FBCSP Toolbox Manual

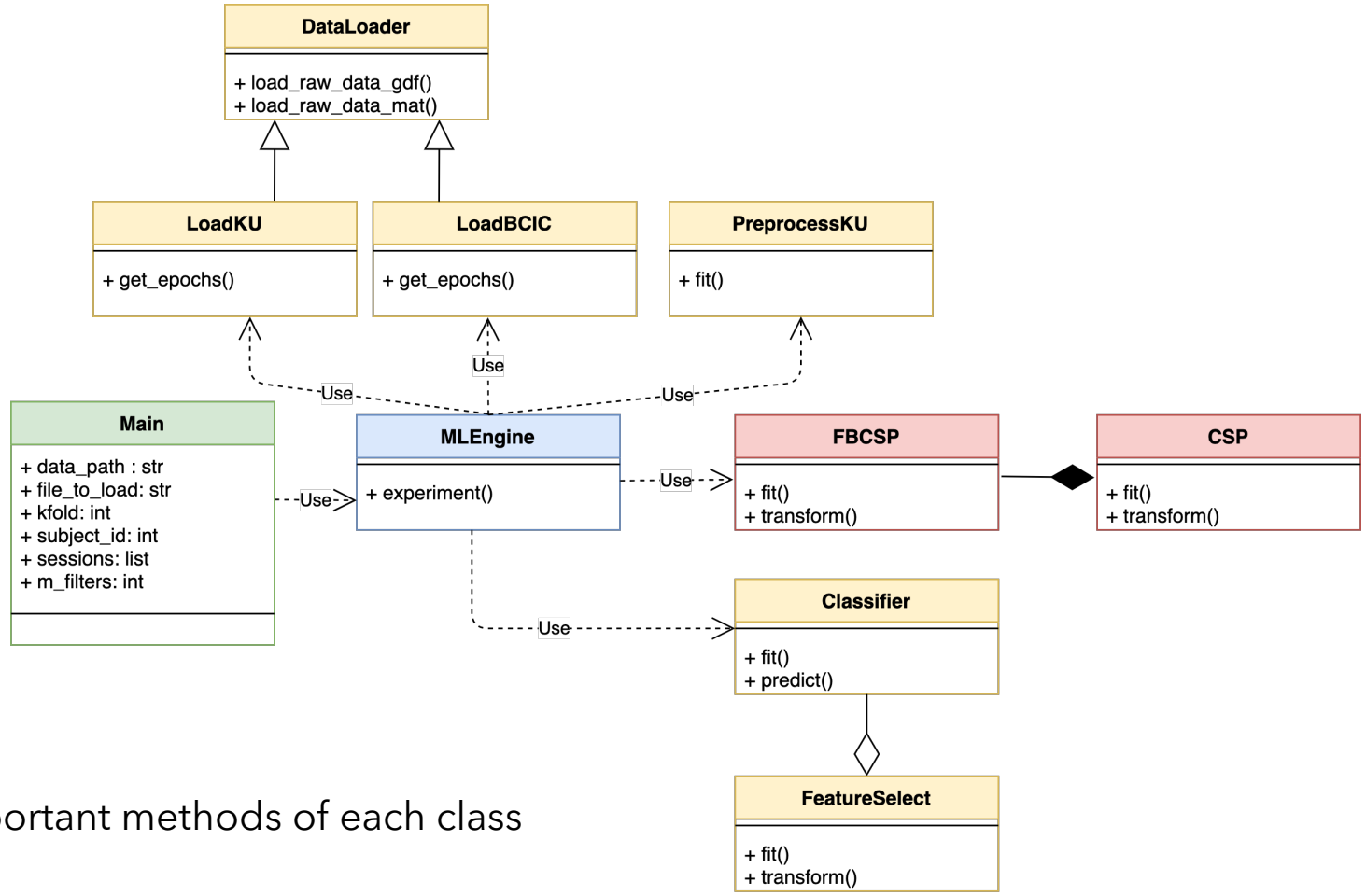
7th July 2020

Use of this toolbox

- This manual provides detailed documentation for the FBCSP Toolbox (fbcsptoolbox.github.io). The architecture of the toolbox has been provided here. A few examples on how to use this toolbox are also provided with a table of the results obtained.
- The toolbox has been made to be independent of the type of EEG acquisition device or format used. All signal processing is done on numpy.ndarrays. The minimum possible input needed to run this toolbox are a 3D numpy array (trials x channels x time samples), a 1D numpy array containing the class labels and the sampling frequency.
- The loading functions should abstract away all other details in other variables. The processing functions for FBCSP directly process the numpy.ndarrays.
- The parameter settings in this toolbox are similar to the original publications that introduced FBCSP. A list of these publications can be found in fbcsptoolbox.github.io/publications

Table of Contents

Architecture of this toolbox



Class diagram showing the important methods of each class

LoadData (bin.LoadData)

Base class that loads EEG data from files. Includes basic functions to load data from .gdf and .mat formats. More functions can be added here for loading data from other formats such as .cnt, .vhdr, etc.

Attributes	Type	Description
eeg_file_path	str	Stores the path to the folder containing the eeg files

Methods	Description
load_raw_data_gdf	Load data from gdf format. Uses MNE package
	Input: ***file_to_load*** (str): Filename of the data file that needs to be loaded. This is used to append to the *eeg_file_path* to load the data. Data is stored in *self.raw_eeg_subject* **Output**: *self*

LoadData (bin.LoadData)

Methods	Description
load_raw_data_mat	Load data from mat format. Uses Scipy package
	Input: ***file_to_load*** (str): The filename of the data file that needs to be loaded. This is used to append to the *eeg_file_path* to load the data. Data is stored in *self.raw_eeg_subject* **Output**: ***self***
get_all_files	Helper function to return all names of files in *eeg_file_path* with a given extension. **Input**: ***file_path_extension*** (str): Extension of the filenames to be listed **Ouput**: List of files with matching extensions

LoadBCIC (bin.LoadData)

Inherits from LoadData to specifically load EEG data from BCI Competition IV Dataset 2a. The default parameters used to reproduce results provided in this manual have been defined. Data is extracted from each trial between [-4.5, 5]s with respect to cue.

Attributes	Type	Description
stimcodes	tuple	Stimulus codes of the trials to be loaded
file_to_load	str	Name of file to be loaded
channels_to_remove	list of str	Names of channels that need to be excluded from analysis

LoadBCIC (bin.LoadData)

Methods	Description
get_epochs	Load EEG data and epoch them from gdf files for BCIC IV 2a.
	Input: ***tmin*** (float): The starting time for each epoch with respect to cue (default: -4.5) ***tmax*** (float): The end time for each epoch with respect to cue (default: 5) ***baseline*** (None or tuple of length 2): Baseline parameter in MNE.Epochs **Output**: ***eeg_data*** (dict): '*x_data*' contains the EEG data epochs in numpy.ndarray format with ndims = 3 (trials x channels x time); '*y_data*' contains the corresponding class labels from 0 to N_classes – 1; '*fs*' contains the sampling frequency

LoadKU (bin.LoadData)

Inherits from LoadData to specifically load EEG data from the OpenBMIDataset from Korea University. Accesses the .mat files containing the epoched data.

Attributes	Type	Description
subject_id	int	Subject number (between 1 to 54) whose data is to be loaded
fs	int or float	Sampling frequency

LoadKU (bin.LoadData)

Methods	Description
get_epochs	Load epoched EEG data from mat files.
	Input: ***sessions*** (list of int): The sessions to be loaded. If list of length 1 is provided, then only that session is loaded and returned. If both session indices are provided, the data is loaded in the specified order of sessions and concatenated (default: [1, 2]) **Output**: ***eeg_data*** (dict): '*x_data*' contains the EEG data epochs in numpy.ndarray format with ndims = 3 (trials x channels x time); '*y_data*' contains the corresponding class labels from 0 to N_classes – 1; '*fs*' contains the sampling frequency; *'ch_names'*: list of str containing the channel names

Preprocess_KU (bin.Preprocess)

Class providing methods to preprocess EEG data from the KU dataset.

Attributes	Type	Description
selected_channels	list of str	Names of channels that need to be retained. All other channels not mentioned in this list will be excluded from the data.

Preprocess_KU (bin.Preprocess)

Methods	Description
select_channels	Retains the channels specified in *selected_channels*
	Input: ***x_data*** (numpy.ndarray): 3D array (trials x channels x time) containing epoched EEG data. ***ch_names*** (list of str): List of channel names corresponding to *x_data*. ***selected_channels*** (list of str): List of channels to be retained. If not provided, channels from *self.selected_channels* are used. **Output**: ***x_data_selected*** (numpy.ndarray): 3D array containing data only from selected channels (trials x channels x time)

MLEngine (bin.MLEngine)

Main class providing methods to carry out data loading, preprocessing and classification of EEG data. All other methods and classes for data processing and classification are called by *MLEngine.experiment()* which contains the main pipeline for classification.

Attributes	Type	Description
data_path	str	Path to folder containing EEG data to be loaded.
subject_id	int	ID of the subject whose data is to be loaded. Used by LoadKU to load the data from the KU dataset. Ignore if loading data from BCIC dataset.
file_to_load	str	Name of file in *data_path* whose data is to be loaded. Used by LoadBCIC. Ignore if loading data from KU dataset.
sessions	list of int	*sessions* parameter from *LoadKU.get_epochs()*

MLEngine (bin.MLEngine)

Attributes	Type	Description
kfold	int	Number of folds of CV to be performed. Data is split into train and test sets accordingly. Used by MLEngine.cross_validate_sequential_split() and MLEngine.cross_validate_Ntimes_Kfold()
ntimes	int	Number of times kfold CV is to be performed. Used by *MLEngine.cross_validate_Ntimes_Kfold()*. To be set to 1 when using *MLEngine.cross_validate_sequential_split()*.
window_details	dict	Indices of window within an extracted epoch that is to be used for classification. For example, used in BCIC pipeline to extract a smaller window within the extracted epoch, from *'tmin'* to *'tmax''*. Ignore if entire epoch is to be used.
m_filters	int	Number of CSP filters to be selected. 2*m_filters corresponding to the top and bottom m_filters are selected for CSP.

MLEngine (bin.MLEngine)

Methods	Description
experiment	The main pipeline for classification of EEG. Calls all other functions and methods within the pipeline.
cross_validate_Ntimes_Kfold	Returns train and test split indices for Ntimes x Kfold cross-validation. *Uses sklearn.model_selection.StratifiedKFold.*
	Input: ***y_labels*** (numpy.ndarray): 1D array containing class labels of epoched EEG data. ***ifold*** (int): The index of the i^{th} Kfold CV used as random_state for reproducibility. **Output**: ***train_indices*** (dict): Dict containing the indices of trials in the training set for each of the Kfold ***test_indices*** (dict): Dict containing the indices of trials in the testing set for each of the Kfold

MLEngine (bin.MLEngine)

Methods	Description
cross_validate_sequential_split	Returns train and test split indices for Kfold cross-validation using the natural order of indices. If used, Ntimes in MLEngine.experiment() should be 1 since the order does not change in successive runs of Kfold CV.
	Input: ***y_labels*** (numpy.ndarray): 1D array containing class labels of epoched EEG data. **Output**: ***train_indices*** (dict): Dict containing the indices of trials in the training set for each of the Kfold ***test_indices*** (dict): Dict containing the indices of trials in the testing set for each of the Kfold

MLEngine (bin.MLEngine)

Methods	Description
cross_validate_half_split	Returns train and test split indices split sequentially into two halves for running train and test once. If used, Ntimes and Kfold in MLEngine.experiment() should both be 1. Additionally, this method also checks to make sure classes are balanced in the split.
	Input: ***y_labels*** (numpy.ndarray): 1D array containing class labels of epoched EEG data. **Output**: ***train_indices*** (dict): Dict containing the indices of trials in the training set. Only contains one key-value pair. ***test_indices*** (dict): Dict containing the indices of trials in the testing set. Only contains one key-value pair.

MLEngine (bin.MLEngine)

Methods	Description
split_xdata	Returns the EEG epochs divided into training and test sets.
	Input: ***eeg_data*** (numpy.ndarray): 4D array (frequency band x trials x channels x time) containing epoched EEG data. ***train_idx*** (numpy.ndarray): 1D array containing indices of trials to be in the training set. ***test_idx*** (numpy.ndarray): 1D array containing indices of trials to be in the test set. **Output**: ***x_train_fb*** (numpy.ndarray): 4D array (frequency band x trials x channels x time) containing the training set. ***x_test_fb*** (numpy.ndarray): 4D array (frequency band x trials x channels x time) containing the test set.

MLEngine (bin.MLEngine)

Methods	Description
split_ydata	Returns the class labels divided into training and test sets.
	Input: ***y_true*** (numpy.ndarray): 1D array containing the class labels for the whole dataset. ***train_idx*** (numpy.ndarray): 1D array containing indices of trials to be in the training set. ***test_idx*** (numpy.ndarray): 1D array containing indices of trials to be in the test set. **Output**: ***y_train_fb*** (numpy.ndarray): 1D array containing the class labels for the training set. ***y_test_fb*** (numpy.ndarray): 1D array containing the class labels for the test set

MLEngine (bin.MLEngine)

Methods	Description
get_multi_class_label	To be used during one-versus-rest when each classified output is a discrete label (0 or 1). Multi-class label is then selected by checking the earliest position of *cls_interest* across all classified labels for each trial.
	Input: ***y_predicted*** (numpy.ndarray): 2D array (Number of trials x Number of classes) containing the predicted class labels. ***cls_interest*** (int): The index of the class that denotes the target class. **Output**: ***y_predict_multi*** (numpy.ndarray): 1D array containing the predicted multi-class labels.

MLEngine (bin.MLEngine)

Methods	Description
get_multi_class_regressed	To be used during one-versus-rest when each classified output is a regressed value (for example, when using SVR). Multi-class label is then selected by checking the index of the minimum regressed value. Here *cls_interest* is assumed to be 0.
	Input: ***y_predicted*** (numpy.ndarray): 2D array (Number of trials x Number of classes) containing the predicted class labels. **Output**: ***y_predict_multi*** (numpy.ndarray): 1D array containing the predicted multi-class labels.

FilterBank (bin.MLEngine)

Class providing methods to carry out filtering of EEG data to be used for FBCSP. The default parameters specified are as pe the original FBCSP settings in [x].

Attributes	Type	Description
fs	int or float	Sampling frequency
f_trans	int or float	Transition bandwidth
f_pass	numpy.ndarray	The pass bands of the frequency bands used in FBCSP
f_width	int	The width of each frequency band
g_pass	int or float	The maximum loss in the passband (dB)
g_stop	int or float	The minimum attenuation in the stopband (dB)
filter_coeff	dict	Contains the filter coefficients ‘b’ and ‘a’

FilterBank (bin.MLEngine)

Methods	Description
get_filter_coeff	Returns the filter coefficients based on filter design parameters.
filter_data	Returns the filtered data into the various frequency bands for FBCSP.
	Input: ***eeg_data*** (numpy.ndarray): 3D array (trials x channels x time) containing epoched EEG data. ***window_details*** (dict): To be used when a smaller window is to be extracted from epoched data after filtering. (default: if used, 4.5 is added to 'tmin' and 'tmax' to correspond to the default epoching segments used with respect to cue in BCIC IV 2a). Ignore if not required. **Output**: ***filtered_data*** (numpy.ndarray): 4D array (frequency band x trials x channels x time) containing filtered epoched EEG data.

FBCSP (bin.FBCSP)

Class providing methods to carry out FBCSP processing. The spatial filters can be obtained using *FBCSP.fit()*. Once they have been obtained, EEG data pre-filtered into the various frequency bands can then be filtered using FBCSP.transform(). This class is to be used along with CSP.

Attributes	Type	Description
m_filters	int	Number of CSP filters to be selected per frequency band. 2*m_filters corresponding to the top and bottom m_filters are selected for CSP.
fbcsp_filters_multi	list of dict of dict	After using *fit()*, contains the spatial filters for multi-class labels in one-versus-rest fashion using the training data. Each list element corresponds to one OVR class and contains a dict for each frequency band. The dict for each frequency band contains the spatial filters '*u_mat*' and the corresponding eigenvalues '*eig_val*' for that frequency band.

FBCSP (bin.FBCSP)

Methods	Description
fit	Estimates the CSP spatial filters in pre-filtered EEG data. Transforms incoming data into one-versus-rest model.
	Input: ***x_train_fb*** (numpy.ndarray): 4D array (frequency band x trials x channels x time) containing filtered epoched EEG data. ***y_train*** (numpy.ndarray): 1D array containing the multi-class labels of the training data.

FBCSP (bin.FBCSP)

Methods	Description
transform	Returns the spatially filtered EEG features for all frequency bands for a specified OVR class.
	Input: ***x_data*** (numpy.ndarray): 4D array (frequency band x trials x channels x time) containing filtered epoched EEG data. ***class_idx*** (int): Index of the OVR class whose FBCSP filters are to be used. **Output**: ***x_features*** (numpy.ndarray): 2D array (trials x total number of features): Concatenates the features across all frequency bands in successive columns.

CSP (bin.CSP)

Class providing methods to carry out CSP processing. The spatial filters can be obtained using *CSP.fit()*. This can be used in isolation if only CSP processing is required or with *FBCSP*.

Attributes	Type	Description
m_filters	int	Number of CSP filters to be selected per frequency band. 2*m_filters corresponding to the top and bottom m_filters are selected for CSP.

CSP (bin.CSP)

Methods	Description
fit	Estimates the CSP spatial filters in EEG data. Assumes incoming data only has binary classes.
	Input: ***x_train*** (numpy.ndarray): 3D array (trials x channels x time) containing filtered epoched EEG data. ***y_train*** (numpy.ndarray): 1D array containing the binary-class labels of the training data.

CSP (bin.CSP)

Methods	Description
transform	Returns the spatially filtered EEG features for a given EEG trial.
	Input: ***x_data*** (numpy.ndarray): 2D array (channels x time) containing trial. ***eig_vectors*** (numpy.ndarray): 2D array containing the CSP spatial filters that were obtained using CSP.fit(). **Output**: ***x_features*** (numpy.ndarray): 1D array (number of features): spatially transformed features for the given trial.

Classifier (bin.Classifier)

Class providing methods to carry out classification training and prediction. The choice of classifier can be specified which will then automatically be used for training and prediction. This class also uses the FeatureSelect class if feature selection is required.

Attributes	Type	Description
model		Holds the instance to the specified classifier model. This will be used for training and prediction. In OVR setting, each OVR iteration should create a new instance of this class with the specified classifier passes as the model.
feature_selection	bool	Whether or not to carry out feature selection before training and prediction

Classifier (bin.Classifier)

Methods	Description
fit	Used to train the model specified in *self.model*. If *self.feature_selection* is true, an instance of FeatureSelect is created which and trained automatically. Returns the predicted labels on the training set. The trained classifier is in *self.model*.
	Input: ***x_features*** (numpy.ndarray): 2D array (trials x features) containing the training feature matrix. ***y_train*** (numpy.ndarray): 1D array containing the binary-class labels of the training data. **Output**: ***y_predicted*** (numpy.ndarray): 1D array containing the predicted binary-class labels on the training data.

Classifier (bin.Classifier)

Methods	Description
predict	Used to carry out classification using the trained model in *self.model*
	Input: ***x_features*** (numpy.ndarray): 2D array (trials x features) containing the training feature matrix. **Output**: ***y_predicted*** (numpy.ndarray): 1D array containing the predicted class labels.

FeatureSelect (bin.Classifier)

Class providing methods to carry out feature selection. Since CSP features are usually used in pairs, it also provides method *FeatureSelect.select_CSP_pairs()* to identify the corresponding CSP pairs after feature selection. *FeatureSelect.MIBIF()* provides feature selection using the Mutual Information Based Individual Feature selection algorithm. This can be easily bypassed to use any other feature selection algorithm as required.

Attributes	Type	Description
n_features_select	int	Number of features selected
n_csp_pairs	int	The number of CSP features to be selected corresponding to the features selected.
Features_selected_indices	numpy.ndarray	Indices of features selected

FeatureSelect (bin.Classifier)

Methods	Description
fit	To train a specified algorithm to perform feature selection. The indices of the selected features are stored in *self.features_selected_indices*. Calls the *FeatureSelect.select_CSP_pairs()* function before returning the selected indices.
	Input: ***x_train_features*** (numpy.ndarray): 2D array (trials x features) containing the training feature matrix. ***y_train*** (numpy.ndarray): 1D array containing the class labels of the training data. **Output**: ***x_train_features_selected*** (numpy.ndarray): 1D array containing the indices of the selected features.

FeatureSelect (bin.Classifier)

Methods	Description
transform	Returns the feature matrix with only the selected features
	Input: ***x_test_features*** (numpy.ndarray): 2D array (trials x features) containing the training feature matrix. ***y_train*** (numpy.ndarray): 1D array containing the class labels of the training data. **Output**: ***x_test_features_selected*** (numpy.ndarray): 2D array (trials x features) containing only the selected features for each trial.

FeatureSelect (bin.Classifier)

Methods	Description
MIBIF	Returns the MIBIF score for each feature with respect to the class label.
	Input: ***x_features*** (numpy.ndarray): 2D array (trials x features) containing the training feature matrix. ***y_labels*** (numpy.ndarray): 1D array containing the class labels. **Output**: ***mifsg*** (numpy.ndarray): 1D array (number of features) containing the MIBIF score for each feature.

Ang, K. K., and Quek, C. (2006). "Rough set-based neuro-fuzzy system," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, Vancouver, BC, 742–749.

FeatureSelect (bin.Classifier)

Methods	Description
select_CSP_pairs	Returns the indices of the features selected by including the corresponding CSP pairs that may not have been selected by the feature selection algorithm.
	Input: ***features_selected*** (numpy.ndarray): 1D array (number of selected features) containing the indices of the selected features. ***n_pairs*** (int): The number of corresponding CSP pairs that need to be selected. Usually takes n_pairs = 2, the other CSP pair will also be included. **Output**: ***paired_features*** (numpy.ndarray): 1D array (number of features) containing the indices of the selected features along with their corresponding CSP pairs.

Using the code - examples

- For subject-specific classification on BCIC dataset IV 2a
  - In mainPipeline.py

```
from bin.MLEngine import MLEngine

dataset_details={
    'data_path' : "/Data/BCICIV_2a_gdf",
    'file_to_load': 'A01T.gdf',
    'ntimes': 10,
    'kfold':10,
    'm_filters':2,
    'window_details':{'tmin':0.5,'tmax':2.5}
}
ML_experiment = MLEngine(**dataset_details)
ML_experiment.experiment()
```

Refer to fbcsptoolbox.github.io/tutorials

Using the code - examples

- For subject-specific classification on KU Dataset
  - In mainPipeline.py

```
from bin.MLEngine import MLEngine

dataset_details = {
    'data_path': "/KU_Dataset/BCI dataset/DB_mat",
    'subject_id': 1,
    'sessions': [1],
    'ntimes': 1,
    'kfold': 10,
    'm_filters': 2,
}
ML_experiment = MLEngine(**dataset_details)
ML_experiment.experiment()
```

Exemplary FBCSP Results

BCI Competition IV 2a: 10 x 10 CV, SVR

FBCSP Settings:
1. Filtering between 4-8, 8-12, … 36-40 Hz
2. Time Window: 0.5 to 2.5s
3. Fs = 250 Hz

Subjects	S1	S2	S3	S4	S5	S6	S7	S8	S9	Mean
Accuracy MIBIF, n=4	82.56	51.64	84.35	57.62	70.31	48.42	87.53	85.11	84.03	**72.40**
Accuracy no MIBIF	81.21	55.20	83.86	59.98	69.35	50.36	90.28	87.36	85.18	**73.64**

Exemplary FBCSP Results

BCI Competition IV 2a: Sequential splitting with balanced class distribution, 10-fold CV, SVR

FBCSP Settings:

1. Filtering between 4-8, 8-12, … 36-40 Hz
2. Time Window: 0.5 to 2.5s
3. Fs = 250 Hz

Subjects	S1	S2	S3	S4	S5	S6	S7	S8	S9	Mean
Accuracy MIBIF, n=4	83.03	48.65	83.97	56.29	72.18	51.02	86.77	83.63	82.94	**72.05**
Accuracy no MIBIF	82.29	55.89	85.39	56.29	68.37	52.06	91.65	86.39	85.03	**73.71**

Exemplary FBCSP Results

BCI Competition IV 2a: Half and half split, SVR

FBCSP Settings:

1. Filtering between 4-8, 8-12, … 36-40 Hz
2. Time Window: 0.5 to 2.5s
3. Fs = 250 Hz

Subjects	S1	S2	S3	S4	S5	S6	S7	S8	S9	Mean
Accuracy MIBIF, n=4	79.17	43.06	79.86	52.08	63.19	43.05	81.94	78.47	77.08	**66.43**
Accuracy no MIBIF	74.31	43.75	76.39	45.83	61.11	45.83	86.81	79.17	75.00	**65.36**

References

1. Ang, Kai Keng, et al. “Filter bank common spatial pattern (FBCSP) in brain-computer interface.” IEEE International Joint Conference on Neural Networks (pp. 2390-2397)
2. Ang, Kai Keng, et al. “Filter bank common spatial pattern algorithm on BCI competition IV datasets 2a and 2b.” Frontiers in neuroscience 6 (2012): 39.
3. Ang, Kai Keng, et al. “Mutual information-based selection of optimal spatial-temporal patterns for single-trial EEG-based BCIs.” Pattern Recognition 45.6 (2012): 2137-2144.
4. Lee, M.-H., Kwon, O.-Y., Kim, Y.-J., Kim, H.-K., Lee, Y.-E., Williamson, J., … Lee, S.-W. (2019). EEG dataset and OpenBMI toolbox for three BCI paradigms: an investigation into BCI illiteracy. GigaScience, 8(5). doi:10.1093/gigascience/giz002