Feature extraction & classification toolbox

(FEATbox documentation)

Radomír Kůs

(September 2016)

FEATbox (Feature Extraction & clAssification Toolbox) is an outcome of attempts to compare feature extraction and selection methods for schizophrenia classification based on magnetic resonance images (MRI) of brains. Thus, the primary focus of the toolbox are various feature extraction techniques, extracting features from 3-D images given in NIfTI format. Namely, Mann-Whitney testing is implemented as a representative of univariate approaches with contrast to multivariate methods such as intersubject PCA (isPCA), the K-SVD algorithm, and pattern-based morphometry (PBM). The extracted features can be either examined more thoroughly or passed to a subsequent leave-one-out cross-validated (LOOCV) linear support vector machine (SVM) classification. Also, several classification measures are implemented in the toolbox for assessing and comparing classification performance of different classification schemes.

Description

Despite its original purpose, the toolbox design enables to use data from various fields and problem domains. The only rule is that the data must be arranged in a matrix with subjects in rows and features in columns. The toolbox provides function to load NIfTI images (either 2-D or 3-D) contained in a folder and put them in a vectorized form in such a matrix. Should the format of data be different, the data matrix must be created manually. As real-world data matrices are often large, the toolbox supplies a dimensionality reduction technique, random projection, in order to reduce computation times.

After data matrix is created, the toolbox can be used either as a feature extraction tool or as a classification tool. Regarding the former, 4 feature extraction functions (one for each of the aforementioned methods) are ready for those who aim to inspect the extracted features more thoroughly or build their own classification scheme.

Speaking about the latter, FEATbox provides 4 classification functions varying in their feature extraction step (again one for each of the aforementioned methods). The output of the functions are classes predicted with SVM classifier. Both feature extraction and classification are nested in a LOOCV loop. Subsequently, the predicted classes can be compared with the true ones in order to receive statistical measures such as classification accuracy, etc. The toolbox also provides a function to visualize the results for one or more classifiers in a multi-bar graph.

Prerequisites

Due to toolbox complexity, the toolbox requires several prerequisites to function properly:

1. NIfTI toolbox ..... for manipulation with data in NIfTI format,
2. ksvdbox by Ron Rubinstein ..... to run the K-SVD algorithm,
3. ompbox by Ron Rubinstein ..... as a pursuit algorithm for ksvdbox.

Their current versions are attached to FEATbox (in the „prerequisites" folder) or they can be downloaded from their respective websites.

When using random projection, make sure MATLAB has assigned enough heap memory as generation of large random projection matrices can be very demanding.

Tutorials

The toolbox comes with example data and 3 tutorial demos. They demonstrate how to:

1. use toolbox build-in functions for simple and automated classification (see demo_class.m in Appendix B: Automated classification tutorial),
2. use toolbox to create own classification scheme (see demo_class_manual.m in Appendix C: Manual classification tutorial)
3. use toolbox to extract and assess features from data (see demo_extraction.m in Appendix D: Feature extraction tutorial).

Notes to all tutorials:

1. In order to run the demo, you must either change „toolbox_path" to location of the demo on your local disk or set your current working folder accordingly.
2. All FEATbox functions are accentuated with white spaces between brackets and arguments in the code (e.g. [output] = func(input1, input2) instead of [output]=func(input1,input2)) so that they can be recognized easily. In order to learn more about the function, please refer to „help *function_name*" or Appendix A: Complete list of FEATbox functions attached below.
3. The example data (with a mask included) are artificial. All demos merely demonstrate how to operate real datasets with the toolbox for feature extraction and classification purposes.

Short list of functions

FEATbox functions are divided into 4 main categories:

1. „ classify-* " functions for automated classification,
2. „ extract-* " functions for feature extraction,
3. other functions serving for data processing and performance evaluation,
4. „ p_* " private functions mainly for inner purposes of the toolbox (p_ prefix = private).

Functions implemented in the toolbox are shortly listed in the following table (Table 1). Detailed description about each of the functions can be found in Appendix A: Complete list of FEATbox functions. Function names in the table serve as hyperlinks to their respective detailed descriptions.

*Table 1: A short list of FEATbox functions (click on *function_name* to see detailed description).*

Function name	Brief description	Input	Output
classify_ispca	Perform a leave-one-out cross-validated linear support vector machine classification based on the most discriminative features extracted with intersubject principal component analysis.	X group_ids c	class_predicts

classify_ksvd	Perform a leave-one-out cross-validated linear support vector machine classification based on the features extracted with the K-SVD algorithm	X group_ids rp_struct a s	class_predicts
classify_mw	Perform a leave-one-out cross-validated linear support vector machine classification based on features selected with Mann-Whitney testing.	X group_ids t	class_predicts
classify_pbm	Perform a leave-one-out cross-validated linear support vector machine classification based on the most discriminative features extracted with pattern-based morphometry.	X group_ids rp_struct a k s	class_predicts
extract_ispca	Transform data into a new feature space spanned by the most discriminative (in terms of ability of distinguishing between the groups in data) isPCA components.	X skip_ids group_ids c	X_ispca X_skipped proj_mat_discr expl_var_discr
extract_ksvd	Transform data into a new feature space spanned by atoms of a dictionary learned by the K-SVD algorithm.	X skip_ids a s	X_ksvd X_skipped dict sparse_coeff
extract_mw	Reduce feature space by selecting the most discriminative (in terms of ability of distinguishing between the groups in data) features on the basis of Mann-Whitney testing.	X skip_ids group_ids t	X_mw X_skipped selected_ids p_MW
extract_pbm	Transform data into a new feature space spanned by atoms of a dictionary learned with pattern-based morphometry.	X skip_ids group_ids a k s	X_pbm X_skipped dict sparse_coeff

create_data_matrix	Create data matrix from NIfTI images (with/without masking) in a folder.	dirname files mask	X n_files n_voxels original_dim
display_nii	Load and display masked/unmasked NIfTI image.	filename mask	
load_prerequisites	Check whether all FEATbox prerequisites are in MATLAB search path and, if not, include them.	dirname remember	
performance_disp	Display classification measures in a grouped multi-bar plot for each classification.	CP measures names f_title f_legend	
performance_eval	Evaluate classification performance.	true_class predictions positive_label	accur sens spec precis cm
random_projection	Reduce matrix dimension with Gaussian or Achlioptas random projection.	X k_desired type	X_red rp_matrix k
p_gen_diff_im	Generate a matrix of a difference images.	group1 group2 k	diff_images
p_ispca	Compute a projection matrix of intersubject principal component analysis.	X	X_proj explained_var
p_preproc	Private function for preprocessing (dividing and centering) data.	X omit_ids group_ids	X_used_c X_omit_c used_g_ids omit_g_ids

Appendix A: Complete list of FEATbox functions

classify_ispca

Perform a leave-one-out cross-validated linear support vector machine classification based on the most discriminative features extracted with intersubject principal component analysis (isPCA).

Syntax:

```
[class_predicts] = classify_ispca(X, group_ids, c)
```

Input arguments:

X ..... data matrix with subjects in rows and features in columns
group_ids ..... vector of group identifiers corresponding to subjects (rows) in X
c ..... number of the most discriminative components to be used for classification; the maximum possible value is N-2, where N is the number of subjects (rows) in X

Output arguments:

class_predicts ..... vector of classes predicted by SVM classifier to each subject

Description:

A linear support vector machine (SVM) classifier is used to perform a leave-one-out cross-validated (CV) classification of subjects given in „X“. In each iteration, the data matrix is divided into training and testing datasets. Next, intersubject PCA [1] is utilized to derive a projection matrix from the training data in order to reduce the dimensionality of the problem. To enhance the classification, p-values resulting from Mann-Whitney testing are exploited to select only „c“ components with the best discriminative power of distinguishing between the groups. Those components span a feature space where the SVM classification takes place. Note that such a CV scheme correctly involves feature extraction in its every step.

Example:

```
predicted_classes = classify_ispca( Subjects, subject_group_ids, 16 );
```

Reference:

[1] E. Janousova, D. Schwarz, and T. Kasparek: „Combining Various Types of Classifiers and Features Extracted from Magnetic Resonance Imaging Data in Schizophrenia Recognition.”, Psychiatry Research: Neuroimaging, Vol. 232, No. 3, 2015, pp. 237–49.

classify_ksvd

Perform a leave-one-out cross-validated linear support vector machine classification based on the features extracted with the K-SVD algorithm.

Syntax:

```
[class_predicts] = classify_ksvd(X, group_ids, rp_struct, a, s)
```

Input arguments:

X ..... data matrix with subjects in rows and features in columns

rp_struct ..... structure array with random projection settings:

- rp_struct.p ..... fraction of columns to retain / desired number of columns in reduced data matrix
- rp_struct.t ..... type of random projection matrix; 'A' (default) for Achlioptas, 'G' for Gaussian

(for more information use „help random_projection“)

a ..... number of atoms in dictionary / dimension of the new feature space; the maximum possible value is N-1, where N is the number of subjects (rows) in X

s ..... sparsity constraint for K-SVD coefficient matrix; default = 5

Output arguments:

class_predicts ..... vector of classes predicted by SVM classifier to each subject

Description:

A linear support vector machine (SVM) classifier is used to perform a leave-one-out cross-validated (CV) classification of subjects given in „X“. In each iteration, the data matrix is divided into training and testing datasets. Next, the training dataset is utilized to learn a data-driven dictionary with the K-SVD algorithm [1]. Atoms in the dictionary span a new feature space where the SVM classification takes place. Note that such a CV scheme correctly involves feature extraction in its every step.

Examples:

```
RP.p = 1/100; % multiplier to reduce the length of descriptors with
RP.t = 'A'; % Achlioptas matrix
predicted_classes = classify_ksvd( Subjects, subject_group_ids, RP, 6, 3 );
predicted_classes = classify_ksvd( Subjects, subject_group_ids, {}, 9 );
```

Reference:

[1] M. Aharon, M. Elad, and A. Bruckstein: „K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation.”, IEEE Transactions on Signal Processing, Vol. 54, No. 11, 2006, pp. 4311–22.

Note:

Requires ksvdbox and ompbox by Ron Rubinstein.

classify_mw

Perform a leave-one-out cross-validated linear support vector machine classification based on features selected with Mann-Whitney testing.

Syntax:

```
[class_predicts] = classify_mw(X, group_ids, t)
```

Input arguments:

X ..... data matrix with subjects in rows and features in columns
group_ids ..... vector of group identifiers corresponding to subjects (rows) in X
t ..... threshold for Mann-Whitney p-values; features with lower p-values than t are selected for the classification

Output arguments:

class_predicts ... vector of classes predicted by SVM classifier to each subject

Description:

A linear support vector machine (SVM) classifier is used to perform a leave-one-out cross-validated (CV) classification of subjects given in „X“. In each iteration, the data matrix is divided into training and testing datasets. Next, the training dataset is subject to Mann-Whitney testing for each feature. The p-values resulting from the testing are exploited to select only the features with a higher discriminative power than given by the threshold „t“. A subsequent SVM classification takes place in the reduced feature space. Note that such a CV scheme correctly involves feature selection in its every step.

Example:

```
predicted_classes = classify_mw(Subjects, subject_group_ids, 0.01);
```

classify_pbm

Perform a leave-one-out cross-validated linear support vector machine classification based on the features extracted with pattern-based morphometry.

Syntax:

```
[class_predicts] = classify_pbm(X, group_ids, rp_struct, a)
[class_predicts] = classify_pbm(X, group_ids, rp_struct, a, k)
[class_predicts] = classify_pbm(X, group_ids, rp_struct, a, k, s)
```

Input arguments:

X ..... data matrix with subjects in rows and features in columns
group_ids ..... vector of group identifiers corresponding to subjects (rows) in X
rp_struct ..... structure array with random projection settings:
- rp_struct.p ..... fraction of columns to retain / desired number of columns in reduced data matrix
- rp_struct.t ..... type of random projection matrix; 'A' (default) for Achlioptas, 'G' for Gaussian

(for more information use „help random_projection“)
a ..... number of atoms in dictionary / dimension of the new feature space; the maximum possible value is (N-1)*k,where N is the number of subjects (rows) in X
k ..... number of nearest neighbours to use when generating difference images; default = 3
(for more information use „help p_gen_diff_im“)
s ..... sparsity constraint for K-SVD coefficient matrix; default = 5

Output arguments:

class_predicts ..... vector of classes predicted by SVM classifier to each subject

Description:

A linear support vector machine (SVM) classifier is used to perform a leave-one-out cross-validated (CV) classification of subjects given in „X“. In each iteration, the data matrix is divided into training and testing datasets. Next, a difference images matrix is generated (see „help p_gen_diff_im“) from dataset of the training dataset. The matrix is utilized to learn a data-driven dictionary with the K-SVD algorithm [1]. Atoms in the dictionary span a new feature space where the SVM classification takes place. Note that such a CV scheme correctly involves feature extraction in its every step.
For further reading on pattern-based morphometry please refer to [2].

Examples:

```
RP.p = 1/100; % multiplier to reduce the length of descriptors with
RP.t = 'A'; % Achlioptas matrix
predicted_classes = classify_pbm(Subjects, subject_group_ids, RP, 49);
predicted_classes = classify_ksvd(Subjects, subject_group_ids, {}, 3, 5, 2);
```

References:

[1] M. Aharon, M. Elad, and A. Bruckstein: „K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation.“, IEEE Transactions on Signal Processing, Vol. 54, No. 11, 2006, pp. 4311–22.

[2] B. Gaonkar, K. Pohl, and C. Davatzikos: „Pattern Based Morphometry.“ Medical Image Computing and Computer-Assisted Intervention, Vol. 14, No. Pt 2, 2011, pp. 459–66.

extract_ispca

Transform data into a new feature space spanned by the most dicriminative (in terms of ability of distinguishing between the groups in data) isPCA components.

Syntax:

```
[X_ispca, X_skipped, proj_mat_discr, expl_var_discr] = extract_ispca(X,
   skip_ids, group_ids, c)
```

Input arguments:

X ..... data matrix with subjects in rows and features in columns

skip_ids ..... vector of subject indices to omit from feature extraction (i.e. rows in X specified by the argument will not be used to derive isPCA projection matrix); to use all the subjects, use an empty vector

group_ids ..... vector of group identifiers corresponding to subjects (rows) in X

c ..... number of the most discriminative components / dimension of the new feature space; the maximum possible value is N-M-1, where N is the number of subjects (rows) in X and M is the number of indices specified in skip_ids

Output arguments:

X_ispca ..... data matrix of subjects not specified in skip_ids (i.e. used in feature extraction) projected onto c most discriminative isPCA components

X_skipped ..... data matrix of subjects specified in skip_ids (i.e. omitted from feature extraction) projected onto c most discriminative isPCA components

proj_mat_discr ..... projection matrix consisting of c most discriminative isPCA components

expl_var_discr ..... vector of relative amount of variability explained by each of c most discriminative isPCA components

Description:

First, subjects specified in „skip_ids“ are taken out of „X“, thus creating a data matrix of retained subjects. Next, intersubject PCA [1] is utilized to derive a projection matrix from the retained data. Subsequently, p-values resulting from Mann-Whitney testing are exploited to select only „c“ components with the best discriminative power of distinguishing between the groups in retained data. Last, both the retained and skipped data are projected onto a feature space spanned by the components.

Examples:

```
X_projected = extract_ispca(X, [], group_ids, 19); % use all subjects
[X_re, X_om, PM, ex_var] = extract_ispca(X, 7, g_ids, 6);
[Subj_proj, T_and_P_proj] = extract_ispca(Subjects,[tested_image paired_image],
   subject_group_ids, 14);
```

Reference:

[1] E. Janousova, D. Schwarz, and T. Kasparek: „Combining Various Types of Classifiers and Features Extracted from Magnetic Resonance Imaging Data in Schizophrenia Recognition.”, Psychiatry Research: Neuroimaging, Vol. 232, No. 3, 2015, pp. 237–49.

extract_ksvd

Transform data into a new feature space spanned by atoms of a dictionary learned by the K-SVD algorithm.

Syntax:

```
[X_ksvd, X_skipped, dict, sparse_coeff] = extract_ksvd(X, skip_ids, a)
[X_ksvd, X_skipped, dict, sparse_coeff] = extract_ksvd(X, skip_ids, a, s)
```

Input arguments:

X ..... data matrix with subjects in rows and features in columns

skip_ids ..... vector of subject indices to omit from feature extraction (i.e. rows in X specified by the argument will not be used to derive K-SVD dictionary); to use all the subjects, use an empty vector

a ..... number of atoms in dictionary / dimension of the new feature space; the maximum possible value is N-M, where N is the number of subjects (rows) in X and M is the number of indices specified in skip_ids

s ..... sparsity constraint for K-SVD coefficient matrix; default = 5

Output arguments:

X_ksvd ..... data matrix of subjects not specified in skip_ids (i.e. used in feature extraction) projected onto K-SVD atoms

X_skipped ..... data matrix of subjects specified in skip_ids (i.e. omitted from feature extraction) projected onto K-SVD atoms

dict ..... K-SVD dictionary (can be considered as a projection matrix)

sparse_coeff ..... sparse coefficient K-SVD matrix

Description:

First, subjects specified in „skip_ids“ are taken out of „X“, thus creating a data matrix of retained subjects. Next, the matrix is utilized to learn a data-driven dictionary with the K-SVD algorithm [1]. Last, both the retained and skipped data are projected onto a feature space spanned by atoms in the dictionary.

Examples:

```
X_pr = extract_ksvd(X, 6, 19);
[X_projected, ~, D, C] = extract_ksvd(X, [], 17); % use all subjects
[Subj_proj, T_and_P_proj] = extract_ksvd(Subjects, [tested_image paired_image], 6,
   3);
```

Reference:

[1] M. Aharon, M. Elad, and A. Bruckstein: „K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation.“, IEEE Transactions on Signal Processing, Vol. 54, No. 11, 2006, pp. 4311–22.

Note:

Requires ksvdbox and ompbox by Ron Rubinstein.

extract_mw

Reduce feature space by selecting the most dicriminative (in terms of ability of distinguishing between the groups in data) features on the basis of Mann-Whitney testing.

Syntax:

```
[X_mw, X_skipped, selected_ids, p_MW] = extract_mw(X, skip_ids, group_ids)
[X_mw, X_skipped, selected_ids, p_MW] = extract_mw(X, skip_ids, group_ids, t)
```

Input arguments:

- X ..... data matrix with subjects in rows and features in columns
- skip_ids ..... vector of subject indices to omit from featureextraction (i.e. rows in X specified by theargument will not be subject to Mann-Whitney testing); to use all the subjects, use an empty vector
- group_ids ..... vector of group identifiers corresponding to subjects (rows) in X
- t ..... threshold for Mann-Whitney p-values; features with lower p-values than t are selected to compose the reduced feature space, the rest are disregarded; default = 0.05

Output arguments:

- X_mw ..... data matrix of subjects not specified in skip_ids (i.e. used in Mann-Whitney testing) in reduced feature space
- X_skipped ..... data matrix of subjects specified in skip_ids (i.e. omitted from Mann-Whitney testing) in reduced feature space
- selected_ids ..... vector of logical values showing whether a feature composes reduced feature space (i.e. was selected) or not
- p_MW ..... vector of p-values resulting from Mann-Whitney testing

Description:

First, subjects specified in „skip_ids“ are taken out of „X“, thus creating a data matrix of retained subjects. Next, the retained data are subject to Mann-Whitney testing for each feature. The resulting p-values are exploited to select only the features with a higher discriminative power than given by the threshold „t“. Subsequently, the features which were not selected in the testing are disregarded from the matrices of both retained and skipped subjects, reducing their dimensionality.

Examples:

```
X_red = extract_mw(X, [], group_ids); % use all subjects
[~, ~, selected_features, p_values] = extract_mw(X, 7, g_ids, 0.01);
[Subj_red, T_and_P_red] = extract_mw(Subjects, [tested_image paired_image],
  subject_group_ids, 0.01);
```

extract_pbm

Transform data into a new feature space spanned by atoms of a dictionary learned with pattern-based morphometry.

Syntax:

```
[X_pbm, X_skipped, dict, sparse_coeff] = extract_pbm(X, skip_ids, group_ids, a)
[X_pbm, X_skipped, dict, sparse_coeff] = extract_pbm(X, skip_ids, group_ids, a, k)
[X_pbm, X_skipped, dict, sparse_coeff] = extract_pbm(X, skip_ids, group_ids, a, k, s)
```

Input arguments:

- X ..... data matrix with subjects in rows and features in columns
- skip_ids ..... vector of subject indices to omit from feature extraction (i.e. rows in X specified by the argument will not be used to derive PBM dictionary); to use all the subjects, use an empty vector
- group_ids ..... vector of group identifiers corresponding to subjects (rows) in X
- a ..... number of atoms in dictionary / dimension of the new feature space; the maximum possible value is (N-M)*k, where N is the number of subjects (rows) in X and M is the number of indices specified in skip_ids
- k ..... number of nearest neighbours to use when generating difference images; default = 3 (for more information use „help p_gen_diff_im“)
- s ..... sparsity constraint for K-SVD coefficient matrix; default = 5

Output arguments:

- X_pbm ..... data matrix of subjects not specified in skip_ids (i.e. used in feature extraction) projected onto PBM atoms
- X_skipped ..... data matrix of subjects specified in skip_ids (i.e. omitted from feature extraction) projected onto PBM atoms
- dict ..... PBM dictionary (can be considered as a projection matrix)
- sparse_coeff ..... sparse coefficient PBM matrix

Description:

First, subjects specified in „skip_ids“ are taken out of „X“, thus creating a data matrix of retained subjects. Next, a difference images matrix is generated (see „help p_gen_diff_im“) from dataset of retained subjects. The matrix is utilized to learn a data-driven dictionary with the K-SVD algorithm [1]. Last, both the retained and skipped data are projected onto a feature space spanned by atoms in the dictionary.

For further reading on pattern-based morphometry please refer to [2].

Examples:

```
X_pr = extract_pbm(X, 6, group_ids, 57);
[X_projected , ~, D, C ] = extract_pbm(X, [], g_ids, 6, 5); % use all subjects
[Subj_proj, T_and_P_proj] = extract_pbm(Subjects, [tested_image paired_image],
   46, 7, 5);
```

References:

[1] M. Aharon, M. Elad, and A. Bruckstein: „K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation.“, IEEE Transactions on Signal Processing, Vol. 54, No. 11, 2006, pp. 4311–22.

[2] B. Gaonkar, K. Pohl, and C. Davatzikos: „Pattern Based Morphometry.“ Medical Image Computing and Computer-Assisted Intervention, Vol. 14, No. Pt 2, 2011, pp. 459–66.

create_data_matrix

Create data matrix from NIfTI images (with/without masking) in a folder.

Syntax:

```
[X, n_files, n_voxels, original_dim] = create_data_matrix(dirname, files)
[X, n_files, n_voxels, original_dim] = create_data_matrix(dirname, files, mask)
```

Input arguments:

dirname ..... name of a folder containing the images
files ..... vector of file names of images in NIfTI format; images must be of the same dimensions
mask ..... mask to be applied to the images; dimensions of the mask and images must agree

Output arguments:

X ..... data matrix with vectorized images in rows
n_files ..... number of processed files (equals number of rows)
n_voxels ..... number of pixels/voxels in one image
original_dim ..... original dimensions of the images

Description:

NIfTI toolbox function „load_nii“ is utilized to load images contained in a folder and assemble them (vectorized) into a matrix as its rows. Moreover, a mask can be applied to the images.

Examples:

```
X = create_data_matrix('C:\Users\...\', [image1.nii image2.nii]);
[X, n_subjects, n_voxels, X_dim] = create_data_matrix('C:\Users\...\',
  [image1.nii image2.nii], mask);
```

Note:

Requires NIfTI toolbox.

display_nii

Load and display masked/unmasked NIfTI image.

Syntax:

```
display_nii(filename)
display_nii(filename, mask)
```

Input arguments:

filename ..... file name of an image in NIfTI format
mask ..... mask to be applied to the image; dimensions of the mask and image specified in filename must agree

Description:

NIfTI toolbox functions „load_nii“ and „view_nii“ are wrapped into a function in order to enable loading and displaying images with one call. Moreover, a mask can be applied to the images.

Example:

```
display_nii('C:\Users\...\image.nii', mask);
```

Note:

Requires NIfTI toolbox.

load_prerequisites

Check whether all FEATbox prerequisites are in MATLAB search path and, if not, include them.

Syntax:

```
load_prerequisites(dirname)
load_prerequisites(dirname, remember)
```

Input arguments:

dirname ..... name of a folder containing FEATbox prerequisites
remember ..... logical value whether to save FEATbox into MATLAB search path permanently or not

Description:

All external functions and toolboxes (namely NIfTI toolbox, ksvdbox, ompbox, multicols.m and normcols.m) are checked to be in MATLAB search path and, if not, they are added. When added permanently („remember“=1), the FEATbox itself is saved in MATLAB search path. Otherwise, FEATbox path must be a current working directory while working with the toolbox.

Examples:

```
load_prerequisites('C:\Users\...\');
load_prerequisites('C:\Users\...\', 1);
```

Note:

The „example_data“ directory is not added to the path as it is not necessary for FEATbox to function.

performance_disp

Display classification measures in a grouped multi-bar plot for each classification.

Syntax:

```
performance_disp(CP, measures)
performance_disp(CP, measures, names)
performance_disp(CP, measures, names, f_title)
```

Input arguments:

CP ..... Classification performance matrix with classification measures (columns) for each classification (rows).

measures ..... vector of 4 logical values implying what statistical measures are in columns of CP:
- accuracy
- sensitivity
- specificity
- precision

e.g. [1, 0, 1, 1] means that CP matrix has 3 columns (accuracy, specificity and precision)

names ..... cell array consisting of names to display on x-axis of the figure corresponding to rows in CP

f_title ..... title of the figure

f_legend ..... location of legend in the figure; when set to 'hide', no legend is displayed; default = 'northeast' (for more information on possible location values use „help legend“ and scroll to „Location“)

Description:

Classification measures for each classifier given in „CP“ are displayed in a grouped multi-bar plot. Required arguments are the classification performance matrix and a vector of statistical measures to display.

Example:

```
CP = [0.8 0.7 0.9; 0.7 0.5 0.9]; % classification performance matrix
measures = [1 1 1 0]; % show accuracy, sensitivity and specificity / disregard
   precision
performance_disp(CP, measures, {'3-NN','5-NN'}, 'My Results', 'northwest');
```

performance_eval

Evaluate classification performance.

Syntax:

```
[accur, sens, spec, precis, cm] = performance_eval(true_class, predictions,
  positive_label)
```

Input arguments:

true_class	vector with true class identifiers
predictions	vector with predicted class identifiers
positive_label	identifier of a group with a positive condition (e.g. patients)

Output arguments:

accur	classification accuracy
sens	sensitivity
spec	specificity
precis	precision
cm	confusion matrix

Description:

Given true and predicted classes, a classification performance can be evaluated.

Examples:

```
[accur, sens, spec, precis, cm] = performance_eval(g_ids, pred_class, 1);
class_accuracy = performance_eval(['A', 'A', 'B', 'A', 'B', 'B'], ['A', 'B', 'A', 'A', 'B', 'B'],
  'B');
```

random_projection

Reduce matrix dimension with Gaussian/Achlioptas random projection.

Syntax:

```
[X_red, rp_matrix, k] = random_projection(X, k_desired)
[X_red, rp_matrix, k] = random_projection(X, k_desired, type)
```

Input arguments:

X ..... data matrix to be reduced [n x m]
k_desired ..... fraction of columns to retain / desired number of columns in reduced data matrix
type ..... type of random projection matrix; 'A' (default) for Achlioptas, 'G' for Gaussian

Output arguments:

X_rp ..... reduced data matrix [n x k]
RP_matrix ..... random projection matrix [m x k]
k ..... number of columns in reduced data matrix

Description:

Random projection is a dimensionality reduction technique which enables reducing dimensionality of a data matrix X [n x m] by multiplying it with a random projection matrix RP [m x k] as follows:

X_red = 1/sqrt(k)*X*RP,

while preserving all pairwise Euclidean distances of X. Such a behaviour is possible when entries of RP matrix follow distribution with zero mean and constant variance (e.g. Gaussian). A simple distribution ('very sparse') complying with the rule was suggested by Achlioptas [1].
For further reading please refer to [2].

Examples:

```
X_red = random_projection(X, 540);
[Subjects_red, RP_matrix, red_n_voxels] = random_projection(Subjects, 1/1000,
   'G');
```

References:

[1] D. Achlioptas. 2003. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of Computer and System Sciences, 66(4), 671–687.

[2] Ping Li, Trevor J. Hastie, and Kenneth W. Church. 2006. Very sparse random projections. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06). ACM, New York, NY, USA, 287-296.

Note:

Make sure MATLAB has assigned enough heap memory as generation of large random projection matrices can be very demanding.

p_gen_diff_im

Generate a matrix of a difference images.

Syntax:

```
[diff_images] = p_gen_diff_im(group1, group2, k)
```

Input arguments:

group1 ..... matrix of subjects from the first group with vectorized subjects in rows and features in columns; the number of features between groups must be the same

group2 ..... matrix of subjects from the first group with vectorized subjects in rows and features in columns; the number of features between groups must be the same

k ..... number of nearest neighbours to use when generating difference images

Output arguments:

diff_images ..... matrix of difference images in rows

Description:

For each image, using the Euclidean distance, a set of its k-nearest neighbors with a different affiliation is found. In other words, for an image a belonging to the group 1, its k most similar images belonging to the group 2 are searched for and vice-versa. Subsequently, the images are subtracted from their neighbors. In the end, the resulting difference images matrices together into a single matrix. Assuming the images are in rows, the new matrix will have k-times more rows than the original matrix.

For further reading please refer to [1].

Example:

```
X = p_gen_diff_im(Patients, Controls, 3);
```

Reference:

[1] B. Gaonkar, K. Pohl, and C. Davatzikos: „Pattern Based Morphometry.” Medical Image Computing and Computer-Assisted Intervention, Vol. 14, No. Pt 2, 2011, pp. 459–66.

p_ispca

Compute a projection matrix of intersubject principal component analysis (isPCA).

Syntax:

```
[X_proj, explained_var] = p_ispca(X)
```

Input arguments:

X ..... centered data matrix with subjects in rows and features in columns

Output arguments:

X_proj ..... isPCA projection matrix consisting of eigenvectors
explained_var ... vector of relative variability explained by eigenvectors in X_proj

Description:

Unlike with PCA, when a projection matrix is based on computation of eigenvectors of covariance matrix of features, the principle of isPCA is to compute eigeinvectors out of intersubject covariance matrix. Those are then transformed [1] into eigenvectors of covariance matrix of features, sorted and assembled together, forming a projection matrix.
For further reading on isPCA please refer to [2].

Example:

```
[A_proj, expl_var] = p_ispca(Subjects);
```

References:

[1] O. Demirci, V.P. Clark, V.A. Magnotta, N.C. Andreasen, J. Lauriello, K.A. Kiehl, G.D. Pearlson, and V.D. Calhoun: „A Review of Challenges in the Use of fMRI for Disease Classification / Characterization and A Projection Pursuit Application from A Multi-Site fMRI Schizophrenia Study.", Brain Imaging and Behavior, Vol. 2, No. 3, 2008, pp. 207–26.

[2] E. Janousova, D. Schwarz, and T. Kasparek: „Combining Various Types of Classifiers and Features Extracted from Magnetic Resonance Imaging Data in Schizophrenia Recognition.", Psychiatry Research: Neuroimaging, Vol. 232, No. 3, 2015, pp. 237–49.

p_preproc

Private function for preprocessing data.

Syntax:

```
[data_used_c, data_omit_c, used_g_ids, omit_g_ids] = p_preproc(data, omit_ids)
[data_used_c, data_omit_c, used_g_ids, omit_g_ids] = p_preproc(data, omit_ids,
   group_ids)
```

Input arguments:

X ..... data matrix to be preprocessed with subjects in rows and features in columns
omit_ids ..... vector of indices pointing to subjects (rows) to omit from the data matrix X
group_ids ..... vector of group identifiers corresponding to subjects (rows) in X

Output arguments:

X_used_c ..... centered data matrix with subjects retained for computation of mean
X_omit_c ..... centered data matrix with subjects specified in omit_ids
used_g_ids ..... group identifiers of subjects retained for computation of mean
omit_g_ids ..... group identifiers of subjects specified in omit_ids

Description:

The data are divided into two datasets -- the subjects to compute a mean from and the subjects taken out of the computation. Both datasets are then centered with the mean. If present, likewise is employed the division of the corresponding group identifiers.

Examples:

```
[X_train, X_test, c_ids_train] = p_preproc(X, tested_image, class_ids);
[X_train, X_skipped] = p_preproc(X, [tested_image paired_image]);
```

Appendix B: Automated classification tutorial

demo_class.m

This demo demonstrates a way to use FEATbox for leave-one-out cross-validated linear support vector machine classification utilizing feature extraction algorithms implemented in the toolbox – namely Mann-Whitney testing, isPCA, K-SVD and pattern-based morphometry (PBM).

First, the toolbox is initialized, example data are loaded and rearranged into a data matrix of subjects in rows and features in columns. The code also shows how masking can be incorporated into data preprocessing and how *.nii images can be visualized (also with masking). Subsequently, for each of the feature extraction algorithms, parameters are set and the classification is employed. In the end, classification performance is evaluated and the results are visually inspected and compared.

Furthermore, random projection reduction can be performed with the use of "random_projection" function. Typically, the data matrix can be firstly reduced and then passed to "extract_*...*" or "classify_*...*" functions to lower computational demands. However, as PBM creates difference images within the "classify_pbm" function, random projection can not take place beforehand. Thus, it can be set in form of a struct in the function call. Possible ways to employ random projection are also demonstrated in the demo on K-SVD classification.

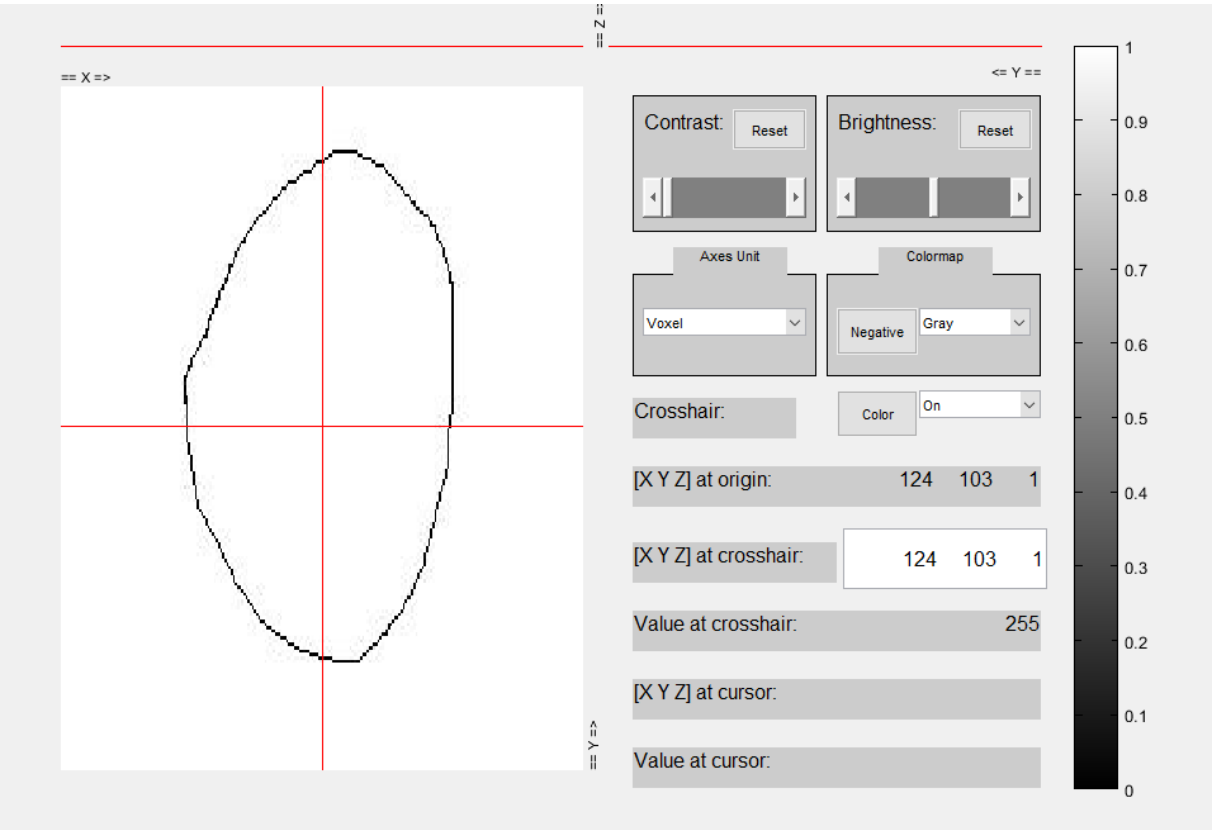
Suggestion: If you uncomment Mann-Whitney testing, the demo would take up to several minutes to compute. Thus, set it off first and take a look at the code (or other demos) while it runs.

1. Initialization

```
toolbox_path = pwd; % set a path to the toolbox
cd(toolbox_path); % change the current working folder
load_prerequisites( [toolbox_path,'\prerequisites'] )

% % optionally, the prerequisites can be saved to the MATLAB search path permanently
% load_prerequisities( [toolbox_path,'\prerequisites'], 1 )
```

2. Data loading & Vizualization

```
data_path = [toolbox_path,'\example_data\'];
[class_ids,files] = xlsread([data_path,'\index_unsorted.xls'],'data','a2:b21'); % read the
indexing table
files = cell2mat(files);

% vizualization
toy_image = 'img24.nii';
display_nii( [data_path,'\',toy_image] ); % display an example image

% % optionally, a mask can be applied to the data
% load([data_path,'\mask.mat']);
% display_nii( [data_path,'\',toy_image], mask ); % display an example image after masking
```



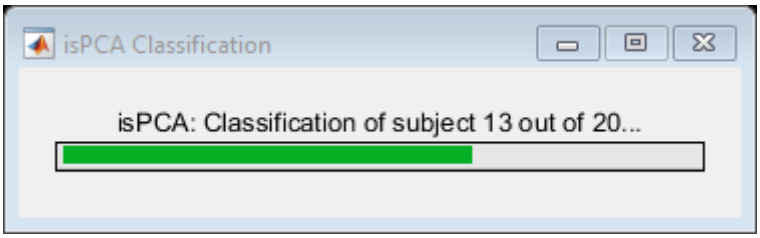
3. Data matrix creation

```
[X,orig_n_subjects,orig_n_voxels,original_size] = create_data_matrix( data_path, files );

% % optionally, a mask can be applied to the data
% [X,orig_n_subjects,orig_n_voxels] = create_data_matrix( data_path, files, mask );
```

4. Feature extraction & Classification

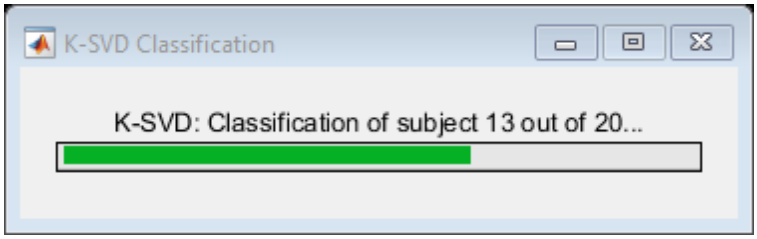
```
% % Mann-Whitney testing ... uncomment to run (computationally demanding)
% p_threshold = 0.01; % threshold for level of significance
% pred_class_mw = classify_mw( X, class_ids, p_threshold ); % vector of predicted classes for
Mann-Whitney
%
% [accur, sens, spec, prec] = performance_eval( class_ids, pred_class_mw );
% perf_mw = [accur, sens, spec, prec];
```

```
% isPCA
n_components = 18; % number of most discriminative components to retain
pred_class_ispca = classify_ispca( X, class_ids, n_components ); % vector of predicted classes
for isPCA
```



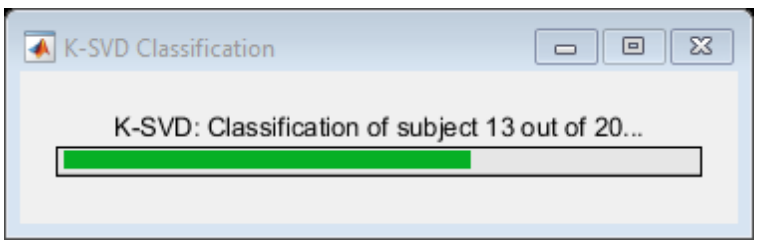

```
[accur, sens, spec, prec] = performance_eval( class_ids, pred_class_ispca ); % classification
performance
perf_ispca = [accur, sens, spec, prec];
```

```
% K-SVD without random projection
n_atoms = 19; % number of atoms
sparsity = 5; % sparsity constraint
RP = {}; % without random projection
pred_class_ksvd = classify_ksvd( X, class_ids, RP, n_atoms, sparsity ); % vector of predicted
classes for K-SVD without random projection
```



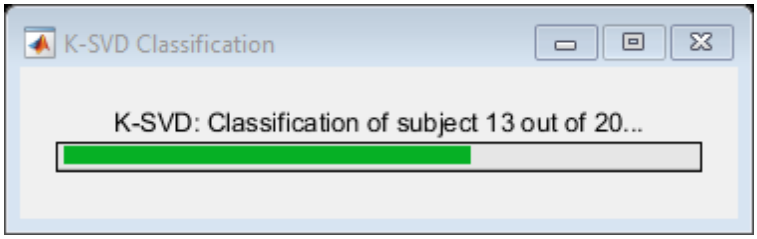

```
[accur, sens, spec, prec] = performance_eval( class_ids, pred_class_ksvd ); % classification
performance
perf_ksvd = [accur, sens, spec, prec];
```

```
% K-SVD with random projection 1
n_atoms = 17; % number of atoms
sparsity = 5; % sparsity constraint
RP.p = 1/100; % multiplier to reduce the length of descriptors with
RP.t = 'A'; % Achlioptas matrix
pred_class_ksvd1 = classify_ksvd( X, class_ids, RP, n_atoms, sparsity ); % vector of predicted
classes for K-SVD with random projection 1
```



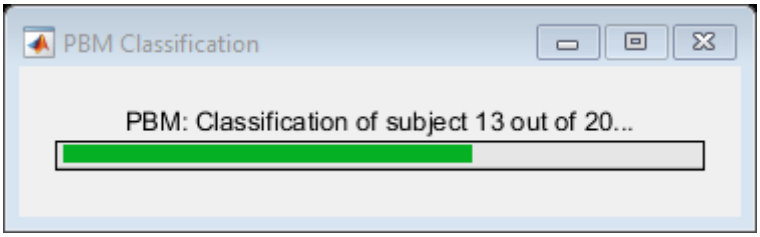

```
[accur, sens, spec, prec] = performance_eval( class_ids, pred_class_ksvd1 ); % classification
performance
perf_ksvd1 = [accur, sens, spec, prec];
```

```
% K-SVD with random projection 2
n_atoms = 17; % number of atoms
sparsity = 5; % sparsity constraint
X_rp = random_projection( X, 1/100, 'A' ); % reduced data matrix
pred_class_ksvd2 = classify_ksvd( X_rp, class_ids, {}, n_atoms, sparsity ); % vector of
predicted classes for K-SVD with random projection 2
```




```
[accur, sens, spec, prec] = performance_eval( class_ids, pred_class_ksvd2 ); % classification
performance
perf_ksvd2 = [accur, sens, spec, prec];
```

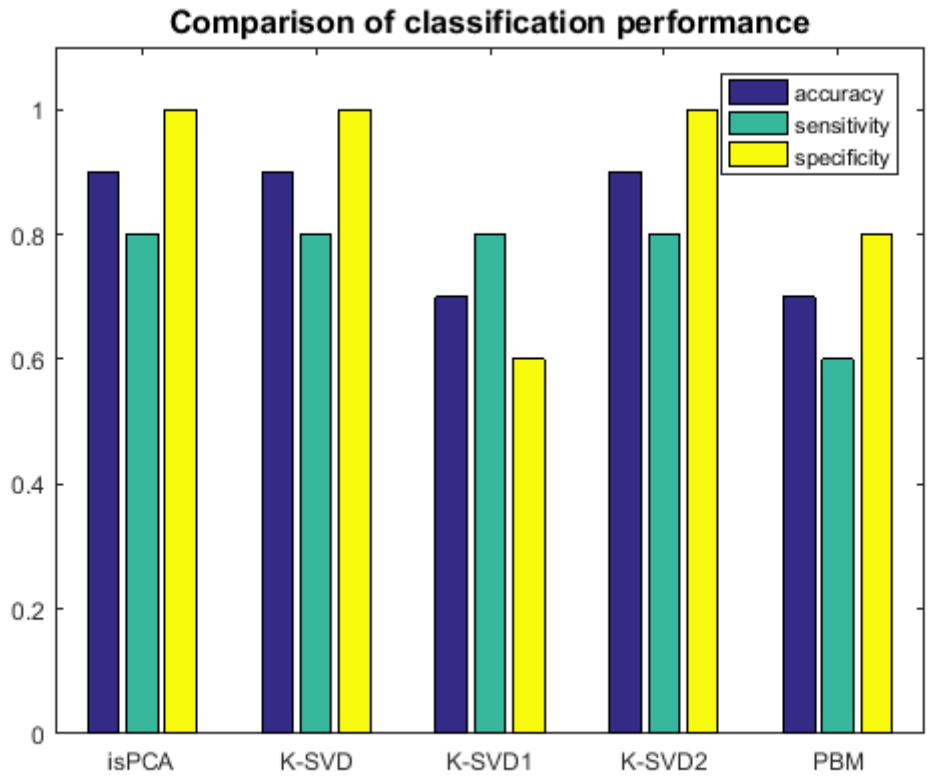
```
% PBM with random projection
n_atoms = 2; % number of atoms
sparsity = 5; % sparsity constraint
n_neighbours = 3; % number of nearest neighbours
RP.p = 1/100; % multiplier to reduce the length of descriptors with
RP.t = 'A'; % Achlioptas matrix
pred_class_pbm = classify_pbm( X, class_ids, RP, n_atoms, n_neighbours, sparsity ); % vector
of predicted classes for PBM with random projection
```




```
[accur, sens, spec, prec] = performance_eval( class_ids, pred_class_pbm ); % classification
performance
perf_pbm = [accur, sens, spec, prec];
```

5. Visualization of classification performance

```
C = [perf_ispca;perf_ksvd;perf_ksvd1;perf_ksvd2;perf_pbm]; % classification performances in
rows
c_names = {'isPCA','K-SVD','K-SVD1','K-SVD2','PBM'}; % names of classifiers
show_m = [1, 1, 1, 0]; % show accuracy, sensitivity and specificity / disregard precision
my_title = 'Comparison of classification performance'
performance_disp( C, show_m, c_names, my_title ); % display multi-bar plot of classification
performance
```


Comparison of classification performance
accuracy
sensitivity
specificity
1
0.8
0.6
0.4
0.2
0
isPCA
K-SVD
K-SVD1
K-SVD2
PBM

Appendix C: Manual classification tutorial

demo_class_manual.m

This demo demonstrates a way to use FEATbox to create your own classification scheme utilizing one of the implemented algorithms, K-SVD.

First, the toolbox is initialized, example data are loaded and rearranged into a data matrix of subjects in rows and features in columns. Second, the data matrix is reduced with Achlioptas random projection creating another data matrix. Subsequently, "extract_ksvd" function is utilized to project the data onto coordinates gained with K-SVD (along with a visual inspection). The reduction of the features space into 2-D facilitates the following k-nearest neighbours classification. Besides, a simplified leave-pair-out cross-validation loop is implemented as follows. In every iteration, a randomly selected pair of subjects from different groups are taken out of a training dataset. The paired image is simply thrown away whereas the tested image is classified. Each of the images is chosen to be the tested one only once. In the end, classification performance is evaluated separately for the full and the reduced data matrix and the results are graphically depicted.

Classification with other algorithms (Mann-Whitney, isPCA, PBM) is analogical – the respective functions are "extract-mw", "extract-ispca", "extract-pbm".

1. Initialization

```
toolbox_path = pwd; % set the location of FEATbox
cd(toolbox_path); % change the current working folder
load_prerequisites( [toolbox_path,'\prerequisites'] ) % check whether all pre-required
functions are in MATLAB search path
```

2. Data loading

```
data_path = [toolbox_path,'\example_data\']; % set the path to data
[class_ids,files] = xlsread([data_path,'\index_unsorted.xls'],'data','a2:b21'); % read the
indexing table
files = cell2mat(files);
```

3. Data matrix creation

```
[ X, orig_n_subjects, orig_n_voxels,original_size ] = create_data_matrix( data_path, files );
% create the data matrix
```

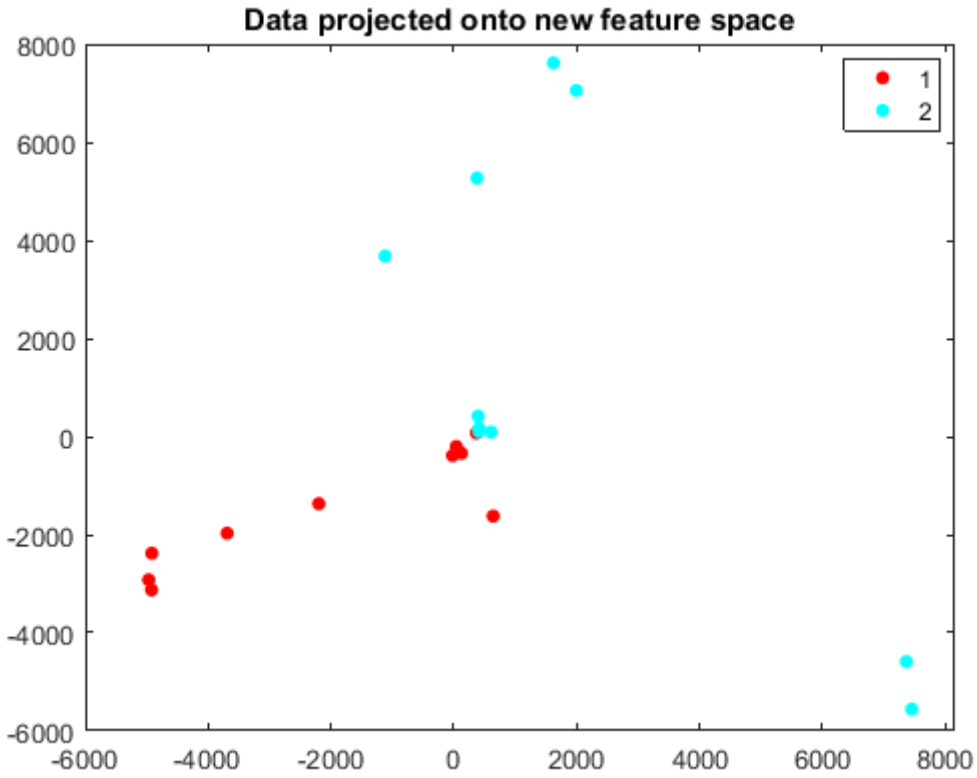
4. Random projection initialization

```
p = 1/100; % set a multiplier to reduce the length of descriptors with
X_rp = random_projection( X, p, 'A' ); % data matrix reduced using (Achlioptas) random
projection
```

5. Use feature extraction to create your own classification scheme

Simplified leave-pair-out classification with k-nearest neighbours classifier based on features extracted with KSVD

```
n_atoms = 2; % number of atoms to be learned

% visual estimation whether 2 atoms span a feature space in which the groups are separable
X_proj = extract_ksvd( X, [], n_atoms ); % data projected onto new feature space
figure;
gscatter(X_proj(:,1),X_proj(:,2),class_ids); % show projected data
title('Data projected onto new feature space');
```



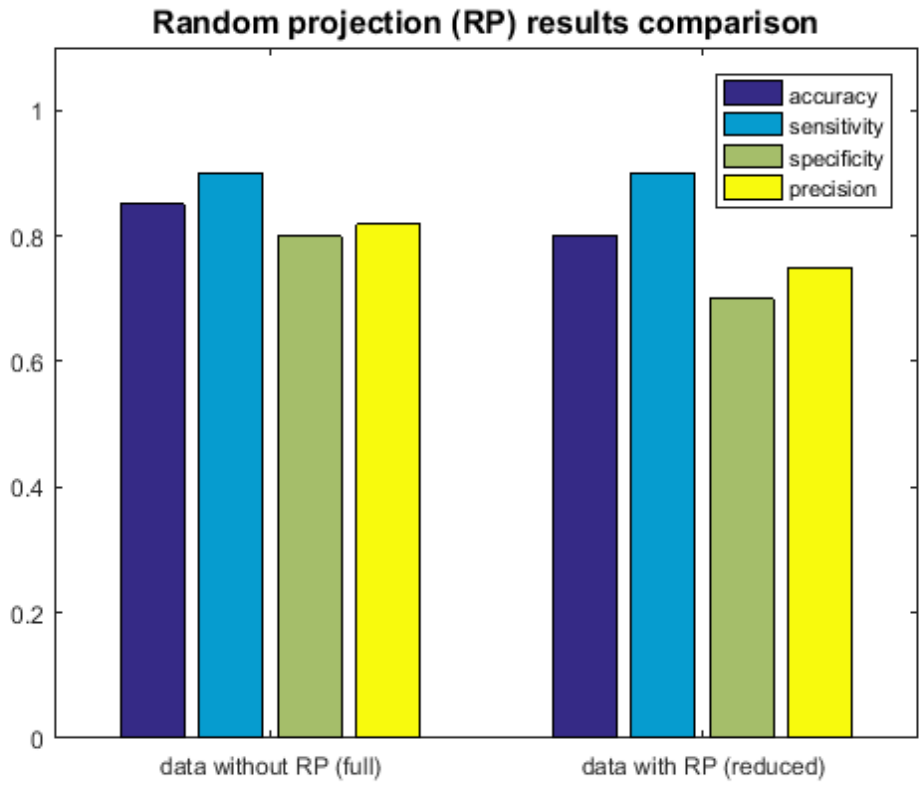

```
% sorting data for leave-pair-out
[class_ids_s,idx] = sort(class_ids); % sorted group identifiers
X_s = X(idx,:); % sorted data matrix
Xrp_s = X_rp(idx,:); % sorted reduced data matrix

predictions_X = zeros(orig_n_subjects,1)-1; % vector of predicted classes (initialized to -1)
predictions_Xrp = zeros(orig_n_subjects,1)-1; % vector of predicted classes for reduced data
(initialized to -1)
```

```
% simplified leave-pair-out classification
for tested_image = 1:orig_n_subjects

    % randomly choose an image (belonging to the other group) to be taken out of the
    % training data set
    if tested_image < (orig_n_subjects/2+1) % here, the data must be sorted
        paired_image = randi([orig_n_subjects/2+1 orig_n_subjects]);
    else
        paired_image = randi([1 orig_n_subjects/2]);
    end
    ci_ksvd = class_ids_s(setdiff(1:end,[tested_image paired_image])); % class identifiers of
training data

    display(sprintf('sLPO * (KSVD + kNN): Computing %d out of %d
...',tested_image,orig_n_subjects));

    % classification
    [ X_ksvd , X_tested ] = extract_ksvd( X_s, [tested_image paired_image], n_atoms ); %
training and tested data in new coordinates
    recognized_index = knnsearch(X_ksvd,X_tested); % k-NN classification
    predictions_X(tested_image) = ci_ksvd(recognized_index(1)); % record the predicted class

    % classification based on reduced data matrix
    [ Xrp_ksvd , Xrp_tested ] = extract_ksvd( Xrp_s, [tested_image paired_image], n_atoms ); %
training and tested data in new coordinates
    recognized_index = knnsearch(Xrp_ksvd,Xrp_tested); % k-NN classification
    predictions_Xrp(tested_image) = ci_ksvd(recognized_index(1)); % record the predicted class


end
```

```
sLPO * (KSVD + kNN): Computing 1 out of 20 ...
sLPO * (KSVD + kNN): Computing 2 out of 20 ...
sLPO * (KSVD + kNN): Computing 3 out of 20 ...
sLPO * (KSVD + kNN): Computing 4 out of 20 ...
sLPO * (KSVD + kNN): Computing 5 out of 20 ...
sLPO * (KSVD + kNN): Computing 6 out of 20 ...
sLPO * (KSVD + kNN): Computing 7 out of 20 ...
sLPO * (KSVD + kNN): Computing 8 out of 20 ...
sLPO * (KSVD + kNN): Computing 9 out of 20 ...
sLPO * (KSVD + kNN): Computing 10 out of 20 ...
sLPO * (KSVD + kNN): Computing 11 out of 20 ...
sLPO * (KSVD + kNN): Computing 12 out of 20 ...
sLPO * (KSVD + kNN): Computing 13 out of 20 ...
sLPO * (KSVD + kNN): Computing 14 out of 20 ...
sLPO * (KSVD + kNN): Computing 15 out of 20 ...
sLPO * (KSVD + kNN): Computing 16 out of 20 ...
sLPO * (KSVD + kNN): Computing 17 out of 20 ...
sLPO * (KSVD + kNN): Computing 18 out of 20 ...
sLPO * (KSVD + kNN): Computing 19 out of 20 ...
sLPO * (KSVD + kNN): Computing 20 out of 20 ...
```

6. Performance Evaluation

```
[ accur, sens, spec, prec, cm ] = performance_eval( class_ids_s, predictions_X, 1 ); % compute
classification performance
[ accur_rp, sens_rp, spec_rp, prec_rp, cm_rp ] = performance_eval( class_ids_s,
predictions_Xrp, 1 ); % compute classification performance

% do some evaluation ...
ClasPer = [[accur,sens,spec,prec];[accur_rp,sens_rp,spec_rp,prec_rp]];
performance_disp( ClasPer, [1 1 1 1], {'data without RP (full)','data with RP (reduced)'},
'Random projection (RP) results comparison' );

display('-----');
display(sprintf('Classification accuracy with full data matrix: %.0f%s.', accur*100,'%'));
display(sprintf('Classification accuracy with data matrix reduced %d times: %.0f%s.', 1/p,
accur_rp*100,'%'));
display('-----');
```




```
-----
Classification accuracy with full data matrix: 85%.
Classification accuracy with data matrix reduced 100 times: 80%.
-----
```

Appendix D: Feature extraction tutorial

demo_extraction.m

This demo demonstrates a way to use FEATbox to extract features using one of the implemented algorithms - Mann-Whithey testing.

First, the toolbox is initialized, example data are loaded and rearranged into a data matrix of subjects in rows and features in columns. Subsequently, "extract_mw" function is utilized to select features significantly distinguishing between two groups in the data. The selection takes place in a leave-one-out cross-validation loop. The selected pixels are then visualized.

Feature extraction with other algorithms (isPCA, K-SVD, PBM) is analogical -- the respective functions are "extract-ispca", "extract-ksvd", "extract-pbm".

Suggestion: This demo takes up to several minutes to compute. Thus, set it off first and take a look at the code (or other demos) while it runs.

1. Initialization

```
toolbox_path = pwd; % set the location of FEATbox
cd(toolbox_path); % change the current working folder
load_prerequisites( [toolbox_path,'\prerequisites'] ) % check whether all pre-required
functions are in MATLAB search path
```

2. Data loading

```
data_path = [toolbox_path,'\example_data\']; % set the path to data
[class_ids,files] = xlsread([data_path,'\index_unsorted.xls'],'data','a2:b21'); % read the
indexing table
files = cell2mat(files);
```

3. Data matrix creation

```
[ X, orig_n_subjects, orig_n_voxels, original_size ] = create_data_matrix( data_path, files );
% create the data matrix
```

4. Feature extraction & Visualization

```
threshold = 0.2; % set a threshold for level of significance
selected_features = zeros(1,orig_n_voxels); % initialize a vector of selected features

% leave-one-out cross-validation
for tested_image = 1:orig_n_subjects

    display(sprintf('MW extraction: Computing %d out of %d
...',tested_image,orig_n_subjects));

    [ ~, ~, selected_ids ] = extract_mw( X, tested_image, class_ids, threshold ); % select
features with p-values lower than the threshold
```

```
    selected_features = selected_features + selected_ids; % cumulate feature across the cross-validation loop

    % the code can be extended with a classification based on selected features
    % ....................................
    % ... (e.g. see "demo_class_manual.m") ...
    % ....................................

end

% visualization of features selected by MW testing cumulated across leave-one-out cross-validation scheme
A = reshape(selected_features,original_size); % rearrange features into a matrix
imagesc(A); % show the features (scale: dark blue = never selected, yellow = selected in each iteration)
title('MW: Set of selected voxels (cumulated across LOO)');
```

```
MW extraction: Computing 1 out of 20 ...
MW extraction: Computing 2 out of 20 ...
MW extraction: Computing 3 out of 20 ...
MW extraction: Computing 4 out of 20 ...
MW extraction: Computing 5 out of 20 ...
MW extraction: Computing 6 out of 20 ...
. . .
MW extraction: Computing 17 out of 20 ...
MW extraction: Computing 18 out of 20 ...
MW extraction: Computing 19 out of 20 ...
MW extraction: Computing 20 out of 20 ...
```