Python for fMRI: A tour of key fMRI packages

March 10, 2019

Those who know me know that I am a huge fan of Python. I find Python to be an incredibly useful programming language for neuroscience and I prefer it to other languages for fMRI research (although I am not interested in engaging in any sort of "language war", see here for a nice discussion on that). Recent commentaries from Poldrack et al and Wessel et al, along with summer school programs aimed towards open source software in science, suggest that Python is growing in neuroscience. Anecdotally, I have seen more and more colleagues show interest in Python, with some making the full switch altogether.

So, I decided to write a blog post that introduces the Python fMRI landscape by describing some key packages. I hope people can find this useful if they're new to fMRI, new to Python, or both!

Python's Scientific Computing Ecosystem

First, just some background on Python in science. Scientific computing with Python is no new concept, but with the rise of data science over the last 10-15 years, and thanks to a number of powerful libraries developed within that time-frame, Python has surged in popularity in recent years for scientific computing and data analysis. Python for data analysis is typically discussed with reference to the "PyData stack", which includes packages such as *numpy* and *scipy*, *pandas*, *matplotlib*, *scikit-learn*, and *jupyter* (to name a few). There are an overwhelming number of resources on this topic, but the best quality resources I've found are the Scipy Lectures, Python Data Science Handbook and Python for Data Analysis (this last one is not freely available, but I still recommend it). If you're not already acquainted with the basics of Python's scientific computing ecosystem, I suggest diving into those resources first.

MRI analysis with Python has its own set of packages, many of which can be found at nipy.org. The packages I am discussing here are ones I use on a regular basis, so this post is definitely biased towards my use case (BOLD fMRI). Packages on *nipy* that are not mentioned here are obviously just as valuable, and I would encourage you to explore their documentation more in depth.

A typical project will involve fMRI-specific packages on top of the general scientific packages I mentioned earlier. While such sets of dependencies might sound foreign to more niche programming languages like MATLAB, this reflects the nature of Python being a general programming language. As well, having the ability to require specific several different projects or domains, because for each project you can only require what you need. Dependency/package management is a whole other topic, and I recommend reading more here and here if you are interested.

fMRI Packages

Nibabel

nibabel is a package for working with directly with NIfTI files and similar neuroimaging formats (e.g., GIFTI). *nibabel* allows you to read in an image file so you can access the data itself in the form of a *numpy* array, as well as the associated header information and affine matrix. A typical use-case for *nibabel* is to load an image, perform some sort of manipulation (e.g., creating a binary mask based on some threshold), and save the results back to a NIfTI image. Because the data is in a *numpy* array, you can conveniently to use whatever *numpy* functionality you want. All packages discussed here rely on *nibabel* in some form or another to read, process, and save fMRI data files, making *nibabel* an essential package for neuroimaging.

Nipype

nipype is a powerful Python interface for interacting with all major neuroimaging software. Your project might consist of several different software packages (e.g., FSL, SPM), which likely requires managing all sorts of scripts. *nipype* offers a solution to this often messy process by providing a single Python interface to join together tools from different software packages. With *nipype*, you essentially create a network of nodes, each of which are a Python class representing a step in your pipeline. *nipype* provides a syntax that allows you to map each node's inputs and outputs, like various parameters or data files, to one another in order to assemble your pipeline node-by-node.

There are two features I particularly enjoy about *nipype*. First, using *nipype* to glue together your pipeline means that you clearly describe your data flow under a unifying syntax. I can read a *nipype* script from top to bottom and immediately understand each step in the process – you could imagine that this is also helpful for general reproducibility purposes. Second, you can easily parallelize your entire pipeline using the multiproc plugin, which really only requires changing one line of code. So, for the exact same code/pipeline, you can cut down on processing time just by specifying how many processors you want to use. I have found this pretty convenient for longer processing tasks (e.g., spatial normalization).

Personally, I think *nipype* has a steep learning curve. To mitigate this, I highly recommend having solid grasp with whatever fMRI software you are using before transforming it into a Nipype pipeline. Also, there are plenty of resources available; Michael Notter's fantastic tutorial series are an absolute must-read when starting out, and the *nipype* documentation/tutorials and publication describe the software in great detail. There is a lot to learn, but *nipype* is definitely worth the investment if you regularly interact with existing fMRI software.

Nilearn

If I have to pick a favorite package discussed here, *nilearn* definitely wins. *nilearn* is a package that provides a number of functions to help perform a variety of multivariate techniques specifically on neuroimaging data. It complements *scikit-learn* (hence the name), the main machine-learning library in Python, and features a similar API design that is straightforward to use.

A number of analyses are possible with *nilearn*. For basic decoding analysis, you can just use one of *scikit-learn*'s classifiers (e.g., support vector machine) for voxel pattern classification. But if you want techniques more specific to fMRI, such as a searchlight or more powerful classifiers, *nilearn* directly provides those options. *nilearn* is also well-equipped for a variety of brain mapping and network approaches, such as functional connectivity analysis, independent component analysis, and clustering techniques for brain parcellation.

But what puts *nilearn* over the top is all of the utilities that make it easy to assemble a complete analysis. Once your data is minimally preprocessed (with a *nipype* preprocessing pipeline, for instance), it is easy to extract voxel/region data and apply the remaining preprocessing for whatever analysis you plan to run. *nilearn* offers ways to do this either by individual regions or by commonly-used brain atlases (many of which, by the way, are included with *nilearn*). On top of this, *nilearn* has a number of plotting functions that allow you to visualize your brain data, such as statistical maps (either in 3D volumetric space or on a 2D surface) and connectivity matrices.

For further reading, I recommend checking out *nilearn*'s user guide, which is an excellent general introduction to multivariate approaches as well. I also recommend reading the publication that discusses machine-learning analyses in neuroimaging with *scikit-learn* and the development of *nilearn*, which provides some additional context to the package.

Note: There is a another machine-learning library for neuroimaging called *PyMVPA*, which predates *nilearn*. I personally find *nilearn* + *scikit-learn* + *numpy/scipy* sufficient for all my needs thus far and haven't much of a chance to use *PyMVPA*. However, it does have some useful routines for specific use cases, such as cluster-thresholding searchlight maps and hyperalignment. I would encourage anyone to check it out if they might find it useful!

Nistats

For conventional mass-univariate analysis, you can use *nipype* to interact with whatever software you want (e.g., SPM). However, a recent package called *nistats* let's you model your fMRI data directly in Python. Running univariate analyses within Python is great if you want to extract voxel beta-estimates for subsequent analyses that you're also running in Python (e.g., a decoding analysis with *nilearn*). Moreover, *nistats* is a breeze if you already know *nilearn* because both packages are largely developed by the same contributors and share similar API design (much like with *scikit-learn*).

nistats has several nice features. It makes it easy to fully run first- and second-level analyses with your data, and like *nilearn*, it comes with useful additional functions. For instance, it provides a quick and easy way to threshold your statistical maps, extract cluster information, or convolve regressors with a hemodynamic response function.

As with any package undergoing development in its early stages, *nistats* might change significantly between versions and/or may be more susceptible to bugs. There is also a likelihood that *nistats* might combine with *nilearn* in the near future in the future. The development of *nistats* is very exciting, but I recommend keeping this all in mind if you are interested in using it now.

Pysurfer

The last package I want to discuss today is *pysurfer*, which lets you interact with Freesurfer to visualize your brain images on surfaces. The package features one major class, `Brain`, which easily lets you add data, regions of interest, contours, and more to your figure. *pysurfer* offers more flexibility than *nilearn* for plotting on 2D surfaces, which makes sense given that visualization is the main purpose. Together with *nilearn*, pretty much all of your common brain plotting tasks can be performed in Python, which ultimately lets you work with Python's plotting ecosystem (e.g., *matplotlib*, *seaborn*) to create fully reproducible figures for publication. Excellent examples of *pysurfer* being used "in the wild" are the codebases for Waskom et al 1 and 2.

Note: There are a few other cool other brain visualization libraries that are definitely worth checking out, like *pycortex* and *visbrain*.

Other Packages

There are several other *nipy* packages out there, including *dipy* for diffusion tensor imaging, *mindboggle* for brain morphometry, and *nitime* for time-series analysis. There are also tons of smaller projects, such as *NiBetaSeries* for beta-series correlation analysis, *atlasreader* for automatically extracting cluster information, and *neurodocker* for encapsulating your entire fMRI software environment within Docker. These are just some of what other packages exist; by no means is this an exhaustive list.

Conclusions

Altogether, Python provides a rich ecosystem for fMRI. Working with the above-mentioned fMRI packages on top of Python's general scientific ecosystem, it is possible to build any step in your fMRI analysis. Once your data is preprocessed with a *nipype* pipeline (or with *fmriprep*, a ready-to-use automated pipeline built with *nipype*), you can work with a variety of packages to analyze and visualize your data all within Python.

It is a pretty exciting time to be a neuroimager and/or a Python user given all the tools and how rapidly the ecosystem is expanding (as is Python's general scientific ecosystem). There are also lots of opportunities to get involved with existing packages or contribute your own project that others may find useful. All in all, I highly recommend exploring what Python has to offer for fMRI if you think it would benefit you and your fMRI research.