Frontiers in Neurosciences 5July 2013

Machine Learning for Neuroimaging with Scikit-Learn

Alexandre Abraham [1,2,*], **Philippe Gervais** [1,2], **Fabian Pedregosa** [1,2], **Andreas Muller, Jean Kossaifi, Michael Eickenberg, Alexandre Gramfort, Bertrand Thirion** [1,2] **and Gaël Varoquaux** [1,2]

[1] *Parietal Team, INRIA Saclay-Île-de-France, Saclay, France*
[2] *Neurospin, I²BM, DSV, CEA, 91191 Gif-Sur-Yvette, France*

Correspondence*:
Alexandre Abraham
Parietal Team, INRIA Saclay-Île-de-France, Saclay, France,
alexandre.abraham@inria.fr

Research Topic

ABSTRACT

Statistical learning methods are increasingly used to perform neuroimaging analysis. Their main virtue for this type of application is their ability to model high-dimensional datasets, e.g. multivariate analysis of activation images, or capturing inter-subject variability. Supervised learning is typically used in decoding setting to relate brain images to behavioral or clinical observations, while unsupervised learning is typically used to uncover hidden structure in sets of images (e.g. resting state functional MRI) or to find sub-populations in large cohorts of subjects. By considering functional neuroimaging use cases, we illustrate how scikit-learn, a Python machine learning library, can be used to perform some key analysis steps. Scikit-learn contains a large set of statistical learning algorithms, both supervised and unsupervised, that can be applied to neuroimaging data after a proper preprocessing. Combined with other Python libraries, neuroimaging data can be loaded, processed and the results can be visualised easily.
Keywords: machine learning, statistical learning, neuroimaging, scikit-learn, python

1 INTRODUCTION

Python is being increasingly used in neuroimaging studies, Millman and Brett (2007); Hanke et al. (2009), which can be explained its versatility, its ability to act as a glue between different components and the maturity of its scientific stack.

1.1 SCIENTIFIC PYTHON AND NEUROIMAGING ECOSYSTEM

1.1.1 Scipy and Numpy SciPy and NumPy packages are the basis of scientific computing in Python. NumPy provides the `ndarray` data type, an efficient n-dimensional data representation. These vectors holds all common operations (transpose...) and can be easily manipulated thanks to a simple, yet powerful, indexing scheme. SciPy provides higher level mathematical functions that operate on ndarrays for a variety of domains including linear algebra, optimization and signal processing. Together, NumPy and SciPy provide a robust scientific environment for numerical computing and they are the elementary bricks we use in all our algorithms.

1.1.2 Matplotlib Matplotlib is a Python plotting library that is tightly integrated into the rest of the scientific python stack. It offers publication quality figures in a variety of formats and allows to display

plots, images or 3D plots in a graphical user interface. All figures in this paper have been generated using this library.

1.1.3 Nibabel Nibabel is a neuroimaging data loading package. Nibabel can load or save data in popular neuroimaging data format. This is indeed an entry point of all our scripts.

1.1.4 nipy

1.1.5 scikit-learn The *scikit-learn* project, Pedregosa et al. (2011), is an open source machine learning library for the Python programming language. The ambition of the project is to provide efficient and well-established machine learning tools within a programming environment that is accessible to non-machine learning experts and reusable in various scientific areas.

2 SCIKIT-LEARN CONCEPTS

All objects within scikit-learn share a uniform common basic API consisting of three complementary interfaces: an *estimator* interface for building and fitting models, a *predictor* interface for making predictions and a *transformer* interface for converting data into a different representation.

Interoperation with third-party code is made not on base on inheriting from a particular class but on the base of implementing the appropriate methods for a given interface.

The *estimator* interface is at the core of the library. It exposes a `fit` method for learning a model from training data. All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface. Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators.

The *predictor* interface extends the notion of an estimator by adding a `predict` method that takes an array `X_test` and produces predictions for `X_test`, based on the learned parameters of the estimator (we call the input to `predict` "`X_test`" in order to emphasize that `predict` generalizes to new data). In the case of supervised learning estimators, this method typically returns the predicted labels or values computed by the model.

Since it is common to modify or filter data before feeding it to a learning algorithm, some estimators in the library implement a *transformer* interface which defines a `transform` method. It takes as input some new data `X_test` and yields as output a transformed version of `X_test`. Preprocessing, feature selection and dimensionality reduction algorithms are all provided as transformers within the library. If the transformation can be inverted, a method called `inverse_transform` also exists.

scikit-learn offers more than bla bla bla

2.1 MODEL SELECTION AND CROSS VALIDATION

It seems more right to me to put it in this part

In the scikit learn and often in statistical machine learning, data is represented in a 2-dimensional matrix of shape $n\_samples \times n\_features$.

3 FROM MR VOLUMES TO A DATA MATRIX

As any domain specific data, MR volumes holds particular properties. Understanding them is crucial to be sure to make proper use of the data.

3.1 TRANSFORMATION MATRIX

Neuroimaging data are represented in a 4D numpy array (3 dimensions + time) and a transformation matrixi (called affine). This affine is used to transpose the data in its original space.

A simple way to apply it is to work with coordinates. One can simply go from numpy indexing `array[i, j, k]` to real coordinates $(x, y, z) \in \mathbb{R}^3$.

$$\begin{bmatrix} r_x & 0 & 0 & o_x \\ 0 & r_y & 0 & o_y \\ 0 & 0 & r_z & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Here is a simple function that gather voxel coordinates in a mask and transpose them into the original space:

```
def get_real_coordinates(mask, affine):
    # get coordinates of non zero voxels
    mask_coords = np.where(mask != 0)
    # add an extra dimension for the offset
    mask_coords = mask_coords + (np.ones(len(mask_coords[0]), dtype=np.int),)
    # apply the transformation
    mask_coords = np.dot(affine, np.asarray(mask_coords))[:3].T
    return mask_coords
```

3.2 DATA PREPARATION

At this point, we suppose that standard preprocessings have been applied to the data (motion correction, slice timing, coregistration on a common template like MNI if necessary). However, data is not yet ready to be processed by the scikit-learn. In fact, preprocessed data may have different shapes. Moreover, it is essential to get rid of some remaining scanner artefacts and individual trends.

3.2.1 Detrending Detrending is an essential step when dealing with fMRI data. It removes a linear trend over the time series of each voxel. It is needed when you want to study the correlation between features.

```
scipy.signal.detrend(data)
```

3.3 RESAMPLING

Resampling consists in changing the spatial resolution of the data. This is typically needed when dealing when data coming from an heterogenous dataset, as shape depends on acquisition parameters.

Resampling is an interpolation and thus may alter the data integrity. That is why it should be used carefully. Oversampling (increasing data resolution) leads to higher memory consumption and computation resources with no gain in information. Downsampling is commonly used to reduce the size of the data to process.

Typical sizes are 2mm or 3mm resolutions, but the spread of high field MR scanner tends to lower these values.

3.4 SIGNAL CLEANING

- Remove high frequency (scanner artifacts)
- Removing confounds is necessary for some processings

3.5 MASKING

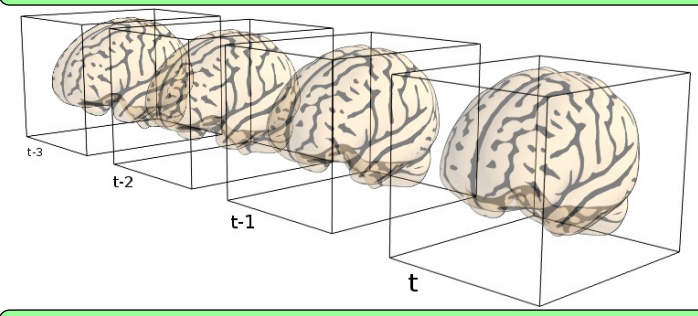
3.5.1 From 4-dimensional image to 2-dimensional array Neuroimaging data are represented in 4 dimensions: 3 dimensions for the scans, which are positioned in a coordinate space, and one dimension for the time. Scikit-learn algorithms, on the other hand, only accept 2-dimensional data: one dimension for the features and one for the samples.

Consequently, in order to use neuroimaging data in the scikit-learn, a conversion is needed. The most simple way to achieve that would be to *flatten* the 3D scans into a 1D array. However, we know that not

every voxels in a neuroimaging scan is useful. In particular, outter-brain voxels are of no use and, worse, they can bring spurious noise and scanner artefacts (such as ghosts).

To sort out voxels of interest, we will have to apply a mask on the data. Most of public datasets provide a mask, come of them even provide several, isolating different functional or anatomical brain regions.

ref to Haxby



Should tell here that some algorithms, like logistic regression, do not like colinear features.

3.5.2 Automatically computing a mask The simplest strategy to compute a mask is a binarization by a selected threshold. Due to the nature of the neuroimaging data, there exists some strategy to choose this threshold in order to obtain a decent segmentation.

There is a reference for the method used in Nisl. We should put it there and in the code. Add a figure with an histogram to illustrate.

Multi subject computation is simply done by intersecting subjects maps relatively to a chosen threshold.

3.5.3 Preserving geometrical structure Applying a mask on the data obviously remove the 3-dimensional structure of the data. However, some algorithms, like the Ward, need this structural information. `sklearn.feature_extraction.image` provides two methods that builds an adjacency matrix based upon your data while taking the mask into account:

- `grid_to_graph` creates a binary adjacency graph based upon the data shape. This is useful for Ward's clustering.
- `image_to_graph` creates a distance matrix using the gradient of the image. This graph can be used in Spectral clustering.

3.6 LABEL SHIFTING

Functional MRI measures brain activity by using the Blood-Oxygen-Level-Dependent contrast (BOLD). In fact, like muscles, brain regions consume more oxygen and nutriments when stimulated. So when a part of the brain starts working, physiological mechanisms induce an oxygen-rich blood flood toward this particular region: this is called haemodynamic response.

However, this reaction takes time, usually around 3-4 seconds. This is the duration between the event and the reaction observed in the brain. To be able to match these two events, we will sometimes have to shift our data. The number of scans that must be shifted depends on the TR (repetition time) of the data. Usually, we remove the two first scans of the data and the two last values of the labels (to keep an homogeneous length).

```
data = data[2:]
labels = labels[:-2]
```

4 DECODING

The process of predicting behavioral or phenotypic data from fMRI scan is called decoding.

4.1 SVM

4.1.1 Haxby dataset For this example and the following, we will use the Haxby dataset (Haxby et al. (2001)). The Haxby dataset is from a study about face and object representation into the brain (in particular in high-level visual cortex). It is composed of 12 runs for each of the 6 subjects. Grayscale images representing faces, houses, cats, bottles, scissors, shoes, chairs and random textures were presented in 24-second blocks separated by rest periods. The repetition time (TR) between each scan is 2.5s. Full acquisition information are available in the reference paper.

To make this example easier, we will work on a subset of this dataset. We will consider only one subject and will classify faces versus houses.

4.1.2 Feature selection: ANOVA F-Test Even if the resolution of brain-imaging data seems low (3mm cubes, around 100000 neurones), from a computational point of view, this is huge. For example, Haxby dataset has a resolution of $64 \times 64 \times 40 = 163840$ voxels. After applying the mask, only 39912 voxels are left, which is still high.

In order to reduce the number of features, we can aggregate them (in regions of interest for example) or we can select only the most relevant ones (those who correlates most with the task). As we expect a lot of feature to be irrelevant for our task, we opt for a feature selection method.

In supervised learning, the most popular feature selection method is the ANalysis Of VAriance (ANOVA) F-Test. This is a generalization of the t-test to more than 2 features. ANOVA compares several groups to determine if they are similar (i.e. randomly drawn from the same population, this is the null hypothesis). We use it to compare the distribution of feature values across classes.

`sklearn.feature_selection` contains a panel of feature selection strategies. One can choose to take a percentile of the features (`SelectPercentile`), or a fixed number of features (`SelectKBest`) for example. All these objects are implemented as transformers. Here we use a fixed number of features and we use the `f_classif` function (ANOVA F-Test) for scoring.

```
from sklearn.feature_selection import SelectKBest, f_classif

### Define the dimension reduction to be used.
# Here we use a classical univariate feature selection based on F-test,
# namely Anova. We set the number of features to be selected to 500
feature_selection = SelectKBest(f_classif, k=500)
```

4.1.3 Classification A Support Vector Classifier (SVC) is a simple classifier that finds a linear hyperplane that separates the samples. Classifying a new sample boils down to seeing on which side of the hyperplane the example is. SVC has the advantage to give reliable results even when the number of dimensions is greater than the number of samples.

The decision is taken based upon a subset of training data called support vectors. We can say that these support vectors hold the information allowing to discriminate the two classes, this is why we will display them and try to see if they match some neuroscientific knowledge.

```
from sklearn.svm import SVC
clf = SVC(kernel='linear', C=1.)
clf.fit(?)

### Look at the discriminating weights
svc = clf.support_vectors_
# reverse feature selection
svc = feature_selection.inverse_transform(svc)
```

4.1.4 Pipeline The workflow described above (feature selection + estimator) is a standard one. In fact, in most cases, the workflow will consist in atomic steps *linked* together (the output of a step is the input of the next one). For this purpose, scikit-learn offers a pipeline object that allows such linking. A pipeline is simply a list of scikit-learn objects through which the input data will be conveyed. The function to call for each object (transform, fit...) depends on its type. This allow the developpers to write a complete processing as a one-liner.

```
from sklearn.pipeline import Pipeline
anova_svc = Pipeline([('anova', feature_selection), ('svc', clf)])
```

4.1.5 Displaying the results

Should we define a visualization function once and for all?

4.2 SEARCHLIGHT

For this example, we use the same dataset as the first one (one subject from Haxby).

4.2.1 Principle This Searchlight algorithm presented in **?** aims to score brain voxels depending on the amount of information they hold. The underlying principle is the same as the previous example: each voxel in the brain is used to predict the labels.

However, considering the fact that the information of a voxel is not only contained in the voxel itself but in its neighbours, Kriegeskorte decided to run a whole classification pipeline (learning and cross validation) in a ball centered on the voxel, instead of applying a smoothing.

4.2.2 Implementation Even if this algorithm seems complicated, it can be easily implemented thanks to some basic bricks provided by the scikit learn. They key elements here are *i)* the neighbourhood (`sklearn.neighobors.NearestNeighbors`), *ii)* the classifier (`sklearn.svm.LinearSVC`), *iii)* the cross-validation (`sklearn.svm.LinearSVC`).

As Searchlight is computationnaly expensive, we will add one constraint to our code: it should be able to run on part of the mask. Thus we will have the traditionnal `mask` and a `process_mask` that restraint computation only to some voxels (in our case, a half axial slice).

First, we compute an adjacency graph between voxels. We will take the neighbourhood of the voxels in the `process_mask`, based on the connectivity informations contained in the `mask`. This can be done by fitting a `NearestNeighbours` on the `mask` and applying it on `process_mask`.

TODO: reformulate

```
    # Compute world coordinates of all in-mask voxels.
    mask_coords = get_real_coordinates(mask, mask_affine)

    # Compute world coordinates of all in-process mask voxels
    process_mask_coords = get_real_coordinates(process_mask,
                                               process_mask_affine)

    clf = neighbors.NearestNeighbors(radius=radius)
    A = clf.fit(mask_coords).radius_neighbors_graph(process_mask_coords)
    del process_mask_coords, mask_coords
    A = A.tolil()

    # Mask 3D data
    X = nifti_file.get_data()[mask].T
```

Then, for each row of our adjacency matrix, we do a cross validation using a given estimator. This is the core of the algorithm and we see that it can be written with a simple for-loop.

```
scores = np.zeros(len(A.rows))
for i, row in enumerate(A.rows):
    print '%d/%d' % (i, len(A.rows))
    scores[i] = np.mean(
            cross_val_score(estimator, X[:, row], y, score_func=score_fun
                                cv=cv, n_jobs=1))
scores_3D = np.zeros(process_mask.shape)
scores_3D[process_mask] = scores
return scores_3D
```

4.3 CLASSIFICATION OF M/EEG SENSOR SPACE DATA

4.4 ORTHOGONAL MATCHING PURSUIT

4.4.1 Kamitani dataset Kamitani dataset is based on a visual task like Haxby. In this experiment, several series of 10×10 binary images are presented to two subjects. Our goal will be to use the scikit-learn to learn a correlation between brain activation and pixel color. Full details are available in the reference paper (Miyawaki et al. (2008)).

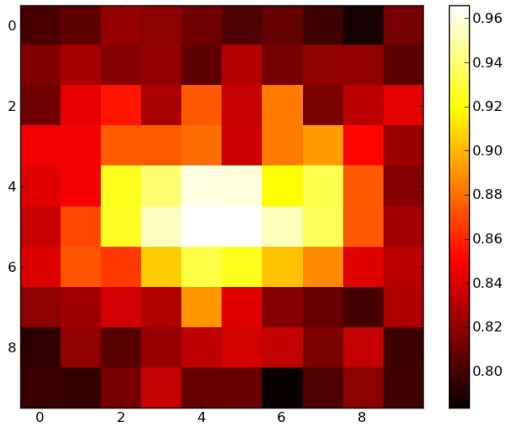
Kamitani training set is composed of random images (where black and white pixels are balanced). The testing set is composed of structured images containing geometric shape (square, cross...) and letters (spelling the word *neuro*).

There are two ways to establish a link between brain voxels and image pixels: we can either try to reconstruct image from brain voxel activation, this is called decoding, or we can try to predict brain activation from an image, this is called encoding.

In the present example, we will do both encoding and decoding and see if the results match.

4.4.2 Preprocessing Common pre-treatment have been applied to the data (detrending and standardization).

4.4.3 Decoding: reconstructing image from brain activity In the original paper (Miyawaki et al. (2008)), Miyawaki uses a sparse multinomial logistic regression to reconstruct the image.



5 ENCODING

After talking with Michael, he told me that he could make a fairly simple example for encoding, which I think is a plus for the paper. The example will be integrated in Nisl.

6 FUNCTIONAL CONNECTIVITY

In Biswal et al. (1995), Biswal was the first to exhibit coherent patterns in the brain activation during resting state. These correlated voxel activations seemed to form functional networks concording with neuroscientific knowledge.

Resting-state fMRI is useful when dealing with subjects that cannot execute a specific task. In fact, no task related protocol is capable of diagnosing precisely damaged brain areas in stroke patients, as no task can highlight these problems. Varoquaux et al. (2010a) exhibits differences in correlation between functional networks between control and post-stroke patients.

As resting state fMRI are unlabeled data, we have to use unsupervised methods to mine them. The most popular method to extract functional networksis the ICA and is the subject of our first use case. We will then show results obtained with clustering methods as several efficient implementations are available in scikit-learn.

The main problem with functional connectivity (and unsupervised methods in general) is the lack of labels or ground truth. It is therefore very difficult to score the computed models and find cross validate it. A first way would be compare your results with some reference atlas or neuroscientific knowledge. In fact, it is believed that functional network should match anatomical division

Well, I believe that. Any reference ?

. It is also possible to relate to networks discovered in task or other resting state experiments (eg default mode network, visual and ausitory networks...). Some researchers even released functional atlases built upon one of their experiment (cite yeo, smith, craddock).

6.1 INDEPENDENT COMPONENT ANALYSIS (ICA)

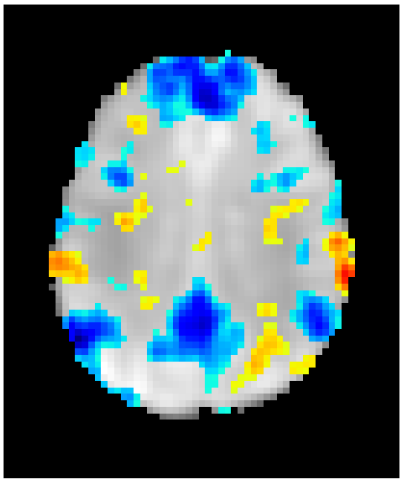
6.1.1 Principle ICA is a blind source separation method. Its principle is to separate a multivariate signal into several components by maximizing their non-gaussianity. A typical example is the *cocktail party problem* where ICA separates the voices of people using signal from several mikes.

It is historically the reference method to extract networks from resting state fMRI Biswal and Ulmer (1999). Several strategies have been used to syndicate ICA results across several subjects. Calhoun et al. (2001) proposes a dimension reduction (using PCA) followed by a concatenation of the time series. Varoquaux et al. (2010b) uses dimension reduction and canonical correlation analysis to aggregate subject data.

6.1.2 Preprocessing Scikit-learn FastICA takes care of whitening of the data, so there is no need to standardize it in our preprocessing. However, we still detrend it because FastICA does not look for linear trends, only constant one.

6.1.3 Application

6.1.4 Postprocessing A thresholding on the maps is necessary.

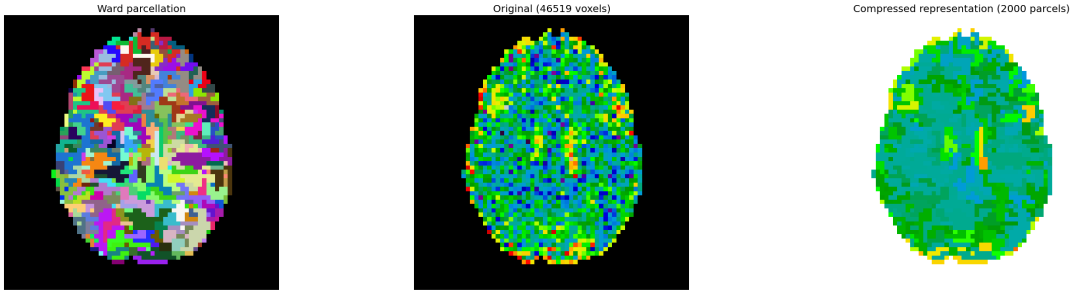


6.2 CLUSTERING

Make an example with Ward Clustering. Indicate then that other algorithms can be used such as KMeans and Spectral clustering and only give results.

We use a PCA here to reduce dimensionality.

Bonus: may be used as dimensionality reduction



DISCLOSURE/CONFLICT-OF-INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ACKNOWLEDGEMENT

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text.

Funding: Text Text Text Text Text Text Text Text.

SUPPLEMENTAL DATA

Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text.

REFERENCES

Millman, K. J. and Brett, M. (2007) Analysis of functional magnetic resonance imaging in python. *Computing in Science & Engineering* 9 52–55.

Hanke, M., Halchenko, Y. O., Sederberg, P. B., Hanson, S. J., Haxby, J. V., and Pollmann, S. (2009) Pymvpa: A python toolbox for multivariate pattern analysis of fmri data. *Neuroinformatics* 7 37–53.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011) Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 2825.

Haxby, J. V., Gobbini, I. M., Furey, M. L., Ishai, A., Schouten, J. L., and Pietrini, P. (2001) Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science* 293 2425.

Miyawaki, Y., Uchida, H., Yamashita, O., Sato, M.-a., Morito, Y., Tanabe, H. C., et al. (2008) Visual image reconstruction from human brain activity using a combination of multiscale local image decoders. *Neuron* 60 915–929.

Biswal, B., Zerrin Yetkin, F., Haughton, V., and Hyde, J. (1995) Functional connectivity in the motor cortex of resting human brain using echo-planar MRI. *Magn Reson Med* 34 53719.

Varoquaux, G., Baronnet, F., Kleinschmidt, A., Fillard, P., and Thirion, B., Detection of brain functional-connectivity difference in post-stroke patients using group-level covariance modeling. *MICCAI* (2010a). 200–208.

Biswal, B. and Ulmer, J. (1999) Blind source separation of multiple signal sources of fMRI data sets using independent component analysis. *Journal of computer assisted tomography* 23 265.

Calhoun, V. D., Adali, T., Pearlson, G. D., and Pekar, J. J. (2001) A method for making group inferences from fMRI data using independent component analysis. *Hum Brain Mapp* 14 140.

Varoquaux, G., Sadaghiani, S., Pinel, P., Kleinschmidt, A., Poline, J. B., and Thirion, B. (2010b) A group model for stable multi-subject ICA on fMRI datasets. *NeuroImage* 51 288.