Title: Nilearn: Streamlined neuroimaging analysis with enhanced infrastructure and surface API integration

Authors: Yasmin Mzayek, Pierre Bellec, Ahmad Chamma, Alexandre Cionca, Jelle Roelof Dalenberg, Jérôme Dockès, Mathieu Dugré, Elizabeth DuPre, Rémi Gau, Nicolas Gensollen, Mathias Goncalves, Anne-Sophie Kieslinger, Alisha Kodibagkar, Steven Meisler, François Paugam, Julio A. Peraza, Jean-Baptiste Poline, Patrick Sadil, Taylor Salo, Kevin Sitek, Maximilian Cosmo Sitter, Alexis Thual, Mohammad Torabi, Konrad Wagstyl, Hao-Ting Wang, Michelle Wang, Bertrand Thirion

Introduction

Nilearn is a widely recognized Python package in the neuroimaging community that offers a comprehensive set of statistical and machine learning tools for analysis of brain images. With over 10 years of ongoing development, it has reached 1000 stars, 576 forks, and over 200 contributors on GitHub, with clear impact as measured by its 163 citations in open access publications[1]. Its continuous growth is backed by its encouraging community, user-friendly API, and comprehensive documentation, solidifying its role as a crucial part of the neuroimaging open-source software ecosystem. Also, Nilearn effectively makes use of powerful Python machine learning libraries, particularly scikit-learn[2], which are extensively utilized by scientific and industrial experts.

Recent work in Nilearn has centered on developing a new API to allow users to seamlessly work with surface data in a manner similar to volumetric data, enhancing support for the General Linear Model (GLM), enhancing the BIDS interface, and improving and updating the infrastructure and codebase.

Methods

Nilearn is designed to be accessible for researchers and developers. The documentation (https://nilearn.github.io) comprises a comprehensive user guide and an illustrative example gallery, along with detailed contribution and maintenance guidelines. Moreover, we actively encourage community engagement by welcoming questions, bug reports, enhancement suggestions, and direct involvement in refining the source code. We use several channels of communication including Neurostars, GitHub, Discord, Twitter, and Mastodon for daily communication with both contributors and users.

Nilearn follows standard software development practices, including version control, unit testing, and rigorous reviews for contributions. Our automated continuous integration infrastructure ensures constant testing and updates for a streamlined development process. On this end, we have been further improving the code quality of and infrastructure around our codebase to meet best practices and ensure quality as the package grows.

Finally, Nilearn is actively showcased in various tutorials and workshops held annually, such as the OHBM Brainhack.

Results

Nilearn supports brain image manipulation, GLM-based analysis, predictive modelling, classification, decoding, and connectivity analysis. It also has tooling for visualizing volumetric and surface brain imaging data. In the latest release (v0.10.2)[3], an experimental surface API has been added to facilitate working with surface data in downstream surface-based analyses.

Key features from the release include the addition of LogisitcRegressionCV and LassoCV estimators to the Decoder module, the ability to compute fixed effects on F contrasts in the GLM module, and the option to enable radiological view for volumetric plotting functions (Fig1). A new surface API has also been released under an experimental module for rapid community feedback. It provides access to a SurfaceImage class that can store mesh and surface data for both hemispheres. This surface object can then be easily passed to other functions for further analysis or visualization (Fig2) mirroring Nilearn's interface for handling volumetric data.

Notable maintenance changes were implemented to improve codebase standardization and consistency, including the use of reformatting tools, such as Black and pre-commit for automatic linting and formatting. Ongoing efforts involve codebase restructuring based on new guidelines and writing reusable Pytest fixtures for efficient unit testing.

Conclusion

The growing reliance on Nilearn underscores its accessibility and utility within the neuroimaging community. Developing the surface API will allow us to better support cortical surface analyses and meet user needs. Further, as the package grows, improved infrastructure and keeping up with new standards will ensure the robustness of this tool.

References

[1] NiLearn (RRID:SCR_001362)

[2] 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.

[3] Nilearn contributors, Chamma, A., Frau-Pascual, A., Rothberg, A., Abadie, A., Abraham, A., Gramfort, A., Savio, A., Cionca, A., Thual, A., Kodibagkar, A., Kanaan, A., Pinho, A. L., Idrobo, A. H., Kieslinger, A.-S., Rokem, A., Mensch, A., Vijayan, A., Duran, A., … Nájera, Ó. (2023). nilearn (0.10.2). Zenodo. https://doi.org/10.5281/zenodo.8397157

Figures

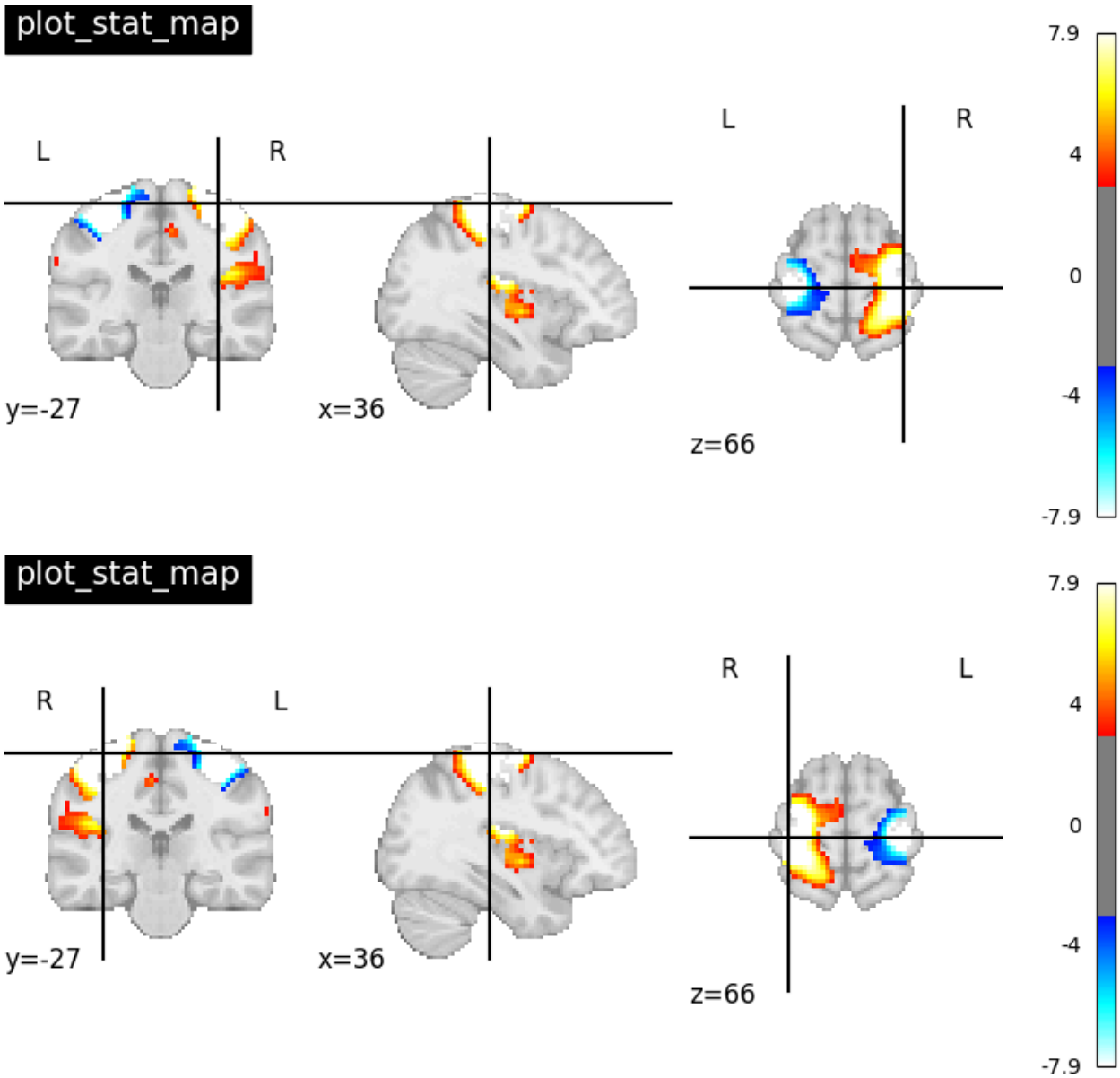

Fig1. Volumetric plotting functions now have a radiological parameter that can be set to True to invert the brain image and the L R labels on the x axis following radiological convention. In this example, the top image shows the default view and the bottom image shows the radiological view.

```
>>> img = surface.fetch_nki()[0]
>>> print(f"NKI image: {img}")
NKI image: <SurfaceImage (895, 20484)>

>>> print(img.data)  # output is truncated
{'left_hemisphere': array([[-0.4938, ..., ]], dtype=object),
'right_hemisphere': array([[-0.5474, ..., ]], dtype=object)}

>>> print(img.mesh)
{'left_hemisphere': <FileMesh with 10242 vertices>,
 'right_hemisphere': <FileMesh with 10242 vertices>}

>>> masker = surface.SurfaceMasker()
>>> masked_data = masker.fit_transform(img)
>>> print(f"Masked data shape: {masked_data.shape}")
Masked data shape: (895, 20484)

>>> first_data = masked_data[0]
>>> first_data = masker.inverse_transform(first_data)
>>> print(f"First timepoint: {first_data}")
First timepoint: <SurfaceImage (20484,)>

>>> plot_surf_img(first_data)
>>> plotting.show()
```

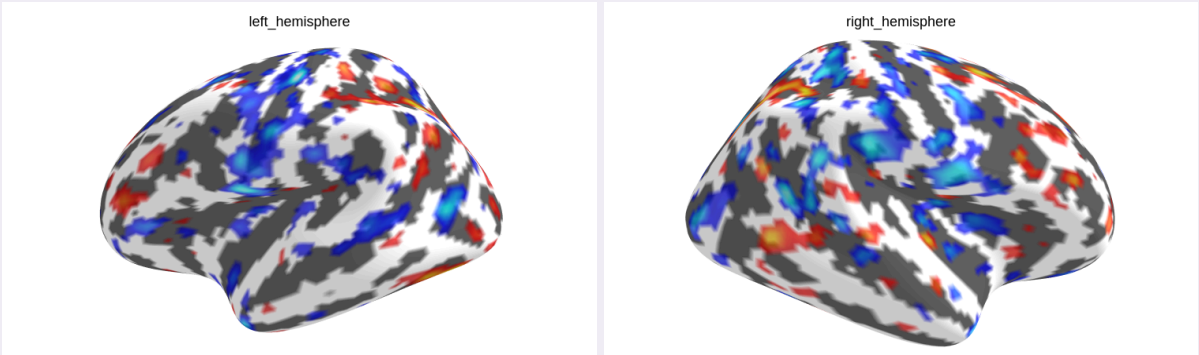

Fig2. Code snippet to demonstrate instantiating an object of the SurfaceImage class and passing it to a SurfaceMasker to perform signal extraction before plotting it with a function that accepts a SurfaceImage object. The SurfaceImage object can store surface data arrays for both hemispheres as well as Mesh objects for each hemisphere. The Mesh objects contain the number of vertices, the coordinates, and the faces for the surface mesh.