skbold Documentation

Release 0.2.0

Lukas Snoek

April 12, 2016

CONTENTS

Contents:

CHAPTER

ONE

SKBOLD - UTILITIES FOR MACHINE LEARNING ANALYSES ON BOLD-FMRI DATA

Functional MRI (fMRI) data has traditionally been analyzed by calculating average signal differences between conditions. In the past decade, however, pattern-based type of analyses have become increasingly popular. Especially machine-learning based analyses experience a surge in popularity among (cognitive) neuroscientists.

While many great resources for domain-general machine learning exists (e.g. scikit-learn, caret, and libsvm), few resources are available specifically for machine learning analyses of neuroimaging data (but see nilearn).

As my PhD involved mainly machine learning analyses of fMRI data, I decided to bundle my (relevant) code into this package, which provides a nice opportunity for me to develop my programming skills by forcing me to write concise, readable, and efficient code.

The skbold-package contains mostly extensions and utilities for machine learning analyses of fMRI data. Its structure/setup draws heavily upon the *scikit-learn* (sklearn, hence the name) machine learning library in Python. Also, credit should be given to this repository, as it has a similar purpose and served as an example for much of my code.

1.1 Installing skbold

Although the package is very much in development, it can be installed using *pip*:

```
pip install skbold
```

However, the pip-version is likely behind compared to the code on Github, so to get the most up to date version, use git:

```
pip install git+https://github.com/lukassnoek/skbold.git@master
```

Or, alternatively, download the package as a zip-file from Github, unzip, and run:

```
python setup.py install
```

1.2 Functionality

Currently, the package contains the following features.

- A class that transforms FSL first-level directories into observation X feature arrays;
- Classes that provide scikit-learn style *transformers*;
- Some custom classifiers (voting- and stacked generalization classifiers).

Below, some basic examples are given.

1.2.1 Generating observation X feature arrays

The class `Mvp` (from the `core.py` module) functions as a container for the objects used in the skbold-package. This object is subclassed to provide methods that allow to generate and load in data. Currently, the only implementation is `Fsl2mvp` (see `data2mvp` module), which converts first-level GLM estimates from an FSL FEAT directory to an array of observations X features. Importantly, this function assumes that a single-trial design is used (i.e. each trial is modelled as a separate regressor) and that the analysis is done within subjects (as opposed to between subjects, with each subject as an 'observation').

For example, given a FEAT directory, the following code would create a trial X voxel array (stored as an Mvp-object; more on this later).

```python
from skbold.data2mvp import Fsl2mvp

# Specify arguments
firstlevel_dir = '/home/user/data/subject_001/firstlevel_dir.feat'
mask_path = '/path/to/mask' # to index the patterns by, e.g., grey-matter mask
mask_threshold = 0 # lower bound for probabilistic masks
beta2tstat = True # convert beta estimates to t-values
ref_space = 'epi' # load patterns in native functional-space (alternative: mni-space)
remove_class = ['condition2', 'condition3'] # conditions to ignore

# Initialize converter
fsl2mvp = Fsl2mvp(directory=firstlevel_dir, mask_path=mask_path, mask_threshold=mask_threshold,
                  beta2tstat=beta2tstat, ref_space=ref_space, remove_class=remove_class)

# Transform directory
fsl2mvp.glm2mvp()
```

Calling the method `glm2mvp()` creates a directory *mvp_data* with a data-file (hdf5) and a corresponding header-file (cPickle).

Alteratively, there is a command line function `glm2mvp` that has the same functionality as outlined in the example above:

```
$ glm2mvp -h
  usage: glm2mvp [-h] [-d DIRECTORY] [-m MASK] [-t THRESHOLD] [-b BETA2T]
         [-s SPACE] [-r [REMOVE [REMOVE ...]]]

$ cd /home/users/data/sub002
$ glm2mvp -d pwd -b True -s epi
```

1.2.2 Loading Mvp objects

To load Mvp-objects, the class DataHandler from the `utils` module can be used. An example is given below:

```
from utils import DataHandler

# Arguments
directory = '/home/user/data/subject_001 # assumes the existence of mvp_data dir!

# Initialize object
loader = DataHandler()

# Load data!
mvp = loader.load_separate_sub(sub_dir=directory)
```

The loaded Mvp-object contains all the necessary data and meta-data necessary for a proper machine learning analysis using scikit-learn.

1.2.3 Structure of Mvp-objects

The Mvp class contains the following main attributes:

- `X` : numpy-ndarray of length = [n_samples, n_features]. This contains the actual patterns!
- `y` : list, containing the target class as numeric labels.

Other useful metadata is stored in the following attributes:

- `mask_index` : index applied to the original whole-brain data
- `mask_shape` : shape of original mask, most likely MNI152 (2mm) shape (91 * 109 * 91)

1.2.4 Transforming data using transformer-classes

A major part of the skbold-package is the `transformers` module, which contains scikit-learn style `transformer`-objects that adhere to the consistent scikit-learn API, using the same `.fit()` and `.transform()` methods. The major advantage of directly inheriting from scikit-learn's Transformer objects is that they can be seamlessly integrated in Pipelines and gridsearch procedures.

In the following example, we'll create a scikit-learn pipeline to extract the patterns from only a single brain region from the whole-brain data contained in mvp.X (using the `RoiIndexer` transformer) and perform a type of univariate feature selection based on the average euclidean distance between classes (using the `MeanEuclidean` transformer).

```python
from utils import DataHandler
from transformers import RoiIndexer, MeanEuclidean
from sklearn.pipeline import Pipeline

loader = DataHandler()
mvp = loader.load_separate_sub('/home/user/data/subject_001')

mask = 'Frontal_pole.nii.gz' # masks are included in skbold!
roiindexer = RoiIndexer(mvp=mvp, mask=mask, mask_threshold=0)
mean_euclidean = MeanEuclidean(cutoff=2)

# You could sequentially transform the data, as such:
X_tmp = roiindexer.fit(mvp.X).transform(mvp.X)
X_final = mean_euclidean.fit(X_tmp, mvp.y).transform(X_tmp)

# Or you could use a pipeline!
pipeline = Pipeline([('roiindex', roiindexer), ('meaneuc', mean_euclidean)])
X_tmp = pipeline.fit_transform(mvp.X, mvp.y)
```

1.2.5 License and contact

The code is BSD (3-clause) licensed. You can find my contact details at my Github profile page.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search