MINING BRAIN IMAGING DATA

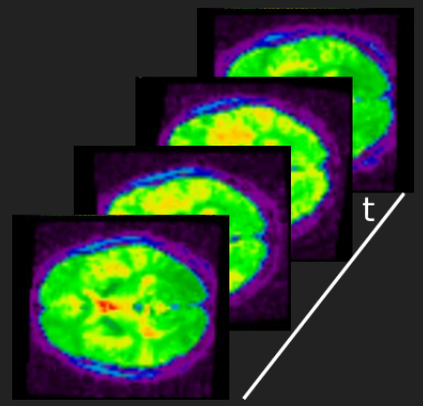
LESSONS LEARNED FROM NILEARN AND JOBLIB

LOÏC ESTÈVE

BIG DATA?

- Google: petabytes of data, distributed storage, map-reduce framework
- Neuroimaging larger datasets: ~100 GB to ~1TB
- Pragmatic approach: data that doesn't fit in RAM

NEUROIMAGING CONTEXT



3D volumes of brain activity along time

Why large datasets matter in neuroimaging:

- Diagnosis through large-scale functional connectivity studies
- More detailed and reliable mapping of brain functions

TALK OVERVIEW

Multi-variate statistical learning on HCP data (fMRI rest data ~2TB). See nilearn example for more details.

Demo @ Scipy 2015:

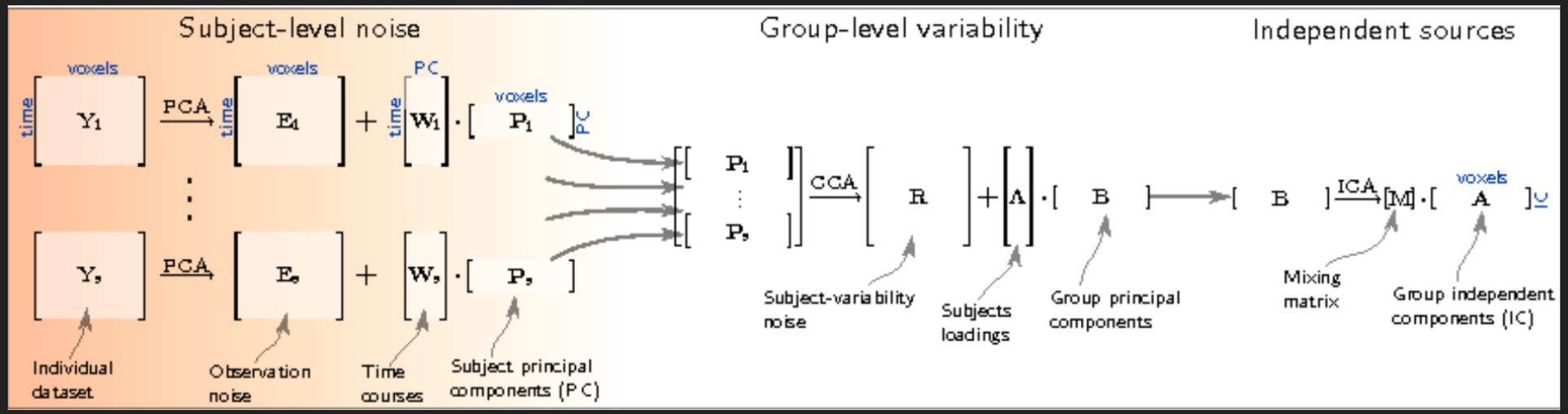
- ~140GB subset of the HCP data on my laptop
- process it with a simple script
- visualize the results in a way that makes sense for domain experts

During this talk, show a few Python patterns that make this possible

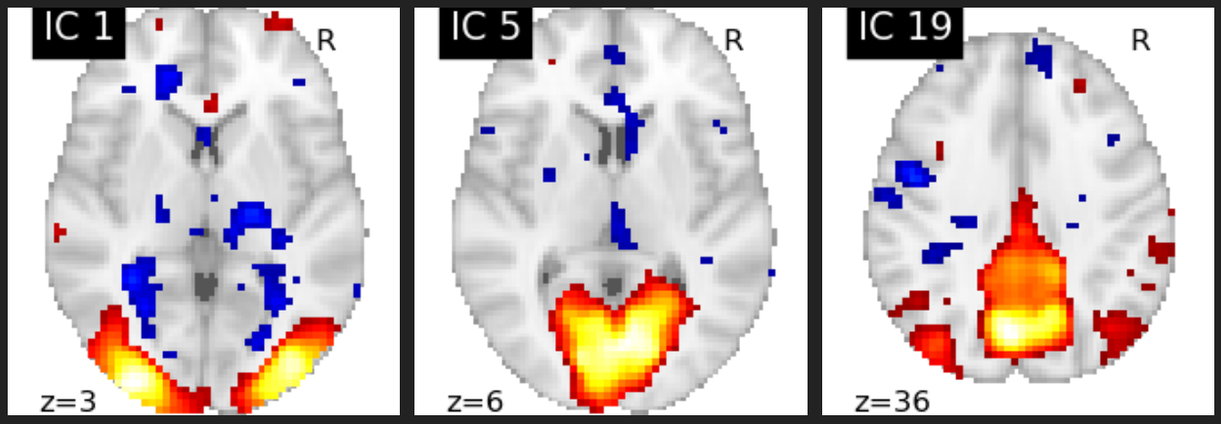
CANICA NILEARN EXAMPLE

```
### Load ADHD rest dataset ####################################################
from nilearn import datasets

adhd_dataset = datasets.fetch_adhd()
func_filenames = adhd_dataset.func

### Apply CanICA ##############################################################
from nilearn.decomposition.canica import CanICA

n_components = 20
canica = CanICA(n_components=n_components, smoothing_fwhm=6.,
                memory="nilearn_cache", memory_level=5,
                threshold=3., verbose=10, random_state=0)
canica.fit(func_filenames)

# Retrieve the independent components in brain space
components_img = canica.masker_.inverse_transform(canica.components_)

### Visualize the results #####################################################
import matplotlib.pyplot as plt
from nilearn.plotting import plot_stat_map
from nilearn.image import iter_img

for i, cur_img in enumerate(iter_img(components_img)):
    plot_stat_map(cur_img, display_mode="z", title="IC %d" % i,
                  cut_coords=1, colorbar=False)
```

CANICA OVERVIEW



- PCA on individual subject data
- concatenation of subject PCAs + group PCA
- ICA on group PCA



JOBLIB

CACHING

- Reproducibility: avoid manually-chained scripts
- Performance: avoid recomputing the same thing twice

joblib.Memory usage:

```
from joblib import Memory

mem = Memory(cachedir='.')

cached_func = mem.cache(func)
cached_func(a)  # computes func using a as input
cached_func(a)  # fetches result from store
```

- Fast input hashing, with optimization for numpy arrays
- Fast I/O for result persistence (joblib.dump/joblib.load)

PARALLEL COMPUTATION

- Focus on embarassingly parallel for-loops
- Both multiprocessing and threading backends

joblib.Parallel usage:

```
>>> from math import sqrt
>>> from joblib import Parallel, delayed
>>> Parallel(n_jobs=2)(delayed(sqrt)(i ** 2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

`n_jobs` parameter in scikit-learn estimators

In CanICA: used to parallelise subject PCAs

joblib.Parallel benefits

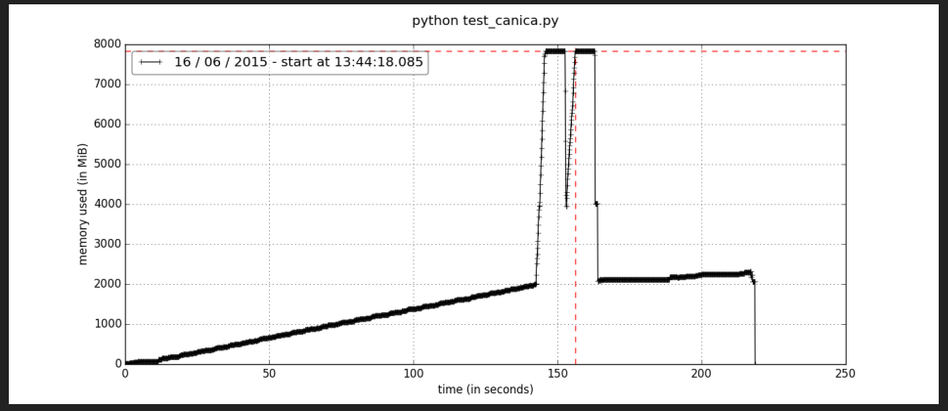
- high-level parallelism without a framework
- no dependencies (numpy optional)
- memmapping of input arrays allows them to be shared between worker processes
- New in 0.9.2 release: : automatic task-batching for multiprocessing backend (O. Grisel, V. Niculae). Useful for short tasks.

ONLINE LEARNING

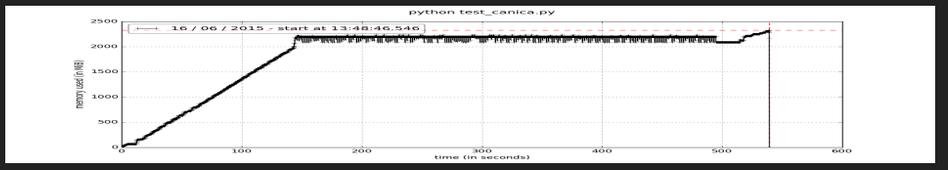
Compute the mean of a gazillion numbers, how hard?

Just do a running mean !

Upcoming in nilearn CanICA



RandomizedPCA



IncrementalPCA
(new in sklearn 0.16)

plots thanks to memory_profiler

HARDWARE ASPECTS

Parietal team biggish iron:

- 48 cores
- 384 GB RAM
- 70 TB storage (SSD cache on RAID controller)

Gets the job done faster than our 800 CPU cluster

Nobody ever got fired for using Hadoop on a cluster
A. Rowstron et al., HotCDP ’12


ni
learn

NILEARN USAGE OVERVIEW

- Easy access to open data

```
haxby_dataset = datasets.fetch_haxby()
```

- Massaging the data for machine learning

```
masker = NiftiMasker(mask_img='mask.nii',
                     standardize=True)
data = masker.fit_transform('fmri.nii')
```

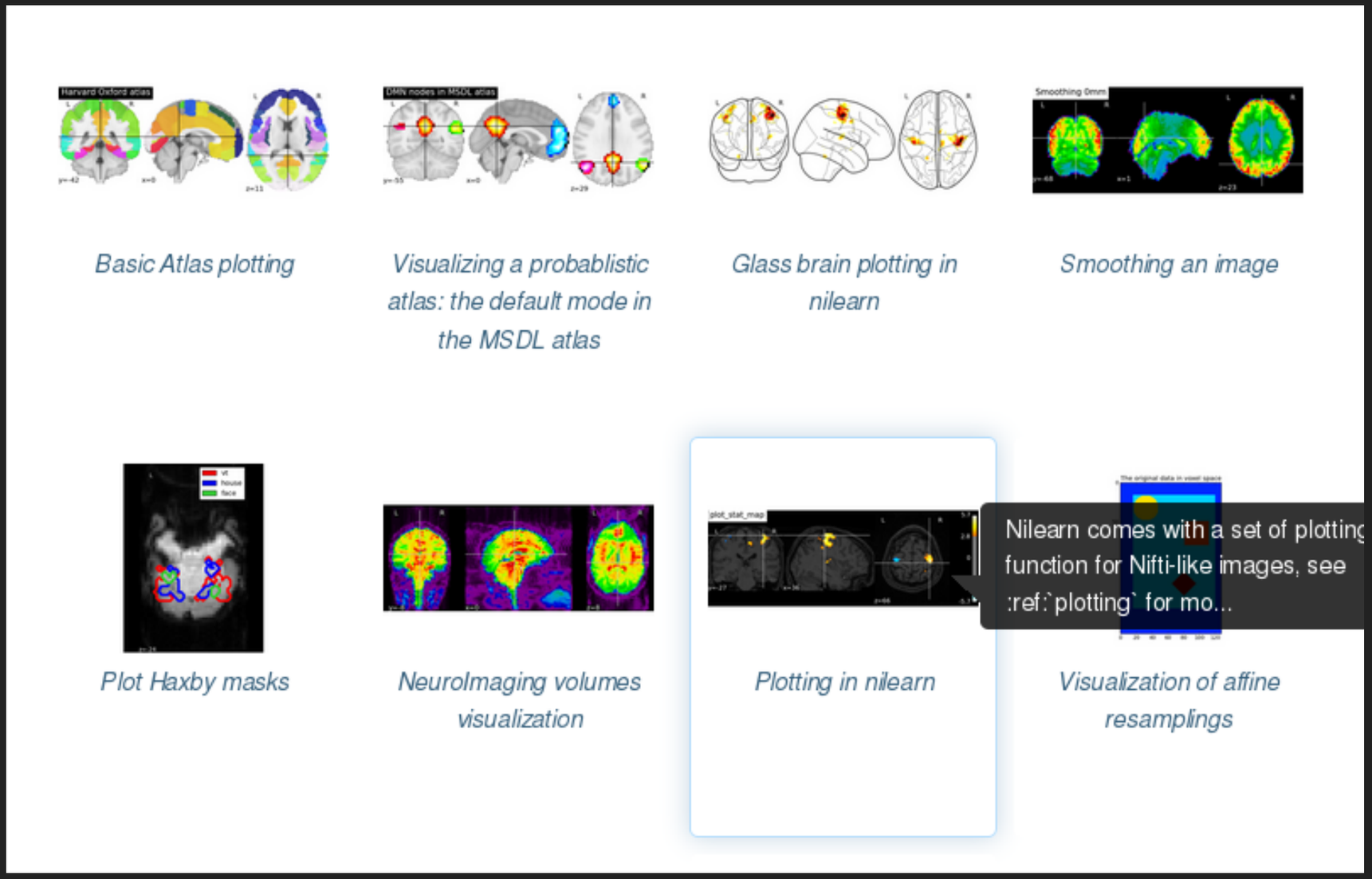
- Learning with scikit-learn: the "easy"

```
estimator.fit(data, labels)
```

- Visualise results

```
plot_stat_map(masker.inverse_transform(
              estimator.weights_))
```

NILEARN GALLERY EXAMPLES



http://nilearn.github.io/auto_examples/index.html

gallery examples generated via sphinx-gallery

FUTURE WORK

joblib

- cache replacement policy
- Improving persistence: joblib.dump/joblib.load (A. Abadie)

nilearn

- Robust connectivity estimation (S. Bougacha)
- Dictionary learning improvement wrt CanICA (A. Mensch)

CONCLUSION

Even if your data doesn't fit in RAM, you can process it

- with Python
- without necessarily using a distributed array/map-reduce framework

Simple Python patterns were presented to tackle such data

- caching (jolib.Memory) + parallel computing (joblib.Parallel)
- dimensionality reduction
- online learning

joblib: https://pythonhosted.org/joblib/
nilearn: machine learning for neuroimaging in Python
http://nilearn.github.io