skbold Documentation

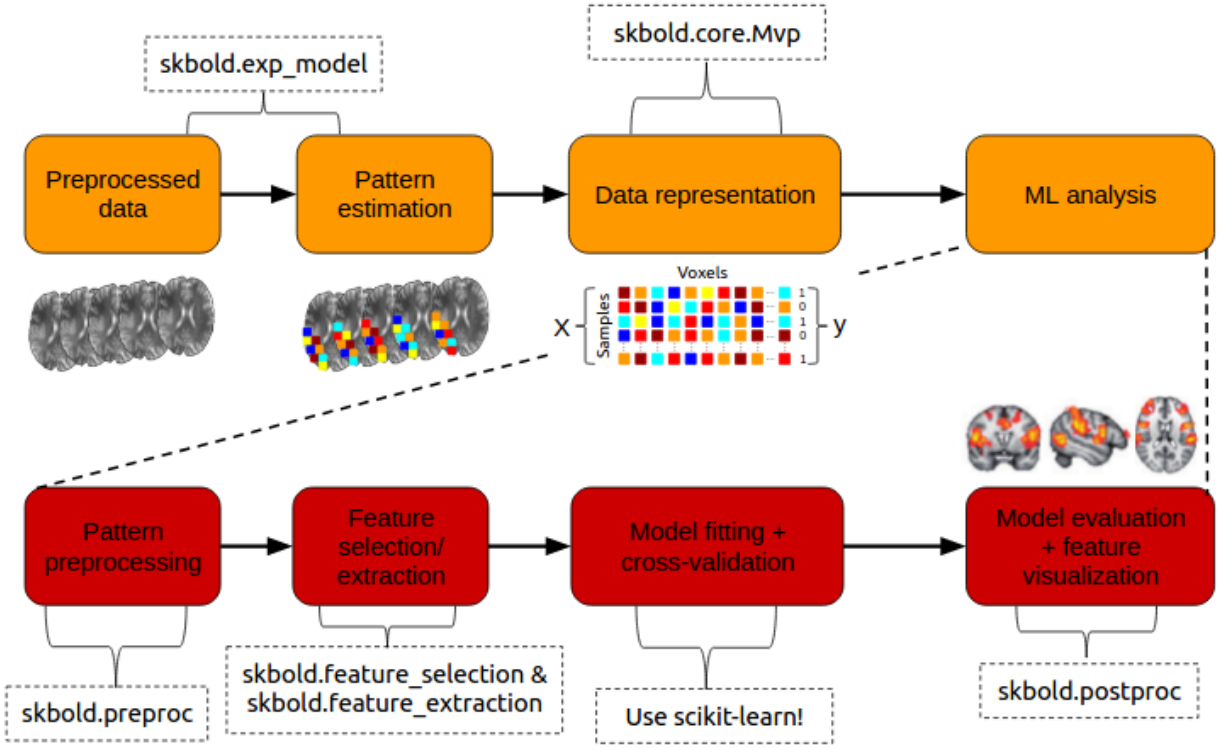
Release 0.1

Lukas Snoek

Jul 13, 2017

Contents

The Python package `skbold` offers a set of tools and utilities for machine learning analyses of functional MRI (BOLD-fMRI) data. Instead of (largely) reinventing the wheel, this package builds upon an existing machine learning framework in Python: scikit-learn. The modules of skbold are applicable in several 'stages' of typical pattern analyses (see image below), including pattern estimation, data representation, pattern preprocessing, feature selection/extraction, and model evaluation/feature visualization.



In what follows, we quickly summarize the main functionality of skbold. For more information (e.g. API documentation) and examples, check out the code documentation at ReadTheDocs!

CHAPTER 1

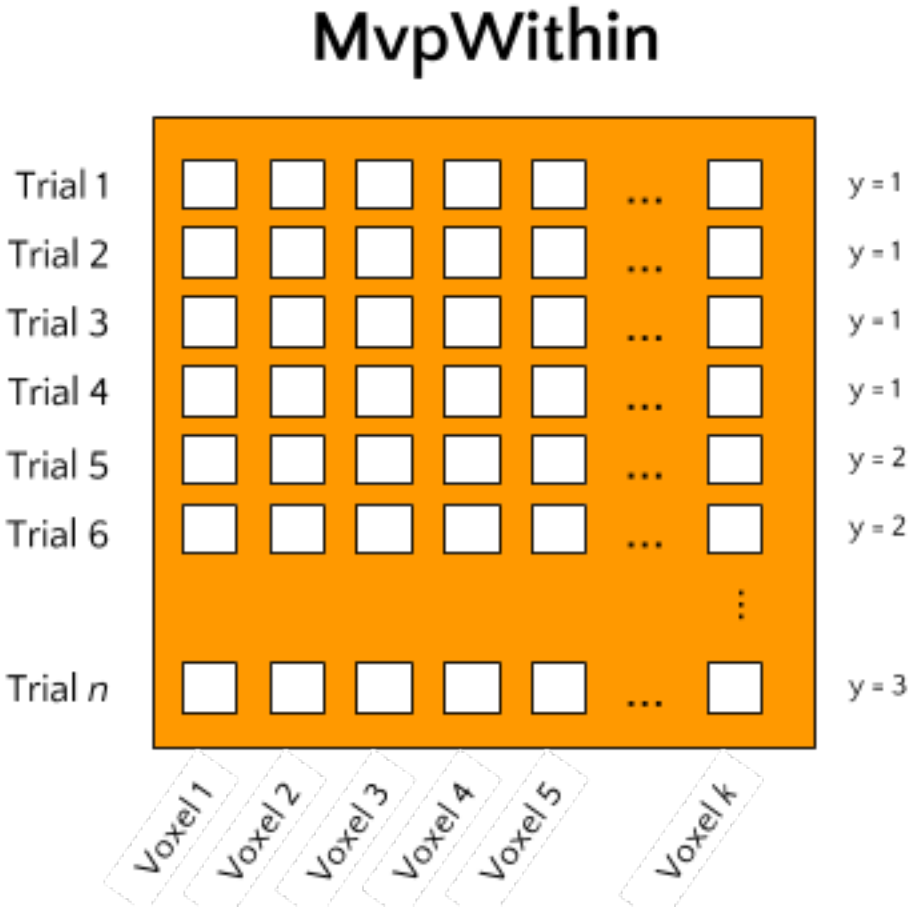
Mvp-objects

One of skbold's main features is the data-structure `Mvp` (an abbreviation of MultiVoxel Pattern). This custom object allows for an efficient way to store and access data and metadata necessary for multivoxel analyses of fMRI data. A nice feature of this `Mvp` objects is that they can easily load data (i.e., sets of nifti-files) from disk and automatically organize it in a format that is used in ML-analyses (i.e., a sample-by-feature matrix).

So, at the core, an `Mvp`-object is simply a collection of data - a 2D array of samples by features - and fMRI-specific metadata necessary to perform customized preprocessing and feature engineering. However, machine learning analyses, or more generally any type of multivoxel-type analysis (i.e. MVPA), can be done in two basic ways, which provide the basis of the two 'flavors' of `Mvp`-objects: `MvpWithin` and `MvpBetween`, as explained in more detail below.

MvpWithin

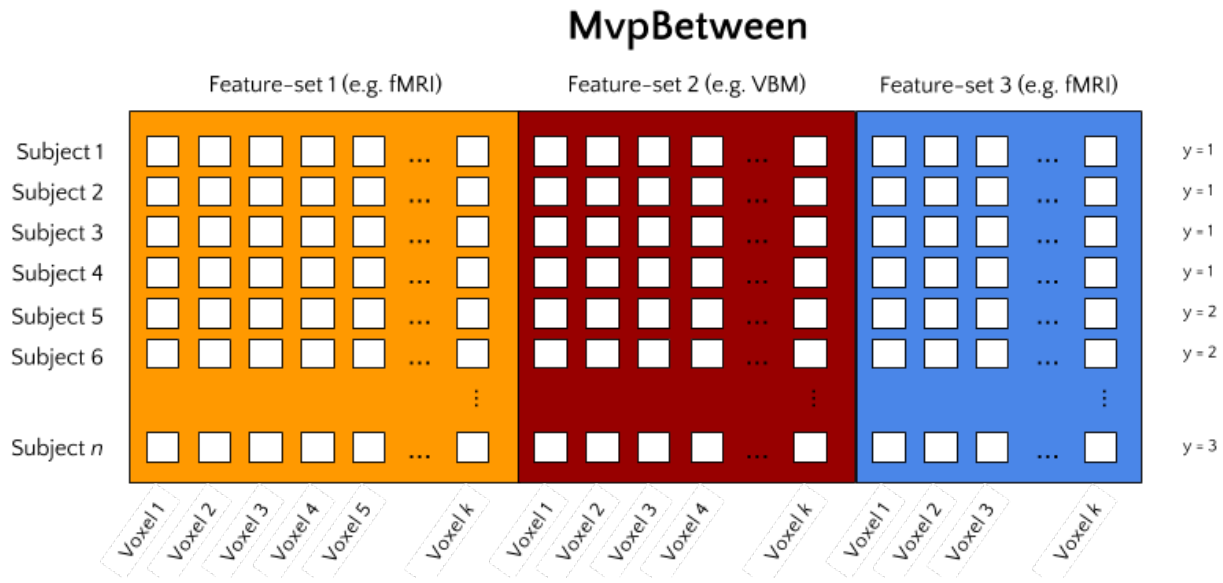
One way is to perform analyses *within subjects*. This means that a model is fit on each subjects' data separately. Data, in this context, often refers to single-trial data, in which each trial comprises a sample in our data-matrix and the values per voxel constitute our features. This type of analysis is alternatively called *single-trial decoding*, and is often performed as an alternative to (whole-brain) univariate analysis.



Ultimately, this type of analysis aims to predict some kind of attribute of the trials (for example condition/class membership in classification analyses or some continuous feature in regression analyses), which skbold calls `y`, based on a model trained on the samples-by-features matrix, which skbold calls `X`. After obtaining model performance scores (such as accuracy, F1-score, or R-squared) for each subject, a group-level random effects (RFX) analysis can be done on these scores. Skbold does not offer any functionality in terms of group-level analyses; we advise researchers to look into the prevalance inference method of Allefeld and colleagues.

MvpBetween

With the increase in large-sample neuroimaging datasets, another type of MVPA starts to become feasible, which we'll call *between subject* analyses. In this type of analysis, single subjects constitute the data's samples and a corresponding single multivoxel pattern constitutes the data's features. The type of multivoxel pattern, or 'feature-set', can be any set of voxel values. For example, features from a single first-level contrast (note: this should be a condition average contrast, as opposed to single-trial contrasts in MvpWithin!) can be used. But voxel patterns from VBM, TBSS (DTI), and dual-regression maps can equally well be used. Crucially, this package allows for the possibility to stack feature-sets such that models can be fit on features from multiple data-types simultaneously.


MvpBetween
Feature-set 1 (e.g. fMRI)
Feature-set 2 (e.g. VBM)
Feature-set 3 (e.g. fMRI)
Subject 1
Subject 2
Subject 3
Subject 4
Subject 5
Subject 6
Subject n
y = 1
y = 1
y = 1
y = 1
y = 2
y = 2
y = 3
Voxel 1
Voxel 2
Voxel 3
Voxel 4
Voxel 5
Voxel k
Voxel 1
Voxel 2
Voxel 3
Voxel 4
Voxel k
Voxel 1
Voxel 2
Voxel 3
Voxel k

CHAPTER 2

MvpResults: model evaluation and feature visualization

Given that an appropriate `Mvp`-object exists, it is really easy to implement a machine learning analysis using standard *scikit-learn* modules. However, as fMRI datasets are often relatively small, K-fold cross-validation is often performed to keep the training-set as large as possible. Additionally, it might be informative to visualize which features are used and are most important in your model. (But, note that feature mapping should not be the main objective of decoding analyses!) Doing this - model evaluation and feature visualization across multiple folds - complicates the process of implementing machine learning pipelines on fMRI data.

The `MvpResults` object offers a solution to the above complications. Simply pass your scikit-learn pipeline to MvpResults after every fold and it automatically calculates a set of model evaluation metrics (accuracy, precision, recall, etc.) and keeps track of which features are used and how 'important' these features are (in terms of the value of their weights).

CHAPTER 3

Feature selection/extraction

The `feature_selection` and `feature_extraction` modules in skbold contain a set of scikit-learn type transformers that can perform various types of feature selection and extraction specific to multivoxel fMRI-data. For example, the RoiIndexer-transformer takes a (partially masked) whole-brain pattern and indexes it with a specific region-of-interest defined in a nifti-file. The transformer API conforms to scikit-learn transformers, and as such, (almost all of them) can be used in scikit-learn pipelines.

To get a better idea of the package's functionality - including the use of Mvp-objects, transformers, and MvpResults - a typical analysis workflow using `skbold` is described below.

CHAPTER 4

Examples

For some example usages of the `Mvp`-objects and how to incorporate them in a `scikit-learn`-based ML-pipeline, check the examples below:

An example workflow: MvpWithin

Suppose you have data from an fMRI-experiment for a set of subjects who were presented with images which were either emotional or neutral in terms of their content. You've modelled them using a single-trial GLM (i.e. each trial is modelled as a separate event/regressor) and calculated their corresponding contrasts against baseline. The resulting FEAT-directory then contains a directory ('stats') with contrast-estimates (COPEs) for each trial. Now, using MvpWithin, it is easy to extract a sample by features matrix and some meta-data associated with it, as shown below.

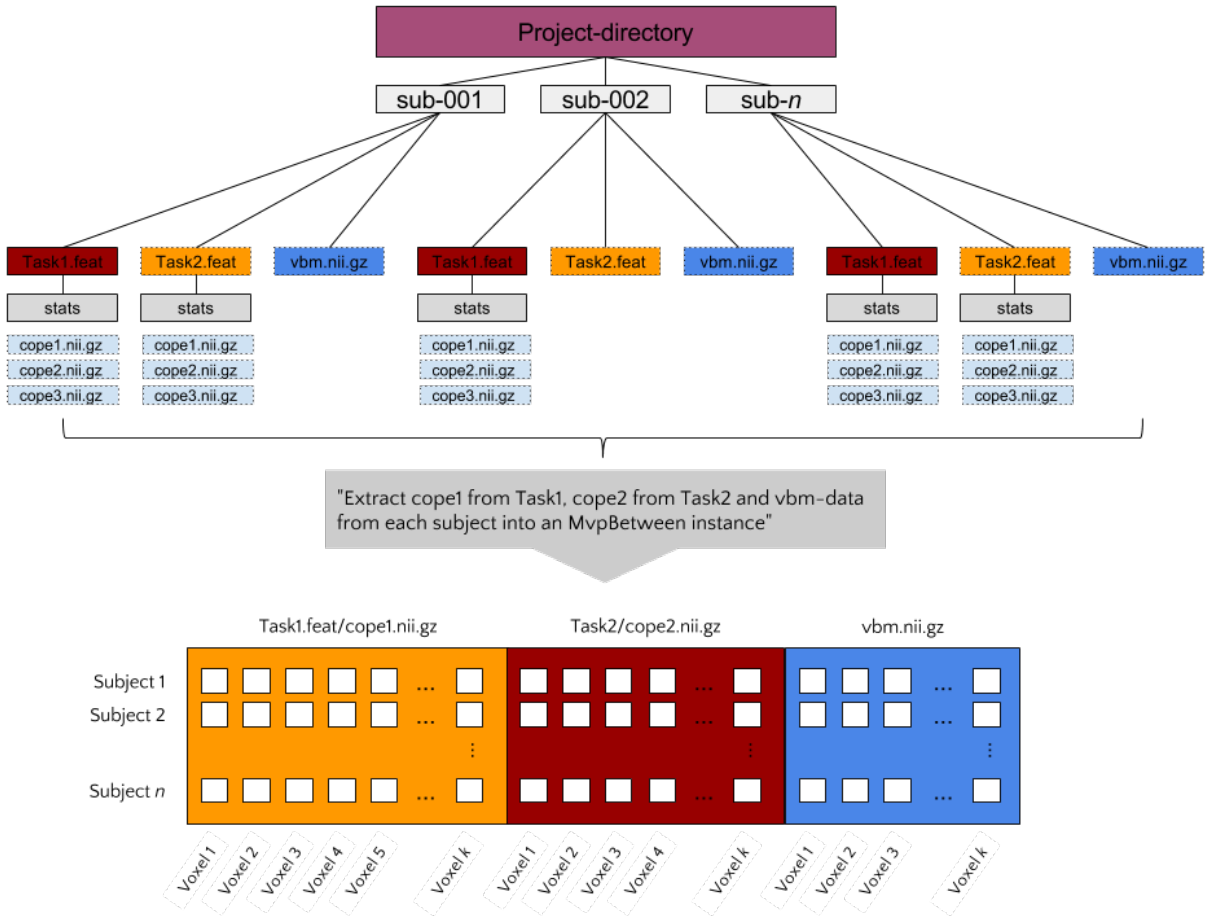
```
from skbold.core import MvpWithin

feat_dir = '~/project/sub001.feat'
mask_file = '~/GrayMatterMask.nii.gz' # mask all non-gray matter!
read_labels = True # parse labels (targets) from design.con file!
remove_contrast = ['nuisance_regressor_x'] # do not load nuisance regressor!
ref_space = 'epi' # extract patterns in functional space (alternatively: 'mni')
statistic = 'tstat' # use the tstat*.nii.gz files (in *.feat/stats) as patterns
remove_zeros = True # remove voxels which are zero in each trial

mvp = MvpWithin(source=feat_dir, read_labels=read_labels,
                remove_contrast=remove_contrast, ref_space=ref_space,
                statistic=statistic, remove_zeros=remove_zeros,
                mask=mask_file)

mvp.create() # extracts and stores (meta)data from FEAT-directory!
mvp.write(path='~/', name='mvp_sub001') # saves to disk!
```

Now, we have an Mvp-object on which machine learning pipeline can be applied:

```
import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold
from skbold.feature_selection import fisher_criterion_score, SelectAboveCutoff
from skbold.feature_extraction import RoiIndexer
from skbold.utils import MvpResultsClassification

mvp = joblib.load('~/mvp_sub001.jl')
roiindex = RoiIndexer(mvp=mvp, mask='Amygdala', atlas_name='HarvardOxford-Subcortical
↪',
                      lateralized=False)  # loads in bilateral mask

# Extract amygdala patterns from whole-brain
mvp.X = roiindex.fit().transform(mvp.X)

# Define pipeline
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('anova', SelectAboveCutoff(fisher_criterion_score, cutoff=5)),
    ('svm', SVC(kernel='linear'))
])

cv = StratifiedKFold(y=mvp.y, n_splits=5)

# Initialization of MvpResults; 'forward' indicates that it keeps track of
# the forward model corresponding to the weights of the backward model
# (see Haufe et al., 2014, Neuroimage)
mvp_results = MvpResultsClassification(mvp=mvp, n_iter=len(cv),
                                       out_path='~/', feature_scoring='forward')

for train_idx, test_idx in cv.split(mvp.X, mvp.y):

    train, test = mvp.X[train_idx, :], mvp.X[test_idx, :]
    train_y, test_y = mvp.y[train_idx], mvp.y[train_idx]

    pipe.fit(train, train_y)
    pred = pipe.predict(test)

    mvp_results.update(test_idx, pred, pipe) # update after each fold!

mvp_results.compute_scores() # compute!
mvp_results.write() # write file with metrics and niftis with feature-scores!
```

An example workflow: MvpBetween

Suppose you have MRI data from a large set of subjects (let's say >50), including (task-based) functional MRI, structural MRI (T1-weighted images, DTI), and behavioral data (e.g. questionnaires, behavioral tasks). Such a dataset would qualify for a *between subject* decoding analysis using the MvpBetween object. To use the MvpBetween functionality effectively, it is important that the data is organized sensibly. An example is given below.



In this example, each subject has three different data-sources: two FEAT- directories (with functional contrasts) and one VBM-file. Let's say that we'd like to use all of these sources of information together to predict some behavioral variable, neuroticism for example (as measured with e.g. the NEO-FFI). The most important argument passed to MvpBetween is `source`. This variable, a dictionary, should contain the data-types you want to extract and their corresponding paths (with wildcards at the place of subject-specific parts):

```
import os
from skbold import roidata_path
gm_mask = os.path.join(roidata_path, 'GrayMatter.nii.gz')

source = {}
source['Contrast_t1cope1'] = {'path': '~/Project_dir/sub*/Task1.feat/cope1.nii.gz'}
source['Contrast_t2cope2'] = {'path': '~/Project_dir/sub*/Task2.feat/cope2.nii.gz'}
source['VBM'] = {'path': '~/Project_dir/sub*/vbm.nii.gz', 'mask': gm_mask}
```

Now, to initialize the MvpBetween object, we need some more info:

```
from skbold.core import MvpBetween

subject_idf='sub-0??' # this is needed to extract the subject names to
                      # cross-reference across data-sources
subject_list=None     # can be a list of subject-names to include

mvp = MvpBetween(source=source, subject_idf=subject_idf, mask=None,
```

```
                  subject_list=None)

# like with MvpWithin, you can simply call create() to start the extraction!
mvp.create()

# and write to disk using write()
mvp.write(path='~/', name='mvp_between') # saves to disk!
```

This is basically all you need to create a MvpBetween object! It is very similar to MvpWithin in terms of attributes (including `X`, `y`, and various meta-data attributes). In fact, MvpResults works exactly in the same way for MvpWithin and MvpBetween! The major difference is that MvpResults keeps track of the feature-information for each feature-set separately and writes out a summarizing nifti file for each feature-set. Transformers also work the same for MvpBetween objects/data, with the exception of the cluster-threshold transformer.

CHAPTER 5

Installation & dependencies

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
```

Skbold is largely Python-only (both Python2.7 and Python3) and is built around the "PyData" stack, including:

- Numpy
- Scipy
- Pandas
- Scikit-learn

And it uses the awesome nibabel package for reading/writing nifti-files. Also, skbold uses FSL (primarily the `FLIRT` and `applywarp` functions) to transform files from functional (native) to standard (here: MNI152 2mm) space. These FSL-calls are embedded in the `convert2epi` and `convert2mni` functions, so avoid this functionality if you don't have a working FSL installation.

CHAPTER 6

Documentation

For those reading this on Github, documentation can be found on ReadTheDocs!

CHAPTER 7

Authos & credits

This package is being develop by Lukas Snoek from the University of Amsterdam with contributions from Steven and help from Joost.

CHAPTER 8

License and contact

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

CHAPTER 9

Code documentation

Below, the links to the documentation for each subpackage of skbold is listed.