictive accuracy, and reduced bias relative to benchmark approaches.

`PyUoI` is a Python package containing implementations of a variety of UoI-based algorithms, encompassing regression, classification, and dimensionality reduction. In order to better facilitate its usage, `PyUoI`'s API is structured similarly to the `scikit-learn` package, which is a commonly used Python machine learning library (Buitinck et al., 2013; Pedregosa et al., 2011).

The UoI framework operates by fitting many models across resamples of the dataset and across a set of regularization parameters. Since these fits can be performed in parallel, the UoI framework is naturally scalable. `PyUoI` is equipped with `mpi4py` functionality to parallelize model fitting on large datasets (Dalcín, Paz, & Storti, 2005).

Background

The Union of Intersections is not a single method or algorithm, but a flexible statistical framework into which other algorithms can be inserted. In this section, we briefly describe

UoI$_{\text{Lasso}}$, the UoI implementation of lasso penalized regression. UoI$_{\text{Lasso}}$ is similar in structure to the UoI versions of other generalized linear models (logistic and poisson). We refer the user to existing literature on the UoI variants of column subset selection and non-negative matrix factorization (Bouchard et al., 2017; Ubaru, Wu, & Bouchard, 2017).

Linear regression consists of estimating parameters $\boldsymbol{\beta} \in \mathbb{R}^{p\times 1}$ that map a p-dimensional vector of features $\mathbf{x} \in \mathbb{R}^{p\times 1}$ to the observation variable $y \in \mathbb{R}$, when the N samples are corrupted by i.i.d Gaussian noise:

$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon \tag{1}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ for each sample. When the true $\boldsymbol{\beta}$ is thought to be sparse (i.e., some subset of the $\boldsymbol{\beta}$ are exactly zero), an estimate of $\boldsymbol{\beta}$ can be found by solving a constrained optimization problem of the form

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\text{argmin}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \boldsymbol{\beta}^T \mathbf{x}_i)^2 + \lambda |\boldsymbol{\beta}|_1 \tag{2}$$

where $|\boldsymbol{\beta}|_1$ is the ℓ_1-norm of the parameters and i indexes data samples. The ℓ_1-norm is a convenient penalty because it will tend to force parameters to be set exactly equal to zero, performing feature selection (Tibshirani, 1994). Typically, λ, the degree to which feature sparsity is enforced, is unknown and must be determined through cross-validation or a penalized score function across a set of hyperparameters $\{\lambda_j\}_{j=1}^{k}$.

The key mathematical idea underlying UoI is to perform model selection through intersection (compressive) operations and model estimation through union (expansive) operations, in that order. This separation of parameter selection and estimation provides selection profiles that are more robust and parameter estimates that have less bias. This can be contrasted with a typical Lasso fit wherein parameter selection and estimation are performed simultaneously. The Lasso procedure can lead to selection profiles that are not robust to data resampling and estimates that are biased by the penalty on $\boldsymbol{\beta}$. For UoI$_{\text{Lasso}}$, the procedure is as follows (see Algorithm 1 for a more detailed pseudocode):

- **Model Selection:** For each λ_j in the Lasso path, generate estimates on N_S resamples of the data (Line 2). The support S_j (i.e., the set of non-zero parameters) for λ_j consists of the features that persist in all model fits across the resamples (i.e., through an intersection) (Line 7).
- **Model Estimation:** For each support S_j, perform Ordinary Least Squares (OLS) on N_E resamples of the data. The final model is obtained by averaging (i.e., taking the union) across the supports chosen according to some model selection criteria for each resample (Lines 15-16). The model selection criteria can be prediction quality on held-out data or penalized likelihood methods (e.g., AIC or BIC).

Thus, the selection module ensures that, for each λ_j, only features that are stable to perturbations in the data (resamples) are allowed in the support S_j. This provides a family of resample-stable model supports with varying levels of sparsity due to λ_j that can be used in estimation. Then, the estimation module ensures that the most predictive supports per resample are averaged together in the final model. The estimation module uses OLS rather than Lasso to provide parameter estimates with low bias. The degree of feature compression via intersections (quantified by N_S) and the degree of feature expansion via unions (quantified by N_E) can be balanced to maximize prediction accuracy for the response variable y.

Algorithm 1 $\textsc{UoI}_{\textsc{Lasso}}$

```
    Input: $X \in \mathbb{R}^{N \times p}$ design matrix
           $\mathbf{y} \in \mathbb{R}^{N \times 1}$ response variable
           Regularization strengths $\{\lambda_j\}_{j=1}^{q}$
           Number of resamples $N_S$ and $N_E$
           Loss function $L(\boldsymbol{\beta}; X, \mathbf{y})$
    Model Selection
 1: for $k = 1$ to $N_S$ do
 2:     Generate resample $X^k$, $\mathbf{y}^k$
 3:     for $j = 1$ to $q$ do
 4:         $\hat{\boldsymbol{\beta}}^{jk} \leftarrow$ Lasso regression (penalty $\lambda_j$) of $\mathbf{y}^k$ on $X^k$
 5:         $S_j^k \leftarrow \{i\}$ where $\hat{\boldsymbol{\beta}}_i^{jk} \neq 0$
 6: for $j = 1$ to $q$ do
 7:     $S_j \leftarrow \bigcap_{k=1}^{N_S} S_j^k$                                  ▷ Intersection
 8: Model Estimation
 9: for $k = 1$ to $N_E$ do
10:     Generate training $\left(X_T^k, \mathbf{y}_T^k\right)$ and evaluation $\left(X_E^k, \mathbf{y}_E^k\right)$ resamples
11:     for $j = 1$ to $q$ do
12:         $X_{T,j}^k, X_{E,j}^k \leftarrow X_T^k, X_E^k$ with features $S_j$ extracted.
13:         $\hat{\boldsymbol{\beta}}^{jk} \leftarrow$ OLS Regression of $\mathbf{y}_T^k$ on $X_{T,j}^k$
14:         $\ell^{jk} \leftarrow L(\hat{\boldsymbol{\beta}}^{jk}; X_{E,j}^k, \mathbf{y}_E^k)$
15:     $\hat{\boldsymbol{\beta}}^k \leftarrow \underset{\hat{\boldsymbol{\beta}}^{jk}}{\text{argmin}}\ \ell^{jk}$
16: $\hat{\boldsymbol{\beta}}^* = \underset{k}{\text{median}}\left(\hat{\boldsymbol{\beta}}^k\right)$                          ▷ Union
17: return $\hat{\boldsymbol{\beta}}^*$
```

Features

`PyUoI` is split up into two modules, with the following UoI algorithms:

- `linear_model` (generalized linear models)
  - Lasso penalized linear regression $\text{UoI}_{\text{Lasso}}$.
  - Elastic-net penalized linear regression ($\text{UoI}_{\text{ElasticNet}}$).
  - Logistic regression (Bernoulli and multinomial) ($\text{UoI}_{\text{Logistic}}$).
  - Poisson regression ($\text{UoI}_{\text{Poisson}}$).
- `decomposition` (dimensionality reduction)
  - Column subset selection (UoI_{CSS}).
  - Non-negative matrix factorization (UoI_{NMF}).

The generalized linear models we have implemented include the most commonly used models in a variety of scientific disciplines, particularly in the fields of neuroscience and genomics. Extensions to other generalized linear models (e.g., negative binomial regression, gamma regression, etc.) are left as future work. However, given the inheritance structure of the `PyUoI` framework, these extensions should be straightforward for the interested user.

Similar to `scikit-learn`, each UoI algorithm has its own Python class. Instantiations of these classes are created with specific hyperparameters and are fit to user-provided datasets. The hyperparameters allow the user to fine-tune the number of resamples, fraction of data in each resample, and the model selection criteria used in the estimation module (in Algorithm 1, test set accuracy is used, but the Akaike and Bayesian Information Criteria are also available (Akaike, 1998; Schwarz, 1978)).

Additionally, UoI is agnostic to the specific solver used for a given model. That is, the UoI framework operates on fits obtained from performing the optimization for a specified model (such as the lasso optimization problem for linear regression). In the case of `PyUoI`, the generalized linear models come equipped with a coordinate descent solver (from `scikit-learn`), a built-in Orthant-Wise Limited memory Quasi-Newton solver (Gong & Ye, 2015), and the `pycasso` solver (Ge, 2019). The choice of solver is left to the user as a hyperparameter. If a different solver is desired, `PyUoI` could be extended by the user to utilize this solver in a straightforward manner.

Applications

We have used `PyUoI` largely in the realm of neuroscience and genomics (Bouchard et al., 2017; Ubaru et al., 2017). A few applications include:

- Interpretable functional connectivity networks from neural populations in the visual, auditory, and motor cortices of various animal models;
- Sparse decoding of behavioral activity from spiking neural activity;
- Parts-based decomposition of electrocorticography recordings in rat auditory cortex that reflect functional cortical organization;
- Extraction of characteristic single nucleotide polymorphisms for the prediction of phenotypes in mice.

However, the algorithms implemented in `PyUoI` are broadly applicable to problems where enforcement of sparsity at minimal cost to bias are desired.

Acknowledgements

We thank the contributors to earlier versions of this software. P.S.S. was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. K.E.B. and J.A.L. were supported through the Lawrence Berkeley National Laboratory-internal LDRD "Deep Learning for Science" led by Prabhat. A.J.T. was supported by the Department of Energy project "Co-design for artificial intelligence coupled with computing at scale for extremely large, complex datasets." K.E.B. was funded by Lawrence Berkeley National Laboratory-internal LDRD "Neuro/Nano-Technology for BRAIN" led by Peter Denes.

References

Akaike, H. (1998). Information theory and an extension of the maximum likelihood principle. In *Selected papers of Hirotugu Akaike* (pp. 199–213). Springer. doi:10.1007/978-1-4612-1694-0_15

Bickel, P. J., Li, B., Tsybakov, A. B., Geer, S. A. van de, Yu, B., Valdés, T., Rivero, C., et al. (2006). Regularization in statistics. *Test*, *15*, 271–344. doi:10.1007/BF02607055

Bouchard, K., Bujan, A., Roosta-Khorasani, F., Ubaru, S., Prabhat, M., Snijders, A., Mao, J.-H., et al. (2017). Union of Intersections (UoI) for interpretable data driven discovery and prediction. In *Advances in Neural Information Processing Systems 30* (pp. 1078–1086).

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., et al. (2013). API design for machine learning software: Experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (pp. 108–122).

Dalcín, L., Paz, R., & Storti, M. (2005). MPI for Python. *Journal of Parallel and Distributed Computing*, *65*(9), 1108–1115.

Ge, J. (2019). PICASSO: PathwIse calibrated sparse shooting algOrithm. *GitHub repository*. https://github.com/jasonge27/picasso; GitHub.

Gong, P., & Ye, J. (2015). A modified orthant-wise limited memory quasi-newton method with convergence analysis. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (pp. 276–284).

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., & Yu, B. (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, *116*(44), 22071–22080. doi:10.1073/pnas.1900654116

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Schwarz. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464. doi:10.1214/aos/1176344136

Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, *58*, 267–288.

Ubaru, S., Wu, K., & Bouchard, K. E. (2017). UoI-NMF cluster: A robust nonnegative matrix factorization algorithm for improved parts-based decomposition and reconstruction of noisy data. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 241–248). doi:10.1109/ICMLA.2017.0-152