# bpkclone.cluster

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cluster__init__.py

## Package Contents

| | | | |
|---|---|---|---|
| cluster | clusteringCombination | templateSelection | |

# bpkclone.datamanager

d:\work\productioncode\clones\biometricspykit\bpkclone\datamanager__init__.py

index

## Package Contents

| | | | |
|---|---|---|---|
| datamanager | datamanagerHelp | parted | sandbox |

# bpkclone.devices

index
d:\work\productioncode\clones\biometricspykit\bpkclone\devices__init__.py

## Package Contents

bitalinoTools

# bpkclone.dimreduction



d:\work\productioncode\clones\biometricspykit\bpkclone\dimreduction__init__.py

## Package Contents

| | |
|---|---|
| LDAopenCV | dimreduction |

# bpkclone.evaluation

index
d:\work\productioncode\clones\biometricspykit\bpkclone\evaluation__init__.py

## Package Contents

evaluation

# bpkclone.featureextraction

d:\work\productioncode\clones\biometricspykit\bpkclone\featureextraction__init__.py

index

## Package Contents

featureextraction

# bpkclone.misc

index
d:\work\productioncode\clones\biometricspykit\bpkclone\misc__init__.py

## Package Contents

misc

# bpkclone.outlier

index
d:\work\productioncode\clones\biometricspykit\bpkclone\outlier__init__.py

## Package Contents

outlier

# bpkclone

index
d:\work\productioncode\clones\biometricspykit\bpkclone__init__.py

## Package Contents

| | | | |
|---|---|---|---|
| BiometricSystemWizard | **cluster** (package) | **featureextraction** (package) | sandbox |
| BiometricSystemWizard_carlos | config | guidelines | **stats** (package) |
| BiometricWizard | cp77mod | hist_mp | test_clustering |
| **Cloud** (package) | **datamanager** (package) | **misc** (package) | test_mimicdb |
| QS_sandbox | **devices** (package) | ms_ | **visualization** (package) |
| bla | **dimreduction** (package) | **outlier** (package) | vitalidiFramework |
| cc_sandbox | **evaluation** (package) | plotWaveletResults_carlos | **wavelets** (package) |
| **classifiers** (package) | fc_sandbox | **preprocessing** (package) | |

# bpkclone.preprocessing

index
d:\work\productioncode\clones\biometricspykit\bpkclone\preprocessing__init__.py

## Package Contents

preprocessing

# bpkclone.visualization

index
d:\work\productioncode\clones\biometricspykit\bpkclone\visualization__init__.py

## Package Contents

visualization

# bpkclone.wavelets

index
d:\work\productioncode\clones\biometricspykit\bpkclone\wavelets__init__.py

## Package Contents

| | | |
|---|---|---|
| sandbox | sandbox_3 | wavelets |
| sandbox_2 | sandbox_dbscan | |

# bpkclone.BiometricWizard

index

d:\work\productioncode\clones\biometricspykit\bpkclone\biometricwizard.py

```
.. module:: BiometricWizard
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Carlos Carreiras
```

## Modules

| | | |
|---|---|---|
| copy | os | traceback |
| numpy | matplotlib.pyplot | |

## Functions

**Classify**(expConfig, run, clf_name, subjects)

**Cluster**(tasks, parameters, case)

**DimReduction**(expConfig, testData, trainData, subjects)

**FeatureExtraction**(tasks, parameters, case)

**FeatureSelection**(tasks, parameters, case)

**Filter**(tasks, parameters, case)

**Load**(expConfig, store)

**Load2Clf**(subjects, case, run)

**Main**(expConfig, expID)

**MongoReport**(expConfig, nbRuns)

**Outlier**(tasks, parameters, case)

**PopulationStatistics**(nbRuns, subjects, key=None)

**Prepare**(tasks, expConfig, case, run)

**Results**(expConfig, run, clf_name)

**Segment**(tasks, parameters, case)

**TemplateSelection**(tasks, parameters, case)

**Visualize**(expConfig, run, clf_name)

**blockPrepare**(data, normalization=None, subpattern=None, number_mean_waves=None, number_median_waves=None, quantization=None, patterns2strings=None, nbTemplates=None, manList=None, subject=None)

**concatenate**(data, axis, vec)

**features**(data, normalization=None, subpattern=None, number_mean_waves=None, number_median_waves=None, quantization=None, patterns2strings=None)

**leadLoader**(expConfig, subjects, lbl2sub, subjectDict, xtrainRecs, xtestRecs, commonRecs)

**leadTaskLoader**(dbParams, lock, tasker, path, subject, trainSR, testSR, commonSR, data_name, data_type, run_setup, concatenate_records=False, concatenate_signals=True, axis=0)

**loocvLoader**(expConfig, subjects, lbl2sub, subjectDict, xtrainRecs, xtestRecs, commonRecs)

**loocvTaskLoader**(dbParams, lock, tasker, path, subject, trainSR, testSR, commonSR, data_name, data_type, run_setup, concatenate_records=False, concatenate_signals=True, axis=0)

**popStatsHelper**(data, key=None)

**prepareTemplates**(tasks, src=None, subject=None, manList=None, normalization=None, subpattern=None, number_mean_waves=None, number_median_waves=None, quantization=None, patterns2strings=None, nbTemplates=None)

**selector**(step)

**simpleLeadLoader**(expConfig, subjects, lbl2sub, subjectDict, xtrainRecs, xtestRecs, commonRecs)

**simpleLeadTaskLoader**(dbParams, lock, tasker, path, subject, trainSR, testSR, commonSR, data_name, data_type, run_setup, concatenate_records=False, concatenate_signals=True, axis=0)

**simpleLoader**(expConfig, subjects, lbl2sub, subjectDict, xtrainRecs, xtestRecs, commonRecs)

**simpleTaskLoader**(dbParams, lock, tasker, path, subject, trainSR, testSR, commonSR, data_name, data_type, run_setup, concatenate_records=False, concatenate_signals=True, axis=0)

**visualizeSubject**(thresholds, rates, filePath)

**windowLoader**(expConfig, subjects, lbl2sub, subjectDict, xtrainRecs, xtestRecs, commonRecs)

## Data

**CLF** = ['SVM', 'KNN', 'Agrafioti', 'Dissimilarity', 'DissimilaritySimple', 'DissimilarityMore']

# bpkclone.classifiers.classifiers

index
d:\work\productioncode\clones\biometricspykit\bpkclone\classifiers\classifiers.py

```
.. module:: classifiers
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Filipe Canento, Carlos Carreiras
```

## Modules

| | | |
|---|---|---|
| scipy.spatial.distance | os | sklearn.svm |
| numpy | shutil | warnings |

## Classes

__builtin__.object

- BaseClassifier
  - KNN
    - Agrafioti
    - Dissimilarity
      - DissimilaritySimple
    - DissimilarityMore
    - Dissimilarity_old
      - DissimilaritySimple_old
  - Odinaka
  - SVM
    - ClusterSVM
    - LeanSVM
  - SifterSVM
- ClassifierDict
- fisher

bidict.bidict(__builtin__.object)

- SubjectDict

exceptions.Exception(exceptions.BaseException)

- EmptyCombination
- UnknownSubjectError
- UntrainedError

class **Agrafioti**(KNN)

```
Class to train and classify AC coefficients from ECG Signals

Performs LDA if coefficient number is bigger than subject number
```

Method resolution order:

- Agrafioti
- KNN
- BaseClassifier
- __builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', h=4.0, M=100.0, w=None, **kwargs)

**autoRejectionThresholds**(self)

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'agrafiotiClassifier'

---

Methods inherited from KNN:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.
```

```
        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**
```
    dictionary for instance variables (if defined)
```

**__weakref__**
```
    list of weak references to the object (if defined)
```

class **BaseClassifier**(__builtin__.object)
```
Base Classifier.
```

Methods defined here:

**__init__**(self, io=None)

**authThreshold**(self, subject, ready=False)

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**autoRejectionThresholds**(self)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)
```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:
```

```
    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**re_train**(self, data)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**train**(self, data=None, updateThresholds=True)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods defined here:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors defined here:

**__dict__**
```
    dictionary for instance variables (if defined)
```

**__weakref__**
```
    list of weak references to the object (if defined)
```

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'BaseClassifier'

class **ClassifierDict**(__builtin__.object)

```
Class to handle hierarchical subject - label - classifier disambiguation.
```

Methods defined here:

**__init__**(self)

**assign**(self, n, label)

**getClassifiers**(self, label)

**getLabel**(self, clf)

**getLabelWeight**(self, label)

**getPairWeight**(self, pair)

---

Data descriptors defined here:

**__dict__**

```
dictionary for instance variables (if defined)
```

**__weakref__**

```
list of weak references to the object (if defined)
```

class **ClusterSVM**(SVM)

```
Perform clustering on the training data.
```

Method resolution order:
ClusterSVM
SVM
BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, nensemble=100, partMethod='average', **kwargs)

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = -1

**EXT** = '.hdf5'

**NAME** = 'clusterSvmClassifier'

---

Methods inherited from SVM:

**autoRejectionThresholds**(self)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.
```

```
    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **Dissimilarity**(KNN)
```
Class to extract multilead distances subjectwise

Performs kNN for classification based on those distances
```

Method resolution order:
- Dissimilarity
- KNN
- BaseClassifier
- __builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', featmetric='euclidean', leads=('I', 'II', 'III'), reflead='I', median=False, **kwargs)

**autoRejectionThresholds**(self)

**dict2list**(self, dictionary)
```
Gets a dictionary as input and returns a list

Needed so as to join all average ECGs leadwise and feed them to scipy.spatial.distance.cdist()
```

**extract_feats**(self, data, label=None)
```
data - dictionary whose entries are discriminated by leads; entries have a matrix of ECG segments
```

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'dissimilarityClassifier'

---

Methods inherited from KNN:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)
```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
    For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
    Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
    Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
    Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
    data (dict): Dictionary holding the testing samples for each subject.

    rejection_thresholds (array): Thresholds used to compute the ROCs.

    dstPath (string): Path for multiprocessing.
```

```
        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**

```
    dictionary for instance variables (if defined)
```

**__weakref__**
list of weak references to the object (if defined)

class **DissimilarityMore**(KNN)

```
Class to extract multilead distances subjectwise

Performs kNN for classification based on those distances
```

Method resolution order:
DissimilarityMore
KNN
BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', featmetric='euclidean', leads=('I', 'II', 'III'), reflead='I', median=False, meansubs=None, percent=0.15, tn=5, **kwargs)

**autoRejectionThresholds**(self)

**dict2list**(self, dictionary)

```
Gets a dictionary as input and returns a list

Needed so as to join all average ECGs leadwise and feed them to scipy.spatial.distance.cdist()
```

**extract_feats**(self, data, label=None)

```
data - dictionary whose entries are discriminated by leads; entries have a matrix of ECG segments
```

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'dissimilarityClassifier'

---

Methods inherited from KNN:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
    For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
    Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
    Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
    Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
    data (dict): Dictionary holding the testing samples for each subject.

    rejection_thresholds (array): Thresholds used to compute the ROCs.

    dstPath (string): Path for multiprocessing.

    log2file (bool): Flag to control the use of logging in multiprocessing.

Kwrvals:
    classification (dict): Results of the classification.

    assessment (dict): Biometric statistics.

See Also:
```

```
    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**

```
    dictionary for instance variables (if defined)
```

**__weakref__**

```
    list of weak references to the object (if defined)
```

class **DissimilaritySimple**(Dissimilarity)

```
Class to extract multilead distances subjectwise
```

Performs kNN for classification based on those distances

Method resolution order:
DissimilaritySimple
Dissimilarity
KNN
BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', featmetric='euclidean', leads=('I', 'II', 'III'), reflead='I', median=False, tn=5, **kwargs)

**extract_feats**(self, data, label=None)
data - dictionary whose entries are discriminated by leads; entries have a matrix of ECG segments

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'dissimilaritySimpleClassifier'

---

Methods inherited from Dissimilarity:

**autoRejectionThresholds**(self)

**dict2list**(self, dictionary)
Gets a dictionary as input and returns a list

Needed so as to join all average ECGs leadwise and feed them to scipy.spatial.distance.cdist()

---

Methods inherited from KNN:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
data (dict): Dictionary holding the testing samples for each subject.

rejection_thresholds (array): Thresholds used to compute the ROCs.

dstPath (string): Path for multiprocessing.

log2file (bool): Flag to control the use of logging in multiprocessing.

Kwrvals:
classification (dict): Results of the classification.

assessment (dict): Biometric statistics.

See Also:

Notes:

Example:

```
    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**

```
    dictionary for instance variables (if defined)
```

**__weakref__**

```
    list of weak references to the object (if defined)
```

class **DissimilaritySimple_old**(Dissimilarity_old)

```
Class to extract multilead distances subjectwise

Performs kNN for classification based on those distances
```

Method resolution order:

DissimilaritySimple_old
Dissimilarity_old
KNN

BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', featmetric='euclidean', leads=('I', 'II', 'III'), reflead='I', **kwargs)

**extract_feats**(self, data, label=None)
```
data - dictionary whose entries are discriminated by leads; entries have a matrix of ECG segments
```

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'dissimilaritySimpleClassifier'

---

Methods inherited from Dissimilarity_old:

**autoRejectionThresholds**(self)

**dict2list**(self, dictionary)
```
Gets a dictionary as input and returns a list

Needed so as to join all average ECGs leadwise and feed them to scipy.spatial.distance.cdist()
```

---

Methods inherited from KNN:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)
```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
    For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
    Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
    Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
    Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
    data (dict): Dictionary holding the testing samples for each subject.

    rejection_thresholds (array): Thresholds used to compute the ROCs.

    dstPath (string): Path for multiprocessing.

    log2file (bool): Flag to control the use of logging in multiprocessing.

Kwrvals:
    classification (dict): Results of the classification.

    assessment (dict): Biometric statistics.

See Also:


Notes:


Example:


References:
    .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**

```
    dictionary for instance variables (if defined)
```

**__weakref__**

```
    list of weak references to the object (if defined)
```

class **Dissimilarity_old**(KNN)

```
Class to extract multilead distances subjectwise

Performs kNN for classification based on those distances
```

Method resolution order:
- Dissimilarity_old
- KNN
- BaseClassifier
- __builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', featmetric='euclidean', leads=('I', 'II', 'III'), reflead='I', **kwargs)

**autoRejectionThresholds**(self)

**dict2list**(self, dictionary)
```
    Gets a dictionary as input and returns a list

    Needed so as to join all average ECGs leadwise and feed them to scipy.spatial.distance.cdist()
```

**extract_feats**(self, data, label=None)
```
    data - dictionary whose entries are discriminated by leads; entries have a matrix of ECG segments
```

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'dissimilarityClassifier'

---

Methods inherited from KNN:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)
```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
    For each test subject and for each threshold, test authentication and identification;
    Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
    Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
    Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
    data (dict): Dictionary holding the testing samples for each subject.

    rejection_thresholds (array): Thresholds used to compute the ROCs.

    dstPath (string): Path for multiprocessing.

    log2file (bool): Flag to control the use of logging in multiprocessing.

Kwrvals:
    classification (dict): Results of the classification.

    assessment (dict): Biometric statistics.

See Also:


Notes:


Example:


References:
    .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **EmptyCombination**(exceptions.Exception)

Method resolution order:
- EmptyCombination
- exceptions.Exception
- exceptions.BaseException
- __builtin__.object

---

Methods defined here:

**__str__**(self)

---

Data descriptors defined here:

**__weakref__**
```
list of weak references to the object (if defined)
```

---

Methods inherited from exceptions.Exception:

**__init__**(...)
```
x.__init__(...) initializes x; see help(type(x)) for signature
```

---

Data and other attributes inherited from exceptions.Exception:

**__new__** = <built-in method __new__ of type object>
```
T.__new__(S, ...) -> a new object with type S, a subtype of T
```

---

Methods inherited from exceptions.BaseException:

**__delattr__**(...)
```
x.__delattr__('name') <==> del x.name
```

**__getattribute__**(...)
```
x.__getattribute__('name') <==> x.name
```

**__getitem__**(...)
```
x.__getitem__(y) <==> x[y]
```

**__getslice__**(...)
```
x.__getslice__(i, j) <==> x[i:j]

Use of negative indices is not supported.
```

**__reduce__**(...)

**__repr__**(...)
```
x.__repr__() <==> repr(x)
```

**__setattr__**(...)
```
x.__setattr__('name', value) <==> x.name = value
```

**__setstate__**(...)

**__unicode__**(...)

---

Data descriptors inherited from exceptions.BaseException:

**__dict__**

**args**

**message**

class **KNN**(BaseClassifier)
```
k-NN classifier.
```

Method resolution order:
KNN
BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, k=3, metric='euclidean', **kwargs)

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**autoRejectionThresholds**(self)

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = 0

**EXT** = '.hdf5'

**NAME** = 'KNN'

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

## class **LeanSVM**(SVM)

```
Lean SVM classifier: the individual SVM classifiers are saved as .dict files.
```

Method resolution order:
- LeanSVM
- SVM
- BaseClassifier
- __builtin__.object

---

Methods defined here:

**__init__**(self, io=None, **kwargs)

**dirIterator**(self)

**fileIterator**(self)

---

Data and other attributes defined here:

**EXT** = '.hdf5'

**NAME** = 'leanSvmClassifier'

---

Methods inherited from SVM:

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**autoRejectionThresholds**(self)

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes inherited from SVM:

**EER_IDX** = -1

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)
```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
    For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
    Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
    Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
    Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
    data (dict): Dictionary holding the testing samples for each subject.

    rejection_thresholds (array): Thresholds used to compute the ROCs.

    dstPath (string): Path for multiprocessing.

    log2file (bool): Flag to control the use of logging in multiprocessing.

Kwrvals:
```

```
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**
```
    dictionary for instance variables (if defined)
```

**__weakref__**
```
    list of weak references to the object (if defined)
```

class **Odinaka**(BaseClassifier)

`Odinaka` classifier.

Method resolution order:
- Odinaka
- BaseClassifier
- __builtin__.object

---

Methods defined here:

**__init__**(self, io=None, n=None, overlap=0, nfft=None, size=None, window='hamming', k=0.0)

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**autoRejectionThresholds**(self)

**identify**(self, data, ready=False, **kwargs)

**re_train**(self, data)

**train**(self, data=None)

---

Data and other attributes defined here:

**EXT** = '.hdf5'

**NAME** = 'odinakaClassifier'

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
Assess the performance of the classifier in both biometric scenarios: authentication and identification.

Workflow:
    For each test subject and for each threshold, test authentication and identification;
    Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
    Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
    Subject and global statistics are then computed by evaluation.assessClassification.

Kwargs:
    data (dict): Dictionary holding the testing samples for each subject.

    rejection_thresholds (array): Thresholds used to compute the ROCs.

    dstPath (string): Path for multiprocessing.

    log2file (bool): Flag to control the use of logging in multiprocessing.

Kwrvals:
    classification (dict): Results of the classification.

    assessment (dict): Biometric statistics.

See Also:


Notes:


Example:


References:
    .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

---

Data and other attributes inherited from BaseClassifier:

**EER_IDX** = 0

class **SVM**(BaseClassifier)

```
SVM classifier.
```

Method resolution order:
SVM
BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, **kwargs)

```
Wrapper for the scikit-learn SVM classifier, using OneClassSVM, SVC or LinearSVC,
depending on usage case.

sklearn.svm.SVC kwargs:
    C (float): Penalty parameter C of the error term. Default=1.0

    kernel (string): The kernel to be used ('linear', 'poly', 'rbf', 'sigmoid'). Default='rbf'

    degree (int): Degree of kernel function; only significant for 'poly' and 'sigmoid'. Default=3
```

```
        gamma (float): Kernel coefficient for 'rbf' and 'poly'. Default=0.0

        coef0 (float): Independent term in kernel function; it is only significant in 'poly' and 'sigmoid'. Default=0.0

        probability (bool): Whether to enable probability estimates. Default=False

        shrinking (bool): Whether to use the shrinking heuristic. Default=True

        tol (float): Tolerance for stopping criterion. Default=1e-3

        cache_size (float): Specify the size of the kernel cache (in MB).

        class_weight (dict, 'auto'): Weights for each class; the 'auto' mode automatically adjusts the
                                     weights inversely proportional to class frequencies.

        verbose (bool): Enable verbose output. Default=False

        max_iter (int): Hard limit on iterations within solver, or -1 for no limit. Default=-1
```

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**autoRejectionThresholds**(self)

**re_train**(self, data)

**train**(self, data=None, updateThresholds=True)

---

Data and other attributes defined here:

**EER_IDX** = -1

**EXT** = '.hdf5'

**NAME** = 'svmClassifier'

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, create a multiprocessing task that tests authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**fileIterator**(self)

**idThreshold**(self, subject, ready=False)

**identify**(self, data, threshold=None, ready=False, **kwargs)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**

```
    dictionary for instance variables (if defined)
```

**__weakref__**

```
    list of weak references to the object (if defined)
```

class **SifterSVM**(BaseClassifier)

```
SVM classifier.
```

Method resolution order:
SifterSVM
BaseClassifier
__builtin__.object

---

Methods defined here:

**__init__**(self, io=None, edges=None, weights=None, **kwargs)

**authenticate**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**autoRejectionThresholds**(self)

**combination**(self, results)

**dirIterator**(self)

**evaluate**(self, data, rejection_thresholds='auto', dstPath=None, log2file=False)

**fileIterator**(self)

**identify**(self, data, ready=False, **kwargs)

**re_train**(self, data)

**train**(self, data=None)

---

Data and other attributes defined here:

**EXT** = '.hdf5'

**NAME** = 'SifterSVM'

---

Methods inherited from BaseClassifier:

**authThreshold**(self, subject, ready=False)

**checkSubject**(self, subject)

**idThreshold**(self, subject, ready=False)

**io_load**(self, label)

**io_save**(self, label, data)

**listSubjects**(self)

**save**(self, dstPath)

**seqEvaluate**(self, data, rejection_thresholds=None, dstPath=None, log2file=False)

```
    Assess the performance of the classifier in both biometric scenarios: authentication and identification.

    Workflow:
        For each test subject and for each threshold, test authentication and identification;
        Authentication results stored in a 3 dimensional array of booleans, shape = (N thresholds, M subjects, K samples);
        Identification results stored in a 2 dimensional array, shape = (N thresholds, K samples);
        Subject and global statistics are then computed by evaluation.assessClassification.

    Kwargs:
        data (dict): Dictionary holding the testing samples for each subject.

        rejection_thresholds (array): Thresholds used to compute the ROCs.

        dstPath (string): Path for multiprocessing.

        log2file (bool): Flag to control the use of logging in multiprocessing.

    Kwrvals:
        classification (dict): Results of the classification.

        assessment (dict): Biometric statistics.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**setAuthThreshold**(self, subject, threshold, ready=False)

**setIdThreshold**(self, subject, threshold, ready=False)

**updateThresholds**(self, overwrite=False, fraction=1.0)

**updateThresholds_old**(self, overwrite=False, N=1)

---

Class methods inherited from BaseClassifier:

**load**(cls, srcPath, dstPath=None) from __builtin__.type

---

Data descriptors inherited from BaseClassifier:

**__dict__**

dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

---

Data and other attributes inherited from BaseClassifier:

**EER_IDX** = 0

class **SubjectDict**(bidict.bidict)

Method resolution order:
SubjectDict
bidict.bidict
__builtin__.object

---

Methods defined here:

**__getitem__**(self, keyorslice)

---

Methods inherited from bidict.bidict:

**__contains__**(self, *args, **kwds)
D.__contains__(k) -> True if D has a key k, else False

**__delitem__**(self, keyorslice)

**__eq__**(self, other)

**__init__**(self, *args, **kwds)

**__invert__**(self)
Called when unary ~ operator is applied.

**__inverted__**(self)

**__iter__**(self, *args, **kwds)
x.__iter__() <==> iter(x)

**__len__**(self, *args, **kwds)
x.__len__() <==> len(x)

**__neq__**(self, other)

**__repr__**(self)

**__setitem__**(self, keyorslice, keyorval)

**__str__** = __repr__(self)

**clear**(self)

**copy**(self)

**get**(self, *args, **kwds)
D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None.

**invert**(self)

**items**(self, *args, **kwds)
D.items() -> list of D's (key, value) pairs, as 2-tuples

**keys**(self, *args, **kwds)
D.keys() -> list of D's keys

**pop**(self, key, *args)

**popitem**(self)

**setdefault**(self, key, default=None)

**update**(self, *args, **kwds)

**values**(self, *args, **kwds)
D.values() -> list of D's values

---

Data descriptors inherited from bidict.bidict:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**inv**
Called when unary ~ operator is applied.

---

Data and other attributes inherited from bidict.bidict:

**methodname** = 'values'

class **UnknownSubjectError**(exceptions.Exception)

Method resolution order:
UnknownSubjectError
exceptions.Exception
exceptions.BaseException
__builtin__.object

---

Methods defined here:

**__init__**(self, subject)

**__str__**(self)

---

Data descriptors defined here:

**__weakref__**
list of weak references to the object (if defined)

---

Data and other attributes inherited from exceptions.Exception:

**__new__** = <built-in method __new__ of type object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from exceptions.BaseException:

**__delattr__**(...)
x.__delattr__('name') <==> del x.name

**__getattribute__**(...)
x.__getattribute__('name') <==> x.name

**__getitem__**(...)
x.__getitem__(y) <==> x[y]

**__getslice__**(...)
x.__getslice__(i, j) <==> x[i:j]

Use of negative indices is not supported.

**__reduce__**(...)

**__repr__**(...)
x.__repr__() <==> repr(x)

**__setattr__**(...)
x.__setattr__('name', value) <==> x.name = value

**__setstate__**(...)

**__unicode__**(...)

---

Data descriptors inherited from exceptions.BaseException:

**__dict__**

**args**

**message**

class **UntrainedError**(exceptions.Exception)

Method resolution order:
UntrainedError
exceptions.Exception
exceptions.BaseException
__builtin__.object

---

Methods defined here:

**__str__**(self)

---

Data descriptors defined here:

**__weakref__**
list of weak references to the object (if defined)

---

Methods inherited from exceptions.Exception:

**__init__**(...)
x.__init__(...) initializes x; see help(type(x)) for signature

---

Data and other attributes inherited from exceptions.Exception:

**__new__** = <built-in method __new__ of type object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from exceptions.BaseException:

**__delattr__**(...)
x.__delattr__('name') <==> del x.name

**__getattribute__**(...)
x.__getattribute__('name') <==> x.name

**__getitem__**(...)
x.__getitem__(y) <==> x[y]

**__getslice__**(...)
x.__getslice__(i, j) <==> x[i:j]

Use of negative indices is not supported.

**__reduce__**(...)

**__repr__**(...)
x.__repr__() <==> repr(x)

**__setattr__**(...)
x.__setattr__('name', value) <==> x.name = value

**__setstate__**(...)

**__unicode__**(...)

---

Data descriptors inherited from exceptions.BaseException:

**__dict__**

**args**

**message**

class **fisher**(__builtin__.object)
# OLD STUFF

Methods defined here:

**__init__**(self, cStep=0.1, **kwargs)
# Init

**authenticate**(self, data, subject, threshold=1.0, labels=False, **kwargs)

**evaluate**(self, data, rejection_thresholds='auto', dstPath=None)

**identify**(self, data, **kwargs)

**train**(self, data=None)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

## Functions

**authenticate_Comb**(self, data, subject, threshold=None, ready=False, labels=False, **kwargs)

**combinationDecorator**(cls)

**eerIndex**(method)

**selector**(method)
```
Selector for the classifier functions and methods.

Input:
    method (str): The desired function or method.

Output:
    fcn (function): The function pointer.

Configurable fields:

See Also:


Example:


References:
    .. [1]
```

**train_Comb**(self, data=None, updateThresholds=True, **kwargs)

**updateThresholds_Comb**(self, overwrite=False, N=1, **kwargs)

# bpkclone.classifiers.rules

index

d:\work\productioncode\clones\biometricspykit\bpkclone\classifiers\rules.py

```
.. module:: rules
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Carlos Carreiras
```

## Modules

numpy

## Classes

exceptions.Exception(exceptions.BaseException)

EmptyCombination

class **EmptyCombination**(exceptions.Exception)

Method resolution order:

EmptyCombination
exceptions.Exception
exceptions.BaseException
__builtin__.object

---

Methods defined here:

**__str__**(self)

---

Data descriptors defined here:

**__weakref__**
list of weak references to the object (if defined)

---

Methods inherited from exceptions.Exception:

**__init__**(...)
x.__init__(...) initializes x; see help(type(x)) for signature

---

Data and other attributes inherited from exceptions.Exception:

**__new__** = <built-in method __new__ of type object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from exceptions.BaseException:

**__delattr__**(...)
x.__delattr__('name') <==> del x.name

**__getattribute__**(...)
x.__getattribute__('name') <==> x.name

**__getitem__**(...)
x.__getitem__(y) <==> x[y]

**__getslice__**(...)
```
x.__getslice__(i, j) <==> x[i:j]

Use of negative indices is not supported.
```

**__reduce__**(...)

**__repr__**(...)
```
x.__repr__() <==> repr(x)
```

**__setattr__**(...)
```
x.__setattr__('name', value) <==> x.name = value
```

**__setstate__**(...)

**__unicode__**(...)

---

Data descriptors inherited from exceptions.BaseException:

**__dict__**

**args**

**message**

## Functions

**combination**(labels)

**majorityRule**(labels)

**pluralityRule**(labels)

# bpkclone.Cloud.client

index d:\work\productioncode\clones\biometricspykit\bpkclone\cloud\client.py

```
.. module:: client
   :platform: Unix, Windows
   :synopsis: This module implements the BITCloud client.

.. moduleauthor:: Carlos Carreiras
```

## Modules

copy os pymongo

## Classes

connection

class **connection**

```
Class to connect to a BITCloud server.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:


References:
    .. [1]
```

Methods defined here:

**__init__**(self, host='localhost', user=None, passwd=None, queue='BiometricsQ', dbName='BiometricsExperiments')

```
Establish the connection to the server.

Kwargs:


Kwrvals:


See Also:


Notes:


Example:
```

```
    References:
        .. [1]
```

**addTask**(self, task=None)

**cancelTask**(self, expID)

**close**(self)

**collectResults**(self, expIDList, mapper, basePath, headerOrder=None)

**getTaskInfo**(self, expID)

**listTasks**(self, status='all', seen=None)

**markSeen**(self, expID)

**requeue**(self, expID)

**search**(self, spec=None, fields=None)

---

Data and other attributes defined here:

**MONGO_HOST** = '193.136.222.234'

**MONGO_PORT** = 27017

**RABBIT_PORT** = 5672

**TRANSPORT** = 'amqp'

**VHOST** = '/'

# bpkclone.Cloud.parallel

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cloud\parallel.py

```
.. module:: server
   :platform: Unix, Windows
   :synopsis: This module implements the BITCloud parallelization services.

.. moduleauthor:: Carlos Carreiras
```

## Modules

errno multiprocessing os traceback
multiprocessing.managers numpy time

## Classes

__builtin__.object

DoWork

DoWork_Clf
DoWork_Distances
DoWork_File

class **DoWork**(__builtin__.object)

Methods defined here:

**__init__**(self, workQueue, store, timeout, loggerDict, lock=None)

**go**(self)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **DoWork_Clf**(DoWork)

Method resolution order:
DoWork_Clf
DoWork
__builtin__.object

---

Methods defined here:

**go**(self)

---

Methods inherited from DoWork:

**__init__**(self, workQueue, store, timeout, loggerDict, lock=None)

Data descriptors inherited from DoWork:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

## class **DoWork_Distances**(DoWork)

Method resolution order:
DoWork_Distances
DoWork
__builtin__.object

Methods defined here:

**go**(self)

Methods inherited from DoWork:

**__init__**(self, workQueue, store, timeout, loggerDict, lock=None)

Data descriptors inherited from DoWork:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

## class **DoWork_File**(DoWork)

Method resolution order:
DoWork_File
DoWork
__builtin__.object

Methods defined here:

**go**(self)

Methods inherited from DoWork:

**__init__**(self, workQueue, store, timeout, loggerDict, lock=None)

Data descriptors inherited from DoWork:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

## Functions

**cleanStore**(store)

**copy**(data)

**getCondition**()

**getDictManager**()

**getEventManager**()

**getIntValue**(value=None)

**getListManager**()

**getLock**()

**getManager**()

**getPipe**()

**getPool**(processes)

**getQueue**()

**loadStore**(store, taskid)

**randomTester**(data)

**runMultiprocess**(workQueue, store, mode='data', numberProcesses=None, lock=None, timeout=None, log2file=False)

**workerProcess**(workClass, workQueue, store, timeout, loggerDict, randomSeed, lock=None)

# bpkclone.Cloud.SerialPortServer

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cloud\serialportserver.py

```
.. module:: WebServer
   :platform: Windows, Linux
   :synopsis: WebSockets server.

.. moduleauthor:: Ana Priscila Alves, Carlos Carreiras
```

## Modules

| | | |
|---|---|---|
| Queue | twisted.internet.protocol | time |
| twisted.internet.error | multiprocessing.queues | traceback |
| multiprocessing | threading | uuid |

## Classes

multiprocessing.queues.Queue(__builtin__.object)

- LockQueue
  - DummyComQueue

threading.Thread(threading._Verbose)

- SPApp
  - Echo
- SPCom

twisted.internet.protocol.Protocol(twisted.internet.protocol.BaseProtocol)

- SP

class **DummyComQueue**(LockQueue)

Method resolution order:
- DummyComQueue
- LockQueue
- multiprocessing.queues.Queue
- __builtin__.object

---

Methods defined here:

**get**(self, *args, **kwargs)

**put**(self, item)

---

Methods inherited from LockQueue:

**__getstate__**(self)

**__init__**(self, cleanup=True, *args, **kwargs)

**__setstate__**(self, state)

**lock**(self)

**unlock**(self)

---

Methods inherited from multiprocessing.queues.Queue:

**cancel_join_thread**(self)

**close**(self)

**empty**(self)

**full**(self)

**get_nowait**(self)

**join_thread**(self)

**put_nowait**(self, obj)

**qsize**(self)

---

Data descriptors inherited from multiprocessing.queues.Queue:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **Echo**(SPApp)

Method resolution order:
Echo
SPApp
threading.Thread
threading._Verbose
__builtin__.object

---

Methods defined here:

**on_close**(self)

**on_message**(self, message)

**on_open**(self)

---

Methods inherited from SPApp:

**__init__**(self, queue=None)

**restartReactor**(self)

**run**(self)

**send**(self, message)

**startCom**(self)

**stopCom**(self)

**stopReactor**(self)

---

Class methods inherited from SPApp:

**instantiate**(cls, *args, **kwargs) from __builtin__.type

---

Methods inherited from threading.Thread:

**__repr__**(self)

**getName**(self)

**isAlive**(self)

**isDaemon**(self)

**is_alive** = isAlive(self)

**join**(self, timeout=None)

**setDaemon**(self, daemonic)

**setName**(self, name)

**start**(self)

---

Data descriptors inherited from threading.Thread:

**daemon**

**ident**

**name**

---

Data descriptors inherited from threading._Verbose:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **LockQueue**(multiprocessing.queues.Queue)

Method resolution order:
- LockQueue
- multiprocessing.queues.Queue
- __builtin__.object

---

Methods defined here:

**__getstate__**(self)

**__init__**(self, cleanup=True, *args, **kwargs)

**__setstate__**(self, state)

**get**(self, *args, **kwargs)

**lock**(self)

**put**(self, *args, **kwargs)

**unlock**(self)

---

Methods inherited from multiprocessing.queues.Queue:

**cancel_join_thread**(self)

**close**(self)

**empty**(self)

**full**(self)

**get_nowait**(self)

**join_thread**(self)

**put_nowait**(self, obj)

**qsize**(self)

---

Data descriptors inherited from multiprocessing.queues.Queue:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
list of weak references to the object (if defined)

class **SP**(twisted.internet.protocol.Protocol)

Method resolution order:
SP
twisted.internet.protocol.Protocol
twisted.internet.protocol.BaseProtocol

---

Methods defined here:

**__init__**(self, app, durable, *args, **kwargs)

**connectionLost**(self, reason)

**connectionMade**(self)

**dataReceived**(self, data)

**send**(self, data)

---

Methods inherited from twisted.internet.protocol.Protocol:

**__provides__**
Special descriptor for class __provides__

The descriptor caches the implementedBy info, so that
we can get declarations for objects without instance-specific
interfaces a bit quicker.

**logPrefix**(self)
Return a prefix matching the class name, to identify log messages
related to this protocol instance.

---

Data and other attributes inherited from twisted.internet.protocol.Protocol:

**__implemented__** = <implementedBy twisted.internet.protocol.Protocol>

---

Methods inherited from twisted.internet.protocol.BaseProtocol:
**__providedBy__** = <zope.interface.declarations.Declaration object>

**makeConnection**(self, transport)
Make a connection to a transport and a server.

This sets the 'transport' attribute of this Protocol, and calls the
connectionMade() callback.

---

Data and other attributes inherited from twisted.internet.protocol.BaseProtocol:

**connected** = 0

**transport** = None

class **SPApp**(threading.Thread)

Method resolution order:
SPApp
threading.Thread
threading._Verbose
__builtin__.object

---

Methods defined here:

**__init__**(self, queue=None)

**on_close**(self)

**on_message**(self, message)

**on_open**(self)

**restartReactor**(self)

**run**(self)

**send**(self, message)

**startCom**(self)

**stopCom**(self)

**stopReactor**(self)

---

Class methods defined here:

**instantiate**(cls, *args, **kwargs) from __builtin__.type

---

Methods inherited from threading.Thread:

**__repr__**(self)

**getName**(self)

**isAlive**(self)

**isDaemon**(self)

**is_alive** = isAlive(self)

**join**(self, timeout=None)

**setDaemon**(self, daemonic)

**setName**(self, name)

**start**(self)

---

Data descriptors inherited from threading.Thread:

**daemon**

**ident**

**name**

---

Data descriptors inherited from threading._Verbose:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **SPCom**(threading.Thread)

Method resolution order:
- SPCom
- threading.Thread
- threading._Verbose
- __builtin__.object

---

Methods defined here:

**__init__**(self, go, queue)

**run**(self)

Methods inherited from threading.Thread:

**__repr__**(self)

**getName**(self)

**isAlive**(self)

**isDaemon**(self)

**is_alive** = isAlive(self)

**join**(self, timeout=None)

**setDaemon**(self, daemonic)

**setName**(self, name)

**start**(self)

---

Data descriptors inherited from threading.Thread:

**daemon**

**ident**

**name**

---

Data descriptors inherited from threading._Verbose:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

## Functions

**serverPort**(cls, port='COM30', baud=9600, setupFcn=None, cleanupFcn=None, durable=True, *args, **kwargs)

## Data

**default_timeout** = None
**reactor** = <twisted.internet.selectreactor.SelectReactor object>

# bpkclone.Cloud.server

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cloud\server.py

```
.. module:: server
   :platform: Unix, Windows
   :synopsis: This module implements the BITCloud server.

.. moduleauthor:: Carlos Carreiras
```

## Modules

| | | | |
|---|---|---|---|
| collections | matplotlib | signal | time |
| json | os | smtplib | |
| logging | pymongo | sys | |

## Classes

__builtin__.object

MQWorker

exceptions.Exception(exceptions.BaseException)

KillSignal

class **KillSignal**(exceptions.Exception)

Method resolution order:
KillSignal
exceptions.Exception
exceptions.BaseException
__builtin__.object

---

Methods defined here:

**__init__**(self, sig)

**__str__**(self)

---

Data descriptors defined here:

**__weakref__**
list of weak references to the object (if defined)

---

Data and other attributes inherited from exceptions.Exception:

**__new__** = <built-in method __new__ of type object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from exceptions.BaseException:

**__delattr__**(...)
x.__delattr__('name') <==> del x.name

**__getattribute__**(...)
```
x.__getattribute__('name') <==> x.name
```

**__getitem__**(...)
```
x.__getitem__(y) <==> x[y]
```

**__getslice__**(...)
```
x.__getslice__(i, j) <==> x[i:j]

Use of negative indices is not supported.
```

**__reduce__**(...)

**__repr__**(...)
```
x.__repr__() <==> repr(x)
```

**__setattr__**(...)
```
x.__setattr__('name', value) <==> x.name = value
```

**__setstate__**(...)

**__unicode__**(...)

---

Data descriptors inherited from exceptions.BaseException:

**__dict__**

**args**

**message**

class **MQWorker**(__builtin__.object)
```
Consumer class
```

Methods defined here:

**__init__**(self, rabbitHost='localhost', mongoHost='localhost', workerName=None, dbName='BiometricsExperiments', queue='BiometricsQ', dstPath='~', mode='production', parameters=None)

**acknowledge_message**(self, message, taskID)

**declare_queue**(self)

**go**(self)

**notify**(self, username, taskID, message)

**on_message**(self, message)

**stop**(self)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
list of weak references to the object (if defined)

---

Data and other attributes defined here:

**EMAIL_SENDER** = 'carlos.carreiras@lx.it.pt'

**EMAIL_SUBJECT** = '[BiometricsPyKit] Status of Task %d'

**FAILURE_MSG** = '\n Dear %s,\n \n Unfortunalety, Task %d ha...le on the %s machine.\n \n Kind regards.\n '

**MONGO_PORT** = 27017

**PASSWD** = 'worker'

**RABBIT_PORT** = 5672

**REPEAT_MSG** = '\n Dear %s,\n \n Task %d has already been ...ht by the %s machine.\n \n Kind regards.\n '

**SUCCESS_MSG** = '\n Dear %s,\n \n Task %d has been success...le on the %s machine.\n \n Kind regards.\n '

**TIMEOUT** = 30

**TRANSPORT** = 'amqp'

**VHOST** = '/'

## Functions

**convertUniDict**(data)

**onKillSignal**(sig, func=None)

**processTask**(conn, task, taskID, parameters)

**processTask_test**(conn, task, taskID, parameters)

**sendEmail**(To, From, CC='', ReplyTo='', Subject='', Text='', FilePath=None, Host='cascais.lx.it.pt', Port=25)

**setKillHandler**(func)

## Data

**Encoders** = <email.LazyImporter object>

# bpkclone.Cloud.webServer

index d:\work\productioncode\clones\biometricspykit\bpkclone\cloud\webserver.py

```
.. module:: WebServer
   :platform: Windows, Linux
   :synopsis: WebSockets server.

.. moduleauthor:: Ana Priscila Alves, Carlos Carreiras
```

## Modules

| | | | |
|---|---|---|---|
| Queue | multiprocessing | sys | traceback |
| ujson | twisted.internet.protocol | threading | uuid |
| logging | multiprocessing.queues | time | websocket |

## Classes

__builtin__.object

- WSClient
  - TestClient

multiprocessing.queues.Queue(__builtin__.object)

- LockQueue
  - DummyComQueue

threading.Thread(threading._Verbose)

- WSApp
  - Echo
- WSCom

twisted.internet.protocol.Factory

- WSFactory

twisted.internet.protocol.Protocol(twisted.internet.protocol.BaseProtocol)

- WS

class **DummyComQueue**(LockQueue)

Method resolution order:

- DummyComQueue
- LockQueue
- multiprocessing.queues.Queue
- __builtin__.object

---

Methods defined here:

**get**(self, *args, **kwargs)

**put**(self, item)

---

Methods inherited from LockQueue:

**__getstate__**(self)

**__init__**(self, cleanup=True, *args, **kwargs)

**__setstate__**(self, state)

**lock**(self)

**unlock**(self)

---

Methods inherited from multiprocessing.queues.Queue:

**cancel_join_thread**(self)

**close**(self)

**empty**(self)

**full**(self)

**get_nowait**(self)

**join_thread**(self)

**put_nowait**(self, obj)

**qsize**(self)

---

Data descriptors inherited from multiprocessing.queues.Queue:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **Echo**(WSApp)

Method resolution order:
- Echo
- WSApp
- threading.Thread
- threading._Verbose
- __builtin__.object

---

Methods defined here:

**on_close**(self)

**on_message**(self, message)

**on_open**(self)

---

Methods inherited from WSApp:

**__init__**(self, queue=None)

**run**(self)

**send**(self, message)

**startCom**(self)

**stopCom**(self)

---

Class methods inherited from WSApp:

**instantiate**(cls, *args, **kwargs) from __builtin__.type

---

Methods inherited from threading.Thread:

**__repr__**(self)

**getName**(self)

**isAlive**(self)

**isDaemon**(self)

**is_alive** = isAlive(self)

**join**(self, timeout=None)

**setDaemon**(self, daemonic)

**setName**(self, name)

**start**(self)

---

Data descriptors inherited from threading.Thread:

**daemon**

**ident**

**name**

---

Data descriptors inherited from threading._Verbose:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **LockQueue**(multiprocessing.queues.Queue)

Method resolution order:
- LockQueue
- multiprocessing.queues.Queue
- __builtin__.object

---

Methods defined here:

**__getstate__**(self)

**__init__**(self, cleanup=True, *args, **kwargs)

**__setstate__**(self, state)

**get**(self, *args, **kwargs)

**lock**(self)

**put**(self, *args, **kwargs)

**unlock**(self)

---

Methods inherited from multiprocessing.queues.Queue:

**cancel_join_thread**(self)

**close**(self)

**empty**(self)

**full**(self)

**get_nowait**(self)

**join_thread**(self)

**put_nowait**(self, obj)

**qsize**(self)

---

Data descriptors inherited from multiprocessing.queues.Queue:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

## class **TestClient**(WSClient)

Method resolution order:
- TestClient
- WSClient
- __builtin__.object

---

Methods defined here:

**__init__**(self, *args, **kwargs)

**on_close**(self)

**on_message**(self, message)

**on_open**(self)

---

Methods inherited from WSClient:

**close**(self)
```
close websocket connection.
```

**on_error**(self, reason)

**run_forever**(self, sockopt=None, sslopt=None, ping_interval=0)
```
run event loop for WebSocket framework.
This loop is infinite loop and is alive during websocket is available.
sockopt: values for socket.setsockopt.
    sockopt must be tuple and each element is argument of sock.setscokopt.
sslopt: ssl socket optional dict.
ping_interval: automatically send "ping" command every specified period(second)
    if set to 0, not send automatically.
```

**send**(self, data, opcode=None)
```
send message.
data: message to send. If you set opcode to OPCODE_TEXT, data must be utf-8 string or unicode.
opcode: operation code of data. default is OPCODE_TEXT.
```

---

Class methods inherited from WSClient:

**instantiate**(cls, url, *args, **kwargs) from __builtin__.type

---

Data descriptors inherited from WSClient:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

## class **WS**(twisted.internet.protocol.Protocol)

Method resolution order:
- WS
- twisted.internet.protocol.Protocol
- twisted.internet.protocol.BaseProtocol

---

Methods defined here:

**__init__**(self, app, durable, *args, **kwargs)

**connectionLost**(self, reason)

**connectionMade**(self)

**dataReceived**(self, data)

**send**(self, data)

---

Methods inherited from twisted.internet.protocol.Protocol:

**__provides__**
```
Special descriptor for class __provides__

The descriptor caches the implementedBy info, so that
we can get declarations for objects without instance-specific
interfaces a bit quicker.
```

**logPrefix**(self)
```
Return a prefix matching the class name, to identify log messages
related to this protocol instance.
```

---

Data and other attributes inherited from twisted.internet.protocol.Protocol:

**__implemented__** = <implementedBy twisted.internet.protocol.Protocol>

---

Methods inherited from twisted.internet.protocol.BaseProtocol:
**__providedBy__** = <zope.interface.declarations.Declaration object>

**makeConnection**(self, transport)
```
Make a connection to a transport and a server.

This sets the 'transport' attribute of this Protocol, and calls the
connectionMade() callback.
```

---

Data and other attributes inherited from twisted.internet.protocol.BaseProtocol:

**connected** = 0

**transport** = None

class **WSApp**(threading.Thread)

Method resolution order:

WSApp
threading.Thread
threading._Verbose
__builtin__.object

---

Methods defined here:

**__init__**(self, queue=None)

**on_close**(self)

**on_message**(self, message)

**on_open**(self)

**run**(self)

**send**(self, message)

**startCom**(self)

**stopCom**(self)

---

Class methods defined here:

**instantiate**(cls, *args, **kwargs) from __builtin__.type

---

Methods inherited from threading.Thread:

**__repr__**(self)

**getName**(self)

**isAlive**(self)

**isDaemon**(self)

**is_alive** = isAlive(self)

**join**(self, timeout=None)

**setDaemon**(self, daemonic)

**setName**(self, name)

**start**(self)

---

Data descriptors inherited from threading.Thread:

**daemon**

**ident**

**name**

---

Data descriptors inherited from threading._Verbose:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **WSClient**(__builtin__.object)

```
Cloned from websocket.WebSocketApp
The interface is like JavaScript WebSocket object.
```

Methods defined here:

**__init__**(self, url, header=[], keep_running=True, get_mask_key=None)

```
 url: websocket url.
 header: custom header for websocket handshake.
keep_running: a boolean flag indicating whether the app's main loop should
  keep running, defaults to True
get_mask_key: a callable to produce new mask keys, see the WebSocket.set_mask_key's
  docstring for more information
```

**close**(self)

```
close websocket connection.
```

**on_close**(self)

**on_error**(self, reason)

**on_message**(self, message)

**on_open**(self)

**run_forever**(self, sockopt=None, sslopt=None, ping_interval=0)

```
run event loop for WebSocket framework.
This loop is infinite loop and is alive during websocket is available.
sockopt: values for socket.setsockopt.
    sockopt must be tuple and each element is argument of sock.setscokopt.
sslopt: ssl socket optional dict.
ping_interval: automatically send "ping" command every specified period(second)
    if set to 0, not send automatically.
```

**send**(self, data, opcode=None)

```
send message.
data: message to send. If you set opcode to OPCODE_TEXT, data must be utf-8 string or unicode.
opcode: operation code of data. default is OPCODE_TEXT.
```

---

Class methods defined here:

**instantiate**(cls, url, *args, **kwargs) from __builtin__.type

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **WSCom**(threading.Thread)

Method resolution order:
WSCom
threading.Thread
threading._Verbose
__builtin__.object

---

Methods defined here:

**__init__**(self, go, queue)

**run**(self)

---

Methods inherited from threading.Thread:

**__repr__**(self)

**getName**(self)

**isAlive**(self)

**isDaemon**(self)

**is_alive** = isAlive(self)

**join**(self, timeout=None)

**setDaemon**(self, daemonic)

**setName**(self, name)

**start**(self)

---

Data descriptors inherited from threading.Thread:

**daemon**

**ident**

**name**

---

Data descriptors inherited from threading._Verbose:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **WSFactory**(twisted.internet.protocol.Factory)

Methods defined here:

**__init__**(self, app, durable, *args, **kwargs)

**buildProtocol**(self, addr)

---

Methods inherited from twisted.internet.protocol.Factory:
**__providedBy__** = <zope.interface.declarations.Declaration object>

**__provides__**
```
Special descriptor for class __provides__
```

```
    The descriptor caches the implementedBy info, so that
    we can get declarations for objects without instance-specific
    interfaces a bit quicker.
```

**doStart**(self)
```
    Make sure startFactory is called.

    Users should not call this function themselves!
```

**doStop**(self)
```
    Make sure stopFactory is called.

    Users should not call this function themselves!
```

**logPrefix**(self)
```
    Describe this factory for log messages.
```

**startFactory**(self)
```
    This will be called before I begin listening on a Port or Connector.

    It will only be called once, even if the factory is connected
    to multiple ports.

    This can be used to perform 'unserialization' tasks that
    are best put off until things are actually running, such
    as connecting to a database, opening files, etcetera.
```

**stopFactory**(self)
```
    This will be called before I stop listening on all Ports/Connectors.

    This can be overridden to perform 'shutdown' tasks such as disconnecting
    database connections, closing files, etc.

    It will be called, for example, before an application shuts down,
    if it was connected to a port. User code should not call this function
    directly.
```

---

Class methods inherited from twisted.internet.protocol.Factory:

**forProtocol**(cls, protocol, *args, **kwargs) from __builtin__.classobj
```
    Create a factory for the given protocol.

    It sets the C{protocol} attribute and returns the constructed factory
    instance.

    @param protocol: A L{Protocol} subclass

    @param args: Positional arguments for the factory.

    @param kwargs: Keyword arguments for the factory.

    @return: A L{Factory} instance wired up to C{protocol}.
```

---

Data and other attributes inherited from twisted.internet.protocol.Factory:

**__implemented__** = <implementedBy twisted.internet.protocol.Factory>

**noisy** = True

**numPorts** = 0

**protocol** = None

## Functions

**connectWS**(cls, host=None, port=9999, SSL=False, *args, **kwargs)

**serveWS**(cls, port=9999, setupFcn=None, cleanupFcn=None, durable=True, SSL=False, *args, **kwargs)

## Data

**SSL_OK** = False
**default_timeout** = None
**logger** = <logging.Logger object>
**reactor** = <twisted.internet.selectreactor.SelectReactor object>
**ssl** = None

# bpkclone.cluster.cluster

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cluster\cluster.py

```
.. module:: cluster
   :platform: Unix, Windows
   :synopsis: This module provides various functions to cluster data

.. moduleauthor:: Filipe Canento, Andre Lourenco, Carlos Carreiras

.. conventions:
    output of type dict {'-1': indexes, '0': indexes, '1': indexes, ...}
```

## Modules

scipy.spatial.distance
scipy.cluster.hierarchy
os
scipy
glob
numpy
pylab

## Functions

**clustersFromFile**(filePath, method, criterion, nsegs=None)

**convertMatFiles**(srcPath, dstPath, method, criterion, nsegs=None)

**dbscan**(data, min_samples=10, eps=0.95, metric='euclidean')

```
Perform clustering using the DBSCAN method. Uses: sklearn.cluster.DBSCAN.

Input:
    data (array): input data array of shape (number samples x number features).

    min_samples (int): minimum number of samples in a cluster.

    eps (float): maximum distance between two samples in the same cluster.

    metric (string): distance metric

Output:
    res (dict): output dict with indexes for each cluster determined.
                Example: res = {'clusters': {{'-1': noise indexes list,
                                              '0': cluster 0 indexes list,
                                              '1': cluster 1 indexes list}}}

Configurable fields:
{"name": "cluster.dbscan", "config": {"min_samples": "10", "eps": "0.95", "metric": "euclidean"}, "inputs": ["data"], "outputs": ["core_sampl

See Also:
    sklearn.cluster.DBSCAN
    scipy.spatial.distance

Notes:
    conversion to similarity uses the same metric distance as dbscan

Example:


References:
    .. [1]    DBSCAN: Ester, M., Kriegel, H.-P., Sander,J. and Xu,X. (1996), "A density-
based algorithm for discovering clusters in large spatial databases with noise". Proc of 2nd Int Conf on Knowledge Discovery and Data Mining
    .. [2]    dbscan: http://scikit-learn.org/dev/modules/generated/sklearn.cluster.DBSCAN.html
    .. [3]    pdist: http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html#scipy.spatial.distance.pdist
```

**floor**(...)

floor(x)

```
Return the floor of x as a float.
This is the largest integral value <= x.
```

**hierarchical**(data, k=0, metric='euclidean', method='average', showDendrogram=True)

```
Clustering using hierarchical agglomerative methods

Input:
    data : (array): input data array of shape (number samples x number features).
    k: (int) : number of clusters (k=0 using life-time criteria)
    method: (str) 'single'   --- nearest distance (default)
                  'complete' --- furthest distance
                  'average'  --- average distance
                  'centroid' --- center of mass distance
                  'ward'     --- inner squared distance

Output:
    res (dict): output dict with indexes for each cluster determined.
                Example: res = {"labels": list(labels),
                                "k":k,
                                "nsamples_in_cluster":list(nsamples_in_cluster),
                                "algorithm":"hierarchical-"+method}

Configurable fields:
{"name": "cluster.dbscan", "config": {"min_samples": "10", "eps": "0.95", "metric": "euclidean"}, "inputs": ["data"], "outputs": ["core_sampl

See Also:
```

Example:

References:
.. [1]

**leadHierarchical**(data, **kwargs)
Perform hierarchical clustering for each lead independently.

**run**(data, tasks, parameters, outPath)
Run the clustering methods for each item in data, according to the given parameters, then perform the prototype selection.

Input:
data

tasks

parameters

outPath

Output:
output

Configurable fields:{"name": "??.??", "config": {}, "inputs": [], "outputs": []}

See Also:

Notes:

Example:

**selector**(method)
Selector for the clustering functions and methods.

Input:
method (str): The desired function or method.

Output:
fcn (function): The function pointer.

Configurable fields:

See Also:

Example:

References:
.. [1]

**templateSelection**(data, clusters=None, method='MDIST', ntemplatesPerCluster=1, metric='euclidean')
Template Selection

This function implements several template selection methods. It may use the information of cluster labels
[passed as argument] in conjunction with the criteria that are implemented. The number of templates per cluster
can be controlled by ntemplatesPerCluster argument. The number returned templates is variable and depends on the
number of clusters and on the number of templates per clusters [If clusters are singletons the number of templates
per cluster may be violated]. It also works with different distance Measures [distMeasure argument]
Currently the implemented criteria are:
1) Centroids: requires that clusters are provided; if number of templates per cluster is 1, then templates are just
the centroids; if the number of clusters is > 1 then k-means is used to determine more templates on each cluster.
2) MDIST: criterion described in U. Uludag, A. Ross, A. Jain, Pattern Recognition 37 (2004). Originally it does not
require any clustering labels, computing the templates based on distances to the mean wave. The number of templates
supplied by the used, in this cases we assume ntemplatesPerCluster argument, as the number of templates. We extended
the idea, including cluster label information. We compute for each cluster ntemplatesPerCluster using the distance to
mean wave criterion.
3) DEND: criterion described in U. Uludag, A. Ross, A. Jain, Pattern Recognition 37 (2004). The implemented version
is an extension that works with any clustering algorithm (uses the clusters labels passed by argument). The original
version used Complete-linkage. It computes medoids on each cluster, based on the average pairwise distance to other
points in the cluster. Templates are the points whose distance is minimum; we allow to chose more than 1 template,
selecting ntemplatesPerCluster using the same criterion

Input:
data : (array): input data array of shape (number samples x number features).
clusters: (dict): dictionary with labels (can be None)
method: (string): method to be used ('Centroids', 'MDIST', 'DEND'/'Medoid')
ntemplatesPerCluster: (int): number of templates on each cluster
distMeasure: (string): distance measure to be used ('Euclidean' or 'Cosine')
Output:
res (dict): output dict with indexes of prototypes for each cluster .
Example: res = {'templates': list(),
'algorithm': method}

Configurable fields:

See Also:

Example:

References: [1] U. Uludag, A. Ross, A. Jain, Pattern Recognition 37 (2004)

**templateSelectionFixingK**(data, clusters=None, method='MDIST', ntemplates=10, distMeasure=None)

```
Template Selection Fixing the number of templates (argument ntemplates)
```

**time**(...)

```
time() -> floating point number

Return the current time in seconds since the Epoch.
Fractions of a second may be present if the system clock provides them.
```

**transform2dissimilaritySpace**(data, templates, distMeasure=None)

```
Transform to Dissimilarity Representation

This function computes a new representation for the data, based on the distance to the templates.
This representation has as number of dimensions the number of templates (columns)
```

# bpkclone.cluster.clusteringCombination

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cluster\clusteringcombination.py

```
###################################
# Evidence Accumulation Clustering (EAC)
# Script used for building the clustering ensembles its combination using EAC
#
# Andre Lourenco, arlourenco@lx.it.pt
# Ana Fred, afred@lx.it.pt
# Instituto de Telecomunicaces, Instituto Superior Tecnico,
# Technical University of Lisbon, Portugal
# IST - Torre Norte, Av. Rovisco Pais, 1049-001, Lisbon, Portugal
###################################
```

## Modules

| | | |
|---|---|---|
| scipy.cluster.hierarchy | os | scipy |
| numpy | pylab | scipy.sparse |

## Functions

**coassociationCreation**(ensemble, ns, debug=False)

```
    Construction of the co-association matrix

    Input:
        ensemble (dic):

        ns (int): number of samples

    Output:
        assocs : co-association matrix


    Configurable fields:
    {"name": "cluster.dbscan", "config": {"min_samples": "10", "eps": "0.95", "metric": "euclidean"}, "inputs": ["data"], "outputs": ["core_sampl

    See Also:

    Example:


    References:
        .. [1]
```

**consensusExtraction**(coassocs, k=0, method='single', showDendrogram=False)

```
    Extraction of the consensus partition using hierarchical agglomerative methods

    Input:
        assos : co-association matrix
        method='single'   --- nearest distance (default)
               'complete' --- furthest distance
               'average'  --- average distance
               'centroid' --- center of mass distance
                'ward'     --- inner squared distance
         k : number of clusters (k=0 using life-time criteria)

    Output:
        res (dict): output dict with indexes for each cluster determined.
                    Example: res = {"labels": list(labels),
                                    "k":k,
                                    "nsamples_in_cluster":list(nsamples_in_cluster),
                                    "algorithm":"EAC-"+method}

    Configurable fields:
    {"name": "cluster.dbscan", "config": {"min_samples": "10", "eps": "0.95", "metric": "euclidean"}, "inputs": ["data"], "outputs": ["core_sampl

    See Also:


    Example:


    References:
        .. [1]
```

**ensembleCreation**(data, nensemble=100, showHist=False, fun='KMeans', kmin=None, kmax=None)

```
    creation of the ensemble using kmeans
    Input:
        data (array): input data array of shape (number samples x number features).

        N_ensemble (int):  number of partitions on the ensemble (if empty using 100)

        eps (float): maximum distance between two samples in the same cluster.

        fun (string): algorithm for producing clusters (default KMeans)

    Output:
        res (dict): output dict with partitions

    Configurable fields:
```

```
    See Also:

    Example:


    References:
        .. [1]
```

**ensembleCreationParallel**(data, nensemble=100, fun='KMeans', kmin=None, kmax=None, dstPath=None)

**ensembleInstance**(data, fun='KMeans', k=1, init='random', max_iter=100, batch_size=100)

**life_time**(Zpy, ns)

```
    life-time criterion for automatic selection of the number of clusters
    [porting from life-time implementation on matlab]

    Input:
        Zpy (array): input data array of shape (number samples x number features).

        ns (int): number of samples.


    Output:
        res (dict): output dict with indexes for each cluster determined.
                    Example: res = {'-1': noise indexes list,
                                    '0': cluster 0 indexes list,
                                    '1': cluster 1 indexes list}

    Configurable fields:
    {"name": "cluster.dbscan", "config": {"min_samples": "10", "eps": "0.95", "metric": "euclidean"}, "inputs": ["data"], "outputs": ["core_sampl

    See Also:

    Example:


    References:
        .. [1]
```

**main**()

```
    ################################################################################
```

**plotClustering**(X, labels, figNumber=2)

```
    plot clustering

    Input:


    Output:
```

**relativeSimilarities**(sim, clusters, n)

```
    sim - similarity matrix
    clusters - clusters
    n - number of samples
```

**runEnsembleCreation**(data, tasks, parameters)

**time**(...)

```
    time() -> floating point number

    Return the current time in seconds since the Epoch.
    Fractions of a second may be present if the system clock provides them.
```

# bpkclone.cluster.templateSelection

index d:\work\productioncode\clones\biometricspykit\bpkclone\cluster\templateselection.py

```
.. module:: templateSelection
   :platform: Unix, Windows
   :synopsis: This module provides various functions to cluster data

.. moduleauthor:: Andre Lourenco, Carlos Carreiras

.. conventions:
    output of type dict {'-1': indexes, '0': indexes, '1': indexes, ...}
```

## Modules

scipy.spatial.distance  numpy

## Functions

**avgDist**(data=None, fdist='Euclidean', norm=None)

**centroids**(data, clusters=None, ntemplates=1)

```
Template Selection: Centroids

Requires that clusters are provided;
if number of templates per cluster is 1, then templates
are just the centroids; if the number of clusters is >1
then k-means is used to determine more templates on each cluster.
```

**highestAveragesAllocator**(cardinals, k, divisor='dHondt', check=False)

**leadCentroids**(data, **kwargs)

```
Perform centroids template selection for each lead independently.

WARNING: Has a dirty hack to be compatible with Francis' dissimilarity code.
```

**mdist**(data, clusters=None, ntemplates=1, distMeasure='Euclidean')

```
Template Selection using MDIST criterion
Proposed in  U. Uludag, A. Ross, A. Jain, Pattern Recognition 37 (2004).
Originally it does not require any clustering labels, computing the templates
based on mean distances. The number of templates supplied by the user,
in this cases we assume ntemplates argument, as the number of templates.
We extended the idea, including cluster label information. We compute,
for each cluster, the mean distance criterion.
We select, at most, ntemplates. With cluster labels, templates are attributed
to clusters using an allocation method (e.g. d'Hondt)
```

**remainderAllocator**(cardinals, k, reverse=True, check=False)

**selector**(method)

```
Selector for the template selection functions and methods.

Input:
    method (str): The desired function or method.

Output:
    fcn (function): The function pointer.

Configurable fields:

See Also:


Example:


References:
    .. [1]
```

# bpkclone.config

index
d:\work\productioncode\clones\biometricspykit\bpkclone\config.py

```
.. module:: config
   :platform: Unix, Windows
   :synopsis: This module provides various configuration parameters to be shared across all modules of BiometricsPyKit.

.. moduleauthor:: Carlos Carreiras
```

## Modules

multiprocessing numpy os pylab

## Functions

**deploy**(folders=None)

**getFilePath**(sequence)

**getPath**(sequence, verify=True)

**getRandomSeed**()

**setRandomSeed**(seed=None)

## Data

**baseFolder** = r'C:\Users\Carlos\BiometricsPyKitRun'
**folder** = r'C:\Users\Carlos\BiometricsPyKitRun'
**font** = {'family': 'Bitstream Vera Sans', 'size': 16, 'weight': 'normal'}
**manager** = None
**nP** = 12
**numberProcesses** = 6
**queueTimeOut** = 2
**random** = <mtrand.RandomState object>
**randomSeed** = None

# bpkclone.datamanager.datamanager

index
d:\work\productioncode\clones\biometricspykit\bpkclone\datamanager\datamanager.py

```
.. module:: datamanager
   :platform: Unix, Windows
   :synopsis: This module provides various functions to load and store data.

.. moduleauthor:: Carlos Carreiras
```

## Modules

| | | | |
|---|---|---|---|
| cPickle | glob | sklearn.externals.joblib | shutil |
| csv | gzip | numpy | warnings |
| errno | h5py | os | zipfile |

## Classes

DBUnit
H5Unit
MPUnit

### class **DBUnit**

```
StorageBIT (online) database manager class.

Input:
    None

Output:
    None

Configurable fields:


See Also:


Notes:


Example:
```

Methods defined here:

**__init__**(self, dbConn, mapper, experiments)

```
Instantiate the class.

Input:
    dbConn (dict): The DB connection parameters.

    mapper (dict): To map user variables to the database representation.

    experiments (string or list of strings): The experiment(s) to load.

Output:
    None

Configurable fields:


See Also:


Notes:


Example:
    dbConn = {'host': '193.136.222.234',
            'dstPath': '/path/to/files',
            'dbName': 'database'}
    mapper = {'raw': 'signals/ECG/hand/raw',
              'filtered': 'signals/ECG/hand/filtered'}
    experiments = ['T1', 'T2', 'test']
    store = DBUnit(dbConn, mapper, experiments)
```

**addExperiment**(self, experiment=None)

```
Add a new experiment to the database.

Input:
    experiment (dict): The new experiment to add. Has to have the key 'name'.

Output:
    experimentName (str): The name of the new experiment.

Configurable fields:


See Also:
```

```
    Notes:


    Example:
        experiment = {'name': 'Test', 'goals': 'To test the addition of experiments.', 'description': 'This is a sample experiment.'}
        experimentName = store.addExperiment(experiment)
```

**addRecord**(self, header)

```
    Add a new record to the database.

    Input:
        header (dict): The new record to add. Has to have 'subject' and 'experiment' keys.

    Output:
        recordId (int): The ID of the new record.

    Configurable fields:


    See Also:


    Notes:


    Example:
        header = {'experiment': 'test', 'subject': 1, 'date': 'yyyy-mm-ddThh:mm:ss'}
        store.addRecord(header)
```

**addSubject**(self, subject=None)

```
    Add a new subject to the database.

    Input:
        experiment (dict): The new subject to add. Has to have the key 'name'.

    Output:
        subjectId (int): The ID of the new subject.

    Configurable fields:


    See Also:


    Notes:


    Example:
        subject = {'name': 'John Smith', 'birthdate': 'yyyy-mm-ddThh:mm:ss', 'email': 'john.smith@domain.com'}
        subjectId = store.addExperiment(subject)
```

**close**(self)

```
    Close the manager.

    Input:
        None

    Output:
        None

    Configurable fields:


    See Also:


    Notes:


    Example:
        store.close()
```

**data2db**(self, data, data_type, updateDuration=False)

```
    Insert data of specified types in the database.

    Input:
        data (dict): Input data where keys correspond to record IDs.
                     There is also one key 'info' with specific information about the data.

        data_type (list of strings): Input data types to be added.

        updateDuration (bool): Flag to update the duration of the record. Default: False.

    Output:
        None

    Configurable fields:


    See Also:


    Notes:
```

```
    Example:
        data = {'info': {'smapleRate': 1000.}, 0: {'raw': [0, 1, 2], 'filtered': [2, 3, 4]}, 1: {'raw': [0, 1, 2], 'filtered': [2, 3, 4]}}
        store.data2db(data, ['raw', 'filtered'])
```

**db2data**(self, data_type, refine=None, tasks=False, mdataSplit=False, concatenate=False, axis=0)

```
    Retrieves data of the specified type from database. Refine rules may be applied.

    Input:
        data_type (string): The data type to retrieve.

        refine (dict): Refine retrieved information.

        tasks (bool): If True, also return task indices for each data element.

        mdataSplit (bool): Flag to separate the metadata from the signals or events.

        concatenate (bool): If True, concatenate the existing datasets into a single array, along axis.

        axis (int): Axis along which to concatenate the datasets.

    Output:
        output (dict): Output data of data_type where keys correspond to record id numbers.

    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.db2data('raw')
        output is {0: [{'signal': [0, 1, 2], 'mdata': {'sampleRate': 1000.}}, ...], 1: ...}
        output = store.db2data('raw', refine={'_id': [0]})
        output is {0: [{'signal': [0, 1, 2], 'mdata': {'sampleRate': 1000.}}, ...]}
        output = store.db2data('R', refine={'_id': [0]})
        output is {0: [{'timeStamps': [0, 1, 2], 'mdata': {'sampleRate': 1000.}}, ...]}
```

**dbmetada**(self, refine=None)

```
    Get metadata of records from database. Refine rules may be applied.

    Input:
        refine (dict): Refine rules

    Output:
        output (dict): Output metadata where keys correspond to record id numbers.

    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.dbmetada()
        output is {0: record 0 metadata, 1: record 1 metadata, ...}
        output = store.dbmetada(refine={'_id': [0]})
        output is {0: record 0 metadata}
```

**getDataSet**(self, tags=None)

```
    Returns the IDs of the records that have the given tags (AND operator).

    Input:
        tags (list): List of tags.

    Output:
        output (list): Output list with the record IDs.

    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.getDataSet(['Tag 1', 'Tag 2', ...])
        output is e.g. [0, 1, 2, ...]
```

**getSubjectInfo**(self, refine=None)

```
    Get metadata of subjects from database. Refine rules may be applied.

    Input:
        refine (dict): Refine rules

    Output:
        output (dict): Output metadata where keys correspond to subject id numbers.
```

```
    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.getSubjectInfo()
        output is {0: subject 0 metadata, 1: subject 1 metadata, ...}
        output = store.getSubjectInfo(refine={'_id': [0]})
        output is {0: subject 0 metadata}
```

**getTTSets**(self, data, tasks, trainTags, testTags)

```
    Split the data and tasks dictionaries into train and test sets.

    Input:
        data (dict): Data dictionary.

        tasks (dict): Tasks dictionary.

        trainTags (list): List of tags for the train set.

        testTags (list): List of tags for the test set.

    Output:
        output (dict): Output dictionary with keys:
            trainData (dict): Train data.
            trainTasks (dict): Train tasks.
            testData (dict): Test data.
            testTasks (dict): Test tasks.

    Configurable fields:


    See Also:


    Notes:


    Example:
```

**subjectTTSets**(self, info)

```
    Returns subjects information according to the parameters in info.

    Input:
        info (dict): Input dict with the following keys:
            'train_set' (list): train tags
            'test_set' (list): test tags
            'train_time' (tuple of ints): train record time in seconds
            'test_time' (tuple of ints): test record time in seconds

    Output:
        output (list): Output list where each entry is tuple (subject id, train record id(s), test record id(s)).

    Configurable fields:


    See Also:


    Notes:


    Example:
        info = {'train_set': ['T1'],
                'test_set': ['T2'],
                'train_time': (0, 60),
                'test_time': (0, 60)}
        output = store.subjectTTSets(info)
        output is [(0, [0, 1], [2]), (1, [3], [4, 5]), (2, [6], [7]), ...]
```

class **H5Unit**

```
StorageBIT (offline) database manager class.

Input:
    None

Output:
    None

Configurable fields:


See Also:


Notes:
```

```
Example:
```

Methods defined here:

**__init__**(self, path, mapper, experiments)

```
    Instantiate the class.

    Input:
        path (string): The path to the HDF5 files.

        mapper (dict): To map user variables to the database representation.

        experiments (string or list of strings): The experiment(s) to load.

    Output:
        None

    Configurable fields:


    See Also:


    Notes:


    Example:
        path = '/path/to/files'
        mapper = {'raw': 'signals/ECG/hand/raw',
                  'filtered': 'signals/ECG/hand/filtered'}
        experiments = ['T1', 'T2', 'test']
        store = H5Unit(dbConn, mapper, experiments)
```

**addExperiment**(self, experiment=None)

```
    Add a new experiment to the database.

    Input:
        experiment (dict): The new experiment to add. Has to have the key 'name'.

    Output:
        experimentName (str): The name of the new experiment.

    Configurable fields:


    See Also:


    Notes:


    Example:
        experiment = {'name': 'Test', 'goals': 'To test the addition of experiments.', 'description': 'This is a sample experiment.'}
        experimentName = store.addExperiment(experiment)
```

**addRecord**(self, header)

```
    Add a new record to the database.

    Input:
        header (dict): The new record to add. Has to have 'subject' and 'experiment' keys.

    Output:
        recordId (int): The ID of the new record.

    Configurable fields:


    See Also:


    Notes:


    Example:
        header = {'experiment': 'test', 'subject': 1, 'date': 'yyyy-mm-ddThh:mm:ss'}
        store.addRecord(header)
```

**addSubject**(self, subject=None)

```
    Add a new subject to the database.

    Input:
        experiment (dict): The new subject to add. Has to have the key 'name'.

    Output:
        subjectId (int): The ID of the new subject.

    Configurable fields:


    See Also:


    Notes:
```

```
    Example:
        subject = {'name': 'John Smith', 'birthdate': 'yyyy-mm-ddThh:mm:ss', 'email': 'john.smith@domain.com'}
        subjectId = store.addExperiment(subject)
```

**close**(self)

```
    Close the manager.

    Input:
        None

    Output:
        None

    Configurable fields:


    See Also:


    Notes:


    Example:
        store.close()
```

**data2db**(self, data, data_type, updateDuration=False)

```
    Insert data of specified types in the database.

    Input:
        data (dict): Input data where keys correspond to record IDs.
                     There is also one key 'info' with specific information about the data.

        data_type (list of strings): Input data types to be added.

        updateDuration (bool): Flag to update the duration of the record. Default: False.

    Output:
        None

    Configurable fields:


    See Also:


    Notes:


    Example:
        data = {'info': {'smapleRate': 1000.}, 0: {'raw': [0, 1, 2], 'filtered': [2, 3, 4]}, 1: {'raw': [0, 1, 2], 'filtered': [2, 3, 4]}}
        store.data2db(data, ['raw', 'filtered'])
```

**db2data**(self, data_type, refine=None, tasks=False, mdataSplit=False, concatenate=False, axis=0)

```
    Retrieves data of the specified type from database. Refine rules may be applied.

    Input:
        data_type (string): The data type to retrieve.

        refine (dict): Refine retrieved information.

        tasks (bool): If True, also return task indices for each data element.

        mdataSplit (bool): Flag to separate the metadata from the signals or events.

        concatenate (bool): If True, concatenate the existing datasets into a single array, along axis.

        axis (int): Axis along which to concatenate the datasets.

    Output:
        output (dict): Output data of data_type where keys correspond to record id numbers.

    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.db2data('raw')
        output is {0: [{'signal': [0, 1, 2], 'mdata': {'sampleRate': 1000.}}, ...], 1: ...}
        output = store.db2data('raw', refine={'_id': [0]})
        output is {0: [{'signal': [0, 1, 2], 'mdata': {'sampleRate': 1000.}}, ...]}
        output = store.db2data('R', refine={'_id': [0]})
        output is {0: [{'timeStamps': [0, 1, 2], 'mdata': {'sampleRate': 1000.}}, ...]}
```

**dbmetada**(self, refine=None)

```
    Get metadata of records from database. Refine rules may be applied.

    Input:
        refine (dict): Refine rules

    Output:
        output (dict): Output metadata where keys correspond to record id numbers.
```

```
    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.dbmetada()
        output is {0: record 0 metadata, 1: record 1 metadata, ...}
        output = store.dbmetada(refine={'_id': [0]})
        output is {0: record 0 metadata}
```

**getDataSet**(self, tags=None)

```
    Returns the IDs of the records that have the given tags (AND operator).

    Input:
        tags (list): List of tags.

    Output:
        output (list): Output list with the record IDs.

    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.getDataSet(['Tag 1', 'Tag 2', ...])
        output is e.g. [0, 1, 2, ...]
```

**getSubjectInfo**(self, refine=None)

```
    Get metadata of subjects from database. Refine rules may be applied.

    Input:
        refine (dict): Refine rules

    Output:
        output (dict): Output metadata where keys correspond to subject id numbers.

    Configurable fields:


    See Also:


    Notes:


    Example:
        output = store.getSubjectInfo()
        output is {0: subject 0 metadata, 1: subject 1 metadata, ...}
        output = store.getSubjectInfo(refine={'_id': [0]})
        output is {0: subject 0 metadata}
```

**getTTSets**(self, data, tasks, trainTags, testTags)

```
    Split the data and tasks dictionaries into train and test sets.

    Input:
        data (dict): Data dictionary.

        tasks (dict): Tasks dictionary.

        trainTags (list): List of tags for the train set.

        testTags (list): List of tags for the test set.

    Output:
        output (dict): Output dictionary with keys:
            trainData (dict): Train data.
            trainTasks (dict): Train tasks.
            testData (dict): Test data.
            testTasks (dict): Test tasks.

    Configurable fields:


    See Also:


    Notes:


    Example:
```

**subjectTTSets**(self, info)

```
    Returns subjects information according to the parameters in info.

    Input:
```

```
            info (dict): Input dict with the following keys:
                'train_set' (list): train tags
                'test_set' (list): test tags
                'train_time' (tuple of ints): train record time in seconds
                'test_time' (tuple of ints): test record time in seconds

        Output:
            output (list): Output list where each entry is tuple (subject id, train record id(s), test record id(s)).

        Configurable fields:


        See Also:


        Notes:


        Example:
            info = {'train_set': ['T1'],
                    'test_set': ['T2'],
                    'train_time': (0, 60),
                    'test_time': (0, 60)}
            output = store.subjectTTSets(info)
            output is [(0, [0, 1], [2]), (1, [3], [4, 5]), (2, [6], [7]), ...]
```

class **MPUnit**

Methods defined here:

**__init__**(self, path, mapper)

## Functions

**Store**(config)

```
    Method to obtain the appropriate database manager: StorageBIT (online or offline), MonitorPlux, GZIP.

    Input:
        config (dict): Configuration parameters.

    Output:
        unit (DBUnit, H5Unit, MPUnit or GZUnit): The database manager instance.

    Configurable fields:


    See Also:


    Notes:


    Example:
```

**allocH5**(path)

**copyExpResults**(basePath, expID, dstPath, images=False)

**cpLoad**(path)

**cpStore**(path, data)

**csvLoad**(path)

**csvStore**(path, data)

**deleteFile**(filePath)

**export2csv**(fname, data)

**fileBenchmark**(path, it)

**gzLoad**(path)

**gzStore**(path, data)

**h5Load**(path, label)

**h5Store**(path, label, data)

**listFiles**(path, spec='*')

**skLoad**(path)

**skStore**(path, data, compress=3)

**taskify**(data)

**zipArchiveLoad**(zipPath, dstPath)

**zipArchiveStore**(srcPath, zipPath)

# bpkclone.datamanager.parted

index
d:\work\productioncode\clones\biometricspykit\bpkclone\datamanager\parted.py

```
.. module:: parted
   :platform: Unix, Windows
   :synopsis: This module provides various functions to partition data into train and test sets.

.. moduleauthor:: Carlos Carreiras
```

## Modules

copy numpy

## Functions

**deterministicFraction**(indx, run, fraction=None, order='start')

```
Deterministically select a fraction of the elements for the training set.

Input:
    indx (list, array): Indices to partition.

    run (int): Number of the run.

    fraction (int, float): The fraction to select.

    order (str): Determines whether the training data is selected from the start ('start', the default), or from the end ('end') of the set.

Output:
    use (list, array): The selected indices.

    unuse (list, array): The exluded indices.

Configurable fields:

See Also:


Example:


References:
    .. [1]
```

**equalFraction**(indx, run)

```
Equal sets (no separation).

Input:
    indx (list, array): Indices to partition.

    run (int): Number of the run.

Output:
    use (list, array): The selected indices.

    unuse (list, array): The exluded indices.

Configurable fields:

See Also:


Example:


References:
    .. [1]
```

**leaveKOut**(indx, run, k=1, random=True)

```
Leave k items out of the training set.

Input:
    indx (list, array): Indices to partition.

    run (int): Number of the run.

    k (int): Number of elements to leave out. Default=1

    random (bool): If True, select randomly the items. Default=True

Output:
    use (list, array): The selected indices.

    unuse (list, array): The exluded indices.

Configurable fields:

See Also:


Example:
```

```
    References:
        .. [1]
```

**randomFraction**(indx, run, fraction=None)

```
    Randomly select a fraction of the elements for the training set.

    Input:
        indx (list, array): Indices to partition.

        run (int): Number of the run.

        fraction (int, float): The fraction to select.

    Output:
        use (list, array): The selected indices.

        unuse (list, array): The exluded indices.

    Configurable fields:

    See Also:


    Example:


    References:
        .. [1]
```

**randomSelection**(indx, run, k=None)

```
    Randomly select a k elements for the training set.

    Input:
        indx (list, array): Indices to partition.

        run (int): Number of the run.

        k (int): The number of elements to select.

    Output:
        use (list, array): The selected indices.

        unuse (list, array): The exluded indices.

    Configurable fields:

    See Also:


    Example:


    References:
        .. [1]
```

**selector**(method)

```
    Selector for the parted functions and methods.

    Input:
        method (str): The desired function or method.

    Output:
        fcn (function): The function pointer.

    Configurable fields:

    See Also:


    Example:


    References:
        .. [1]
```

# bpkclone.devices.bitalinoTools

index d:\work\productioncode\clones\biometricspykit\bpkclone\devices\bitalinotools.py

```
.. module:: bitalinoProcess
   :platform: Windows
   :synopsis: BITalino acquisition process.

.. moduleauthor:: Filipe Canento, Carlos Carreiras
```

## Modules

collections numpy time

## Classes

__builtin__.object

- CheckHands
- CheckHands2
- CheckHands3
- CheckHandsNotch
- OnlineFilter
  - OnlineFilterBA
- SSFSegmenter
- SegmentECG
- WSPlotter

multiprocessing.process.Process(__builtin__.object)

- BITalinoProcess

class **BITalinoProcess**(multiprocessing.process.Process)

```
BITalino acquisition process.
```

Method resolution order:

- BITalinoProcess
- multiprocessing.process.Process
- __builtin__.object

---

Methods defined here:

**__init__**(self, outQueue, goFlag, acqFlag, mac=None, channels=[0], step=100, SamplingRate=1000, timeout=5, bitCls=None)

**connect**(self)

**exit**(self)

**on_start**(self)

**on_stop**(self)

**processData**(self, data)

**run**(self)

**setup**(self)

**trigger**(self, data=[0, 0, 0, 0])

---

Class methods defined here:

**instaStart**(cls, *args, **kwargs) from __builtin__.type

---

Methods inherited from multiprocessing.process.Process:

**__repr__**(self)

**is_alive**(self)
```
Return whether process is alive
```

**join**(self, timeout=None)
```
Wait until child process terminates
```

**start**(self)
```
Start child process
```

**terminate**(self)
```
Terminate process; sends SIGTERM signal or uses TerminateProcess()
```

---

Data descriptors inherited from multiprocessing.process.Process:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

**authkey**

**daemon**
```
Return whether process is a daemon
```

**exitcode**
```
Return exit code of process or `None` if it has yet to stop
```

**ident**
```
Return identifier (PID) of process or `None` if it has yet to start
```

**name**

**pid**
```
Return identifier (PID) of process or `None` if it has yet to start
```

class **CheckHands**(__builtin__.object)
```
Class to check if hands are on the ECG electrodes.
```

Methods defined here:

**__init__**(self, step=150, size=300, pad=256, signalFc=None, highFc=None, SamplingRate=1000.0, nbits=10, tol=24, delay=3)

**check**(self, signal)

**reset**(self)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **CheckHands2**(__builtin__.object)
```
Class to check if hands are on the ECG electrodes.
```

Methods defined here:

**__init__**(self, step=150, size=300, SamplingRate=1000.0, nbits=10, delay=3)

**check**(self, signal)

**reset**(self)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

---

Data and other attributes defined here:

**HF** = [45, 65]

**PAD** = 512

**TH_BP** = 0.3

**TH_BV** = 0.0234375

**TH_HF** = 0.6

**TH_TH** = 0.02

**TH_TL** = 2e-05

class **CheckHands3**(__builtin__.object)
```
Class to check if hands are on the ECG electrodes.
```

Methods defined here:

**__init__**(self, step=150, size=500, SamplingRate=1000.0, nbits=10, delay=3, bsize=5)

**check**(self, signal)

**reset**(self)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

---

Data and other attributes defined here:

**HF** = [1.953125, 3.90625]

**PAD** = 512

**THRH** = 100.0

**THRL** = 1.0

**TH_BV** = 0.0234375

class **CheckHandsNotch**(__builtin__.object)
```
Class to check if hands are on the ECG electrodes.
```

Methods defined here:

**__init__**(self, step=150, size=300, SamplingRate=1000.0, nbits=10, delay=3)

**check**(self, signal)

**reset**(self)

**test**(self, signal)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

---

Data and other attributes defined here:

**HF** = [45, 65]

**PAD** = 512

**TH_BP** = 0.15

**TH_BV** = 0.0234375

**TH_TH** = 0.015529624809234088

**TH_TL** = 0.00015529624809234089

class **OnlineFilter**(__builtin__.object)
Online Filtering class.

Methods defined here:

**__init__**(self, truncate=False, *args, **kwargs)

**filter**(self, signal)

**reset**(self)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **OnlineFilterBA**(OnlineFilter)
OnlineFilter with explicitly given coefficients.

Method resolution order:
OnlineFilterBA
OnlineFilter
__builtin__.object

---

Methods defined here:

**__init__**(self, b, a, truncate=False)

---

Methods inherited from OnlineFilter:

**filter**(self, signal)

**reset**(self)

---

Data descriptors inherited from OnlineFilter:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **SSFSegmenter**(__builtin__.object)
Online ECG segmentation using the SSF algorithm.

Methods defined here:

**__init__**(self, SamplingRate=1000.0, size=1.5, threshold=20, before=0.03, after=0.01)

**reset**(self)

**resetBuffer**(self)

**segment**(self, signal)

---

Static methods defined here:

**concatenateSegments**(*args)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **SegmentECG**(__builtin__.object)
Online ECG segmentation.

Methods defined here:

**__init__**(self, size=1500, SamplingRate=1000.0)

**reset**(self)

**resetBuffer**(self)

**segment**(self, signal)

---

Static methods defined here:

**concatenateSegments**(*args)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **WSPlotter**(__builtin__.object)

```
Prepare the signal plot for communication to the webpage via websockets.
```

Methods defined here:

**__init__**(self, screenSize=1280, decimation=1, scale=None)

**prepare**(self, signal)

**reset**(self)

---

Data descriptors defined here:

**__dict__**

```
dictionary for instance variables (if defined)
```

**__weakref__**

```
list of weak references to the object (if defined)
```

# bpkclone.dimreduction.dimreduction

index
d:\work\productioncode\clones\biometricspykit\bpkclone\dimreduction\dimreduction.py

```
.. module:: dimreduction
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Filipe Canento
```

## Modules

| | | |
|---|---|---|
| scipy.linalg | pylab | sklearn.decomposition |
| numpy | scipy | traceback |

## Classes

__builtin__.object

- lda
- pca_Ind
- pca_OPop

class **lda**(__builtin__.object)

```
# Linear Discriminant Analysis
```

Methods defined here:

**__init__**(self)

```
# Init
```

**project**(self, data=None)

```
Transforms input data. Train method must have been performed previously.

Input:
    data (array): input data array of shape (number samples x number features).

Output:
    transformed_data (matrix): output data array.

Configurable fields:{"name": "dimreduction.lda.train", "config": {"": ""}, "inputs": ["data"], "outputs": ["transformed_data"]}

See Also:


Notes:


Example:


References:
    .. [1]    ...
    .. [2]    ...
    .. [3]    ...
```

**train**(self, data=None, label=None, energy=1)

```
Perform Linear Discriminant Analysis on input data.

Input:
    data (array): input data array of shape (number samples x number features).

    label (array): input data label array of shape (number of samples x 1). Must start in 0 (e.g., label = [0,0,0,1,2] for 5 samples).

Output:
    success (boolean): indicates whether training was successful (True) or not (False).

Configurable fields:{"name": "dimreduction.lda.train", "config": {"": ""}, "inputs": ["data", "label"], "outputs": ["success"]}

See Also:


Notes:


Example:


References:
    .. [1]    ...
    .. [2]    ...
    .. [3]    ...
```

---

Data descriptors defined here:

**__dict__**

dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **pca_lnd**(__builtin__.object)

Methods defined here:

**__init__**(self)
# Init

**project**(self, data=None)
Input:
data : test data, a 2 dimentional list or array [heartbeat, values] with the data of ONE individual
eigen_vectors : eigenvectors' matrix coming from data training
train_mean : mean of the training data
Output:
Projections : projection coeficients of each observation of the test data projected in each individual of the train set
*projections[testID][trainID] = set of projections of the individual testID in the individual trainID

**train**(self, data=None, label=None, energy=1)
Input:
data : 3 dimentional list or array [individual, heartbeat, values]
energy : value between 0 and 1 for dimentionality reduction (fraction)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **pca_OPop**(__builtin__.object)
# Principal Component Analysis - Overall Population Eigen-Heartbeat

Methods defined here:

**__init__**(self)
# Init

**project**(self, data=None)
Input:
data : test data, a 2 dimentional list or array [heartbeat, values] with the segments of one individue
eigen_vectors : eigenvectors' matrix coming from data training
train_mean : mean of the training data
Output:
Projections : projection coeficients of the individual's observation in the test data

**train**(self, data=None, label=None, energy=1)
Input:
data : 3 dimentional list or array [individual, heartbeat, values]
energy : value between 0 and 1 for dimentionality reduction #### nao sei se e para fazer isto aqui

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

## Functions

**lda_project**(data=None, transform_matrix=None)
Transforms input data. Train method must have been performed previously.

Input:
data (array): input data array of shape (number samples x number features).

Output:
transformed_data (matrix): output data array.

Configurable fields:{"name": "dimreduction.lda.train", "config": {"": ""}, "inputs": ["data"], "outputs": ["transformed_data"]}

See Also:

Notes:

Example:

```
    References:
        .. [1]    ...
        .. [2]    ...
        .. [3]    ...
```

**lda_train**(data=None, label=None)

**lda_train2**(data=None, label=None)

```
    Perform Linear Discriminant Analysis on input data.

    Input:
        data (array): input data array of shape (number samples x number features).

        label (array): input data label array of shape (number of samples x 1). Must start in 0 (e.g., label = [0,0,0,1,2] for 5 samples).

    Output:
        success (boolean): indicates whether training was successful (True) or not (False).

    Configurable fields:{"name": "dimreduction.lda.train", "config": {"": ""}, "inputs": ["data", "label"], "outputs": ["success"]}

    See Also:


    Notes:


    Example:


    References:
        .. [1]    ...
        .. [2]    ...
        .. [3]    ...
```

**selector**(method)

```
    Selector for the dimensionality reduction functions and methods.

    Input:
        method (str): The desired function or method.

    Output:
        fcn (function): The function pointer.

    Configurable fields:

    See Also:


    Example:


    References:
        .. [1]
```

# bpkclone.evaluation.evaluation

index d:\work\productioncode\clones\biometricspykit\bpkclone\evaluation\evaluation.py

```
.. module:: evaluation
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Filipe Canento, Carlos Carreiras
```

## Modules

copy
numpy
scipy
scipy.interpolate
scipy.optimize

## Functions

**assessClassification**(results, rejection_thresholds, ignSubjects=[], dstPath=None, log2file=False)

**assessRuns**(runResults, subjects)

**combineClassifiers**(clf, *p)

**combineSubjectClass**(data, label2subject, subject2label)

**computeAuthRates**(TP, FP, TN, FN, thresholds)

**computeIdRates**(H, M, R, N, thresholds)

**distanceStats**(data, distanceFcn)

**findEqual**(x1, y1, x2, y2, alpha=1.5, xtol=1e-06, ytol=1e-06)

**interDistances**(data, data2=None, distanceFcn=None)

**intraDistances**(data, distanceFcn=None)

**partialDistStats**(pdata)

**pdiff**(x, p1, p2)

**results2report**(title, info, image_path, outfile='report.pdf')

**subjectResults**(results, subject, rejection_thresholds, subjects, subjectDict, subjectIdx)

## Data

**A4** = (595.275590551181, 841.8897637795275)
**inch** = 72.0

# bpkclone.featureextraction.featureextraction d:\work\productioncode\clones\biometricspykit\bpkclone\featureextraction\featureextra

```
.. module:: featureextraction
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Filipe Canento
```

## Modules

h5py
math
numpy
pylab
scipy
traceback

## Classes

__builtin__.object

odinaka

class **odinaka**(__builtin__.object)

Methods defined here:

**__init__**(self, fs, framesz, hop, sinal_length)

**classify**(self, data=None)

**train**(self, data=None, data_idx=None, label=None)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

## Functions

**all_indices**(value, qlist)

**getWindow**(name, n)

**odinakaSTFT**(data, n, overlap, nfft, window, mask=None, scale=False)

**selector**(method)
```
Selector for the feature extraction functions and methods.

Input:
    method (str): The desired function or method.

Output:
    fcn (function): The function pointer.

Configurable fields:

See Also:


Example:


References:
    .. [1]
```

**shortTimeFT**(signal, n, overlap, nfft, window, scale=False)

**stft**(x, fs, framesz, hop, nfft, ftype)

**stft0**(x, fs, framesz, hop)

**stft1**(x, fs, framesz, hop, nfft)

**subpattern**(patterns, cols, lines='all')

# bpkclone.misc.misc

index
d:\work\productioncode\clones\biometricspykit\bpkclone\misc\misc.py

```
.. module:: misc
   :platform: Unix, Windows
   :synopsis: This module provides miscellaneous tools.

.. moduleauthor:: Filipe Canento
```

## Modules

| | | | |
|---|---|---|---|
| cPickle | itertools | numpy | time |
| ctypes | scipy.linalg | pylab | |
| gzip | logging | scipy | |

## Classes

__builtin__.object

- BDIterator
- FiniteCycle
- InfiniteBiterator

_ctypes.Structure(_ctypes._CData)

- HardwareInput
- Input
- KeyBdInput
- MouseInput

_ctypes.Union(_ctypes._CData)

- Input_I

class **BDIterator**(__builtin__.object)

Methods defined here:

**__init__**(self, collection, prev=0)

**__iter__**(self)

**item**(self)

**next**(self)

**prev**(self)

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

class **FiniteCycle**(__builtin__.object)

```
Cycles the given finite iterable indefinitely.
Subclasses ``itertools.cycle`` and adds pickle support.
```

Methods defined here:

**__init__**(self, finite_iterable)

**__reduce__**(self)

**__setstate__**(self, state)

**next**(self)

**peek**(self)
    Return the next value in the cycle without moving the iterable forward.

---

Data descriptors defined here:

**__dict__**
    dictionary for instance variables (if defined)

**__weakref__**
    list of weak references to the object (if defined)

**index**

class **HardwareInput**(_ctypes.Structure)

Method resolution order:
    HardwareInput
    _ctypes.Structure
    _ctypes._CData
    __builtin__.object

---

Data descriptors defined here:

**__dict__**
    dictionary for instance variables (if defined)

**__weakref__**
    list of weak references to the object (if defined)

**uMsg**
    Structure/Union member

**wParamH**
    Structure/Union member

**wParamL**
    Structure/Union member

---

Methods inherited from _ctypes.Structure:

**__init__**(...)
    x.__init__(...) initializes x; see help(type(x)) for signature

---

Data and other attributes inherited from _ctypes.Structure:

**__new__** = <built-in method __new__ of _ctypes.PyCStructType object>
    T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from _ctypes._CData:

**__ctypes_from_outparam__**(...)

**__hash__**(...)

x.__hash__() <==> hash(x)

**__reduce__**(...)

**__setstate__**(...)

class **InfiniteBiterator**(__builtin__.object)

This infinite iterator swings both ways.

Methods defined here:

**__init__**(self, collection)
The iterator infinitely loops over the given collection, either forwards or backwards.

**cur**(self)
Return the current element.

**next**(self)
Get the next element from the collection.

**prev**(self)
Get the previous element from the collection.

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**index**
Returns current index.

class **Input**(_ctypes.Structure)

Method resolution order:
Input
_ctypes.Structure
_ctypes._CData
__builtin__.object

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**ii**
Structure/Union member

**type**
Structure/Union member

---

Methods inherited from _ctypes.Structure:

**__init__**(...)
x.__init__(...) initializes x; see help(type(x)) for signature

---

Data and other attributes inherited from _ctypes.Structure:

**__new__** = <built-in method __new__ of _ctypes.PyCStructType object>
```
T.__new__(S, ...) -> a new object with type S, a subtype of T
```

---

Methods inherited from _ctypes._CData:

**__ctypes_from_outparam__**(...)

**__hash__**(...)
```
x.__hash__() <==> hash(x)
```

**__reduce__**(...)

**__setstate__**(...)

## class **Input_I**(_ctypes.Union)

Method resolution order:
- Input_I
- _ctypes.Union
- _ctypes._CData
- __builtin__.object

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

**hi**
```
Structure/Union member
```

**ki**
```
Structure/Union member
```

**mi**
```
Structure/Union member
```

---

Methods inherited from _ctypes.Union:

**__init__**(...)
```
x.__init__(...) initializes x; see help(type(x)) for signature
```

---

Data and other attributes inherited from _ctypes.Union:

**__new__** = <built-in method __new__ of _ctypes.UnionType object>
```
T.__new__(S, ...) -> a new object with type S, a subtype of T
```

---

Methods inherited from _ctypes._CData:

**__ctypes_from_outparam__**(...)

**__hash__**(...)
```
x.__hash__() <==> hash(x)
```

**__reduce__**(...)

**__setstate__**(...)

## class **KeyBdInput**(_ctypes.Structure)

Method resolution order:
KeyBdInput
_ctypes.Structure
_ctypes._CData
__builtin__.object

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**dwExtraInfo**
Structure/Union member

**dwFlags**
Structure/Union member

**time**
Structure/Union member

**wScan**
Structure/Union member

**wVk**
Structure/Union member

---

Methods inherited from _ctypes.Structure:

**__init__**(...)
x.__init__(...) initializes x; see help(type(x)) for signature

---

Data and other attributes inherited from _ctypes.Structure:

**__new__** = <built-in method __new__ of _ctypes.PyCStructType object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from _ctypes._CData:

**__ctypes_from_outparam__**(...)

**__hash__**(...)
x.__hash__() <==> hash(x)

**__reduce__**(...)

**__setstate__**(...)

class **MouseInput**(_ctypes.Structure)

Method resolution order:
MouseInput
_ctypes.Structure
_ctypes._CData
__builtin__.object

---

Data descriptors defined here:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**dwExtraInfo**
Structure/Union member

**dwFlags**
Structure/Union member

**dx**
Structure/Union member

**dy**
Structure/Union member

**mouseData**
Structure/Union member

**time**
Structure/Union member

---

Methods inherited from _ctypes.Structure:

**__init__**(...)
x.__init__(...) initializes x; see help(type(x)) for signature

---

Data and other attributes inherited from _ctypes.Structure:

**__new__** = <built-in method __new__ of _ctypes.PyCStructType object>
T.__new__(S, ...) -> a new object with type S, a subtype of T

---

Methods inherited from _ctypes._CData:

**__ctypes_from_outparam__**(...)

**__hash__**(...)
x.__hash__() <==> hash(x)

**__reduce__**(...)

**__setstate__**(...)

## Functions

**ar**(x, M)

**centeredFFT**(x, fs, oneside=False, ax=None)

**checkMouseMov**(d)

**chunks**(l, n)

**clickMouse**(key)

**cosdistance**(s1, s2)

**cosv**(a, b)

**dtwdistance**(s1, s2)

**getLogger**(name, logPath=None, level='debug')

**get_subgroup_by_tags**(st, tags)

**keyboardPress**(key, duration=0)

**load_information**(filename)

**mean_waves**(segments, nwaves, jump=0)

**median_waves**(data, nwaves, jump=0)

**merge_clusters**(data)

```
    Merge information from different clusters. Ignores cluster -1 (noise).

    Input:

        data (dict): input data.
                    example: data = {1: {'segments': record 1 ECG segments, 'R': record 1 r peaks},
                                     2: {'segments': record 2 ECG segments, 'R': record 2 r peaks},
                                     ...}

        data_type (string): data type to be analyzed

        parameters (dict): filter parameters.
                           Example: parameters = {'method': 'dbscan', ...}

    Output:

        output (dict): output data where keys correspond to record id numbers.
                       example: output = { 1: {-1: record 1 outlier indexes, '0': record 1 cluster 0 indexes},
                                           2: {-1: record 2 outlier indexes, '0': record 2 cluster 0 indexes},
                                           ...}

    Configurable fields:
    {"name": "??.??", "config": {}, "inputs": ["data", "data_type", "parameters"], "outputs": ["output"]}

    See Also:

    Notes:

    Example:
```

**moveMouse**(dx, dy, absolute=True)

**msedistance**(s1, s2)

**plot_data**(data, ax, title)

**plot_mean_curve**(data)

**pressKey**(key)

**quantize**(signal, levels=256)

**random_idx**(low, high, size, exclude=[])

**releaseKey**(key)

**save_information**(info, dformat, filename=None)

**slasherDict**(d, path)

**test_real_time**()

**unclickMouse**(key)

**wavedistance**(testwave, trainwaves, fdistance)

## Data

**SendInput** = <_FuncPtr object>

**WIN_KEYS** = {'*': 106, '+': 187, ',': 188, '-': 189, '.': 190, '/': 111, '0': 48, '1': 49, '2': 50, '3': 51, ...}

# bpkclone.outlier.outlier

index
d:\work\productioncode\clones\biometricspykit\bpkclone\outlier\outlier.py

```
.. module:: outlier detection
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Filipe Canento, Andre Lourenco
```

## Modules

scipy.spatial.distance  os  scipy
numpy  pylab

## Functions

**dmean**(data=None, R_Position=200, alpha=0.5, beta=1.5, metric='euclidean', checkR=True)

```
DMEAN outlier detection heuristic.

Determines which (if any) samples in a block of ECG templates are outliers.
A sample is considered valid if it cumulatively verifies:
    - distance to average template smaller than a (data derived) threshold T;
    - sample minimum greater than a (data derived) threshold M;
    - sample maximum smaller than a (data derived) threshold N;
    - [optional] position of the sample maximum is the same as the given R position.

For a set of {X_1, ..., X_n} n samples,
Y = rac{1}{n} \sum_{i=1}^{n}{X_i}
d_i = dist(X_i, Y)
D_m = rac{1}{n} \sum_{i=1}^{n}{d_i}
D_s = \sqrt{rac{1}{n - 1} \sum_{i=1}^{n}{(d_i - D_m)^2}}
T = D_m + lpha * D_s
M = eta * median({max(X_i), i=1, ..., n})
N = eta * median({min(X_i), i=1, ..., n})

Input:
    data (array): input data (number of samples x number of features)

    R_Position (int): Position of the R peak.

    alpha (float): Parameter for the distance threshold.

    beta (float): Parameter for the maximum and minimum thresholds.

    metric (string): Distance metric to use (euclidean or cosine).

    checkR (bool): If True checks the R peak position.


Output:
    '0' (list): Indices of the normal samples.

    '-1' (list): Indices of the outlier samples.

Configurable fields:{}

References:
    .. [1]
```

**dmean_old**(data=None, R_Position=200, alpha=0.5, metric='euclidean')

**dmean_tst**(data=None, R_Position=200, alpha=0.5, beta=1.5, metric='euclidean', checkR=True, absolute=False)

**eleazar_eskin**(data=None, metric='euclidean', threshold=0.99)

```
A Geometric Framework for Unsupervised Anomaly Detection:
Detecting based on Fixed widht-clustering (Alg1)
intrusions in Unlabeled Data Eleazar Eskin et Al....

Input:
    data (array): input data (number of observations x number of features).

    metric (string): distance metric to be used.

    threshold (float): threshold between 0.0 and 1.0.

Output:
    res (dict): output dict with keys "normal" and "outlier" and the corresponding data indexes.

Configurable fields:{"name": "outlier.eleazar", "config": {"metric": "euclidean", "threshold": 0.99}, "inputs": ["data"]}
```

```
    See Also:


    Notes:


    Example:


    References:
        .. [1] Eleazar Eskin et Al ...
```

**fiducialMedianFilter**(data=None, fil=0.4, NS=20)

```
    Detect outliers using features and medians

    Input:
        data (array): input data (number of observations x number of features).

        fil (float): percentage of median

        NS (int): number of points

    Output:
        res (dict): output dict with keys "0":normal and "-1":outliers and the corresponding data indexes.

    Configurable fields:{"name": "outlier.fiducialMedianFilter", "config": {"fil": 0.4, "NS": 20}, "inputs": ["data"]}

    References:
        .. [1] Marta Santos et Al, "EIGEN HEARTBEATS FOR USER IDENTIFICATION"
```

**meanfilter**(data=None, metric='euclidean', th=0.8)

```
    Detect outliers using distances over the mean

    Input:
        data (array): input data (number of observations x number of features).

        metric : type of metric

        th (float): percentage to be removed


    Output:
        res (dict): output dict with keys "0":normal and "-1":outliers and the corresponding data indexes.

    Configurable fields:{"name": "outlier.modefilter", "config": {"fil": 0.4, "NS": 20}, "inputs": ["data"]}

    References:
        .. [1]
```

**outliers_dbscan**(data=None, min_samples=10, eps=0.95, metric='euclidean')

```
    Detect outliers using DBSCAN

    Input:
        data (array): input data (number of observations x number of features).

        min_samples (int): minimum number of samples in a cluster.

        eps (float): maximum distance between two samples in the same cluster.

        metric (string): distance metric


    Output:
        res (dict): output dict with keys "0":normal and "-1":outliers and the corresponding data indexes.

    Configurable fields:
    {"name": "outlier.outliers_dbscan", "config": {"metric": "euclidean", "eps": 0.99, "min_samples": 10}, "inputs": ["data"]}

    See Also:
        BiometricsPyKit.cluster.dbscan

    Notes:


    Example:
```

**plotECG**(data, res, figProps)

**removeOutliers**(data=None, outliers=None)

```
    Method to remove the outliers from a given data set.

    Input:
        data (array): The initial data set.
```

```
        outliers (dict): The dictionary with the oulier labels.

    Output:
        templates (array): The data set without the outliers.

    Configurable fields:

    See Also:


    Example:


    References:
        .. [1]
```

**run**(data, tasks, parameters, outPath)

```
    Run the outlier detection methods for each item in data, according to the given parameters.

    Input:
        data

        tasks

        parameters

        outPath

    Output:
        output

    Configurable fields:{"name": "??.??", "config": {}, "inputs": [], "outputs": []}

    See Also:

    Notes:

    Example:
```

**runMethod**(data=None, method=None, **kwargs)

```
    Runs an outlier detection and removal method.

    Input:
        data (array): The initial data set.

        method (str): The desired function or method.

        Additional kwargs are passed to the outlier detection method.

    Output:
        templates (array): The data set without the outliers.

        partition (dict): The partition dictionary between normal and outlier templates.

        nbGood (int): Number of normal templates

    Configurable fields:

    See Also:


    Example:


    References:
        .. [1]
```

**selector**(method)

```
    Selector for the outlier detection functions and methods.

    Input:
        method (str): The desired function or method.

    Output:
        fcn (function): The function pointer.

    Configurable fields:

    See Also:


    Example:


    References:
```

```
.. [1]
```

# bpkclone.preprocessing.preprocessing

index
d:\work\productioncode\clones\biometricspykit\bpkclone\preprocessing\preprocessing.py

```
.. module:: preprocessing
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Filipe Canento, Carlos Carreiras
```

## Modules

numpy  scipy  scipy.signal  scipy.stats

## Functions

**PLA**(signal=None, step=1, epsilon=0.1)

**butter**(signal=None, low_cutoff=None, high_cutoff=None, zi=None, order=4, sampling_rate=1000.0)

```
Butterworth filtering.

Input:
    signal (array): input signal.

    'order' (int): filter order.

    'low_cutoff' (float): low cutoff frequency.

    'high_cutoff' (float): high cutoff frequency.

    'sampling rate' (float): sampling rate.

    'zi' (array): initial conditions

Output:
    output_signal (array): output FIR-filtered signal. Output signal length = length(signal) - filter order (n)

Configurable fields:
{"name": "preprocessing.butter", "config": {"order": 4, "sampling_rate": 1000., "zi": None}, "inputs": ["signal", "low_cutoff", "high_cutoff"

See Also:

Notes:


Example:


References:
    .. [1]
```

**butterFilt**(signal=None, low_cutoff=None, high_cutoff=None, order=1, sampling_rate=1000.0)

```
Butterworth IIR filtering. Uses filtfilt to avoid phase distortions.

Input:
    signal (array): input signal.

    low_cutoff (float): low cutoff frequency.

    high_cutoff (float): high cutoff frequency.

    order (int): filter order.

    sampling_rate (float): sampling rate.

Output:
    output_signal (array): output filtered signal.

Configurable fields:
{"name": "preprocessing.butterFilt", "config": {"order" : 300, "sampling_rate": 1000.}, "inputs": ["signal", "low_cutoff", "high_cutoff"], "c

See Also:
    scipy.signal.filtfilt
    scipy.signal.butter

Notes:


Example:


References:
    .. [1]
```

**butterNotch**(signal=None, center=None, width=None, order=1, sampling_rate=1000.0, harmonics=False)

```
Butterworth IIR notch filtering. Uses filtfilt to avoid phase distortions.

Input:
    signal (array): input signal.

    low_cutoff (float): low cutoff frequency.
```

```
        high_cutoff (float): high cutoff frequency.

        order (int): filter order.

        sampling_rate (float): sampling rate.

        harmonics (bool): Flag to also remove harmonics; the default is False.

    Output:
        output_signal (array): output FIR-filtered signal.

    Configurable fields:
    {"name": "preprocessing.firfilt", "config": {"order" : 300, "sampling_rate": 1000.}, "inputs": ["signal", "low_cutoff", "high_cutoff"], "outp

    See Also:
        scipy.signal.filtfilt
        scipy.signal.butter

    Notes:


    Example:


    References:
        .. [1]
```

**do_filter_work**(work_queue, output)

**filter_data**(data, parameters)

```
    Filters input data according to specified parameters.

    Input:

        data (dict): input data.
                    example: data = {1: record 1 signal, 2: record 2 signal, ...}

        parameters (dict): filter parameters.
                           Example: parameters = {'filter': 'butter',
                                                  'order': 4,
                                                  'low_cutoff': 100.,
                                                  'sampling_rate': 1000.}

    Output:

        output (dict): output filtered data where keys correspond to record id numbers.
                       example: output = {1: record 1 filtered signal, 2: record 2 filtered signal}

    Configurable fields:{"name": "??.??", "config": {}, "inputs": ["data", "parameters"], "outputs": ["output"]}

    See Also:

    Notes:

    Example:
```

**firfilt**(signal=None, low_cutoff=None, high_cutoff=None, order=300, sampling_rate=1000.0)

```
    FIR Filtering. Uses firwin and lfilter from scipy.signal.

    Input:
        signal (array): input signal.

        low_cutoff (float): low cutoff frequency.

        high_cutoff (float): high cutoff frequency.

        order (int): filter order.

        sampling_rate (float): sampling rate.

    Output:
        output_signal (array): output FIR-filtered signal. Output signal length = length(signal) - filter order (n)

    Configurable fields:
    {"name": "preprocessing.firfilt", "config": {"order" : 300, "sampling_rate": 1000.}, "inputs": ["signal", "low_cutoff", "high_cutoff"], "outp

    See Also:
        scipy.signal.firwin
        scipy.signal.lfilter

    Notes:


    Example:


    References:
        .. [1]
```

**heartRateExtractor**(data, sift, selection=None)

**heartRateSifter**(signal, R, Fs=1000.0, edges=None)

**medFIR**(signal, sampleRate=1000.0)

**norm1**(data)

**norm2**(data)

**norm3**(data)

**norm4**(data, R=200)

**norm5**(data)

**norm6**(data)

**norm7**(data)

**norm8**(data, R=200)

**norm9**(data, R=200, sigma=1.5)

**normArcLength**(data)

**normSelector**(method)

**normalWindow**(nb, mu, sigma)

**normalization**(data, ntype)

**notchFilter**(signal=None, center=None, width=None, order=300, sampling_rate=1000.0, harmonics=False)

```
FIR notch Filtering. Uses firwin and lfilter from scipy.signal.

Input:
    signal (array): input signal.

    low_cutoff (float): low cutoff frequency.

    high_cutoff (float): high cutoff frequency.

    order (int): filter order.

    sampling_rate (float): sampling rate.

    harmonics (bool): Flag to also remove harmonics; the default is False.

Output:
    output_signal (array): output FIR-filtered signal. Output signal length = length(signal) - filter order (n)

Configurable fields:
{"name": "preprocessing.firfilt", "config": {"order" : 300, "sampling_rate": 1000.}, "inputs": ["signal", "low_cutoff", "high_cutoff"], "outp

See Also:
    scipy.signal.firwin
    scipy.signal.lfilter

Notes:


Example:


References:
    .. [1]
```

**patterns2strings**(patterns)

**sparseNorm**(data)

## Data

**NUMBER_PROCESSES** = 5

# bpkclone.visualization.visualization

index

d:\work\productioncode\clones\biometricspykit\bpkclone\visualization\visualization.py

```
.. module:: visualization
   :platform: Unix, Windows
   :synopsis: This module provides various functions to...

.. moduleauthor:: Carlos Carreiras, Andre Lourenco
```

## Modules

matplotlib.cm
csv
matplotlib.pyplot
scipy
copy
numpy
pylab

## Functions

**EERCurves**(results, idx, figsize=None)

**FARFRRCurves**(ths, results, figsize=None)

**addEERCurve**(ax, thresholds, rates, variables, lw=1, colors=None, alpha=1, drawEER=False, labels=False, scale=1.0)

**addFARFRR**(ax, thresholds, rates, lw=1, colors=None, alpha=1, drawEER=False, labels=False)

**buildCSVTable**(filePath, resultsCollection)

**buildLatexTable**(resultsCollection, keyOrder)

**clfConfusionMatrix**(subjects, rejection_thresholds, clfResults, assessResults)

**close**(fig=None)

**clusterOverTime**(clusterlabels, linelabels=None, xlabel=None, ylabel=None, ncols=20, figureNumber=1, figsize=None)

**confusionMatrix**(confMatrix)

**figure**(*args, **kwargs)

**multiBarPLot**(data, xticklabels, yerr=None, labels=None, xlabel=None, ylabel=None, xlim=None, ylim=None, width=0.15, loc='best', legendAnchor=None, figsize=None, rotation=30, xtickssize=12, vlines=False, btext=False, btextsize=12, xgap=3.0)

**multiBarPLotAxis**(ax, data, xticklabels, yerr=None, labels=None, xlabel=None, ylabel=None, xlim=None, ylim=None, width=0.15, loc='best', legendAnchor=None, rotation=30, xtickssize=12, vlines=False, btext=False, btextsize=12, xgap=3.0)

**multiBarPLotAxisText**(ax, data, ypos=0, textalpha=4.5)

**nestedMultiBarPLot**(data, xticklabels1, xticklabels2, yerr=None, labels=None, xlabel1=None, xlabel2=None, ylabel=None, xlim=None, ylim=None, width=0.15, loc='best', legendAnchor=(0.75, 1.115), figsize=None, rotation=30)

**plotClusters**(data, clusters, figsize=None, alpha=1)

```
    Plot a clustering partition.

    Kwargs:
        data (array): The (2D) feature array; each line corresponds to a sample.

        clusters (dict): The data partition.

        figsize (tuple): The size of the figure to create.

        alpha (float): The transparency of the plot lines.

    Kwrvals:
        fig (matplotlib.figure.Figure): The figure object.

    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**plotMeanWaves**(data, labels=None, figureNumber=1)

**plotOutliers**(data, partition, alpha=0.7, figsize=None)

**plotSegments**(data, labels, linewidth=2, legenda=None, figureNumber=1, figsize=None)

**plotTTTemplates**(x, trainData, testData, alpha=0.7, linewidth=1, xlabel=None, ylabel=None, title=None, loc='best', figsize=None)

**rocCurves**(results, outfile=None, figNum=None, label=None)

**selector**(method)

**show**()

**singleFARFRRCurve**(thresholds, rates, figsize=None)

**superFARFRRCurves**(results, figsize=None)

# bpkclone.wavelets.wavelets

index
d:\work\productioncode\clones\biometricspykit\bpkclone\wavelets\wavelets.py

```
.. module:: wavelets
   :platform: Unix, Windows
   :synopsis: This module provides various functions to ...

.. moduleauthor:: Carlos Carreiras
```

## Modules

numpy  pylab  pywt  scipy.signal

## Functions

**RDWT**(signal, wavelet, level, matrix=True, **kwargs)

```
    Perform the Redundant Discrete Wavelet Transform (RDWT). This transform is different from the
    Discrete Wavelet Transform (DWT) by skipping the decimation step at each decomposition level.
    One advantage of RDWT over DWT is that it is time-invariant.
    This transform is also known as:
        Stationary wavelet transform,
        Algorithme a trous,
        Quasi-continuous wavelet transform,
        Translation invariant wavelet transform,
        Shift invariant wavelet transform,
        Undecimated wavelet transform.

    Kwargs:
        signal (array): The input signal.

        wavelet (string, pywt.Wavelet): The mother wavelet.

        level (int): The number of decomposition levels.

        matrix (bool): Flag to return the output in matrix form. Default: True.

    Kwrvals:


    See Also:


    Notes:
        The approximation and detail coefficients are arranged by level: [(cAn, cDn), ..., (cA1, cD1)].
        In matrix form the output is [cD1, cD2, ..., cDn, cAn].

    Example:
        signal = ones(1000)
        wavelet = pywt.Wavelet('rbio3.3')
        coefficients, H, G = RDWT(signal, wavelet, 3)

    References:
        .. [1] Fowler, The Redundant Discrete Wavelet Transform and Additive Noise, 2005
```

**RPeaks**(matrix, sampleRate)

```
    Determine locations of R peaks in the wavelet domain.

    Kwargs:
        matrix (array): Set of coefficients.

        sampleRate (float): The sample rate.

    Kwrvals:


    See Also:


    Notes:


    Example:
```

```
    References:
        .. [1] Rooijakkers, Low-complexity R-peak Detection in ECG Signals, 2011
```

**applyThreshold**(coefficients, threshold, hard=False, last=False)

```
    Function to apply a threshold to each of the decomposition levels.

    Kwargs:
        coefficients (array): List of the wavelet decomposition coefficients in matrix form.

        threshold (list): List of length L with the threshold to apply to each level.

        hard (bool): Flag to perform hard thresholding. Default: False.

        last (bool): Flag to also apply the threshold to the last level. Default: False.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**changThr**(coefficients)

```
    Compute the Chang Threshold for each level of the RDWT. Noise is estimated from the first level.

    Kwargs:
        coefficents (array): The coefficients from RDWT in matrix form.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1] Chang, Adaptive Wavelet Thresholding for Image Denoising and Compression, 2000
```

**coeffs2Matrix**(coefficients)

```
    Convert the coefficients list to a matrix format. Note that only the approximation coefficients
    from the last level are kept.

    Kwargs:
        coefficents (list): The coefficients from RDWT.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**conv**(x, h)

```
    Convolution operator. To avoid boundary effects, the input signal is padded (length of the filter) at both ends.

    Kwargs:
        x (array): The input signal.
```

```
        h (list): The FIR filter.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**cutLevel**(coefficients, level)

```
    Set detail coefficients to zero up to (excluding) a given level.

    Kwargs:
        coefficients (array): List of the wavelet decomposition coefficients in matrix form.

        level (int): The level at which to cut.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**extractSegments**(matrix, R, sampleRate)

```
    Extract the wavelet segments.

    Kwargs:
        matrix (array): Set of coefficients.

        R (list): List of indexes locating the R peaks.

        sampleRate (float): The sample rate.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**iRDWT**(coefficients, wavelet, zeros=False)

```
    The inverse Redundant Discrete Wavelet Transform.

    Kwargs:
        coefficients (array): The coefficients as produced by the RDWT function (matrix form).

        wavelet (string, pywt.Wavelet): The mother wavelet.

        zeros (bool): Flag to replace approximation coefficients with zeros.

    Kwrvals:
```

```
    See Also:


    Notes:
        The approximation and detail coefficients are arranged by level: [(cAn, cDn), ..., (cA1, cD1)].

    Example:
        signal = ones(1000)
        wavelet = pywt.Wavelet('rbio3.3')
        coefficients, H, G = RDWT(signal, wavelet, 3)
        back = iRDWT(coefficients, H, G)

    References:
        .. [1]
```

**maxLevel**(signal, wavelet)

```
    Determine the maximum decomposition level that is possible to apply the
    RDWT to a given signal, with the specified mother wavelet.

    Kwargs:
        signal (int, array): The input signal, or the length of the signal to decompose.

        wavelet (string, pywt.Wavelet): The mother wavelet.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1] Fowler, The Redundant Discrete Wavelet Transform and Additive Noise, 2005
```

**mchangThr**(coefficients)

```
    Compute the modified Chang Threshold for each level of the RDWT. Noise is estimated for each level.

    Kwargs:
        coefficents (array): The coefficients from RDWT in matrix form.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1] Chang, Adaptive Wavelet Thresholding for Image Denoising and Compression, 2000
```

**meanWave**(data, axis=2)

**medianWave**(data, axis=2)

**muniversalThr**(coefficients)

```
    Compute the modifief Universal Threshold for each level of the RDWT.

    Kwargs:
        coefficents (array): The coefficients from RDWT in matrix form.

    Kwrvals:


    See Also:


    Notes:
```

```
    Example:


    References:
        .. [1] Donoho, De-Noising by Soft Thresholding, 1995
```

**plotCoeffs**(signal, coefficients, start=None, stop=None, step=None)

```
    Plot the decomposition.

    Kwargs:
        signal (array): The original signal.

        coefficients (array): Set of coefficients in matrix form.

        start (int): Starting time index to plot. Default: None.

        stop (int): Ending time index to plot. Default: None.

        step (int): The time step to plot. Default: None.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**plotOutliers**(data=None, outliersResult=None, offset=0)

```
    Plot the results of the outlier removal procedure.

    Kwargs:
        data (array): The segments of wavelet coefficients.

        outliersResult (dict): Result of the outlier removal procedure;
                               has keys '0' (good) and '-1' (bad).

        offset (int): Offset to correct the display of the levels. Default: 0

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**plotWaveletSpectrum**(wavelet, level, sampleRate, npoints=1024, start=None, stop=None)

```
    Plot the transfer function (amplitude) of the wavelet decomposition filters.

    Kwargs:
        wavelet (string, pywt.Wavelet): The mother wavelet.

        level (int): The number of decomposition levels.

        sampleRate (float): The sample rate.

        npoints (int): Number of points in which to compute the transfer function. Default: 1024.

        start (int): Starting frequency to plot. Default: None.

        stop (int): Ending frequency to plot. Default: None.

    Kwrvals:
```

```
    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**rcosdmean**(data=None, th_min=1.5, th_max=1.5, th_dist=0.5)

```
    Outlier detection based on the mean segment and the medians of maxima and minima (wavelet version).

    Kwargs:
        data (array): The segments of wavelet coefficients.

        th_min (float): Parameter for median of minima. Default: 1.5

        th_max (float): Parameter for median of maxima. Default: 1.5

        th_dist (float): Parameter for distances. Default: 0.5

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**reverse**(x)

```
    Method to reverse a signal.
    y[n] = x[-n]

    Kwargs:
        x (array): The input signal.

    Kwrvals:


    See Also:


    Notes:


    Example:


    References:
        .. [1]
```

**trous**(h)

```
    Add zeros (holes - trous) between the filter coefficients.

    Kwargs:
        h (list): The FIR filter.

    Kwrvals:


    See Also:


    Notes:


    Example:
```

References:
    .. [1]

**universalThr**(coefficients)
Compute the Universal Threshold for each level of the RDWT.

Kwargs:
    coefficents (array): The coefficients from RDWT in matrix form.

Kwrvals:

See Also:

Notes:

Example:

References:
    .. [1] Donoho, De-Noising by Soft Thresholding, 1995

**waveDist**(coefficients1, coefficients2, thr=400.0, levels=None, indexes=None)
Compute distance between a pair of wavelet coefficients (in matrix form).

Kwargs:
    coefficients1 (array): First set of coefficients.

    coefficients2 (array): Second set of coefficients.

    levels (list, slice): Levels to use in the distance (e.g [0, 1, 2]). Default: all levels.

    indexes (list, slice): Time indexes to use in the distance. Default: all.

Kwrvals:

See Also:

Notes:

Example:

References:
    .. [1] Chan, Wavelet disntance measure for person identification using ECG, 2008

**waveletSegments**(matrix, sampleRate, **kwargs)
Determine R peak locations and extract the wavelet segments.

Kwargs:
    matrix (array): Set of coefficients.

    sampleRate (float): The sample rate.

Kwrvals:
    segments (array): The segmented wavelet coefficients (time, levels, segments).

    R (list): List of indexes locating the R peaks.

See Also:

Notes:

Example:

References:
    .. [1]

# bpkclone.vitalidiFramework

index
d:\work\productioncode\clones\biometricspykit\bpkclone\vitalidiframework.py

```
.. module:: vitalidiFramework
   :platform: Unix, Windows
   :synopsis: This module interfaces between the Vitalidi demonstrator Python back-end and BiometricsPyKit.

.. moduleauthor:: Ana Priscila Alves, Carlos Carreiras
```

## Modules

| | | | |
|---|---|---|---|
| copy | glob | numpy | shutil |
| datetime | json | os | traceback |

## Classes

__builtin__.object

BaseBrain

DoorBrain
KeyBrain
MixBrain
MyBrain
QuickBrain

multiprocessing.process.Process(__builtin__.object)

RunEvaluate

class **BaseBrain**(__builtin__.object)

```
Base class for Vitalidi brains.
```

Methods defined here:

**__init__**(self, path, settings, ignorePrevious=False, deployLocations=None)

**authenticate**(self, data, subject, ready=False)

**checkSubject**(self, subject)

**combination**(self, results)

**createClassifiers**(self)

**debugTest**(self, data)

**deploy**(self, deployLocations=None)

**evaluate**(self, recordTypes=None)

**groupAuthenticate**(self, data, names=None, ready=False)

**identify**(self, data, ready=False)

**listClassSubjects**(self)

**listSubjects**(self)

**loadSegments**(self, subject, recordTypes)

**loadSettings**(self, settings)

**prepareData**(self, data)

**removeSubject**(self, subject, updateThresholds=False)

**saveFile**(self, command, name, Rawsignal=None, FilteredSignal=None, Segments=None, currentTime=None, eventValue=None, good_segments=None, res=None, mode='a')

**saveOutliersPlot**(self, segments, partition, currentTime=None)

**subjectPerformance**(self, subject)

**train**(self, data, subject, updateThresholds=True)

**trainFromFiles**(self, recordTypes=None, ignoreSubjects=None)

**trainFromFolder**(self, path=None, ignoreSubjects=None)

**updateFile**(self, name)

**updateThresholds**(self)

**validateAutoAuth**(self, segments)

**validation**(self, nsegments, segments, case, autoAuth=False)

**validationStatus**(self, nsegments, case)

---

Class methods defined here:

**getCurrentTime**(cls) from __builtin__.type

---

Static methods defined here:

**listFolderSubjects**(path)

---

Data descriptors defined here:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

---

Data and other attributes defined here:

**CLF_NAMES** = []

**CLF_WEIGHTS** = {}

class **DoorBrain**(BaseBrain)
```
Brain using an SVM classifier with 5 median waves.
Templates are scaled with a gaussian window.
```

Method resolution order:
DoorBrain
BaseBrain
__builtin__.object

---

Methods defined here:

**prepareData**(self, data)

---

Data and other attributes defined here:

**CLF_NAMES** = ['5-medianWaves']

**CLF_WEIGHTS** = {'5-medianWaves': 1.0}

---

Methods inherited from BaseBrain:

**__init__**(self, path, settings, ignorePrevious=False, deployLocations=None)

**authenticate**(self, data, subject, ready=False)

**checkSubject**(self, subject)

**combination**(self, results)

**createClassifiers**(self)

**debugTest**(self, data)

**deploy**(self, deployLocations=None)

**evaluate**(self, recordTypes=None)

**groupAuthenticate**(self, data, names=None, ready=False)

**identify**(self, data, ready=False)

**listClassSubjects**(self)

**listSubjects**(self)

**loadSegments**(self, subject, recordTypes)

**loadSettings**(self, settings)

**removeSubject**(self, subject, updateThresholds=False)

**saveFile**(self, command, name, Rawsignal=None, FilteredSignal=None, Segments=None, currentTime=None, eventValue=None, good_segments=None, res=None, mode='a')

**saveOutliersPlot**(self, segments, partition, currentTime=None)

**subjectPerformance**(self, subject)

**train**(self, data, subject, updateThresholds=True)

**trainFromFiles**(self, recordTypes=None, ignoreSubjects=None)

**trainFromFolder**(self, path=None, ignoreSubjects=None)

**updateFile**(self, name)

**updateThresholds**(self)

**validateAutoAuth**(self, segments)

**validation**(self, nsegments, segments, case, autoAuth=False)

**validationStatus**(self, nsegments, case)

---

Class methods inherited from BaseBrain:

**getCurrentTime**(cls) from __builtin__.type

---

Static methods inherited from BaseBrain:

**listFolderSubjects**(path)

---

Data descriptors inherited from BaseBrain:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **KeyBrain**(BaseBrain)
```
Brain using SVM classifiers, combining all segments, 3 median waves and 5 median waves.
Templates are scaled with a gaussian window.
```

Method resolution order:
KeyBrain
BaseBrain
__builtin__.object

---

Methods defined here:

**prepareData**(self, data)

---

Data and other attributes defined here:

**CLF_NAMES** = ['segments', '3-medianWaves', '5-medianWaves']

**CLF_WEIGHTS** = {'3-medianWaves': 2.0, '5-medianWaves': 3.0, 'segments': 1.0}

---

Methods inherited from BaseBrain:

**__init__**(self, path, settings, ignorePrevious=False, deployLocations=None)

**authenticate**(self, data, subject, ready=False)

**checkSubject**(self, subject)

**combination**(self, results)

**createClassifiers**(self)

**debugTest**(self, data)

**deploy**(self, deployLocations=None)

**evaluate**(self, recordTypes=None)

**groupAuthenticate**(self, data, names=None, ready=False)

**identify**(self, data, ready=False)

**listClassSubjects**(self)

**listSubjects**(self)

**loadSegments**(self, subject, recordTypes)

**loadSettings**(self, settings)

**removeSubject**(self, subject, updateThresholds=False)

**saveFile**(self, command, name, Rawsignal=None, FilteredSignal=None, Segments=None, currentTime=None, eventValue=None, good_segments=None, res=None, mode='a')

**saveOutliersPlot**(self, segments, partition, currentTime=None)

**subjectPerformance**(self, subject)

**train**(self, data, subject, updateThresholds=True)

**trainFromFiles**(self, recordTypes=None, ignoreSubjects=None)

**trainFromFolder**(self, path=None, ignoreSubjects=None)

**updateFile**(self, name)

**updateThresholds**(self)

**validateAutoAuth**(self, segments)

**validation**(self, nsegments, segments, case, autoAuth=False)

**validationStatus**(self, nsegments, case)

---

Class methods inherited from BaseBrain:

**getCurrentTime**(cls) from __builtin__.type

---

Static methods inherited from BaseBrain:

**listFolderSubjects**(path)

---

Data descriptors inherited from BaseBrain:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **MixBrain**(BaseBrain)
```
Brain
```

Method resolution order:
MixBrain
BaseBrain
__builtin__.object

---

Methods defined here:

**prepareData**(self, data)

---

Data and other attributes defined here:

**CLF_NAMES** = ['segments', '3-medianWaves-norm', '3-medianWaves', '5-medianWaves']

**CLF_WEIGHTS** = {'3-medianWaves': 1.0, '3-medianWaves-norm': 1.0, '5-medianWaves': 2.0, 'segments': 1.0}

---

Methods inherited from BaseBrain:

**__init__**(self, path, settings, ignorePrevious=False, deployLocations=None)

**authenticate**(self, data, subject, ready=False)

**checkSubject**(self, subject)

**combination**(self, results)

**createClassifiers**(self)

**debugTest**(self, data)

**deploy**(self, deployLocations=None)

**evaluate**(self, recordTypes=None)

**groupAuthenticate**(self, data, names=None, ready=False)

**identify**(self, data, ready=False)

**listClassSubjects**(self)

**listSubjects**(self)

**loadSegments**(self, subject, recordTypes)

**loadSettings**(self, settings)

**removeSubject**(self, subject, updateThresholds=False)

**saveFile**(self, command, name, Rawsignal=None, FilteredSignal=None, Segments=None, currentTime=None, eventValue=None, good_segments=None, res=None, mode='a')

**saveOutliersPlot**(self, segments, partition, currentTime=None)

**subjectPerformance**(self, subject)

**train**(self, data, subject, updateThresholds=True)

**trainFromFiles**(self, recordTypes=None, ignoreSubjects=None)

**trainFromFolder**(self, path=None, ignoreSubjects=None)

**updateFile**(self, name)

**updateThresholds**(self)

**validateAutoAuth**(self, segments)

**validation**(self, nsegments, segments, case, autoAuth=False)

**validationStatus**(self, nsegments, case)

---

Class methods inherited from BaseBrain:

**getCurrentTime**(cls) from __builtin__.type

---

Static methods inherited from BaseBrain:

**listFolderSubjects**(path)

---

Data descriptors inherited from BaseBrain:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

## class **MyBrain**(BaseBrain)

Brain using SVM classifiers, combining all segments, 3 mean-waves and 5 mean-waves.

Method resolution order:
MyBrain
BaseBrain
__builtin__.object

---

Methods defined here:

**prepareData**(self, data)

---

Data and other attributes defined here:

**CLF_NAMES** = ['segments', '3-meanWaves', '5-meanWaves']

**CLF_WEIGHTS** = {'3-meanWaves': 1.0, '5-meanWaves': 1.0, 'segments': 1.0}

---

Methods inherited from BaseBrain:

**__init__**(self, path, settings, ignorePrevious=False, deployLocations=None)

**authenticate**(self, data, subject, ready=False)

**checkSubject**(self, subject)

**combination**(self, results)

**createClassifiers**(self)

**debugTest**(self, data)

**deploy**(self, deployLocations=None)

**evaluate**(self, recordTypes=None)

**groupAuthenticate**(self, data, names=None, ready=False)

**identify**(self, data, ready=False)

**listClassSubjects**(self)

**listSubjects**(self)

**loadSegments**(self, subject, recordTypes)

**loadSettings**(self, settings)

**removeSubject**(self, subject, updateThresholds=False)

**saveFile**(self, command, name, Rawsignal=None, FilteredSignal=None, Segments=None, currentTime=None, eventValue=None, good_segments=None, res=None, mode='a')

**saveOutliersPlot**(self, segments, partition, currentTime=None)

**subjectPerformance**(self, subject)

**train**(self, data, subject, updateThresholds=True)

**trainFromFiles**(self, recordTypes=None, ignoreSubjects=None)

**trainFromFolder**(self, path=None, ignoreSubjects=None)

**updateFile**(self, name)

**updateThresholds**(self)

**validateAutoAuth**(self, segments)

**validation**(self, nsegments, segments, case, autoAuth=False)

**validationStatus**(self, nsegments, case)

---

Class methods inherited from BaseBrain:

**getCurrentTime**(cls) from __builtin__.type

---

Static methods inherited from BaseBrain:

**listFolderSubjects**(path)

---

Data descriptors inherited from BaseBrain:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **QuickBrain**(BaseBrain)
```
Brain using SVM classifiers, combining all segments, 3 mean-waves and 5 mean-waves.
```

Method resolution order:
QuickBrain
BaseBrain
__builtin__.object

---

Methods defined here:

**prepareData**(self, data)

---

Data and other attributes defined here:

**CLF_NAMES** = ['5-meanWaves']

**CLF_WEIGHTS** = {'5-meanWaves': 1.0}

---

Methods inherited from BaseBrain:

**__init__**(self, path, settings, ignorePrevious=False, deployLocations=None)

**authenticate**(self, data, subject, ready=False)

**checkSubject**(self, subject)

**combination**(self, results)

**createClassifiers**(self)

**debugTest**(self, data)

**deploy**(self, deployLocations=None)

**evaluate**(self, recordTypes=None)

**groupAuthenticate**(self, data, names=None, ready=False)

**identify**(self, data, ready=False)

**listClassSubjects**(self)

**listSubjects**(self)

**loadSegments**(self, subject, recordTypes)

**loadSettings**(self, settings)

**removeSubject**(self, subject, updateThresholds=False)

**saveFile**(self, command, name, Rawsignal=None, FilteredSignal=None, Segments=None, currentTime=None, eventValue=None, good_segments=None, res=None, mode='a')

**saveOutliersPlot**(self, segments, partition, currentTime=None)

**subjectPerformance**(self, subject)

**train**(self, data, subject, updateThresholds=True)

**trainFromFiles**(self, recordTypes=None, ignoreSubjects=None)

**trainFromFolder**(self, path=None, ignoreSubjects=None)

**updateFile**(self, name)

**updateThresholds**(self)

**validateAutoAuth**(self, segments)

**validation**(self, nsegments, segments, case, autoAuth=False)

**validationStatus**(self, nsegments, case)

---

Class methods inherited from BaseBrain:

**getCurrentTime**(cls) from __builtin__.type

---

Static methods inherited from BaseBrain:

**listFolderSubjects**(path)

---

Data descriptors inherited from BaseBrain:

**__dict__**
```
dictionary for instance variables (if defined)
```

**__weakref__**
```
list of weak references to the object (if defined)
```

class **RunEvaluate**(multiprocessing.process.Process)

Run classifier evaluation

Method resolution order:
RunEvaluate
multiprocessing.process.Process
__builtin__.object

---

Methods defined here:

**__init__**(self, brainCls=None, stateDict=None, *args, **kwargs)

**run**(self)

---

Methods inherited from multiprocessing.process.Process:

**__repr__**(self)

**is_alive**(self)
Return whether process is alive

**join**(self, timeout=None)
Wait until child process terminates

**start**(self)
Start child process

**terminate**(self)
Terminate process; sends SIGTERM signal or uses TerminateProcess()

---

Data descriptors inherited from multiprocessing.process.Process:

**__dict__**
dictionary for instance variables (if defined)

**__weakref__**
list of weak references to the object (if defined)

**authkey**

**daemon**
Return whether process is a daemon

**exitcode**
Return exit code of process or `None` if it has yet to stop

**ident**
Return identifier (PID) of process or `None` if it has yet to start

**name**

**pid**
Return identifier (PID) of process or `None` if it has yet to start

## Data

**ERR_MSG** = {'AUT-0': 'Authentication: Success', 'AUT-1': 'Authentication: Error: UntrainedError', 'AUT-2': 'Authentication: Error: UnknownSubjectError', 'AUT-3': 'Authentication: Error: Combination confidence below threshold', 'AUT-4': 'Authentication: Error: Empty combination', 'ID-0': 'Identification: Success', 'ID-1': 'Identification: Error: UntrainedError', 'ID-2': 'Identification: Error: Default subject', 'ID-3': 'Identification: Error: Combination confidence below threshold', 'ID-4': 'Identification: Error: Empty combination', ...}
**settings** = {'MAC': '', 'autoAuth': {'k': 0, 'threshold': 0}, 'classification': {'clf_0': {'method': '', 'parameters': {}}, 'clf_N': {'method': '', 'parameters': {}}, 'confidence_threshold': 0}, 'outlier': {'enroll': {'method': '', 'parameters': {}}, 'id': {'method':

", 'parameters': {}}}, 'validation': {'enroll': {'minimum_templates': 0, 'time': 0}, 'id': {'minimum_templates': 0, 'time': 0}}}

# bpkclone.classifiers

index
d:\work\productioncode\clones\biometricspykit\bpkclone\classifiers__init__.py

## Package Contents

classifiers
rules

# bpkclone.Cloud

index
d:\work\productioncode\clones\biometricspykit\bpkclone\cloud__init__.py

## Package Contents

| | | | |
|---|---|---|---|
| SerialPortServer | parallel | server | txws_old |
| client | remind | txws | webServer |