The Journal of Open Source Software

PySensors: A Python package for sparse sensor placement

Brian M. de Silva[1], Krithika Manohar[2], Emily Clark[3], Bingni W. Brunton[4], J. Nathan Kutz[1], and Steven L. Brunton[2]

1 Department of Applied Mathematics, University of Washington **2** Department of Mechanical Engineering, University of Washington **3** Department of Physics, University of Washington **4** Department of Biology, University of Washington

DOI: 10.21105/joss.02828

Software

- Review
- Repository
- Archive

Editor: Pierre de Buyl

Reviewers:

- @jordanperr
- @tuelwer

Submitted: 24 October 2020
Published: 21 February 2021

Summary

Successful predictive modeling and control of engineering and natural processes is often entirely determined by *in situ* measurements and feedback from sensors (S. L. Brunton & Kutz, 2019), which provide measurements of the state of these processes at specific points in space and time. However, deploying sensors into complex environments, including in application areas such as manufacturing (Manohar, Hogan, et al., 2018), geophysical environments (Yildirim et al., 2009), and biological processes (Colvert et al., 2017; Mohren et al., 2018), is often expensive and challenging. Furthermore, modeling outcomes are extremely sensitive to the location and number of these sensors, motivating optimization strategies for the principled placement of sensors for different decision-making tasks. In general, choosing the globally optimal placement within the search space of a large-scale complex system is an intractable computation, in which the number of possible placements grows combinatorially with the number of candidates (Ko et al., 1995). While sensor placement has traditionally been guided by expert knowledge and first principles models, increases in system complexity, emerging sensor technologies, and innovations in data-driven modeling strategies motivates automated algorithms for optimizing sensor placements.

`PySensors` is a Python package for the scalable optimization of sensor placement from data. In particular, `PySensors` provides tools for sparse sensor placement optimization approaches that employ data-driven dimensionality reduction (B. W. Brunton et al., 2016; Manohar, Brunton, et al., 2018). This approach results in near-optimal placements for various decision-making tasks and can be readily customized using different optimization algorithms and objective functions.

The `PySensors` package can be used by both researchers looking to advance state-of-the-art methods and practitioners seeking simple sparse sensor selection methods for their applications of interest. Straightforward methods and abundant examples help new users to quickly and efficiently leverage existing methods to their advantage. At the same time, modular classes leave flexibility for users to experiment with and plug in new sensor selection algorithms or dimensionality reduction techniques. Users of `scikit-learn` will find `PySensors` objects familiar, intuitive, and compatible with existing `scikit-learn` routines such as cross-validation (Pedregosa et al., 2011).

Statement of need

Maximizing the impact of sensor placement algorithms requires tools to make them accessible to scientists and engineers across various domains and at various levels of mathematical expertise and sophistication. `PySensors` unifies the algorithms developed in the papers (B. W.

Brunton et al., 2016; Clark et al., 2018; Manohar, Brunton, et al., 2018) and their accompanying codes `SSPOR_pub` and `SSPOC_pub` into one software package. The only other packages in this domain of which we are aware are `Chama` (Klise et al., 2017) and `Polire` (Narayanan et al., 2020). While these packages and `PySensors` all enable sparse sensor placement optimization, `Chama` and `Polire` are geared towards event detection and Gaussian processes respectively, whereas `PySensors` is aimed at signal reconstruction and classification tasks. As such, there are marked differences in the objective functions optimized by `PySensors` and its precursors. In addition to these two packages, researchers and practitioners have made available various custom scripts for sensor placement. Currently, researchers seeking to employ modern sensor placement methods must choose between implementing them from scratch or manually augmenting existing unpolished codes.

Reconstruction and classification tasks often arise in the modeling, prediction, and control of complex processes in geophysics, fluid dynamics, biology, and manufacturing. The goal of *reconstruction* is to recover a high-dimensional signal $\mathbf{x} \in \mathbb{R}^N$ from a limited number of p measurements $y_i = \mathbf{c}_i^\top \mathbf{x}$, where each $\mathbf{c}_i \in \mathbb{R}^N$ represents the action of a sensor. For example, $\mathbf{c}_i^\top = [1, 0, 0, \ldots, 0]$ represents a sensor which takes a point measurement of the first dimension of the signal $\mathbf{x}$. `PySensors` selects a set of p sensors out of N candidates $\mathbf{c}_i^\top$ (rows of a measurement matrix $\mathbf{C} : \mathbf{y} = \mathbf{C}\mathbf{x}$) that minimize reconstruction error in a data-dependent basis $\mathbf{\Phi} \in \mathbb{R}^{N \times r}$

$$\mathbf{C}^\star = \underset{\mathbf{C} \in \mathbb{R}^{p \times N}}{\arg\min} \|\mathbf{x} - \mathbf{\Phi}(\mathbf{C}\mathbf{\Phi})^\dagger \mathbf{y}\|_2^2,$$

where $\dagger$ denotes the Moore-Penrose pseudoinverse. The key innovation is to recover the low-dimensional representation $\mathbf{x}_r \in \mathbb{R}^r$ satisfying $\mathbf{x} = \mathbf{\Phi}\mathbf{x}_r$ via the reconstruction map $\mathbf{\Phi}(\mathbf{C}\mathbf{\Phi})^\dagger$, ultimately reducing sensor placement to a highly efficient matrix pivoting operation (Manohar, Brunton, et al., 2018). Similarly, sensor placement for *classification* (B. W. Brunton et al., 2016) optimizes the sparsest vector $\mathbf{s}^\star$ that reconstructs $\mathbf{w} : \mathbf{\Phi}^\dagger \mathbf{s} = \mathbf{w}$ in the low-dimensional feature space, where $\mathbf{w}$ is the the set of weights learned by a linear classifier fit to $\mathbf{x}_r$. In this case, the optimal sensor locations are determined by the nonzero components of $\mathbf{s}^\star$.

The basis $\mathbf{\Phi}$ is explicitly computed from the data using powerful dimensionality reduction techniques such as principal components analysis (PCA) and random projections, which enable significant compression of most signals to $r \ll N$ dimensions. PCA extracts the dominant spatial correlations or *principal components*, the leading eigenvectors of the data covariance matrix. It is computed using the matrix singular value decomposition (SVD) and is closely related to proper orthogonal decomposition (POD); POD modes and principal components are equivalent. Other basis choices are possible, such as dynamic mode decomposition for extracting temporally correlated features (Manohar et al., 2019).

Features

`PySensors` enables the sparse placement of sensors for two classes of problems: reconstruction and classification. For reconstruction problems the package implements a unified `SensorSelector` class, with methods for efficiently analyzing the effects that data or sensor quantity have on reconstruction performance (Manohar, Brunton, et al., 2018). Sensor selection is based on the computationally efficient QR algorithm. Often different sensor locations impose variable costs, e.g. if measuring sea-surface temperature, it may be more expensive to place buoys/sensors in the middle of the ocean than close to shore. These costs can be taken into account during sensor selection via a built-in cost-sensitive optimization routine (Clark et al., 2018). For classification tasks, the package implements the Sparse Sensor Placement Optimization for Classification (SSPOC) algorithm (B. W. Brunton et al., 2016), allowing one to optimize sensor placement for classification accuracy. The algorithm is related to compressed sensing optimization (Baraniuk, 2007; Candès et al., 2006; Donoho, 2006), but

identifies the sparsest set of sensors that reconstructs a discriminating plane in a feature subspace. This SSPOC implementation is fully general in the sense that it can be used in conjunction with any linear classifier. Additionally, `PySensors` provides methods to enable straightforward exploration of the impacts of primary hyperparameters like the number of sensors or basis modes.

It is well known (Manohar, Brunton, et al., 2018) that the basis in which one represents measurement data can have a pronounced effect on the sensors that are selected and the quality of the reconstruction. Users can readily switch between different bases typically employed for sparse sensor selection, including POD modes and random projections. Because `PySensors` was built with `scikit-learn` compatibility in mind, it is easy to use cross-validation to select among possible choices of bases, basis modes, and other hyperparameters.

Finally, included with `PySensors` is a large suite of examples, implemented as Jupyter notebooks. Some of the examples are written in a tutorial format and introduce new users to the objects, methods, and syntax of the package. Other examples demonstrate intermediate-level concepts such as how to visualize model parameters and performance, how to combine `scikit-learn` and `PySensors` objects, selecting appropriate parameter values via cross-validation, and other best-practices. Further notebooks use `PySensors` to solve challenging real-world problems. The notebooks reproduce many of the examples from the papers upon which the package is based (B. W. Brunton et al., 2016; Clark et al., 2018; Manohar, Brunton, et al., 2018). To help users begin applying `PySensors` to their own datasets even faster, interactive versions of every notebook are available on Binder. Together with comprehensive documentation, the examples will compress the learning curve of learning a new software package.

Acknowledgments

The authors acknowledge support from the Air Force Office of Scientific Research (AFOSR FA9550-19-1-0386) and The Boeing Corporation. The work of KM is supported by the National Science Foundation Mathematical Sciences Postdoctoral Research Fellowship (award 1803663). JNK acknowledges support from the Air Force Office of Scientific Research (AFOSR FA9550-19-1-0011)

References

Baraniuk, R. G. (2007). Compressive sensing. *IEEE Signal Processing Magazine*, *24*(4), 118–120. https://doi.org/10.1109/MSP.2007.4286571

Brunton, B. W., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Sparse sensor placement optimization for classification. *SIAM Journal on Applied Mathematics*, *76*(5), 2099–2122. https://doi.org/10.1137/15M1036713

Brunton, S. L., & Kutz, J. N. (2019). *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press. https://doi.org/10.1017/9781108380690

Candès, E. J., Romberg, J. K., & Tao, T. (2006). Stable signal recovery from incomplete and inaccurate measurements. *Communications in Pure and Applied Mathematics*, *59*(8). https://doi.org/10.1002/cpa.20124

Clark, E., Askham, T., Brunton, S. L., & Kutz, J. N. (2018). Greedy sensor placement with cost constraints. *IEEE Sensors Journal*, *19*(7), 2642–2656. https://doi.org/10.1109/JSEN.2018.2887044

Colvert, B., Chen, K., & Kanso, E. (2017). Local flow characterization using bioinspired sensory information. *Journal of Fluid Mechanics*, *818*, 366–381. https://doi.org/10.1017/jfm.2017.137

Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory*, *52*(4), 1289–1306. https://doi.org/10.1109/TIT.2006.871582

Klise, K. A., Nicholson, B., & Laird, C. D. (2017). Sensor placement optimization using chama. *Number Sand2017-11472. Albuquerque, NM: Sandia National Laboratories*. https://doi.org/10.2172/1405271

Ko, C.-W., Lee, J., & Queyranne, M. (1995). An exact algorithm for maximum entropy sampling. *Operations Research*, *43*(4), 684–691. https://doi.org/10.1287/opre.43.4.684

Manohar, K., Brunton, B. W., Kutz, J. N., & Brunton, S. L. (2018). Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns. *IEEE Control Systems Magazine*, *38*(3), 63–86. https://doi.org/10.1109/mcs.2018.2810460

Manohar, K., Hogan, T., Buttrick, J., Banerjee, A. G., Kutz, J. N., & Brunton, S. L. (2018). Predicting shim gaps in aircraft assembly with machine learning and sparse sensing. *Journal of Manufacturing Systems*, *48*, 87–95. https://doi.org/10.1016/j.jmsy.2018.01.011

Manohar, K., Kaiser, E., Brunton, S. L., & Kutz, J. N. (2019). Optimized sampling for multiscale dynamics. *Multiscale Modeling & Simulation*, *17*(1), 117–136. https://doi.org/10.1137/17M1162366

Mohren, T. L., Daniel, T. L., Brunton, S. L., & Brunton, B. W. (2018). Neural-inspired sensors enable sparse, efficient classification of spatiotemporal data. *Proceedings of the National Academy of Sciences*, *115*(42), 10564–10569. https://doi.org/10.1073/pnas.1808909115

Narayanan, S. D., Patel, Z. B., Agnihotri, A., & Batra, N. (2020). A toolkit for spatial interpolation and sensor placement. *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 653–654. https://doi.org/10.1145/3384419.3430407

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Yildirim, B., Chryssostomidis, C., & Karniadakis, G. (2009). Efficient sensor placement for ocean measurements using low-dimensional concepts. *Ocean Modelling*, *27*, 160–173. https://doi.org/10.1016/j.ocemod.2009.01.001