PySensors: A Python package for sparse sensor placement

7 October 2020

Extra material added in by Brian

TODO: remove extra material

What should the paper contain?

From the JOSS submission guide:

"JOSS welcomes submissions from broadly diverse research areas. For this reason, we require that authors include in the paper some sentences that explain the software functionality and domain of use to a non-specialist reader. We also require that authors explain the research applications of the software. The paper should be between 250-1000 words.

Your paper should include:

- A list of the authors of the software and their affiliations, using the correct format (see the example below).
- A summary describing the high-level functionality and purpose of the software for a diverse, non-specialist audience.
- A clear Statement of Need that illustrates the research purpose of the software.
- A list of key references, including to other software addressing related needs.
- Mention (if applicable) a representative set of past or ongoing research projects using the software and recent scholarly publications enabled by it.
- Acknowledgment of any financial support."

See also the review checklist to get an idea of what the reviewers are looking for.

Summary

Successful predictive modeling and control of engineering and natural processes is often entirely determined by in situ measurements and feedback from sensors.

However, deploying sensors into complex environments in manufacturing, geophysical and biological processes is often expensive and challenging. Furthermore, modeling outcomes are extremely sensitive to the location and number of these sensors, motivating the optimization of sensor placements 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. While sensor placements have traditionally been guided by expert knowledge and first principles models, the explosion in system complexity, data collection and data-driven modeling motivates automated algorithms for optimizing sensor placements.

A number of automated sensor placement methods have been developed in recent years, designed to optimize outcomes in the design of experiments, convex (Joshi and Boyd 2008; Brunton et al. 2016) and submodular objective functions (Summers, Cortesi, and Lygeros 2015), information theoretic criteria (Krause, Singh, and Guestrin 2008), optimal control, and reduced order modeling (Willcox 2006; Manohar et al. 2018; Clark et al. 2018). Maximizing the impact of sensor placement algorithms requires tools to make them accessible to scientists across various domains and at various levels of mathematical expertise.

`PySensors` is a Python package for the scalable optimization of sensor placements from data. In particular, `PySensors` provides tools for sparse sensor placement optimization approaches that employ data-driven dimensionality reduction (Brunton et al. 2016; Manohar 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 the state of the art and practitioners seeking simple sparse sensor selection methods for their applications of interest. Straightforward methods and abundant examples help new users to hit the ground running. 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.

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 data or sensor quantity have on reconstruction performance. 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 dur-

ing 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 (Brunton et al. 2016), allowing one to optimize sensor placement for classification accuracy. 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 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 PCA 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 (Manohar et al. 2018; Clark et al. 2018; Brunton et al. 2016). To help users begin applying `PySensors` to their own datasets even faster, interactive versions of every notebook are available on Binder. Overall, the examples will compress the learning curve of learning a new software package.

Acknowledgments

TODO: write acknowledgments section

References

Brunton, Bingni W, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. 2016. "Sparse Sensor Placement Optimization for Classification." *SIAM Journal on Applied Mathematics* 76 (5): 2099–2122.

Clark, Emily, Travis Askham, Steven L Brunton, and J Nathan Kutz. 2018. "Greedy Sensor Placement with Cost Constraints." *IEEE Sensors Journal* 19 (7): 2642–56.

Joshi, Siddharth, and Stephen Boyd. 2008. "Sensor Selection via Convex Optimization." *IEEE Transactions on Signal Processing* 57 (2): 451–62.

Krause, Andreas, Ajit Singh, and Carlos Guestrin. 2008. "Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies." *Journal of Machine Learning Research* 9 (Feb): 235–84.

Manohar, Krithika, Bingni W Brunton, J Nathan Kutz, and Steven L Brunton. 2018. "Data-Driven Sparse Sensor Placement for Reconstruction: Demonstrating the Benefits of Exploiting Known Patterns." *IEEE Control Systems Magazine* 38 (3): 63–86.

Summers, Tyler H, Fabrizio L Cortesi, and John Lygeros. 2015. "On Submodularity and Controllability in Complex Dynamical Networks." *IEEE Transactions on Control of Network Systems* 3 (1): 91–101.

Willcox, Karen. 2006. "Unsteady Flow Sensing and Estimation via the Gappy Proper Orthogonal Decomposition." *Computers & Fluids* 35 (2): 208–26.