Title: Nilearn: Streamlined neuroimaging analysis now with surface data support

Authors: Rémi Gau, Himanshu Aggarwal, Alexis Thual, Elizabeth DuPre, Hande Gözükan, Hao-Ting Wang, Jérôme Dockès, Michelle Wang, Mohammad Torabi, Nicolas Gensollen, Taylor Salo, Victoria Shevchenko, Yasmin Mzayek, Bertrand Thirion

Introduction

Nilearn is a Python package that aims to simplify statistical analysis and machine learning on brain images, with an increasing user base and over 200 contributors on GitHub. It leverages scikit-learn's **[1]** established API design philosophy and extends its machine-learning models specifically for neuroimaging data.

Nilearn is widely recognized within the neuroscience research community, with 277 citations in open-access publications **[2]**. Our recently integrated support for surface data (i.e., GIfTI files), therefore, promises to be of significant interest to a wide portion of the neuroimaging community. Over the past year, surface data support has moved from an experimental feature to fully integrated across all modules, such as decoding, general linear modeling (GLM), and visualization.

In addition, Nilearn's documentation now consists of several guides and examples to help users interact with this new surface API. As Nilearn follows "documentation-driven development," this adoption provides opportunities for additional, significant advances in surface support in the Python neuroimaging open-source software ecosystem as additional user-stories are identified. Other highlights include improved reporting, enforcing code formatting rules via Ruff **[3]**, and making Nilearn's estimators more compliant with scikit-learn's guidelines **[4]**.

Methods

Nilearn is designed to be accessible to researchers and developers. The documentation comprises a comprehensive user guide and an illustrative example gallery tailored for specific research use cases, along with detailed contribution and maintenance guidelines.

We actively engage with the community via Neurostars, GitHub, Discord, X, and Bluesky and welcome questions, bug reports, enhancement suggestions, and direct involvement in refining the source code. Nilearn is also actively showcased in various tutorials and workshops held annually, such as the OHBM Brainhack and the Montreal Artificial Intelligence and Neuroscience (MAIN) Educational Workshop.

We follow standard software development practices, including version control, automated continuous integration infrastructure that ensures constant testing, and rigorous reviews for contributions.

Results

With over 10 years of ongoing development, Nilearn has reached more than 1200 stars, 600 forks, and is used by more than 200 packages on GitHub. Its continuous growth is evident via an increase in downloads over the years, reaching a peak of 600k in 2024 (**Fig.1**).

Nilearn now supports brain image manipulation, GLM-based analysis, predictive modeling, classification, decoding, connectivity analysis, and visualization of brain imaging data. In the latest release (v0.11.0) **[5]**, we extended all of these modules to work with surface data (i.e., GIfTI). This ensures that all the analyses that were previously only possible on volumetric data can now also be done on the surface representation of brain imaging data (**Fig. 2**).

Furthermore, now all the maskers – the module that allows users to extract brain signals from fMRI data – can produce reports that contain visualizations and other important information, providing an overview of the manipulations done during data processing.

Maintenance is critical for user trust in a software. We have thus continued to focus significant attention on code maintenance, including the adoption of Ruff formatting and linting rules to improve codebase standardization and consistency. In addition, we also extended our test suite to include scikit-learn's estimator checks to ensure that Nilearn's estimators comply with scikit-learn's recommendations.

Conclusion

The growing reliance on Nilearn underscores its accessibility and utility within the neuroimaging community. Future planned work includes extending further support for cortical surface analyses, including support of the CIFTI file format. In addition, as the package grows, improving the codebase to meet the new standards will ensure the robustness of this tool.

References

[1] 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.

[2] NiLearn (RRID:SCR_001362)

[3] https://docs.astral.sh/ruff/

[4] https://scikit-learn.org/stable/developers/develop.html

[5] Nilearn contributors, Chamma, A., Frau-Pascual, A., Rothberg, A., Abadie, A., Abraham, A., Gramfort, A., Savio, A., Cionca, A., Sayal, A., Thual, A., Kodibagkar, A., Kanaan, A., Pinho, A. L., Joshi, A., Idrobo, A. H., Kieslinger, A.-S., Kumari, A., Rokem, A., … Nájera, Ó. (2024). nilearn (0.11.0). Zenodo. https://doi.org/10.5281/zenodo.14259676

Figures

Fig. 1: Nilearn package downloads over the years via PyPI (or pip) since 2016.

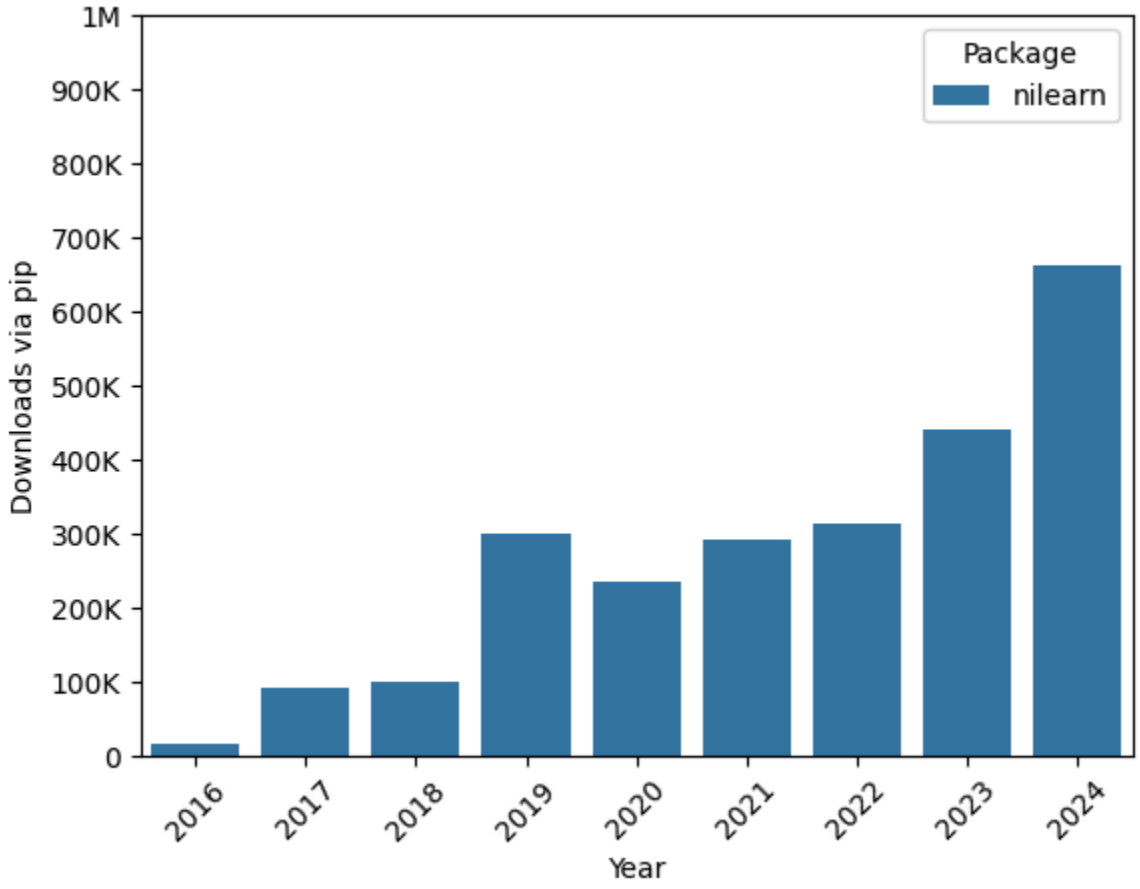

Fig. 2: Code snippet demonstrating loading of surface data, creating a SurfaceImage object that can then be used with SurfaceMasker to extract brain signals and finally for visualization.

```python
from nilearn import datasets, plotting, surface, maskers

# Fetch the fsaverage surface dataset
fsaverage = datasets.fetch_surf_fsaverage()
# Fetch the nki_enhanced surface dataset
nki_enhanced = datasets.fetch_surf_nki_enhanced()

# Define mesh files
mesh = {"left": fsaverage["pial_left"], "right": fsaverage["pial_right"]}
# Define data files
data = {"left": nki_enhanced["func_left"][0], "right": nki_enhanced["func_right"][0]}
# combine the mesh and data files in a SurfaceImage object
img = surface.SurfaceImage(mesh=mesh, data=data)

# Extract first subject data
masker = maskers.SurfaceMasker()
masked_data = masker.fit_transform(img)
first_subject = masked_data[0]
first_subject_img = masker.inverse_transform(first_subject)

# Plot the SurfaceImage object
plotting.view_surf(surf_map=first_subject_img)
```