skbold Documentation

Release 0.1

Lukas Snoek

Jul 10, 2017

Contents

The Python package `skbold` offers a set of tools and utilities for machine learning (and soon also RSA-type) 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, including data loading/organization, feature selection/extraction, model evaluation, and feature visualization.

An important feature of `skbold` is the data-structure `Mvp` (Multivoxel pattern), that allows for an efficient way to store and access data and metadata necessary for multivoxel analyses of fMRI data. A novel feature of this data-structure is that it is able to easily load data from FSL-FEAT output directories. As the `Mvp` object is available in two 'options', they are explained in more detail below.

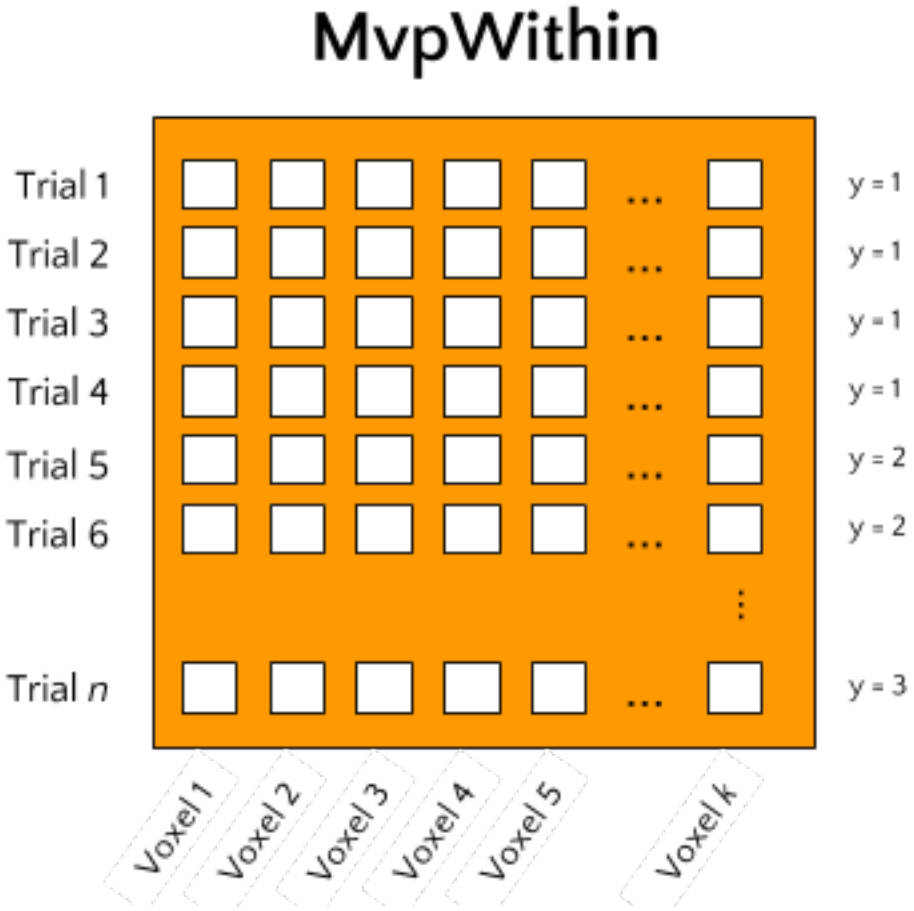
CHAPTER 1

Mvp-objects

At the core, an `Mvp`-object is simply a collection of data - a 2D array of samples by features - and fMRI-specific meta-data 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.

MvpWithin

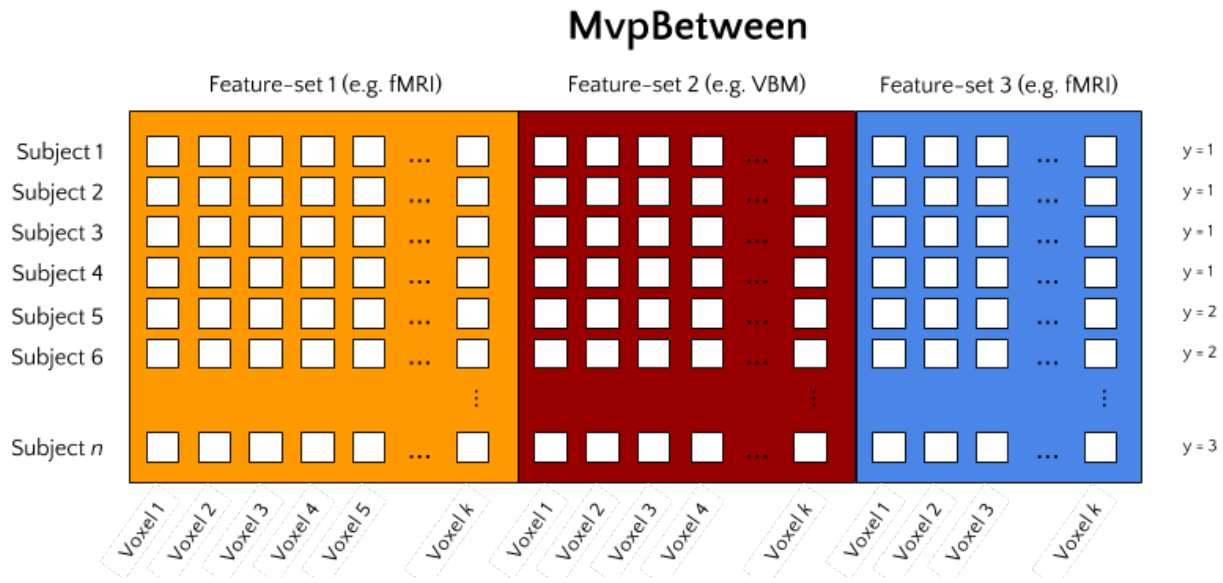
One way is to perform analyses *within subjects*. This means that a model is fit on each subjects' data separately. Data, in this context, often refers to single-trial data, in which each trial comprises a sample in our data-matrix and the values per voxel constitute our features. This type of analysis is alternatively called *single-trial decoding*, and is often performed as an alternative to massively (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). Ultimately, group-analyses may be done on subject-specific analysis metrics (such as classification accuracy or R2-score) and group-level feature-importance maps may be calculated to draw conclusions about the model's predictive power and the spatial distribution of informative features, respectively.

MvpBetween

With the apparent increase in large-sample neuroimaging datasets, another type of analysis 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

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')
beta2tstat = True # convert beta-estimates of COPEs to tstats
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,
                beta2tstat=beta2tstat, 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:

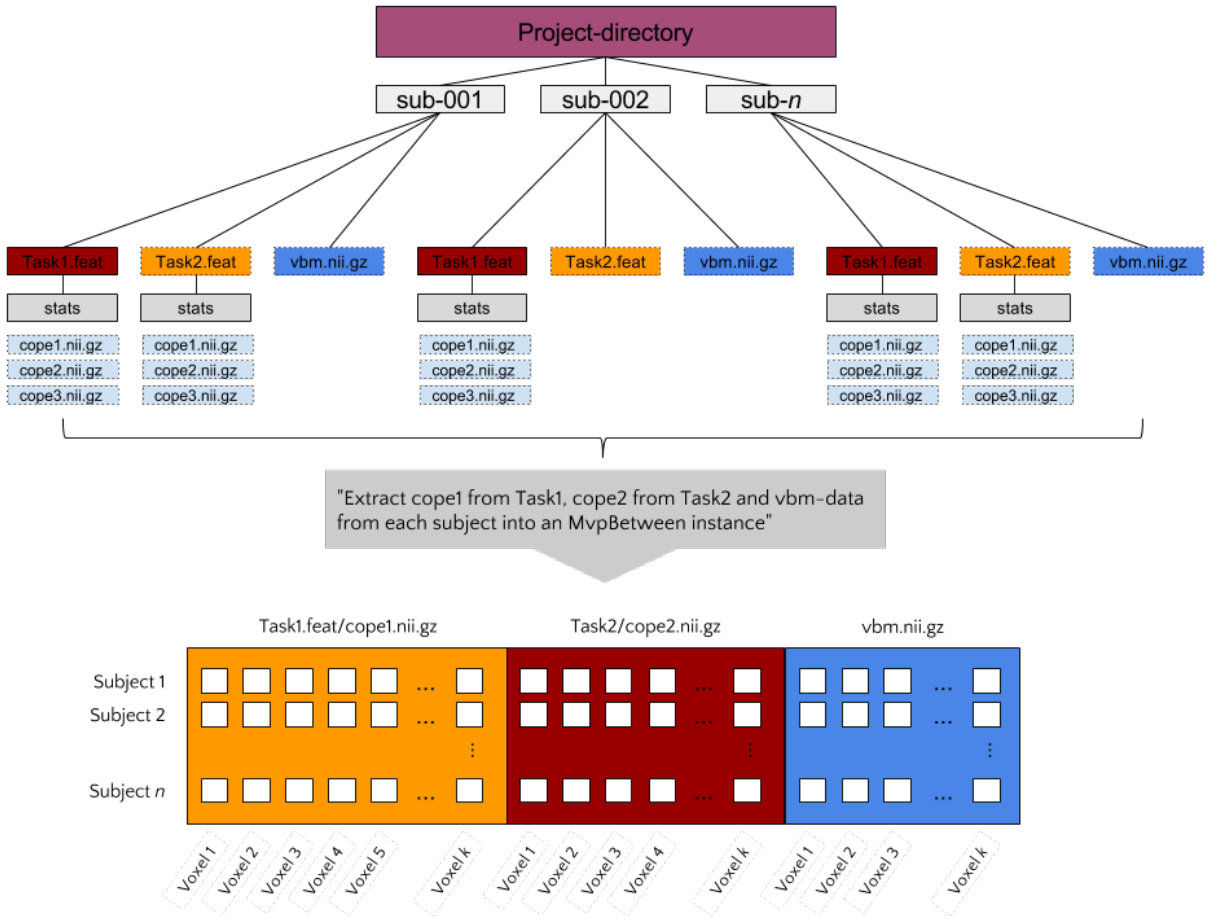
```
import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import StratifiedKFold
from skbold.feature_selection import fisher_criterion_score, SelectAboveCutoff
from skbold.feature_extraction import 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_folds=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:

    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!
```

CHAPTER 5

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 6

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
```

CHAPTER 7

Documentation

For those reading this on Github, documentation can be found on readthedocs.org!

CHAPTER 8

Credits

When I started writingthis package, I knew next to nothing about Python programming in general and packaging in specific. The mlxtend package has been a great 'template' and helped a great deal in structuring the current package. Also, Steven has contributed some very nice features as part of his internship. Lastly, Joost has beena major help in virtually every single phase of this package!

CHAPTER 9

License and contact

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

CHAPTER 10

Code documentation:

skbold - utilities and tools for machine learning on BOLD-fMRI data

The Python package `skbold` offers a set of tools and utilities for machine learning (and soon also RSA-type) 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, including data loading/organization, feature selection/extraction, model evaluation, and feature visualization.

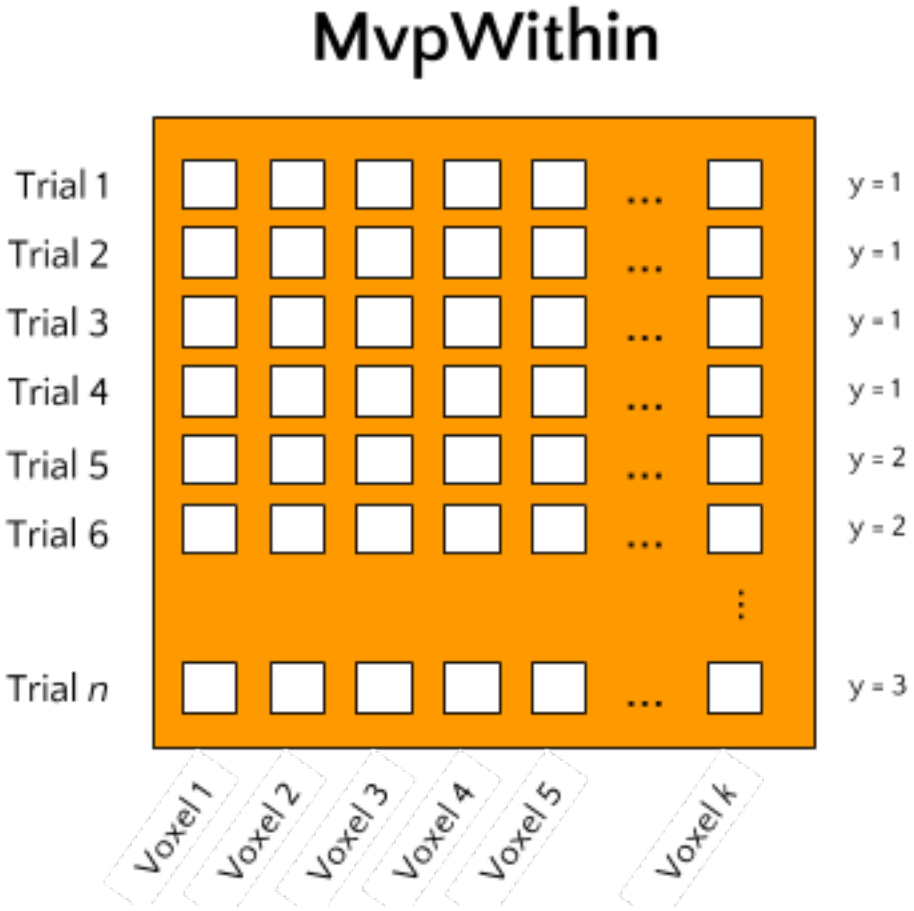
An important feature of `skbold` is the data-structure `Mvp` (Multivoxel pattern), that allows for an efficient way to store and access data and metadata necessary for multivoxel analyses of fMRI data. A novel feature of this data-structure is that it is able to easily load data from FSL-FEAT output directories. As the `Mvp` object is available in two 'options', they are explained in more detail below.

Mvp-objects

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.

MvpWithin

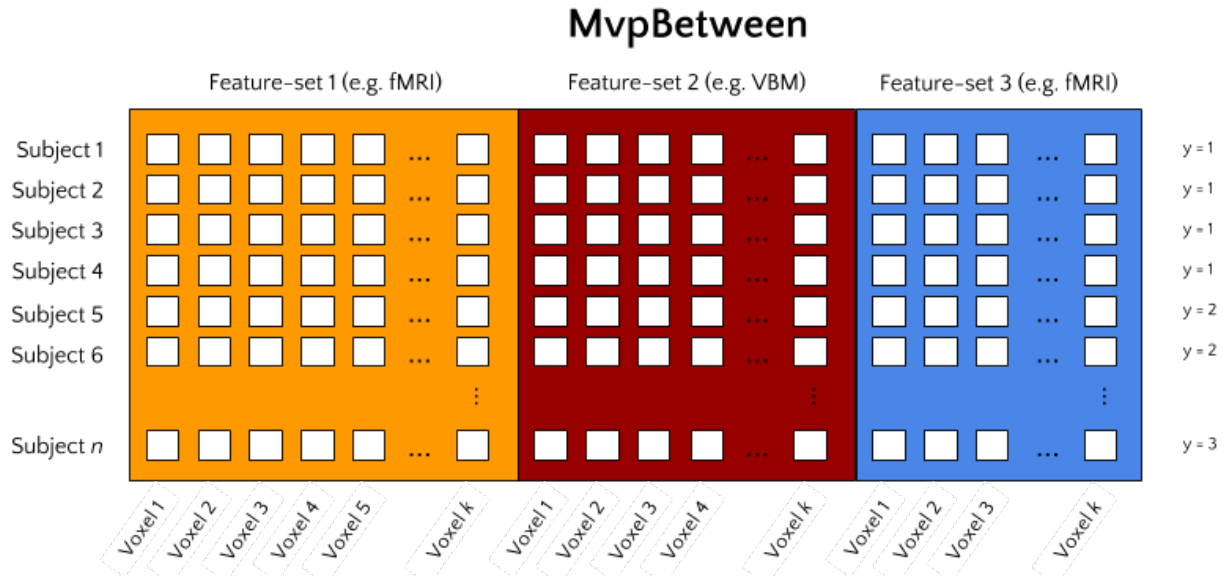
One way is to perform analyses *within subjects*. This means that a model is fit on each subjects' data separately. Data, in this context, often refers to single-trial data, in which each trial comprises a sample in our data-matrix and the values per voxel constitute our features. This type of analysis is alternatively called *single-trial decoding*, and is often performed as an alternative to massively (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). Ultimately, group-analyses may be done on subject-specific analysis metrics (such as classification accuracy or R2-score) and group-level feature-importance maps may be calculated to draw conclusions about the model's predictive power and the spatial distribution of informative features, respectively.

MvpBetween

With the apparent increase in large-sample neuroimaging datasets, another type of analysis 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.



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).

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.

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')
beta2tstat = True # convert beta-estimates of COPEs to tstats
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,
                beta2tstat=beta2tstat, 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!
```

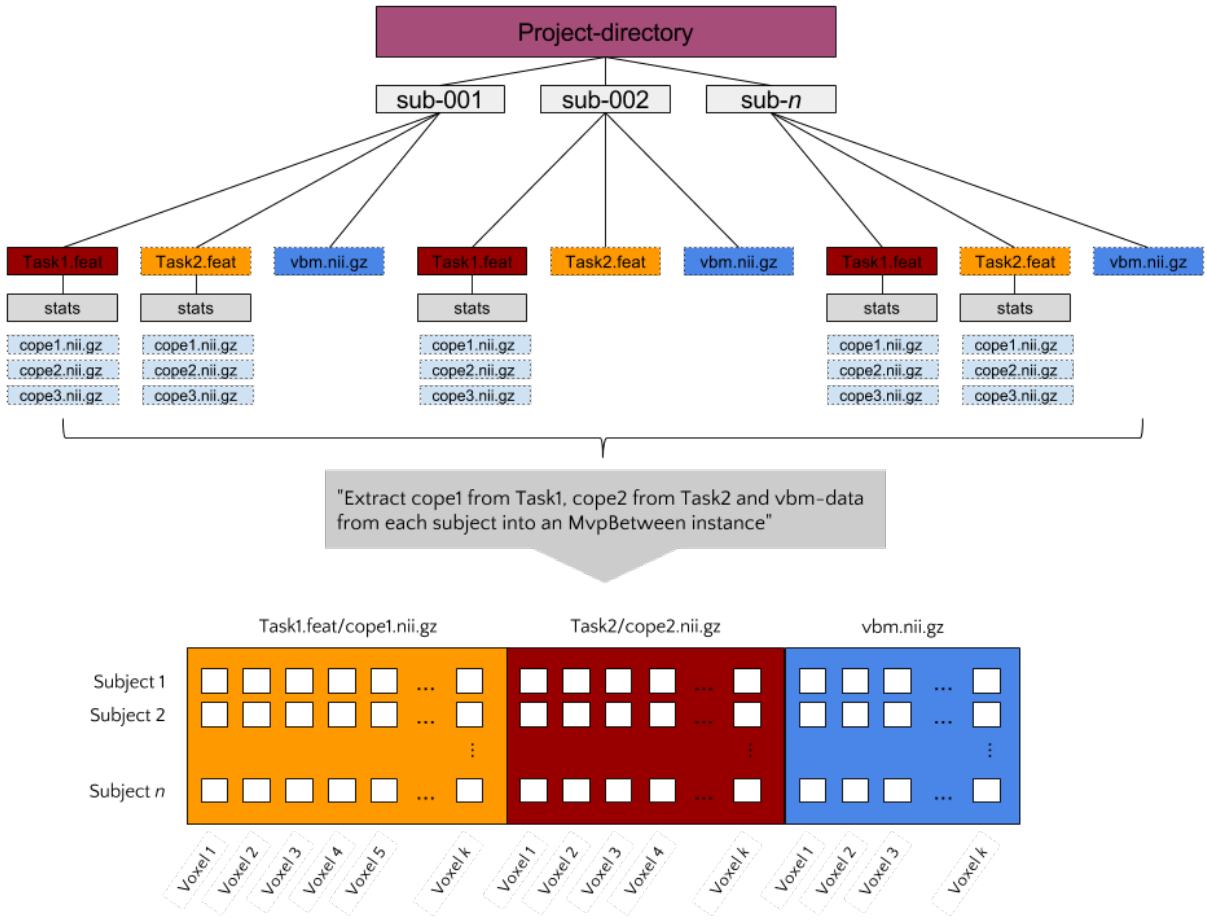
Now, we have an Mvp-object on which machine learning pipeline can be applied:

```
import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.cross_validation 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_folds=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:

    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.

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
```

Documentation

For those reading this on Github, documentation can be found on readthedocs.org!

Credits

When I started writingthis package, I knew next to nothing about Python programming in general and packaging in specific. The mlxtend package has been a great 'template' and helped a great deal in structuring the current package.

Also, Steven has contributed some very nice features as part of his internship. Lastly, Joost has beena major help in virtually every single phase of this package!

License and contact

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

skbold package

Subpackages

skbold.core package

The `core` subpackage contains skbold's most important data-structure: the `Mvp`. This class forms the basis of the 'multivoxel-patterns' (i.e. mvp) that are used throughout the package. Subclasses of Mvp (`MvpWithin` and `MvpBetween`) are also defined in this core module.

The `MvpWithin` object is meant as a data-structure that contains a set of multivoxel fMRI patterns of *single trials, for a single subject*, hence the 'within' part (i.e. within-subjects). Currently, it has a single public method, `create()`, loading a set of contrasts from a FSL-firstlevel directory (i.e. a .feat-directory). Thus, importantly, it assumes that the single-trial patterns are already modelled, on a single-trial basis, using some kind of GLM. These trialwise patterns are then horizontally stacked to create a 2D samples by features matrix, which is set to the `X` attribute of MvpWithin.

The `MvpBetween` object is meant as a data-structure that contains a set of multivoxel fMRI patterns of *single conditions, for a set of subjects*. It is, so to say, a 'between-subjects' multivoxel pattern, in which subjects are 'samples'. In contrast to MvpWithin, contrasts that will be loaded are less restricted in terms of their format; the only requisite is that they are nifti files. Notably, the MvpBetween format allows to vertically stack different kind of 'feature-sets' in a single MvpBetween object. For example, it is possible to, for a given set of subjects, stack a functional contrast (e.g. a high-load minus low-load functional contrast) with another functional contrast (e.g. a conflict minus no-conflict functional contrast) in order to use features from both sets to predict a certain psychometric or behavioral variable of the corresponding subjects (such as, e.g., intelligence). Also, the MvpBetween format allows to load (and stack!) VBM, TBSS, resting-state (to extract connectivity measures), and dual-regression data. More information can be found below in the API. A use case can be found on the main page of ReadTheDocs.

Also, functional-to-standard (i.e. `convert2mni`) and standard-to-functional (i.e. `convert2epi`) warp-functions for niftis are defined here, because they have caused circular import errors in the past.

class **Mvp** (*X=None*, *y=None*, *mask=None*, *mask_thres=0*)

Bases: `object`

Mvp (multiVoxel Pattern) class. Creates an object, specialized for storing fMRI data that will be analyzed using machine learning or RSA-like analyses, that stores both the data (X: an array of samples by features, y: numeric labels corresponding to X's classes/conditions) and the corresponding meta-data (e.g. nifti header, mask info, etc.).

Parameters

- **X** (*ndarray*) – A 2D numpy-array with rows indicating samples and columns indicating features.
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.
- **mask** (*str*) – Absolute path to nifti-file that will mask (index) the patterns.
- **mask_thres** (*int or float*) – Minimum value for mask (in cases of probabilistic masks).

Variables

- **mask_shape** (*tuple*) – Shape of mask that patterns will be indexed with.
- **nifti_header** (*Nifti1Header object*) – Nifti-header from corresponding mask.
- **affine** (*ndarray*) – Affine corresponding to nifti-mask.
- **voxel_idx** (*ndarray*) – Array with integer-indices indicating which voxels are used in the patterns relative to whole-brain space. In other words, it allows to map back the patterns to a whole-brain orientation.
- **X** (*ndarray*) – The actual patterns (2D: samples X features)
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.

Notes

This class is mainly meant to serve as a parent-class for `MvpWithin` and `MvpBetween`, but it can alternatively be used as an object to store a 'custom' multivariate-pattern set with meta-data.

update_mask(*mask*, *threshold=0*)

write(*path=None*, *name='mvp'*, *backend='joblib'*)
Writes the Mvp-object to disk.

Parameters

- **path** (*str*) – Absolute path where the file will be written to.
- **name** (*str*) – Name of to-be-written file.
- **backend** (*str*) – Which format will be used to save the files. Default is 'joblib', which conveniently saves the Mvp-object as one file. Alternatively, and if the Mvp-object is too large to be save with joblib, a data-header format will be used, in which the data (`X`) will be saved using Numpy and the meta-data (everythin except `X`) will be saved using joblib.

convert2epi(*file2transform*, *reg_dir*, *out_dir=None*, *interpolation='trilinear'*, *suffix='epi'*, *overwrite=False*)
Transforms a nifti from mni152 (2mm) to EPI (native) format. Assuming that reg_dir is a directory with transformation-files (warps) including standard2example_func warps, this function uses nipype's fsl interface to flirt a nifti to EPI format.

Parameters

- **file2transform** (*str or list*) – Absolute path(s) to nifti file(s) that needs to be transformed
- **reg_dir** (*str*) – Absolute path to registration directory with warps
- **out_dir** (*str*) – Absolute path to desired out directory. Default is same directory as the to-be transformed file.
- **interpolation** (*str*) – Interpolation used by flirt. Default is 'trilinear'.
- **suffix** (*str*) – What to suffix the transformed file with (default : 'epi')
- **overwrite** (*bool*) – Whether to overwrite existing transformed files

Returns **out_all** – Absolute path(s) to newly transformed file(s).

Return type list

convert2mni (*file2transform*, *reg_dir*, *out_dir=None*, *interpolation='trilinear'*, *suffix=None*, *overwrite=False*, *apply_warp=True*)

Transforms a nifti to mni152 (2mm) format. Assuming that reg_dir is a directory with transformation-files (warps) including example_func2standard warps, this function uses nipype's fsl interface to flirt a nifti to mni format.

Parameters

- **file2transform** (*str or list*) – Absolute path to nifti file(s) that needs to be transformed
- **reg_dir** (*str*) – Absolute path to registration directory with warps
- **out_dir** (*str*) – Absolute path to desired out directory. Default is same directory as the to-be transformed file.
- **interpolation** (*str*) – Interpolation used by flirt. Default is 'trilinear'.
- **suffix** (*str*) – What to append to name when converted (default : basename file2transform).
- **overwrite** (*bool*) – Whether to overwrite already existing transformed file(s)
- **apply_warp** (*bool*) – Whether to use the non-linear warp transform (if available).

Returns **out_all** – Absolute path(s) to newly transformed file(s).

Return type list

class MvpBetween (*source*, *subject_idf='sub0???'*, *remove_zeros=True*, *X=None*, *y=None*, *mask=None*, *mask_thres=0*, *subject_list=None*)

Bases: *skbold.core.mvp.Mvp*

Extracts and stores multivoxel pattern information across subjects. The MvpBetween class allows for the extraction and storage of multivoxel (MRI) pattern information across subjects. The MvpBetween class can handle various types of information, including functional contrasts, 3D (subject-specific) and 4D (subjects stacked) VBM and TBSS data, dual-regression data, and functional-connectivity data from resting-state scans (experimental).

Parameters

- **source** (*dict*) – Dictionary with types of data as keys and data-specific dictionaries as values. Keys can be 'Contrast_*' (indicating a 3D functional contrast), '4D_anat' (for 4D anatomical - VBM/TBSS - files), 'VBM', 'TBSS', and 'dual_reg' (a subject-spedific 4D file with components as fourth dimension).

  The dictionary passed as values must include, for each data-type, a path with wildcards to the corresponding (subject-specific) data-file. Other, optional, key-value pairs per data-type can be assigned, including 'mask': 'path', to use data-type-specific masks.

  An example:

```
>>> source = {}
>>> path_emo = '~/data/sub0*/*.feat/stats/tstat1.nii.gz'
>>> source['Contrast_emo'] = {'path': path_emo}
>>> vbm_mask = '~/vbm_mask.nii.gz'
>>> path_vbm = '~/data/sub0*/*vbm.nii.gz'
>>> source['VBM'] = {'path': path_vbm, 'mask': vbm_mask}
```

- **subject_idf** (*str*) – Subject-identifier. This identifier is used to extract subject-names from the globbed directories in the 'path' keys in source, so that it is known which pattern belongs to which subject. This way, MvpBetween can check which subjects contain complete data!

- **X** (*ndarray*) – Not necessary to pass MvpWithin, but needs to be defined as it is needed in the super-constructor.
- **y** (*ndarray or list*) – Labels or targets corresponding to the samples in X.
- **mask** (*str*) – Absolute path to nifti-file that will be used as a common mask. Note: this will only be applied if its shape corresponds to the to-be-indexed data. Otherwise, no mask is applied. Also, this mask is 'overridden' if source[data_type] contains a 'mask' key, which implies that this particular data-type has a custom mask.
- **mask_threshold** (*int or float*) – Minimum value to binarize the mask when it's probabilistic.

Variables

- **mask_shape** (*tuple*) – Shape of mask that patterns will be indexed with.
- **nifti_header** (*list of Nifti1Header objects*) – Nifti-headers from original data-types.
- **affine** (*list of ndarray*) – Affines corresponding to nifti-masks of each data-type.
- **X** (*ndarray*) – The actual patterns (2D: samples X features)
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.
- **common_subjects** (*list*) – List of subject-names that have complete data specified in source.
- **featureset_id** (*ndarray*) – Array with integers of size X.shape[1] (i.e. the amount of features in X). Each unique integer, starting at 0, refers to a different feature-set.
- **voxel_idx** (*ndarray*) – Array with integers of size X.shape[1]. Per feature-set, these voxel- indices allow the features to be mapped back to whole-brain space. For example, to map back the features in X from feature set 1 to MNI152 (2mm) space, do:

```
>>> mni_vol = np.zeros((91, 109, 91))
>>> tmp_idx = mvp.featureset_id == 0
>>> mni_vol[mvp.featureset_id[tmp_idx]] = mvp.X[0, tmp_idx]
```

- **data_shape** (*list of tuples*) – Original (whole-brain) shape of the loaded data, per data-type.
- **data_name** (*list of str*) – List of names of data-types.

add_y (*file_path*, *col_name*, *sep='\t'*, *index_col=0*, *normalize=False*, *remove=None*, *ensure_balanced=False*, *nan_strategy='remove'*, ***kwargs*)
Sets y attribute to an outcome-variable (target).

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the outcome var.
- **col_name** (*str*) – Column name in spreadsheet containing the outcome variable
- **sep** (*str*) – Separator to parse the spreadsheet-like file.
- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **normalize** (*bool*) – Whether to normalize (0 mean, unit std) the outcome variable.
- **remove** (*int or float or str*) – Removes instances in which y == remove from MvpBetween object.
- **ensure_balanced** (*bool*) – Whether to ensure balanced classes (if True, done by undersampling the majority class).

- **nan_strategy** (*str*) – Strategy on how to deal with NaNs. Default: 'remove'. Also, a specific string, int, or float can be specified when you want to impute a specific value. Other options, see: sklearn.preprocessing.Imputer.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

apply_binarization_params(*param_file*, *ensure_balanced=False*)
Applies binarization-parameters to y.

binarize_y(*params*, *save_path=None*, *ensure_balanced=False*)
Binarizes mvp's y-attribute using a specified method.

Parameters

- **params** (*dict*) – The outcome variable (y) will be binarized along the key-value pairs in the params-argument. Options:

```
>>> params = {'type': 'percentile', 'high': 75, 'low': 25}
>>> params = {'type': 'zscore', 'std': 1}
>>> params = {'type': 'constant', 'cutoff': 10}
>>> params = {'type': 'median'}
```

- **save_path** (*str*) – If not None (default), this should be an absolute path referring to where the binarization-params should be saved.
- **ensure_balanced** (*bool*) – Whether to ensure balanced classes (if True, done by undersampling the majority class).

calculate_confound_weighting(*file_path*, *col_name*, *sep='\t'*, *index_col=0*, *estimator=None*, *nan_strategy='depends'*, ***kwargs*)
Calculates inverse probability weighting for confounds.

Note: should be moved to mvp-core

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the confounding variable.
- **col_name** (*str or List[str]*) – Column name in spreadsheet containing the confouding variable
- **sep** (*str*) – Separator to parse the spreadsheet-like file.
- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **estimator** (*scikit-learn estimator*) – Estimator used to calculate p(y=1 | confound-array)
- **nan_strategy** (*str*) – How to impute NaNs.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

Returns **ipw** – Array with inverse probability weights for the samples, based on the confounds indicated by col_name.

Return type array

References

Linn, K.A., Gaonkar, B., Doshi, J., Davatzikos, C., & Shinohara, R. (2016). Addressing confounding in predictive models with an application to neuroimaging. Int. J. Biostat., 12(1): 31-44.

Code adapted from https://github.com/kalinn/IPW-SVM.

create()
Extracts and stores data as specified in source.

Raises `ValueError` – If data-type is not one of ['VBM', 'TBSS', '4D_anat*', 'dual_reg', 'Contrast*']

regress_out_confounds(*file_path*, *col_name*, *backend='numpy'*, *sep='\t'*, *index_col=0*, *nan_strategy='depends'*, ***kwargs*)
Regresses out a confounding variable from X.

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the confounding variable.
- **col_name** (*str or List[str]*) – Column name in spreadsheet containing the confouding variable
- **backend** (*str*) – Which algorithm to use to regress out the confound. The option 'numpy' uses np.linalg.lstsq() and 'sklearn' uses the LinearRegression estimator.
- **sep** (*str*) – Separator to parse the spreadsheet-like file.
- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **nan_strategy** (*str*) – How to impute NaNs.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

run_searchlight(*out_dir*, *name='sl_results'*, *n_folds=10*, *radius=5*, *mask=None*, *estimator=None*, ***kwargs*)
Runs a searchlight on the mvp object.

Parameters

- **out_dir** (*str*) – Path to which to save the searchlight results
- **name** (*str*) – Name for the searchlight-results-file (nifti)
- **n_folds** (*int*) – The amount of folds in sklearn's StratifiedKFold.
- **radius** (*int/list*) – Radius for the searchlight. If list, it iterates over radii.
- **mask** (*str*) – Path to mask to apply to mvp. If nothing is listed, it will use the masks applied when the mvp was created.
- **estimator** (*sklearn estimator or pipeline*) – Estimator to use in the classification process.
- ****kwargs** – Other keyword arguments for initializing nilearn's searchlight object (see nilearn.github.io/decoding/searchlight.html).

split(*file_path*, *col_name*, *target*, *sep='\t'*, *index_col=0*, *nan_strategy='train'*, ***kwargs*)
Splits an MvpBetween object based on some external index.

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the outcome var.
- **col_name** (*str*) – Column name in spreadsheet containing the outcome variable
- **target** (*str or int or float*) – Target to which the data in col_name needs to be compared to, in order to create an index.
- **sep** (*str*) – Separator to parse the spreadsheet-like file.

- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **nan_strategy** (*str*) – Which value to impute if the labeling is absent. Default: 'train'.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

update_sample(*idx*)
Updates the data matrix and associated attributes.

write_4D(*path=None*, *return_nimg=False*)
Writes a 4D nifti (subs = 4th dimension) of X.

Parameters

- **path** (*str*) – Absolute path to save nifti to.
- **return_nimg** (*bool*) – Whether to actually return the Nifti1-image object.

class MvpWithin(*source*, *read_labels=True*, *remove_contrast=[]*, *invert_selection=None*, *ref_space='epi'*, *beta2tstat=True*, *remove_zeros=True*, *X=None*, *y=None*, *mask=None*, *mask_threshold=0*)
Bases: *skbold.core.mvp.Mvp*

Extracts and stores subject-specific single-trial multivoxel-patterns The MvpWithin class allows for the extraction of subject-specific single-trial, multivoxel fMRI patterns from a FSL feat-directory.

Parameters

- **source** (*str*) – An absolute path to a subject-specific first-level FEAT directory.
- **read_labels** (*bool*) – Whether to read the labels/targets (i.e. `y`) from the contrast names defined in the design.con file.
- **remove_contrast** (*list*) – Given that all contrasts (COPEs) are loaded from the FEAT-directory, this argument can be used to remove irrelevant contrasts (e.g. contrasts of nuisance predictors). Entries in remove_contrast do not have to literal; they may be a substring of the full name of the contrast.
- **invert_selection** (*bool*) – Sometimes, instead of loading in all contrasts and excluding some, you might want to load only a single or a couple contrasts, and exclude all other. By setting invert_selection to True, it treats the remove_contrast variable as a list of contrasts to include.
- **ref_space** (*str*) – Indicates in which 'space' the patterns will be stored. The default is 'epi', indicating that the patterns will be loaded and stored in subject-specific (native) functional space. The other option is 'mni', which indicates that MvpWithin will first transform contrasts to MNI152 (2mm) space before it loads them. This option assumes that a 'reg' directory is present in the .feat-directory, including warp-files from functional to mni space (i.e. example_func2standara.nii.gz).
- **beta2tstat** (*bool*) – Whether to convert beta-values from COPEs to t-statistics by dividing them by the square-root of the res4d.
- **remove_zeros** (*bool*) – Whether to remove features (i.e. voxels) which are 0 across all trials (due to, e.g., being located outside the brain).
- **X** (*ndarray*) – Not necessary to pass MvpWithin, but needs to be defined as it is needed in the super-constructor.
- **y** (*ndarray or list*) – Labels or targets corresponding to the samples in `X`. This can be used when read_labels is set to False.
- **mask** (*str*) – Absolute path to nifti-file that will be used as mask.
- **mask_threshold** (*int or float*) – Minimum value to binarize the mask when it's probabilistic.

Variables

- **mask_shape** (*tuple*) – Shape of mask that patterns will be indexed with.
- **nifti_header** (*Nifti1Header object*) – Nifti-header from corresponding mask.
- **affine** (*ndarray*) – Affine corresponding to nifti-mask.
- **voxel_idx** (*ndarray*) – Array with integer-indices indicating which voxels are used in the patterns relative to whole-brain space. In other words, it allows to map back the patterns to a whole-brain orientation.
- **X** (*ndarray*) – The actual patterns (2D: samples X features)
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.
- **contrast_labels** (*list*) – List of names corresponding to the y-values.

create()

Extracts (meta-)data from FEAT-directory given appropriate settings during initialization.

Raises

- ValueError – If the 'source'-directory doesn't exist.
- ValueError – If the number of loaded contrasts does not equal the number of extracted labels.

Submodules

skbold.core.convert_to_epi module

convert2epi(*file2transform*, *reg_dir*, *out_dir=None*, *interpolation='trilinear'*, *suffix='epi'*, *overwrite=False*)

Transforms a nifti from mni152 (2mm) to EPI (native) format. Assuming that reg_dir is a directory with transformation-files (warps) including standard2example_func warps, this function uses nipype's fsl interface to flirt a nifti to EPI format.

Parameters

- **file2transform** (*str or list*) – Absolute path(s) to nifti file(s) that needs to be transformed
- **reg_dir** (*str*) – Absolute path to registration directory with warps
- **out_dir** (*str*) – Absolute path to desired out directory. Default is same directory as the to-be transformed file.
- **interpolation** (*str*) – Interpolation used by flirt. Default is 'trilinear'.
- **suffix** (*str*) – What to suffix the transformed file with (default : 'epi')
- **overwrite** (*bool*) – Whether to overwrite existing transformed files

Returns **out_all** – Absolute path(s) to newly transformed file(s).

Return type list

skbold.core.convert_to_mni module

convert2mni (*file2transform*, *reg_dir*, *out_dir=None*, *interpolation='trilinear'*, *suffix=None*, *overwrite=False*, *apply_warp=True*)

Transforms a nifti to mni152 (2mm) format. Assuming that reg_dir is a directory with transformation-files (warps) including example_func2standard warps, this function uses nipype's fsl interface to flirt a nifti to mni format.

Parameters

- **file2transform** (*str or list*) – Absolute path to nifti file(s) that needs to be transformed
- **reg_dir** (*str*) – Absolute path to registration directory with warps
- **out_dir** (*str*) – Absolute path to desired out directory. Default is same directory as the to-be transformed file.
- **interpolation** (*str*) – Interpolation used by flirt. Default is 'trilinear'.
- **suffix** (*str*) – What to append to name when converted (default : basename file2transform).
- **overwrite** (*bool*) – Whether to overwrite already existing transformed file(s)
- **apply_warp** (*bool*) – Whether to use the non-linear warp transform (if available).

Returns **out_all** – Absolute path(s) to newly transformed file(s).

Return type list

skbold.core.mvp module

class Mvp (*X=None*, *y=None*, *mask=None*, *mask_thres=0*)

Bases: object

Mvp (multiVoxel Pattern) class. Creates an object, specialized for storing fMRI data that will be analyzed using machine learning or RSA-like analyses, that stores both the data (X: an array of samples by features, y: numeric labels corresponding to X's classes/conditions) and the corresponding meta-data (e.g. nifti header, mask info, etc.).

Parameters

- **X** (*ndarray*) – A 2D numpy-array with rows indicating samples and columns indicating features.
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.
- **mask** (*str*) – Absolute path to nifti-file that will mask (index) the patterns.
- **mask_thres** (*int or float*) – Minimum value for mask (in cases of probabilistic masks).

Variables

- **mask_shape** (*tuple*) – Shape of mask that patterns will be indexed with.
- **nifti_header** (*Nifti1Header object*) – Nifti-header from corresponding mask.
- **affine** (*ndarray*) – Affine corresponding to nifti-mask.

- **voxel_idx** (*ndarray*) – Array with integer-indices indicating which voxels are used in the patterns relative to whole-brain space. In other words, it allows to map back the patterns to a whole-brain orientation.
- **X** (*ndarray*) – The actual patterns (2D: samples X features)
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.

Notes

This class is mainly meant to serve as a parent-class for `MvpWithin` and `MvpBetween`, but it can alternatively be used as an object to store a 'custom' multivariate-pattern set with meta-data.

update_mask(*mask*, *threshold=0*)

write(*path=None*, *name='mvp'*, *backend='joblib'*)
Writes the Mvp-object to disk.

Parameters

- **path** (*str*) – Absolute path where the file will be written to.
- **name** (*str*) – Name of to-be-written file.
- **backend** (*str*) – Which format will be used to save the files. Default is 'joblib', which conveniently saves the Mvp-object as one file. Alternatively, and if the Mvp-object is too large to be save with joblib, a data-header format will be used, in which the data (`X`) will be saved using Numpy and the meta-data (everythin except `X`) will be saved using joblib.

skbold.core.mvp_between module

class MvpBetween(*source*, *subject_idf='sub0???'*, *remove_zeros=True*, *X=None*, *y=None*, *mask=None*, *mask_thres=0*, *subject_list=None*)
Bases: *skbold.core.mvp.Mvp*

Extracts and stores multivoxel pattern information across subjects. The MvpBetween class allows for the extraction and storage of multivoxel (MRI) pattern information across subjects. The MvpBetween class can handle various types of information, including functional contrasts, 3D (subject-specific) and 4D (subjects stacked) VBM and TBSS data, dual-regression data, and functional-connectivity data from resting-state scans (experimental).

Parameters

- **source** (*dict*) – Dictionary with types of data as keys and data-specific dictionaries as values. Keys can be 'Contrast_*' (indicating a 3D functional contrast), '4D_anat' (for 4D anatomical - VBM/TBSS - files), 'VBM', 'TBSS', and 'dual_reg' (a subject-spedific 4D file with components as fourth dimension).

  The dictionary passed as values must include, for each data-type, a path with wildcards to the corresponding (subject-specific) data-file. Other, optional, key-value pairs per data-type can be assigned, including 'mask': 'path', to use data-type-specific masks.

  An example:

```
>>> source = {}
>>> path_emo = '~/data/sub0*/*.feat/stats/tstat1.nii.gz'
>>> source['Contrast_emo'] = {'path': path_emo}
>>> vbm_mask = '~/vbm_mask.nii.gz'
```

```
>>> path_vbm = '~/data/sub0*/*vbm.nii.gz'
>>> source['VBM'] = {'path': path_vbm, 'mask': vbm_mask}
```

- **subject_idf** (*str*) – Subject-identifier. This identifier is used to extract subject-names from the globbed directories in the 'path' keys in source, so that it is known which pattern belongs to which subject. This way, MvpBetween can check which subjects contain complete data!
- **X** (*ndarray*) – Not necessary to pass MvpWithin, but needs to be defined as it is needed in the super-constructor.
- **y** (*ndarray or list*) – Labels or targets corresponding to the samples in X.
- **mask** (*str*) – Absolute path to nifti-file that will be used as a common mask. Note: this will only be applied if its shape corresponds to the to-be-indexed data. Otherwise, no mask is applied. Also, this mask is 'overridden' if source[data_type] contains a 'mask' key, which implies that this particular data-type has a custom mask.
- **mask_threshold** (*int or float*) – Minimum value to binarize the mask when it's probabilistic.

Variables

- **mask_shape** (*tuple*) – Shape of mask that patterns will be indexed with.
- **nifti_header** (*list of Nifti1Header objects*) – Nifti-headers from original data-types.
- **affine** (*list of ndarray*) – Affines corresponding to nifti-masks of each data-type.
- **X** (*ndarray*) – The actual patterns (2D: samples X features)
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.
- **common_subjects** (*list*) – List of subject-names that have complete data specified in source.
- **featureset_id** (*ndarray*) – Array with integers of size X.shape[1] (i.e. the amount of features in X). Each unique integer, starting at 0, refers to a different feature-set.
- **voxel_idx** (*ndarray*) – Array with integers of size X.shape[1]. Per feature-set, these voxel- indices allow the features to be mapped back to whole-brain space. For example, to map back the features in X from feature set 1 to MNI152 (2mm) space, do:

```
>>> mni_vol = np.zeros((91, 109, 91))
>>> tmp_idx = mvp.featureset_id == 0
>>> mni_vol[mvp.featureset_id[tmp_idx]] = mvp.X[0, tmp_idx]
```

- **data_shape** (*list of tuples*) – Original (whole-brain) shape of the loaded data, per data-type.
- **data_name** (*list of str*) – List of names of data-types.

add_y (*file_path*, *col_name*, *sep='\t'*, *index_col=0*, *normalize=False*, *remove=None*, *ensure_balanced=False*, *nan_strategy='remove'*, ***kwargs*)

Sets y attribute to an outcome-variable (target).

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the outcome var.
- **col_name** (*str*) – Column name in spreadsheet containing the outcome variable
- **sep** (*str*) – Separator to parse the spreadsheet-like file.

- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **normalize** (*bool*) – Whether to normalize (0 mean, unit std) the outcome variable.
- **remove** (*int or float or str*) – Removes instances in which y == remove from MvpBetween object.
- **ensure_balanced** (*bool*) – Whether to ensure balanced classes (if True, done by undersampling the majority class).
- **nan_strategy** (*str*) – Strategy on how to deal with NaNs. Default: 'remove'. Also, a specific string, int, or float can be specified when you want to impute a specific value. Other options, see: sklearn.preprocessing.Imputer.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

apply_binarization_params (*param_file*, *ensure_balanced=False*)
Applies binarization-parameters to y.

binarize_y (*params*, *save_path=None*, *ensure_balanced=False*)
Binarizes mvp's y-attribute using a specified method.

Parameters

- **params** (*dict*) – The outcome variable (y) will be binarized along the key-value pairs in the params-argument. Options:

```
>>> params = {'type': 'percentile', 'high': 75, 'low': 25}
>>> params = {'type': 'zscore', 'std': 1}
>>> params = {'type': 'constant', 'cutoff': 10}
>>> params = {'type': 'median'}
```

- **save_path** (*str*) – If not None (default), this should be an absolute path referring to where the binarization-params should be saved.
- **ensure_balanced** (*bool*) – Whether to ensure balanced classes (if True, done by undersampling the majority class).

calculate_confound_weighting (*file_path*, *col_name*, *sep='\t'*, *index_col=0*, *estimator=None*, *nan_strategy='depends'*, ***kwargs*)
Calculates inverse probability weighting for confounds.

Note: should be moved to mvp-core

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the confounding variable.
- **col_name** (*str or List[str]*) – Column name in spreadsheet containing the confouding variable
- **sep** (*str*) – Separator to parse the spreadsheet-like file.
- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **estimator** (*scikit-learn estimator*) – Estimator used to calculate p(y=1 | confound-array)
- **nan_strategy** (*str*) – How to impute NaNs.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

Returns **ipw** – Array with inverse probability weights for the samples, based on the confounds indicated by col_name.

Return type array

References

Linn, K.A., Gaonkar, B., Doshi, J., Davatzikos, C., & Shinohara, R. (2016). Addressing confounding in predictive models with an application to neuroimaging. Int. J. Biostat., 12(1): 31-44.

Code adapted from https://github.com/kalinn/IPW-SVM.

create()
Extracts and stores data as specified in source.

Raises `ValueError` – If data-type is not one of ['VBM', 'TBSS', '4D_anat*', 'dual_reg', 'Contrast*']

regress_out_confounds(*file_path*, *col_name*, *backend='numpy'*, *sep='\t'*, *index_col=0*, *nan_strategy='depends'*, ***kwargs*)
Regresses out a confounding variable from X.

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the confounding variable.
- **col_name** (*str or List[str]*) – Column name in spreadsheet containing the confouding variable
- **backend** (*str*) – Which algorithm to use to regress out the confound. The option 'numpy' uses np.linalg.lstsq() and 'sklearn' uses the LinearRegression estimator.
- **sep** (*str*) – Separator to parse the spreadsheet-like file.
- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **nan_strategy** (*str*) – How to impute NaNs.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

run_searchlight(*out_dir*, *name='sl_results'*, *n_folds=10*, *radius=5*, *mask=None*, *estimator=None*, ***kwargs*)
Runs a searchlight on the mvp object.

Parameters

- **out_dir** (*str*) – Path to which to save the searchlight results
- **name** (*str*) – Name for the searchlight-results-file (nifti)
- **n_folds** (*int*) – The amount of folds in sklearn's StratifiedKFold.
- **radius** (*int/list*) – Radius for the searchlight. If list, it iterates over radii.
- **mask** (*str*) – Path to mask to apply to mvp. If nothing is listed, it will use the masks applied when the mvp was created.
- **estimator** (*sklearn estimator or pipeline*) – Estimator to use in the classification process.
- ****kwargs** – Other keyword arguments for initializing nilearn's searchlight object (see nilearn.github.io/decoding/searchlight.html).

split(*file_path*, *col_name*, *target*, *sep='\t'*, *index_col=0*, *nan_strategy='train'*, ***kwargs*)
Splits an MvpBetween object based on some external index.

Parameters

- **file_path** (*str*) – Absolute path to spreadsheet-like file including the outcome var.
- **col_name** (*str*) – Column name in spreadsheet containing the outcome variable
- **target** (*str or int or float*) – Target to which the data in col_name needs to be compared to, in order to create an index.
- **sep** (*str*) – Separator to parse the spreadsheet-like file.
- **index_col** (*int*) – Which column to use as index (should correspond to subject-name).
- **nan_strategy** (*str*) – Which value to impute if the labeling is absent. Default: 'train'.
- ****kwargs** – Arbitrary keyword arguments passed to pandas read_csv.

update_sample(*idx*)
Updates the data matrix and associated attributes.

write_4D(*path=None*, *return_nimg=False*)
Writes a 4D nifti (subs = 4th dimension) of X.

Parameters

- **path** (*str*) – Absolute path to save nifti to.
- **return_nimg** (*bool*) – Whether to actually return the Nifti1-image object.

check_zeropadding_and_sort(*lst*)

skbold.core.mvp_within module

class MvpWithin(*source*, *read_labels=True*, *remove_contrast=[]*, *invert_selection=None*, *ref_space='epi'*, *beta2tstat=True*, *remove_zeros=True*, *X=None*, *y=None*, *mask=None*, *mask_threshold=0*)
Bases: *skbold.core.mvp.Mvp*

Extracts and stores subject-specific single-trial multivoxel-patterns The MvpWithin class allows for the extraction of subject-specific single-trial, multivoxel fMRI patterns from a FSL feat-directory.

Parameters

- **source** (*str*) – An absolute path to a subject-specific first-level FEAT directory.
- **read_labels** (*bool*) – Whether to read the labels/targets (i.e. `y`) from the contrast names defined in the design.con file.
- **remove_contrast** (*list*) – Given that all contrasts (COPEs) are loaded from the FEAT-directory, this argument can be used to remove irrelevant contrasts (e.g. contrasts of nuisance predictors). Entries in remove_contrast do not have to literal; they may be a substring of the full name of the contrast.
- **invert_selection** (*bool*) – Sometimes, instead of loading in all contrasts and excluding some, you might want to load only a single or a couple contrasts, and exclude all other. By setting invert_selection to True, it treats the remove_contrast variable as a list of contrasts to include.
- **ref_space** (*str*) – Indicates in which 'space' the patterns will be stored. The default is 'epi', indicating that the patterns will be loaded and stored in subject-specific (native) functional space. The other option is 'mni', which indicates that MvpWithin will first transform contrasts to MNI152 (2mm) space before it loads them. This option assumes that a 'reg' directory is present in the .feat-directory, including warp-files from functional to mni space (i.e. example_func2standara.nii.gz).

- **beta2tstat** (*bool*) – Whether to convert beta-values from COPEs to t-statistics by dividing them by the square-root of the res4d.
- **remove_zeros** (*bool*) – Whether to remove features (i.e. voxels) which are 0 across all trials (due to, e.g., being located outside the brain).
- **X** (*ndarray*) – Not necessary to pass MvpWithin, but needs to be defined as it is needed in the super-constructor.
- **y** (*ndarray or list*) – Labels or targets corresponding to the samples in X. This can be used when read_labels is set to False.
- **mask** (*str*) – Absolute path to nifti-file that will be used as mask.
- **mask_threshold** (*int or float*) – Minimum value to binarize the mask when it's probabilistic.

Variables

- **mask_shape** (*tuple*) – Shape of mask that patterns will be indexed with.
- **nifti_header** (*Nifti1Header object*) – Nifti-header from corresponding mask.
- **affine** (*ndarray*) – Affine corresponding to nifti-mask.
- **voxel_idx** (*ndarray*) – Array with integer-indices indicating which voxels are used in the patterns relative to whole-brain space. In other words, it allows to map back the patterns to a whole-brain orientation.
- **X** (*ndarray*) – The actual patterns (2D: samples X features)
- **y** (*list or ndarray*) – Array/list with labels/targets corresponding to samples in X.
- **contrast_labels** (*list*) – List of names corresponding to the y-values.

create()

Extracts (meta-)data from FEAT-directory given appropriate settings during initialization.

Raises

- ValueError – If the 'source'-directory doesn't exist.
- ValueError – If the number of loaded contrasts does not equal the number of extracted labels.

skbold.estimators package

The classifiers subpackage provides two ensemble-type classifiers that aim at aggregating multivoxel information from multiple local sources in the brain. They do so by allowing to fit a model on different brain areas, which predictions are subsequently combined using either a stacked (meta) model (i.e. the RoiStackingClassifier) or using a voting-strategy (i.e. the RoiVotingClassifier). The structure and API of these classifiers adhere to the scikit-learn estimator object.

class RoiStackingClassifier(*mvp*, *preproc_pipe='default'*, *base_clf=None*, *meta_clf=None*, *mask_type='unilateral'*, *proba=True*, *folds=10*, *meta_fs='univar'*, *meta_gs=None*, *n_cores=1*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

This scikit-learn-style classifier implements a stacking classifier that fits a base-classifier on multiple brain-regions separately and subsequently trains a meta-classifier on the outputs of the base- classifiers on the separate brain-regions.

Parameters

- **mvp** (*mvp-object*) – An custom object from the skbold package containing data (X, y) and corresponding meta-data (e.g. mask info)
- **preproc_pipe** (*object*) – A scikit-learn Pipeline object with desired preprocessing steps (e.g. scaling, additional feature selection). Defaults to only scaling and univariate-feature-selection by means of highest mean-euclidean differences (see `skbold.transformers.mean_euclidean`).
- **base_clf** (*object*) – A scikit-learn style classifier (implementing fit(), predict(), and predict_proba()), that is able to be used in Pipelines.
- **meta_clf** (*object*) – A scikit-learn style classifier.
- **mask_type** (*str*) – Can be 'unilateral' or 'bilateral', which will use all masks from the corresponding Harvard-Oxford Cortical (lateralized) atlas. Alternatively, it may be an absolute path to a directory containing a custom set of masks as nifti-files (default: 'unilateral').
- **meta_gs** (*list or ndarray*) – Optional parameter-grid over which to perform grid-search.
- **n_cores** (*int*) – Number of CPU-cores on which to perform the fitting procedure (here, outer-folds are parallelized).

Variables

- **train_scores** (*ndarray*) – Accuracy-scores per brain region (averaged over outer-folds) on the training (fit) phase.
- **test_scores** (*ndarray*) – Accuracy-scores per brain region (averaged over outer- and inner-folds) on the test phase.
- **masks** (*list of str*) – List of absolute paths to found masks.
- **stck_train** (*ndarray*) – Array with outputs from base-classifiers fit on train-set.
- **stck_test** (*ndarray*) – Array with outputs from base-classifiers generalized to test-set.

fit (*X*, *y*)

Fits RoiStackingClassfier.

Parameters

- **X** (*ndarray*) – Array of shape = [n_samples, n_features].
- **y** (*list or ndarray of int or float*) – List or ndarray with floats/ints corresponding to labels.

Returns **self** – RoiStackingClassifier instance with fitted parameters.

Return type object

predict (*X*, *y=None*)

Predict class given RoiStackingClassifier.

Parameters **X** (*ndarray*) – Array of shape = [n_samples, n_features].

Returns **meta_pred** – Array with class predictions.

Return type ndarray

score (*X*, *y*)

Scoring function calculating accuracy given predictions.

X [ndarray] Array of shape = [n_samples, n_features]

y [list or ndarray of int or float] List or ndarray with floats/ints corresponding to labels.

Returns **score** – Accuracy of predictions on the test-set.

Return type float

class **RoiVotingClassifier**(*mvp*, *preproc_pipeline=None*, *clf=None*, *mask_type='unilateral'*, *voting='soft'*, *weights=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

This classifier fits a base-estimator (by default a linear SVM) on different feature sets (i.e. voxels) from different regions of interest (which are drawn from the Harvard-Oxford Cortical atlas), and subsequently the final prediction is derived through a max-voting rule, which can be either 'soft' (argmax of mean class probability) or 'hard' (max of class prediction).

Notes

This classifier has not been tested!

Parameters

- **mvp** (*mvp-object*) – An custom object from the skbold package containing data (X, y) and corresponding meta-data (e.g. mask info)
- **preproc_pipeline** (*object*) – A scikit-learn Pipeline object with desired preprocessing steps (e.g. scaling, additional feature selection)
- **clf** (*object*) – A scikit-learn style classifier (implementing fit(), predict(), and predict_proba()), that is able to be used in Pipelines.
- **mask_type** (*str*) – Can be 'unilateral' or 'bilateral', which will use all masks from the corresponding Harvard-Oxford Cortical (lateralized) atlas. Alternatively, it may be an absolute path to a directory containing a custom set of masks as nifti-files (default: 'unilateral').
- **voting** (*str*) – Either 'hard' or 'soft' (default: 'soft').
- **weights** (*list (or ndarray)*) – List/array of shape [n_rois] with a relative weighting factor to be used in the voting procedure.

fit(*X=None*, *y=None*)

Fits RoiVotingClassifier.

Parameters

- **X** (*ndarray*) – Array of shape = [n_samples, n_features].
- **y** (*list or ndarray of int or float*) – List or ndarray with floats/ints corresponding to labels.

Returns **self** – RoiStackingClassifier instance with fitted parameters.

Return type object

predict(*X*)

Predict class given fitted RoiVotingClassifier.

Parameters **X** (*ndarray*) – Array of shape = [n_samples, n_features].

Returns **maxvotes** – Array with class predictions for all classes of X.

Return type ndarray

class **MultimodalVotingClassifier**(*mvp*, *clf=None*, *voting='soft'*, *weights=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

This classifier fits a base-estimator (by default a linear SVM) on different feature sets of different modalities (i.e. VBM, TBSS, BOLD, etc), and subsequently the final prediction is derived through a max-voting rule, which can be either 'soft' (argmax of mean class probability) or 'hard' (max of class prediction).

Notes

This classifier has not been tested!

Parameters

- **mvp** (*mvp-object*) – An custom object from the skbold package containing data (X, y) and corresponding meta-data (e.g. mask info)
- **preproc_pipeline** (*object*) – A scikit-learn Pipeline object with desired preprocessing steps (e.g. scaling, additional feature selection)
- **clf** (*object*) – A scikit-learn style classifier (implementing fit(), predict(), and predict_proba()), that is able to be used in Pipelines.
- **voting** (*str*) – Either 'hard' or 'soft' (default: 'soft').
- **weights** (*list (or ndarray)*) – List/array of shape [n_rois] with a relative weighting factor to be used in the voting procedure.

fit (*X=None*, *y=None*, *iterations=1*)
Fits RoiVotingClassifier.

Parameters

- **X** (*ndarray*) – Array of shape = [n_samples, n_features].
- **y** (*list or ndarray of int or float*) – List or ndarray with floats/ints corresponding to labels.

Returns **self** – RoiStackingClassifier instance with fitted parameters.

Return type object

predict (*X*)
Predict class given fitted RoiVotingClassifier.

Parameters **X** (*ndarray*) – Array of shape = [n_samples, n_features].

Returns **maxvotes** – Array with class predictions for all classes of X.

Return type ndarray

Submodules

skbold.estimators.multimodal_voting_classifier module

class **MultimodalVotingClassifier** (*mvp*, *clf=None*, *voting='soft'*, *weights=None*)
Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

This classifier fits a base-estimator (by default a linear SVM) on different feature sets of different modalities (i.e. VBM, TBSS, BOLD, etc), and subsequently the final prediction is derived through a max-voting rule, which can be either 'soft' (argmax of mean class probability) or 'hard' (max of class prediction).

Notes

This classifier has not been tested!

Parameters

- **mvp** (*mvp-object*) – An custom object from the skbold package containing data (X, y) and corresponding meta-data (e.g. mask info)
- **preproc_pipeline** (*object*) – A scikit-learn Pipeline object with desired preprocessing steps (e.g. scaling, additional feature selection)
- **clf** (*object*) – A scikit-learn style classifier (implementing fit(), predict(), and predict_proba()), that is able to be used in Pipelines.
- **voting** (*str*) – Either 'hard' or 'soft' (default: 'soft').
- **weights** (*list (or ndarray)*) – List/array of shape [n_rois] with a relative weighting factor to be used in the voting procedure.

fit (*X=None*, *y=None*, *iterations=1*)
Fits RoiVotingClassifier.

Parameters

- **X** (*ndarray*) – Array of shape = [n_samples, n_features].
- **y** (*list or ndarray of int or float*) – List or ndarray with floats/ints corresponding to labels.

Returns **self** – RoiStackingClassifier instance with fitted parameters.

Return type object

predict (*X*)
Predict class given fitted RoiVotingClassifier.

Parameters **X** (*ndarray*) – Array of shape = [n_samples, n_features].

Returns **maxvotes** – Array with class predictions for all classes of X.

Return type ndarray

skbold.estimators.roi_stacking_classifier module

class RoiStackingClassifier (*mvp*, *preproc_pipe='default'*, *base_clf=None*, *meta_clf=None*, *mask_type='unilateral'*, *proba=True*, *folds=10*, *meta_fs='univar'*, *meta_gs=None*, *n_cores=1*)
Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

This scikit-learn-style classifier implements a stacking classifier that fits a base-classifier on multiple brain-regions separately and subsequently trains a meta-classifier on the outputs of the base- classifiers on the separate brain-regions.

Parameters

- **mvp** (*mvp-object*) – An custom object from the skbold package containing data (X, y) and corresponding meta-data (e.g. mask info)
- **preproc_pipe** (*object*) – A scikit-learn Pipeline object with desired preprocessing steps (e.g. scaling, additional feature selection). Defaults to only scaling and univariate-feature-selection by means of highest mean-euclidean differences (see skbold.transformers.mean_euclidean).

- **base_clf** (*object*) – A scikit-learn style classifier (implementing fit(), predict(), and predict_proba()), that is able to be used in Pipelines.
- **meta_clf** (*object*) – A scikit-learn style classifier.
- **mask_type** (*str*) – Can be 'unilateral' or 'bilateral', which will use all masks from the corresponding Harvard-Oxford Cortical (lateralized) atlas. Alternatively, it may be an absolute path to a directory containing a custom set of masks as nifti-files (default: 'unilateral').
- **meta_gs** (*list or ndarray*) – Optional parameter-grid over which to perform grid-search.
- **n_cores** (*int*) – Number of CPU-cores on which to perform the fitting procedure (here, outer-folds are parallelized).

Variables

- **train_scores** (*ndarray*) – Accuracy-scores per brain region (averaged over outer-folds) on the training (fit) phase.
- **test_scores** (*ndarray*) – Accuracy-scores per brain region (averaged over outer- and inner-folds) on the test phase.
- **masks** (*list of str*) – List of absolute paths to found masks.
- **stck_train** (*ndarray*) – Array with outputs from base-classifiers fit on train-set.
- **stck_test** (*ndarray*) – Array with outputs from base-classifiers generalized to test-set.

fit (*X*, *y*)

Fits RoiStackingClassfier.

Parameters

- **X** (*ndarray*) – Array of shape = [n_samples, n_features].
- **y** (*list or ndarray of int or float*) – List or ndarray with floats/ints corresponding to labels.

Returns **self** – RoiStackingClassifier instance with fitted parameters.

Return type object

predict (*X*, *y=None*)

Predict class given RoiStackingClassifier.

Parameters **X** (*ndarray*) – Array of shape = [n_samples, n_features].

Returns **meta_pred** – Array with class predictions.

Return type ndarray

score (*X*, *y*)

Scoring function calculating accuracy given predictions.

X [ndarray] Array of shape = [n_samples, n_features]

y [list or ndarray of int or float] List or ndarray with floats/ints corresponding to labels.

Returns **score** – Accuracy of predictions on the test-set.

Return type float

skbold.estimators.roi_voting_classifier module

class **RoiVotingClassifier**(*mvp*, *preproc_pipeline=None*, *clf=None*, *mask_type='unilateral'*, *voting='soft'*, *weights=None*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.ClassifierMixin`

This classifier fits a base-estimator (by default a linear SVM) on different feature sets (i.e. voxels) from different regions of interest (which are drawn from the Harvard-Oxford Cortical atlas), and subsequently the final prediction is derived through a max-voting rule, which can be either 'soft' (argmax of mean class probability) or 'hard' (max of class prediction).

Notes

This classifier has not been tested!

Parameters

- **mvp** (*mvp-object*) – An custom object from the skbold package containing data (X, y) and corresponding meta-data (e.g. mask info)
- **preproc_pipeline** (*object*) – A scikit-learn Pipeline object with desired preprocessing steps (e.g. scaling, additional feature selection)
- **clf** (*object*) – A scikit-learn style classifier (implementing fit(), predict(), and predict_proba()), that is able to be used in Pipelines.
- **mask_type** (*str*) – Can be 'unilateral' or 'bilateral', which will use all masks from the corresponding Harvard-Oxford Cortical (lateralized) atlas. Alternatively, it may be an absolute path to a directory containing a custom set of masks as nifti-files (default: 'unilateral').
- **voting** (*str*) – Either 'hard' or 'soft' (default: 'soft').
- **weights** (*list (or ndarray)*) – List/array of shape [n_rois] with a relative weighting factor to be used in the voting procedure.

fit(*X=None*, *y=None*)

Fits RoiVotingClassifier.

Parameters

- **X** (*ndarray*) – Array of shape = [n_samples, n_features].
- **y** (*list or ndarray of int or float*) – List or ndarray with floats/ints corresponding to labels.

Returns **self** – RoiStackingClassifier instance with fitted parameters.

Return type object

predict(*X*)

Predict class given fitted RoiVotingClassifier.

Parameters **X** (*ndarray*) – Array of shape = [n_samples, n_features].

Returns **maxvotes** – Array with class predictions for all classes of X.

Return type ndarray

skbold.exp_model package

The exp_model subpackage contains some (pre)processing functions and classes that help in preparing to fit a first-level GLM on fMRI data across multiple subjects.

The PresentationLogfileCrawler (and its function-equivalent 'parse_presentation_logfile') can be used to parse Presentation-logfile, which are often used at the University of Amsterdam.

Also, there is an experimental Eprime-logfile converter, which converts the Eprime .txt-file to a tsv-file format.

parse_presentation_logfile(*in_file*, *con_names*, *con_codes*, *con_design=None*, *con_duration=None*, *pulsecode=30*)

Function-interface for PresentationLogfileCrawler. Can be used to create a Nipype node.

Parameters

- **in_file** (*str or list*) – Absolute path to logfile (can be a list of paths).
- **con_names** (*list*) – List with names for each condition
- **con_codes** (*list*) – List with codes for conditions. Can be a single integer or string (in the latter case, it may be a substring) or a list with possible values.
- **con_design** (*list or str*) – Which 'design' to assume for events (if 'multivar', all events - regardless of condition - are treated as a separate condition/regressor; if 'univar', all events from a single condition are treated as a single condition). Default: 'univar' for all conditions.
- **con_duration** (*list*) – If the duration cannot be parsed from the logfile, you can specify them here manually (per condition).
- **pulsecode** (*int*) – Code with which the first (or any) pulse is logged.

class **PresentationLogfileCrawler**(*in_file*, *con_names*, *con_codes*, *con_design=None*, *con_duration=None*, *pulsecode=30*, *write_bfsl=False*, *verbose=True*)

Bases: object

Logfile crawler for Presentation (Neurobs) files; cleans logfile, calculates event onsets and durations, and (optionally) writes out .bfsl files per condition.

Parameters

- **in_file** (*str or list*) – Absolute path to logfile (can be a list of paths).
- **con_names** (*list*) – List with names for each condition
- **con_codes** (*list*) – List with codes for conditions. Can be a single integer or string (in the latter case, it may be a substring) or a list with possible values.
- **con_design** (*list or str*) – Which 'design' to assume for events (if 'multivar', all events - regardless of condition - are treated as a separate condition/regressor; if 'univar', all events from a single condition are treated as a single condition). Default: 'univar' for all conditions.
- **con_duration** (*list*) – If the duration cannot be parsed from the logfile, you can specify them here manually (per condition).
- **pulsecode** (*int*) – Code with which the first (or any) pulse is logged.
- **write_bfsl** (*bool*) – Whether to write out a .bfsl file per condition.
- **verbose** (*bool*) – Print out intermediary output.

Variables **df** (*Dataframe*) – Dataframe with cleaned and parsed logfile.

parse()
Parses logfile, writes bfsl (optional), and return subject-info.

Returns **subject_info_list** – Bunch object to be used in Nipype pipelines.

Return type Nilearn bunch object

class **Eprime2tsv**(*in_file*)
Bases: object

Converts Eprime txt-files to tsv.

Parameters **in_file** (*str*) – Absolute path to Eprime txt-file.

Variables **df** (*Dataframe*) – Pandas dataframe with parsed and cleaned txt-file

convert(*out_dir=None*)
Converts txt-file to tsv.

Parameters **out_dir** (*str*) – Absolute path to desired directory to save tsv to (default: current directory).

class **FsfCrawler**(*preproc_dir*, *run_idf*, *template='mvpa'*, *mvpa_type='trial_wise'*, *output_dir=None*, *subject_idf='sub'*, *event_file_ext='bfsl'*, *func_idf='func'*, *prewhiten=True*, *derivs=False*, *mat_suffix=None*, *sort_by_onset=False*, *n_cores=1*)
Bases: object

Given an fsf-template, this crawler creates subject-specific fsf-FEAT files assuming that appropriate .bfsl files exist.

Parameters

- **template** (*str*) – Absolute path to template fsf-file. Default is 'mvpa', which models each bfsl-file as a separate regressor (and contrast against baseline).
- **mvpa_type** (*str*) – Whether to estimate patterns per trial (mvpa_type='trial_wise') or to estimate patterns per condition (or per run, mvpa_type='run_wise')
- **preproc_dir** (*str*) – Absolute path to directory with preprocessed files.
- **run_idf** (*str*) – Identifier for run to apply template fsf to.
- **output_dir** (*str*) – Path to desired output dir of first-levels.
- **subject_idf** (*str*) – Identifier for subject-directories.
- **event_file_ext** (*str*) – Extension for event-file; if 'bfsl' (default, for legacy reasons), then assumes single event-file per predictor. If 'tsv' (cf. BIDS), then assumes a single tsv-file with all predictors.
- **func_idf** (*str*) – Identifier for which functional should be use.
- **prewhiten** (*bool*) – Whether the data should be prewhitened in model fitting
- **derivs** (*bool*) – Whether to model derivatives of original regressors
- **mat_suffix** (*str*) – Identifier (suffix) for design.mat and batch.fsf file (such that it does not overwrite older files).
- **sort_by_onset** (*bool*) – Whether to sort predictors by onset (first trial = first predictor), or, when False, sort by condition (all trials condition A, all trials condition B, etc.).
- **n_cores** (*int*) – How many CPU cores should be used for the batch-analysis.

crawl()
Crawls subject-directories and spits out subject-specific fsf.

class **MelodicCrawler**(*preproc_dir*, *run_idf*, *template=None*, *output_dir=None*, *subject_idf='sub'*, *func_idf='func'*, *copy_reg=True*, *copy_mc=True*, *varnorm=True*, *n_cores=1*)

Bases: object

__init__(*preproc_dir*, *run_idf*, *template=None*, *output_dir=None*, *subject_idf='sub'*, *func_idf='func'*, *copy_reg=True*, *copy_mc=True*, *varnorm=True*, *n_cores=1*)

Given an fsf-template (Melodic), this crawler creates subject- specific fsf-melodic files and (optionally) copies the corresponding registration and mc directories to the out-directory.

Parameters

- **template** (*str*) – Absolute path to template fsf-file
- **preproc_dir** (*str*) – Absolute path to the directory with preprocessed files
- **run_idf** (*str*) – Identifier for run to apply template fsf to
- **output_dir** (*str*) – Path to desired output dir of Melodic-ica results.
- **subject_idf** (*str*) – Identifier for subject-directories.
- **func_idf** (*str*) – Identifier for which functional should be use.
- **copy_reg** (*bool*) – Whether to copy the subjects' registration directory
- **copy_mc** (*bool*) – Whether to copy the subjects' mc directory
- **varnorm** (*bool*) – Whether to apply variance-normalization (melodic option)
- **n_cores** (*int*) – How many CPU cores should be used for the batch-analysis.

crawl()

Crawls subject-directories and spits out subject-specific fsf.

Submodules

skbold.exp_model.batch_fsf module

class **FsfCrawler**(*preproc_dir*, *run_idf*, *template='mvpa'*, *mvpa_type='trial_wise'*, *output_dir=None*, *subject_idf='sub'*, *event_file_ext='bfsl'*, *func_idf='func'*, *prewhiten=True*, *derivs=False*, *mat_suffix=None*, *sort_by_onset=False*, *n_cores=1*)

Bases: object

Given an fsf-template, this crawler creates subject-specific fsf-FEAT files assuming that appropriate .bfsl files exist.

Parameters

- **template** (*str*) – Absolute path to template fsf-file. Default is 'mvpa', which models each bfsl-file as a separate regressor (and contrast against baseline).
- **mvpa_type** (*str*) – Whether to estimate patterns per trial (mvpa_type='trial_wise') or to estimate patterns per condition (or per run, mvpa_type='run_wise')
- **preproc_dir** (*str*) – Absolute path to directory with preprocessed files.
- **run_idf** (*str*) – Identifier for run to apply template fsf to.
- **output_dir** (*str*) – Path to desired output dir of first-levels.
- **subject_idf** (*str*) – Identifier for subject-directories.

- **event_file_ext** (*str*) – Extension for event-file; if 'bfsl' (default, for legacy reasons), then assumes single event-file per predictor. If 'tsv' (cf. BIDS), then assumes a single tsv-file with all predictors.
- **func_idf** (*str*) – Identifier for which functional should be use.
- **prewhiten** (*bool*) – Whether the data should be prewhitened in model fitting
- **derivs** (*bool*) – Whether to model derivatives of original regressors
- **mat_suffix** (*str*) – Identifier (suffix) for design.mat and batch.fsf file (such that it does not overwrite older files).
- **sort_by_onset** (*bool*) – Whether to sort predictors by onset (first trial = first predictor), or, when False, sort by condition (all trials condition A, all trials condition B, etc.).
- **n_cores** (*int*) – How many CPU cores should be used for the batch-analysis.

crawl()
Crawls subject-directories and spits out subject-specific fsf.

class MelodicCrawler(*preproc_dir*, *run_idf*, *template=None*, *output_dir=None*, *subject_idf='sub'*, *func_idf='func'*, *copy_reg=True*, *copy_mc=True*, *varnorm=True*, *n_cores=1*)
Bases: object

__init__(*preproc_dir*, *run_idf*, *template=None*, *output_dir=None*, *subject_idf='sub'*, *func_idf='func'*, *copy_reg=True*, *copy_mc=True*, *varnorm=True*, *n_cores=1*)
Given an fsf-template (Melodic), this crawler creates subject- specific fsf-melodic files and (optionally) copies the corresponding registration and mc directories to the out-directory.

Parameters

- **template** (*str*) – Absolute path to template fsf-file
- **preproc_dir** (*str*) – Absolute path to the directory with preprocessed files
- **run_idf** (*str*) – Identifier for run to apply template fsf to
- **output_dir** (*str*) – Path to desired output dir of Melodic-ica results.
- **subject_idf** (*str*) – Identifier for subject-directories.
- **func_idf** (*str*) – Identifier for which functional should be use.
- **copy_reg** (*bool*) – Whether to copy the subjects' registration directory
- **copy_mc** (*bool*) – Whether to copy the subjects' mc directory
- **varnorm** (*bool*) – Whether to apply variance-normalization (melodic option)
- **n_cores** (*int*) – How many CPU cores should be used for the batch-analysis.

crawl()
Crawls subject-directories and spits out subject-specific fsf.

skbold.exp_model.convert_eprime module

class Eprime2tsv(*in_file*)
Bases: object

Converts Eprime txt-files to tsv.

Parameters **in_file** (*str*) – Absolute path to Eprime txt-file.

Variables **df** (*Dataframe*) – Pandas dataframe with parsed and cleaned txt-file

convert(*out_dir=None*)
Converts txt-file to tsv.

Parameters **out_dir** (*str*) – Absolute path to desired directory to save tsv to (default: current directory).

skbold.exp_model.parse_presentation_logfile module

class PresentationLogfileCrawler(*in_file*, *con_names*, *con_codes*, *con_design=None*, *con_duration=None*, *pulsecode=30*, *write_bfsl=False*, *verbose=True*)
Bases: object

Logfile crawler for Presentation (Neurobs) files; cleans logfile, calculates event onsets and durations, and (optionally) writes out .bfsl files per condition.

Parameters

- **in_file** (*str or list*) – Absolute path to logfile (can be a list of paths).
- **con_names** (*list*) – List with names for each condition
- **con_codes** (*list*) – List with codes for conditions. Can be a single integer or string (in the latter case, it may be a substring) or a list with possible values.
- **con_design** (*list or str*) – Which 'design' to assume for events (if 'multivar', all events - regardless of condition - are treated as a separate condition/regressor; if 'univar', all events from a single condition are treated as a single condition). Default: 'univar' for all conditions.
- **con_duration** (*list*) – If the duration cannot be parsed from the logfile, you can specify them here manually (per condition).
- **pulsecode** (*int*) – Code with which the first (or any) pulse is logged.
- **write_bfsl** (*bool*) – Whether to write out a .bfsl file per condition.
- **verbose** (*bool*) – Print out intermediary output.

Variables **df** (*Dataframe*) – Dataframe with cleaned and parsed logfile.

parse()
Parses logfile, writes bfsl (optional), and return subject-info.

Returns **subject_info_list** – Bunch object to be used in Nipype pipelines.

Return type Nilearn bunch object

parse_presentation_logfile(*in_file*, *con_names*, *con_codes*, *con_design=None*, *con_duration=None*, *pulsecode=30*)
Function-interface for PresentationLogfileCrawler. Can be used to create a Nipype node.

Parameters

- **in_file** (*str or list*) – Absolute path to logfile (can be a list of paths).
- **con_names** (*list*) – List with names for each condition
- **con_codes** (*list*) – List with codes for conditions. Can be a single integer or string (in the latter case, it may be a substring) or a list with possible values.
- **con_design** (*list or str*) – Which 'design' to assume for events (if 'multivar', all events - regardless of condition - are treated as a separate condition/regressor; if 'univar',

all events from a single condition are treated as a single condition). Default: 'univar' for all conditions.

- **con_duration** (*list*) – If the duration cannot be parsed from the logfile, you can specify them here manually (per condition).
- **pulsecode** (*int*) – Code with which the first (or any) pulse is logged.

skbold.feature_extraction package

This module contains some feature-extraction methods/transformers.

class PatternAverager(*method='mean'*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Reduces the set of features to its average.

Parameters **method** (*str*) – method of averaging (either 'mean' or 'median')

fit(*X=None*, *y=None*)

Does nothing, but included to be used in sklearn's Pipeline.

transform(*X*)

Transforms patterns to its average.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X_new** – Transformed ndarray of shape = [n_samples, 1]

Return type ndarray

class ArrayPermuter

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Permutes (shuffles) rows of matrix.

__init__()

Initializes ArrayPermuter object.

fit(*X=None*, *y=None*)

Does nothing, but included to be used in sklearn's Pipeline.

transform(*X*)

Permutes rows of data input.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X_new** – ndarray with permuted rows

Return type ndarray

class AverageRegionTransformer(*atlas='HarvardOxford-All'*, *mask_threshold=0*, *mvp=None*, *reg_dir=None*, *orig_mask=None*, *data_shape=None*, *ref_space=None*, *affine=None*, ***kwargs*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Transforms a whole-brain voxel pattern into a region-average pattern Computes the average from different regions from a given parcellation and returns those as features for X.

Parameters

- **mask_type** (*List[str]*) – List with absolute paths to nifti-images of brain masks in MNI152 (2mm) space.

- **mvp** (*Mvp-object (see core.mvp)*) – Mvp object that provides some metadata about previous masks
- **mask_threshold** (*int (default: 0)*) – Minimum threshold for probabilistic masks (such as Harvard-Oxford)

fit (*X=None*, *y=None*)
Does nothing, but included to be used in sklearn's Pipeline.

transform (*X*, *y=None*)
Transforms features from X (voxels) to region-average features.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*Optional[List[str] or numpy ndarray[str]]*) – List of ndarray with strings indicating label-names

Returns **X_new** – array with transformed data of shape = [n_samples, n_features] in which features are region-average values.

Return type ndarray

class PCAfilter (*n_components=5*, *reject=None*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Filters out a (set of) PCA component(s) and transforms it back to original representation.

Parameters

- **n_components** (*int*) – number of components to retain.
- **reject** (*list*) – Indices of components which should be additionally removed.

Variables **pca** (*scikit-learn PCA object*) – Fitted PCA object.

fit (*X*, *y=None*, **args*)
Fits PcaFilter.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*List of str*) – List or ndarray with floats corresponding to labels

transform (*X*)
Transforms a pattern (X) by the inverse PCA transform with removed components.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the PCA calculated during fit().

Return type ndarray

class RoiIndexer (*mask*, *mask_threshold=0*, *mvp=None*, *orig_mask=None*, *ref_space=None*, *reg_dir=None*, *data_shape=None*, *affine=None*, ***kwargs*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Indexes a whole-brain pattern with a certain ROI. Given a certain ROI-mask, this class allows transformation from a whole-brain pattern to the mask-subset.

Parameters

- **mvp** (*mvp-object (see scikit_bold.core)*) – Mvp-object, necessary to extract some pattern metadata. If no mvp object has been supplied, you have to set which original mask has been used (e.g. graymatter mask) and what the reference-space is ('epi' or 'mni').
- **mask** (*str*) – Absolute paths to nifti-images of brain masks in MNI152 space
- **mask_threshold** (*Optional[int, float]*) – Threshold to be applied on mask-indexing (given a probabilistic mask).

fit (*X=None*, *y=None*)
Fits RoiIndexer.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*List of str*) – List or ndarray with floats corresponding to labels

transform (*X*, *y=None*)
Transforms features from X (voxels) to a mask-subset.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*Optional[List[str] or numpy ndarray[str]]*) – List of ndarray with strings indicating label-names

Returns **X_new** – array with transformed data of shape = [n_samples, n_features] in which features are region-average values.

Return type ndarray

class RowIndexer (*mvp*, *train_idx*)
Bases: object

Selects a subset of rows from an Mvp object.

Notes

NOT a scikit-learn style transformer.

Parameters

- **idx** (*ndarray*) – Array with indices.
- **mvp** (*mvp-object*) – Mvp-object to drawn metadata from.

transform ()

Returns

- **mvp** (*mvp-object*) – Indexed mvp-object.
- **X_not_selected** (*ndarray*) – Data which has not been selected.
- **y_not_selected** (*ndarray*) – Labels which have not been selected.

class ClusterThreshold (*mvp*, *min_score*, *selector=<function f_classif>*, *min_cluster_size=20*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Implements a cluster-based feature selection method. This feature selection method performs a univariate feature selection method to yield a set of voxels which are then cluster-thresholded using a minimum (contiguous) cluster size. These clusters are then averaged to yield a set of cluster-average features. This method is described in detail in my master's thesis: github.com/lukassnoek/MSc_thesis.

Parameters

- **transformer** (*scikit-learn style transformer class*) – transformer class used to perform some kind of univariate feature selection.
- **mvp** (*Mvp-object (see core.mvp)*) – Necessary to provide mask metadata (index, shape).
- **min_cluster_size** (*int*) – minimum cluster size to be set for cluster-thresholding

fit(*X*, *y*, **args*)
Fits ClusterThreshold transformer.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*List[str] or numpy ndarray[str]*) – List of ndarray with floats corresponding to labels

transform(*X*)
Transforms a pattern (X) given the indices calculated during fit().

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X_cl** – Transformed array of shape = [n_samples, n_clusters] given the indices calculated during fit().

Return type ndarray

class SelectFeatureset(*mvp*, *featureset_idx*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Selects only columns of a certain featureset. CANNOT be used in a scikit-learn pipeline!

Parameters

- **mvp** (*mvp-object*) – Used to extract meta-data.
- **featureset_idx** (*ndarray*) – Array with indices which map to unique feature-set voxels.

fit()
Does nothing, but included due to scikit-learn API.

transform(*X=None*)
Transforms mvp.

class IncrementalFeatureCombiner(*scores*, *cutoff*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Indexes a set of features with a number of (sorted) features.

Parameters

- **scores** (*ndarray*) – Array of shape = n_features, or [n_features, n_class] in case of soft/hard voting in, e.g., a roi_stacking_classifier (see classifiers.roi_stacking_classifier).
- **cutoff** (*int or float*) – If int, it refers the absolute number of features included, sorted from high to low (w.r.t. scores). If float, it selects a proportion of features.

fit(*X*, *y=None*)
Fits IncrementalFeatureCombiner transformer.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

transform(*X*, *y=None*)
Transforms a pattern (X) given the indices calculated during fit().

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the indices calculated during fit().

Return type ndarray

Submodules

skbold.feature_extraction.transformers module

class ArrayPermuter
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Permutes (shuffles) rows of matrix.

__init__()
Initializes ArrayPermuter object.

fit(*X=None*, *y=None*)
Does nothing, but included to be used in sklearn's Pipeline.

transform(*X*)
Permutes rows of data input.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X_new** – ndarray with permuted rows

Return type ndarray

class AverageRegionTransformer(*atlas='HarvardOxford-All'*, *mask_threshold=0*, *mvp=None*, *reg_dir=None*, *orig_mask=None*, *data_shape=None*, *ref_space=None*, *affine=None*, ***kwargs*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Transforms a whole-brain voxel pattern into a region-average pattern Computes the average from different regions from a given parcellation and returns those as features for X.

Parameters

- **mask_type** (*List[str]*) – List with absolute paths to nifti-images of brain masks in MNI152 (2mm) space.
- **mvp** (*Mvp-object (see core.mvp)*) – Mvp object that provides some metadata about previous masks
- **mask_threshold** (*int (default: 0)*) – Minimum threshold for probabilistic masks (such as Harvard-Oxford)

fit(*X=None*, *y=None*)
Does nothing, but included to be used in sklearn's Pipeline.

transform(*X*, *y=None*)
Transforms features from X (voxels) to region-average features.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

- **y** (*Optional[List[str] or numpy ndarray[str]]*) – List of ndarray with strings indicating label-names

Returns **X_new** – array with transformed data of shape = [n_samples, n_features] in which features are region-average values.

Return type ndarray

class **ClusterThreshold**(*mvp*, *min_score*, *selector=<function f_classif>*, *min_cluster_size=20*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Implements a cluster-based feature selection method. This feature selection method performs a univariate feature selection method to yield a set of voxels which are then cluster-thresholded using a minimum (contiguous) cluster size. These clusters are then averaged to yield a set of cluster-average features. This method is described in detail in my master's thesis: github.com/lukassnoek/MSc_thesis.

Parameters

- **transformer** (*scikit-learn style transformer class*) – transformer class used to perform some kind of univariate feature selection.
- **mvp** (*Mvp-object (see core.mvp)*) – Necessary to provide mask metadata (index, shape).
- **min_cluster_size** (*int*) – minimum cluster size to be set for cluster-thresholding

fit(*X*, *y*, **args*)

Fits ClusterThreshold transformer.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*List[str] or numpy ndarray[str]*) – List of ndarray with floats corresponding to labels

transform(*X*)

Transforms a pattern (X) given the indices calculated during fit().

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X_cl** – Transformed array of shape = [n_samples, n_clusters] given the indices calculated during fit().

Return type ndarray

class **IncrementalFeatureCombiner**(*scores*, *cutoff*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Indexes a set of features with a number of (sorted) features.

Parameters

- **scores** (*ndarray*) – Array of shape = n_features, or [n_features, n_class] in case of soft/hard voting in, e.g., a roi_stacking_classifier (see classifiers.roi_stacking_classifier).
- **cutoff** (*int or float*) – If int, it refers the absolute number of features included, sorted from high to low (w.r.t. scores). If float, it selects a proportion of features.

fit(*X*, *y=None*)

Fits IncrementalFeatureCombiner transformer.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

transform(*X*, *y=None*)

Transforms a pattern (X) given the indices calculated during fit().

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the indices calculated during fit().

Return type ndarray

class **PCAfilter**(*n_components=5*, *reject=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Filters out a (set of) PCA component(s) and transforms it back to original representation.

Parameters

- **n_components** (*int*) – number of components to retain.
- **reject** (*list*) – Indices of components which should be additionally removed.

Variables **pca** (*scikit-learn PCA object*) – Fitted PCA object.

fit(*X*, *y=None*, **args*)

Fits PcaFilter.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*List of str*) – List or ndarray with floats corresponding to labels

transform(*X*)

Transforms a pattern (X) by the inverse PCA transform with removed components.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the PCA calculated during fit().

Return type ndarray

class **PatternAverager**(*method='mean'*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Reduces the set of features to its average.

Parameters **method** (*str*) – method of averaging (either 'mean' or 'median')

fit(*X=None*, *y=None*)

Does nothing, but included to be used in sklearn's Pipeline.

transform(*X*)

Transforms patterns to its average.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X_new** – Transformed ndarray of shape = [n_samples, 1]

Return type ndarray

class **RoiIndexer**(*mask*, *mask_threshold=0*, *mvp=None*, *orig_mask=None*, *ref_space=None*, *reg_dir=None*, *data_shape=None*, *affine=None*, ***kwargs*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Indexes a whole-brain pattern with a certain ROI. Given a certain ROI-mask, this class allows transformation from a whole-brain pattern to the mask-subset.

Parameters

- **mvp** (*mvp-object (see scikit_bold.core)*) – Mvp-object, necessary to extract some pattern metadata. If no mvp object has been supplied, you have to set which original mask has been used (e.g. graymatter mask) and what the reference-space is ('epi' or 'mni').
- **mask** (*str*) – Absolute paths to nifti-images of brain masks in MNI152 space
- **mask_threshold** (*Optional[int, float]*) – Threshold to be applied on mask-indexing (given a probabilistic mask).

fit (*X=None*, *y=None*)
Fits RoiIndexer.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*List of str*) – List or ndarray with floats corresponding to labels

transform (*X*, *y=None*)
Transforms features from X (voxels) to a mask-subset.

Parameters

- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]
- **y** (*Optional[List[str] or numpy ndarray[str]]*) – List of ndarray with strings indicating label-names

Returns **X_new** – array with transformed data of shape = [n_samples, n_features] in which features are region-average values.

Return type ndarray

class RowIndexer (*mvp*, *train_idx*)
Bases: object

Selects a subset of rows from an Mvp object.

Notes

NOT a scikit-learn style transformer.

Parameters

- **idx** (*ndarray*) – Array with indices.
- **mvp** (*mvp-object*) – Mvp-object to drawn metadata from.

transform ()

Returns

- **mvp** (*mvp-object*) – Indexed mvp-object.
- **X_not_selected** (*ndarray*) – Data which has not been selected.
- **y_not_selected** (*ndarray*) – Labels which have not been selected.

class SelectFeatureset (*mvp*, *featureset_idx*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Selects only columns of a certain featureset. CANNOT be used in a scikit-learn pipeline!

Parameters

- **mvp** (*mvp-object*) – Used to extract meta-data.

- **featureset_idx** (*ndarray*) – Array with indices which map to unique feature-set voxels.

fit()
Does nothing, but included due to scikit-learn API.

transform(*X=None*)
Transforms mvp.

skbold.feature_selection package

The transformer subpackage provides several scikit-learn style transformers that perform feature selection and/or extraction of multivoxel fMRI patterns. Most of them are specifically constructed with fMRI data in mind, and thus often need an Mvp object during initialization to extract necessary metadata. All comply with the scikit-learn API, using fit() and transform() methods.

class GenericUnivariateSelect(*score_func=<function f_classif>*, *mode='percentile'*, *param=1e-05*)
Bases: sklearn.feature_selection.univariate_selection._BaseFilter

Univariate feature selector with configurable strategy.

Updated version from scikit-learn: http://scikit-learn.org/‘.

Parameters

- **score_func** (*callable*) – Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues). For modes ‘percentile’ or ‘kbest’ it can return a single array scores.
- **mode** (*{'percentile', 'k_best', 'fpr', 'fdr', 'fwe', 'cutoff'}*) – Feature selection mode.
- **param** (*float or int depending on the feature selection mode*) – Parameter of the corresponding mode.

Variables

- **scores** (*array-like, shape=(n_features,)*) – Scores of features.
- **pvalues** (*array-like, shape=(n_features,)*) – p-values of feature scores, None if *score_func* returned scores only.

class SelectAboveCutoff(*cutoff*, *score_func=<function f_classif>*)
Bases: sklearn.feature_selection.univariate_selection._BaseFilter

Filter: Select features with a score above some cutoff.

Parameters

- **cutoff** (*int/float*) – Cutoff for feature-scores to be selected.
- **score_func** (*callable*) – Function that takes a 2D array X (samples x features) and returns a score reflecting a univariate difference (higher is better).

fisher_criterion_score(*X*, *y*, *norm='l1'*, *balance=False*)
Calculates fisher score.

See [1]_ for more info.

References

[1] P. E. H. R. O. Duda and D. G. Stork. Pattern Classification. Wiley-Interscience Publication, 2001.

Parameters

- **X** (*{array-like, sparse matrix} shape = (n_samples, n_features)*) – The set of regressors that will be tested sequentially.
- **y** (*array of shape(n_samples)*) – The data matrix
- **norm** (*str*) – Whether to use the l1-norm or l2-norm.

Returns **scores_** – Fisher criterion scores for each feature.

Return type array, shape=(n_features,)

Submodules

skbold.feature_selection.filters module

class **GenericUnivariateSelect**(*score_func=<function f_classif>*, *mode='percentile'*, *param=1e-05*)

Bases: sklearn.feature_selection.univariate_selection._BaseFilter

Univariate feature selector with configurable strategy.

Updated version from scikit-learn: http://scikit-learn.org/‘.

Parameters

- **score_func** (*callable*) – Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues). For modes ‘percentile’ or ‘kbest’ it can return a single array scores.
- **mode** (*{'percentile', 'k_best', 'fpr', 'fdr', 'fwe', 'cutoff'}*) – Feature selection mode.
- **param** (*float or int depending on the feature selection mode*) – Parameter of the corresponding mode.

Variables

- **scores** (*array-like, shape=(n_features,)*) – Scores of features.
- **pvalues** (*array-like, shape=(n_features,)*) – p-values of feature scores, None if *score_func* returned scores only.

class **SelectAboveCutoff**(*cutoff*, *score_func=<function f_classif>*)

Bases: sklearn.feature_selection.univariate_selection._BaseFilter

Filter: Select features with a score above some cutoff.

Parameters

- **cutoff** (*int/float*) – Cutoff for feature-scores to be selected.
- **score_func** (*callable*) – Function that takes a 2D array X (samples x features) and returns a score reflecting a univariate difference (higher is better).

skbold.feature_selection.selectors module

fisher_criterion_score(*X*, *y*, *norm='l1'*, *balance=False*)

Calculates fisher score.

See [1]_ for more info.

References

[1] P. E. H. R. O. Duda and D. G. Stork. Pattern Classification. Wiley-Interscience Publication, 2001.

Parameters

- **X** (*{array-like, sparse matrix} shape = (n_samples, n_features)*) – The set of regressors that will be tested sequentially.
- **y** (*array of shape(n_samples)*) – The data matrix
- **norm** (*str*) – Whether to use the l1-norm or l2-norm.

Returns **scores_** – Fisher criterion scores for each feature.

Return type array, shape=(n_features,)

skbold.pipelines package

The pipelines module contains some standard MVPA pipelines using the scikit-learn style Pipeline objects.

create_ftest_kbest_svm(*kernel='linear'*, *k=100*, ***kwargs*)
Creates an svm-pipeline with f-test feature selection.

Uses SelectKBest from scikit-learn.feature_selection.

Parameters

- **kernel** (*str*) – Kernel for SVM (default: 'linear')
- **k** (*int*) – How many voxels to select (from the k best)
- ****kwargs** – Arbitrary keyword arguments for SVC() initialization.

Returns **ftest_svm** – Pipeline with f-test feature selection and svm.

Return type scikit-learn Pipeline object

create_ftest_percentile_svm(*kernel='linear'*, *perc=10*, ***kwargs*)
Creates an svm-pipeline with f-test feature selection.

Uses SelectPercentile from scikit-learn.feature_selection.

Parameters

- **kernel** (*str*) – Kernel for SVM (default: 'linear')
- **perc** (*int or float*) – Percentage of voxels to select
- ****kwargs** – Arbitrary keyword arguments for SVC() initialization.

Returns **ftest_svm** – Pipeline with f-test feature selection and svm.

Return type scikit-learn Pipeline object

create_pca_svm(*kernel='linear'*, *n_comp=10*, *whiten=False*, ***kwargs*)
Creates an svm-pipeline with f-test feature selection.

Parameters

- **kernel** (*str*) – Kernel for SVM (default: 'linear')
- **n_comp** (*int*) – How many PCA-components to select
- **whiten** (*bool*) – Whether to use whitening in PCA
- ****kwargs** – Arbitrary keyword arguments for SVC() initialization.

Returns **pca_svm** – Pipeline with PCA feature extraction and svm.

Return type scikit-learn Pipeline object

skbold.postproc package

The postproc subpackage contains all off skbold's 'postprocessing' tools. Most prominently, it contains the MvpResults objects (both MvpResultsClassification and MvpResultsRegression) which can be used in analyses to keep track of model performance across iterations/folds (in cross-validation). Additionally, it allows for keeping track of feature-scores (e.g. f-values from the univariate feature selection procedure) or model weights (e.g. SVM-coefficients). These coefficients can kept track of as raw weights[1] or as 'forward-transformed' weights[2].

The postproc subpackage additionally contains the function 'extract_roi_info', which allows to calculate the amount of voxels (and other statistics) per ROI in a single statistical brain map and output a csv-file.

The cluster_size_threshold function allows you to set voxels to zero which do not belong to a cluster of a given extent/size. This is NOT a statistical procedure (like GRF thresholding), but merely a tool for visualization purposes.

References

R., and Turner, R. (2014). Prioritizing spatial accuracy in high-resolution fMRI data using multivariate feature weight mapping. Front. Neurosci., http://dx.doi.org/10.3389/fnins.2014.00066.

Blankertz, B., and Biessmann, F. et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. Neuroimage, 87, 96-110.

extract_roi_info(*statfile*, *stat_name=None*, *roi_type='unilateral'*, *per_cluster=True*, *cluster_engine='scipy'*, *min_clust_size=20*, *stat_threshold=None*, *mask_threshold=20*, *save_indices=True*, *verbose=True*)

Extracts information per ROI for a given statistics-file. Reads in a thresholded (!) statistics-file (such as a thresholded z- or t-stat from a FSL first-level directory) and calculates for a set of ROIs the number of significant voxels included and its maximum value (+ coordinates). Saves a csv-file in the same directory as the statistics-file. Assumes that the statistics file is in MNI152 2mm space.

Parameters

- **statfile** (*str*) – Absolute path to statistics-file (nifti) that needs to be evaluated.
- **stat_name** (*str*) – Name for the contrast/stat-file that is being analyzed.
- **roi_type** (*str*) – Whether to use unilateral or bilateral masks (thus far, only Harvard-Oxford atlas masks are supported.)
- **per_cluster** (*bool*) – Whether to evaluate the statistics-file as a whole (per_cluster=False) or per cluster separately (per_cluster=True).
- **cluster_engine** (*str*) – Which 'engine' to use for clustering; can be 'scipy' (default), using scipy.ndimage.measurements.label, or 'fsl' (using FSL's cluster commmand).
- **min_clust_size** (*int*) – Minimum cluster size (i.e. clusters with fewer voxels than this number are discarded; also, ROIs containing fewer voxels than this will not be listed on the CSV.
- **stat_threshold** (*int or float*) – If the stat-file contains uncorrected data, stat_threshold can be used to set a lower bound.

[1] Stelzer, J., Buschmann, T., Lohmann, G., Margulies, D.S., Trampel,
[2] Haufe, S., Meineck, F., Gorger, K., Dahne, S., Haynes, J-D.,

- **mask_threshold** (*bool*) – Threshold for probabilistics masks, such as the Harvard-Oxford masks. Default of 25 is chosen as this minimizes overlap between adjacent masks while still covering most of the entire brain.
- **save_indices** (*bool*) – Whether to save the indices (coordinates) of peaks of clusters.
- **verbose** (*bool*) – Whether to print some output regarding the parsing process.

Returns **df** – Dataframe corresponding to the written csv-file.

Return type Dataframe

class **MvpResultsClassification**(*mvp*, *n_iter*, *feature_scoring='fwm'*, *verbose=False*, *out_path=None*)

Bases: *skbold.postproc.mvp_results.MvpResults*

MvpResults class specifically for classification analyses.

Parameters

- **mvp** (*mvp-object*) – Necessary to extract some metadata from.
- **n_iter** (*int*) – Number of folds that will be kept track of.
- **out_path** (*str*) – Path to save results to.
- **feature_scoring** (*str*) – Which method to use to calculate feature-scores with. Can be: 1) 'coef': keep track of raw voxel-weights (coefficients) 2) 'forward': transform raw voxel-weights to corresponding forward- model (see Haufe et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. Neuroimage, 87, 96-110.)
- **verbose** (*bool*) – Whether to print extra output.

compute_scores()

Computes scores across folds.

update(*test_idx*, *y_pred*, *pipeline=None*)

Updates with information from current fold.

Parameters

- **test_idx** (*ndarray*) – Indices of current test-trials.
- **y_pred** (*ndarray*) – Predictions of current test-trials.
- **values** (*ndarray*) – Values of features for model in the current fold. This can be the entire pipeline (in this case, it is extracted automaticlly). When a pipeline is passed, the idx-parameter does not have to be passed.
- **idx** (*ndarray*) – Index mapping the 'values' back to whole-brain space.

class **MvpResultsRegression**(*mvp*, *n_iter*, *feature_scoring=''*, *verbose=False*, *out_path=None*)

Bases: *skbold.postproc.mvp_results.MvpResults*

MvpResults class specifically for Regression analyses.

Parameters

- **mvp** (*mvp-object*) – Necessary to extract some metadata from.
- **n_iter** (*int*) – Number of folds that will be kept track of.
- **out_path** (*str*) – Path to save results to.

- **feature_scoring** (*str*) – Which method to use to calculate feature-scores with. Can be: 1) 'coef': keep track of raw voxel-weights (coefficients) 2) 'forward': transform raw voxel-weights to corresponding forward- model (see Haufe et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. Neuroimage, 87, 96-110.)
- **verbose** (*bool*) – Whether to print extra output.

:param .. warning:: Has not been tested with MvpWithin!:

compute_scores()
Computes scores across folds.

update(*test_idx*, *y_pred*, *pipeline=None*)
Updates with information from current fold.

Parameters

- **test_idx** (*ndarray*) – Indices of current test-trials.
- **y_pred** (*ndarray*) – Predictions of current test-trials.
- **pipeline** (*scikit-learn Pipeline object*) – pipeline from which relevant scores/coefficients will be extracted.

class MvpAverageResults(*out_dir*, *type='classification'*)
Bases: object

Averages results from MVPA analyses on, for example, different subjects or different ROIs.

Parameters **out_dir** (*str*) – Absolute path to directory where the results will be saved.

compute(*mvp_list*, *identifiers*, *metric='f1'*, *h0=0.5*)

write(*path*, *name='average_results'*)

cluster_size_threshold(*data*, *thresh=None*, *min_size=20*, *save=False*)
Removes clusters smaller than a prespecified number in a stat-file.

Parameters

- **data** (*numpy-array or str*) – 3D Numpy-array with statistic-value or a string to a path pointing to a nifti-file with statistic values.
- **thresh** (*int, float*) – Initial threshold to binarize the image and extract clusters.
- **min_size** (*int*) – Minimum size (i.e. amount of voxels) of cluster. Any cluster with fewer voxels than this amount is set to zero ('removed').
- **save** (*bool*) – If data is a file-path, this parameter determines whether the cluster- corrected file is saved to disk again.

Submodules

skbold.postproc.extract_roi_info module

extract_roi_info(*statfile*, *stat_name=None*, *roi_type='unilateral'*, *per_cluster=True*, *cluster_engine='scipy'*, *min_clust_size=20*, *stat_threshold=None*, *mask_threshold=20*, *save_indices=True*, *verbose=True*)
Extracts information per ROI for a given statistics-file. Reads in a thresholded (!) statistics-file (such as a thresholded z- or t-stat from a FSL first-level directory) and calculates for a set of ROIs the number of significant

voxels included and its maximum value (+ coordinates). Saves a csv-file in the same directory as the statistics-file. Assumes that the statistics file is in MNI152 2mm space.

Parameters

- **statfile** (*str*) – Absolute path to statistics-file (nifti) that needs to be evaluated.
- **stat_name** (*str*) – Name for the contrast/stat-file that is being analyzed.
- **roi_type** (*str*) – Whether to use unilateral or bilateral masks (thus far, only Harvard-Oxford atlas masks are supported.)
- **per_cluster** (*bool*) – Whether to evaluate the statistics-file as a whole (per_cluster=False) or per cluster separately (per_cluster=True).
- **cluster_engine** (*str*) – Which 'engine' to use for clustering; can be 'scipy' (default), using scipy.ndimage.measurements.label, or 'fsl' (using FSL's cluster commmand).
- **min_clust_size** (*int*) – Minimum cluster size (i.e. clusters with fewer voxels than this number are discarded; also, ROIs containing fewer voxels than this will not be listed on the CSV.
- **stat_threshold** (*int or float*) – If the stat-file contains uncorrected data, stat_threshold can be used to set a lower bound.
- **mask_threshold** (*bool*) – Threshold for probabilistics masks, such as the Harvard-Oxford masks. Default of 25 is chosen as this minimizes overlap between adjacent masks while still covering most of the entire brain.
- **save_indices** (*bool*) – Whether to save the indices (coordinates) of peaks of clusters.
- **verbose** (*bool*) – Whether to print some output regarding the parsing process.

Returns **df** – Dataframe corresponding to the written csv-file.

Return type Dataframe

skbold.postproc.mvp_results module

class **MvpAverageResults** (*out_dir*, *type='classification'*)
Bases: object

Averages results from MVPA analyses on, for example, different subjects or different ROIs.

Parameters **out_dir** (*str*) – Absolute path to directory where the results will be saved.

compute (*mvp_list*, *identifiers*, *metric='f1'*, *h0=0.5*)

write (*path*, *name='average_results'*)

class **MvpResults** (*mvp*, *n_iter*, *out_path=None*, *feature_scoring=''*, *verbose=False*)
Bases: object

Class to keep track of model evaluation metrics and feature scores. See the ReadTheDocs homepage for more information on its API and use.

Parameters

- **mvp** (*mvp-object*) – Necessary to extract some metadata from.
- **n_iter** (*int*) – Number of folds that will be kept track of.
- **out_path** (*str*) – Path to save results to.

- **feature_scoring** (*str*) – Which method to use to calculate feature-scores with. Can be: 1) 'fwm': feature weight mapping[1] - keep track of raw voxel-weights (coefficients) 2) 'forward': transform raw voxel-weights to corresponding forward- model[2].
- **verbose** (*bool*) – Whether to print extra output.

References

load_model (*path*, *param=None*)
Load model or pipeline from disk.

Parameters

- **path** (*str*) – Absolute path to model.
- **param** (*str*) – Which, if any, specific param needs to be loaded.

save_model (*model*)
Method to serialize model(s) to disk.

Parameters **model** (*pipeline or scikit-learn object.*) – Model to be saved.

write (*feature_viz=True*, *confmat=True*, *to_tstat=True*, *multiclass='ovr'*)
Writes results to disk.

Parameters **to_tstat** (*bool*) – Whether to convert averaged coefficients to t-tstats (by dividing them by sqrt(coefs.std(axis=0)).

class **MvpResultsClassification** (*mvp*, *n_iter*, *feature_scoring='fwm'*, *verbose=False*, *out_path=None*)
Bases: *skbold.postproc.mvp_results.MvpResults*

MvpResults class specifically for classification analyses.

Parameters

- **mvp** (*mvp-object*) – Necessary to extract some metadata from.
- **n_iter** (*int*) – Number of folds that will be kept track of.
- **out_path** (*str*) – Path to save results to.
- **feature_scoring** (*str*) – Which method to use to calculate feature-scores with. Can be: 1) 'coef': keep track of raw voxel-weights (coefficients) 2) 'forward': transform raw voxel-weights to corresponding forward- model (see Haufe et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. Neuroimage, 87, 96-110.)
- **verbose** (*bool*) – Whether to print extra output.

compute_scores ()
Computes scores across folds.

update (*test_idx*, *y_pred*, *pipeline=None*)
Updates with information from current fold.

Parameters

- **test_idx** (*ndarray*) – Indices of current test-trials.

[1] Stelzer, J., Buschmann, T., Lohmann, G., Margulies, D.S., Trampel, R., and Turner, R. (2014). Prioritizing spatial accuracy in high-resolution fMRI data using multivariate feature weight mapping. Front. Neurosci., http://dx.doi.org/10.3389/fnins.2014.00066.

[2] Haufe, S., Meineck, F., Gorger, K., Dahne, S., Haynes, J-D., Blankertz, B., and Biessmann, F. et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. Neuroimage, 87, 96-110.

- **y_pred** (*ndarray*) – Predictions of current test-trials.
- **values** (*ndarray*) – Values of features for model in the current fold. This can be the entire pipeline (in this case, it is extracted automaticlly). When a pipeline is passed, the idx-parameter does not have to be passed.
- **idx** (*ndarray*) – Index mapping the 'values' back to whole-brain space.

class MvpResultsRegression(*mvp*, *n_iter*, *feature_scoring=''*, *verbose=False*, *out_path=None*)
Bases: *skbold.postproc.mvp_results.MvpResults*

MvpResults class specifically for Regression analyses.

Parameters

- **mvp** (*mvp-object*) – Necessary to extract some metadata from.
- **n_iter** (*int*) – Number of folds that will be kept track of.
- **out_path** (*str*) – Path to save results to.
- **feature_scoring** (*str*) – Which method to use to calculate feature-scores with. Can be: 1) 'coef': keep track of raw voxel-weights (coefficients) 2) 'forward': transform raw voxel-weights to corresponding forward- model (see Haufe et al. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. Neuroimage, 87, 96-110.)
- **verbose** (*bool*) – Whether to print extra output.

:param .. warning:: Has not been tested with MvpWithin!:

compute_scores()
Computes scores across folds.

update(*test_idx*, *y_pred*, *pipeline=None*)
Updates with information from current fold.

Parameters

- **test_idx** (*ndarray*) – Indices of current test-trials.
- **y_pred** (*ndarray*) – Predictions of current test-trials.
- **pipeline** (*scikit-learn Pipeline object*) – pipeline from which relevant scores/coefficients will be extracted.

skbold.preproc package

class LabelFactorizer(*grouping*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Transforms labels according to a given factorial grouping.

Factorizes/encodes labels based on part of the string label. For example, the label-vector ['A_1', 'A_2', 'B_1', 'B_2'] can be grouped based on letter (A/B) or number (1/2).

Parameters **grouping** (*List of str*) – List with identifiers for condition names as strings

Variables **new_labels** (*list*) – List with new labels.

fit(*y=None*, *X=None*)
Does nothing, but included to be used in sklearn's Pipeline.

get_new_labels()
Returns new labels based on factorization.

transform(*y*, *X=None*)
Transforms label-vector given a grouping.

Parameters

- **y** (*List/ndarray of str*) – List of ndarray with strings indicating label-names
- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns

- **y_new** (*ndarray*) – array with transformed y-labels
- **X_new** (*ndarray*) – array with transformed data of shape = [n_samples, n_features] given new factorial grouping/design.

class MajorityUndersampler(*verbose=False*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Undersamples the majority-class(es) by selecting random samples.

Parameters **verbose** (*bool*) – Whether to print downsamples number of samples.

__init__(*verbose=False*)
Initializes MajorityUndersampler object.

fit(*X=None*, *y=None*)
Does nothing, but included for scikit-learn pipelines.

transform(*X*, *y*)
Downsamples majority-class(es).

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the indices calculated during fit().

Return type ndarray

class LabelBinarizer(*params*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

__init__(*params*)
Initializes LabelBinarizer object.

fit(*X=None*, *y=None*)
Does nothing, but included for scikit-learn pipelines.

transform(*X*, *y*)
Binarizes y-attribute.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the indices calculated during fit().

Return type ndarray

Submodules

skbold.preproc.label_preproc module

class LabelBinarizer(*params*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

__init__ (*params*)
Initializes LabelBinarizer object.

fit (*X=None*, *y=None*)
Does nothing, but included for scikit-learn pipelines.

transform (*X*, *y*)
Binarizes y-attribute.

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the indices calculated during fit().

Return type ndarray

class LabelFactorizer (*grouping*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Transforms labels according to a given factorial grouping.

Factorizes/encodes labels based on part of the string label. For example, the label-vector ['A_1', 'A_2', 'B_1', 'B_2'] can be grouped based on letter (A/B) or number (1/2).

Parameters **grouping** (*List of str*) – List with identifiers for condition names as strings

Variables **new_labels** (*list*) – List with new labels.

fit (*y=None*, *X=None*)
Does nothing, but included to be used in sklearn's Pipeline.

get_new_labels ()
Returns new labels based on factorization.

transform (*y*, *X=None*)
Transforms label-vector given a grouping.

Parameters

- **y** (*List/ndarray of str*) – List of ndarray with strings indicating label-names
- **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns

- **y_new** (*ndarray*) – array with transformed y-labels
- **X_new** (*ndarray*) – array with transformed data of shape = [n_samples, n_features] given new factorial grouping/design.

class MajorityUndersampler (*verbose=False*)
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Undersamples the majority-class(es) by selecting random samples.

Parameters **verbose** (*bool*) – Whether to print downsamples number of samples.

__init__ (*verbose=False*)
Initializes MajorityUndersampler object.

fit (*X=None*, *y=None*)
Does nothing, but included for scikit-learn pipelines.

transform (*X*, *y*)
Downsamples majority-class(es).

Parameters **X** (*ndarray*) – Numeric (float) array of shape = [n_samples, n_features]

Returns **X** – Transformed array of shape = [n_samples, n_features] given the indices calculated during fit().

Return type ndarray

skbold.utils package

The utils subpackage contains some extra utilities for machine learning pipelines on fMRI data. Most notably, the CrossvalSplitter class allows for the construction of counterbalanced splits between train- and test-sets (e.g. counterbalancing a certain confounding variable in the train-set and between the train- and test-set).

More information can be found on the homepage of ReadTheDocs.

To do: - extend crossvalsplitter to create 3 groups (train, cv, test)

sort_numbered_list(*stat_list*)
Sorts a list containing numbers.

Sorts list with paths to statistic files (e.g. COPEs, VARCOPES), which are often sorted wrong (due to single and double digits). This function extracts the numbers from the stat files and sorts the original list accordingly.

Parameters **stat_list** (*list or str*) – list with absolute paths to files

Returns **sorted_list** – sorted stat_list

Return type list of str

class CrossvalSplitter(*data*, *train_size*, *vars*, *cb_between_splits=False*, *binarize=None*, *include=None*, *exclude=None*, *interactions=True*, *sep='t'*, *index_col=0*, *ignore=None*, *iterations=1000*)
Bases: object

plot_results(*out_dir*)

save(*out_dir*, *save_plots=True*)

split(*verbose=False*)

parse_roi_labels(*atlas_type='Talairach'*, *lateralized=False*, *debug=False*)
Parses xml-files belonging to FSL atlases.

Parameters

- **atlas_type** (*str*) – String identifying which atlas needs to be parsed.
- **lateralized** (*bool*) – Whether to use the lateralized version of the atlas (only applicable to HarvardOxford masks)

Returns **info_dict** – Dictionary with indices and coordinates (values) per ROI (keys).

Return type dict

Submodules

skbold.utils.crossval_splitter module

class CrossvalSplitter(*data*, *train_size*, *vars*, *cb_between_splits=False*, *binarize=None*, *include=None*, *exclude=None*, *interactions=True*, *sep='t'*, *index_col=0*, *ignore=None*, *iterations=1000*)
Bases: object

plot_results(*out_dir*)

save (*out_dir*, *save_plots=True*)

split (*verbose=False*)

skbold.utils.load_roi_mask module

load_nifti_and_check_space (*nifti*, *reg_dir*, *return_array=True*, ***kwargs*)

load_roi_mask (*roi_name*, *atlas_name='HarvardOxford-Cortical'*, *resolution='2mm'*, *lateralized=False*, *which_hemifield=None*, *threshold=0*, *maxprob=False*, *yeo_conservative=False*, *reg_dir=None*, *verbose=False*)

Loads a mask (from an atlas).

Parameters

- **roi_name** (*str*) – Name of the ROI (as specified in the FSL XML-files)
- **atlas_name** (*str*) – Name of the atlas. Choose from: 'HarvardOxford-Cortical', 'HarvardOxford-Subcortical', 'MNI', 'JHU-labels', 'JHU-tracts', 'Talairach', 'Yeo2011'.
- **resolution** (*str*) – Resolution of the mask/atlas ('1mm' or '2mm')
- **lateralized** (*bool*) – Whether to use lateralized masks (only available for Harvard-Oxford atlases). If this variable is specified, you have to specify which_hemifield too.
- **which_hemifield** (*str*) – If lateralized is True, then which hemifield should be used?
- **threshold** (*int*) – Threshold for probabilistic masks (everything below this threshold is set to zero before creating the mask).
- **maxprob** (*bool*) – Whether to select only the voxels that have the highest probability of that particular ROI for a given threshold. Setting this option to true ensures that each mask has unique voxels (substantially slows down the function, though).
- **yeo_conservative** (*bool*) – If Yeo2011 atlas is picked, whether the conservative or liberal atlas should be used.
- **reg_dir** (*str*) – Absolute path to directory with registration info (in FSL format), for if you want to automatically warp the mask to native (EPI) space!
- **return_path** (*bool*) – Whether to return the path to the ROI/mask or the loaded corresponding numpy array.

Returns **mask** – Boolean numpy array(s) indicating the ROI-mask(s).

Return type (list of) numpy-array(s)

skbold.utils.parse_roi_labels module

parse_roi_labels (*atlas_type='Talairach'*, *lateralized=False*, *debug=False*)

Parses xml-files belonging to FSL atlases.

Parameters

- **atlas_type** (*str*) – String identifying which atlas needs to be parsed.
- **lateralized** (*bool*) – Whether to use the lateralized version of the atlas (only applicable to HarvardOxford masks)

Returns **info_dict** – Dictionary with indices and coordinates (values) per ROI (keys).

Return type dict

skbold.utils.roi_globals module

skbold.utils.sort_numbered_list module

sort_numbered_list(*stat_list*)
Sorts a list containing numbers.

Sorts list with paths to statistic files (e.g. COPEs, VARCOPES), which are often sorted wrong (due to single and double digits). This function extracts the numbers from the stat files and sorts the original list accordingly.

Parameters **stat_list** (*list or str*) – list with absolute paths to files

Returns **sorted_list** – sorted stat_list

Return type list of str

Python Module Index

S

Index

Symbols

A

B

C

E

F

G

I

L

M

P

R

S

T

U

W