adenine Documentation

Release 0.1.4

Samuele Fiorini - Federico Tomasi - Annalisa Barla

Mar 01, 2017

CONTENTS

adenine is a machine learning and data mining Python library for exploratory data analysis.

The main structure of **adenine** can be summarized in the following 4 steps.

1. **Imputing:** Does your dataset have missing entries? In the first step you can fill the missing values choosing between different strategies: feature-wise median, mean and most frequent value or k-NN imputing.
2. **Preprocessing:** Have you ever wondered what would have changed if only your data have been preprocessed in a different way? Or is it data preprocessing a good idea after all? **adenine** includes several preprocessing procedures, such as: data recentering, Min-Max scaling, standardization and normalization. **adenine** also allows you to compare the results of the analysis made with different preprocessing strategies.
3. **Dimensionality Reduction:** In the context of data exploration, this phase becomes particularly helpful for high dimensional data. This step includes manifold learning (such as isomap, multidimensional scaling, etc) and unsupervised feature learning (principal component analysis, kernel PCA, Bernoulli RBM, etc) techniques.
4. **Clustering:** This step aims at grouping data into clusters in an unsupervised manner. Several techniques such as k-means, spectral or hierarchical clustering are offered.

The final output of **adenine** is a compact, textual and graphical representation of the results obtained from the pipelines made with each possible combination of the algorithms selected at each step.

adenine can run on multiple cores/machines* and it is fully *scikit-learn* compliant.

CHAPTER

ONE

USER DOCUMENTATION

1.1 Quick start tutorial

Adenine* can be installed using standard Python tools (with administrative or sudo permissions on GNU-Linux platforms):

```
$ pip install adenine
```

1.1.1 Installation from sources

If you like to manually install **Adenine**, download the .zip or .tar.gz archive from http://slipguru.github.io/adenine/. Then extract it and move into the root directory:

```
$ unzip slipguru-adenine-|release|.zip
$ cd adenine-|release|/
```

or:

```
$ tar xvf slipguru-adenine-|release|.tar.gz
$ cd adenine-|release|/
```

Otherwise you can clone our GitHub repository:

```
$ git clone https://github.com/slipguru/adenine.git
```

From here, you can follow the standard Python installation step:

```
$ python setup.py install
```

After **Adenine** installation, you should have access to two scripts, named with a common `ade_` prefix:

```
$ ade_<TAB>
ade_analysis.py    ade_run.py
```

This tutorial assumes that you downloaded and extracted **Adenine** source package which contains a `examples\data` directory with some data files (`.npy` or `.csv`) which will be used to show **Adenine** functionalities.

Adenine needs only 3 ingredients:

- `n_samples x n_variables` input matrix
- `n_samples x 1` output vector (optional)

- `configuration` file

1.1.2 Input data format

Input data are assumed to be:

- `numpy` array stored in `.npy` files organized with a row for each sample and a column for each feature,
- tabular data stored in comma separated `.csv` files presenting the variables header on the first row and the sample indexes on the first column,
- toy examples available from `adenine.utils.data_source` function.

1.1.3 Configuration File

Adenine configuration file is a standard Python script. It is imported as a module, then all the code is executed. In this file the user can define all the option needed to read the data and to create the pipelines.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""Configuration file for adenine."""

from adenine.utils import data_source

# -------------------------  EXPERMIENT INFO ------------------------- #
exp_tag = '_experiment'
output_root_folder = 'results'
plotting_context = 'notebook'  # one of {paper, notebook, talk, poster}
file_format = 'pdf'  # or 'png'

# ----------------------------  INPUT DATA ---------------------------- #
# Load an example dataset or specify your input data in tabular format
data_file = 'data.csv'
labels_file = 'labels.csv'  # OPTIONAL
samples_on = 'rows'  # if samples lie on columns use 'cols' or 'col'
data_sep = ','  # the data separator. e.g., ',', '\t', ' ', ...
X, y, feat_names, index = data_source.load('custom',
                                           data_file, labels_file,
                                           samples_on=samples_on,
                                           sep=data_sep)

# ----------------------  PIPELINES DEFINITION ----------------------- #
# --- Missing values imputing --- #
step0 = {'Impute': [False, {'missing_values': 'NaN',
                            'strategy': ['median',
                                         'mean',
                                         'nearest_neighbors']}]}

# --- Data preprocessing --- #
step1 = {'None': [False], 'Recenter': [False], 'Standardize': [False],
         'Normalize': [False, {'norm': ['l1', 'l2']}],
         'MinMax': [False, {'feature_range': [(0, 1), (-1, 1)]}]}

# --- Unsupervised features learning --- #
# affinity ca be precumputed for SE
step2 = {'PCA': [False, {'n_components': 3}],
         'IncrementalPCA': [False],
```

```
         'RandomizedPCA': [False],
         'KernelPCA': [False, {'kernel': ['linear', 'rbf', 'poly']}],
         'Isomap': [False, {'n_neighbors': 5}],
         'LLE': [False, {'n_neighbors': 5,
                         'method': ['standard', 'modified',
                                    'hessian', 'ltsa']}],
         'SE': [False, {'affinity': ['nearest_neighbors', 'rbf']}],
         'MDS': [False, {'metric': True}],
         'tSNE': [False],
         'RBM': [False, {'n_components': 256}],
         'None': [False]
         }

# --- Clustering --- #
# affinity ca be precumputed for AP, Spectral and Hierarchical
step3 = {'KMeans': [False, {'n_clusters': [3, 'auto']}],
         'AP': [False, {'preference': ['auto']}],
         'MS': [False],
         'Spectral': [False, {'n_clusters': [3, 8]}],
         'Hierarchical': [False, {'n_clusters': [3, 8],
                                  'affinity': ['manhattan', 'euclidean'],
                                  'linkage':  ['ward', 'complete', 'average']}]
         }
```

1.1.4 Experiment runner

The `ade_run.py` script, executes the full **Adenine** framework. The prototype is the following:

```
$ ade_run.py ade_config.py
```

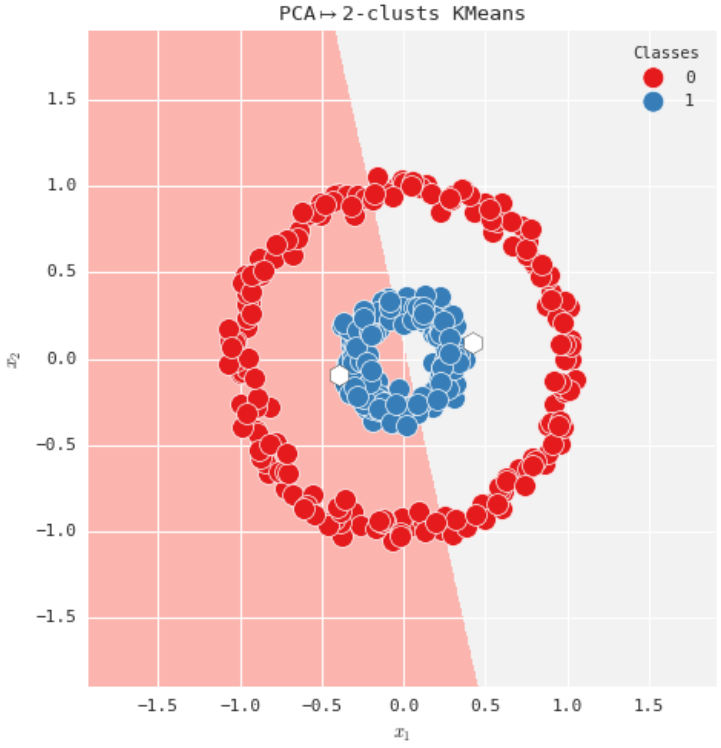
When launched, the script reads the data, then it creates and runs each pipeline saving the results in a tree-like structure which has the current folder as root.

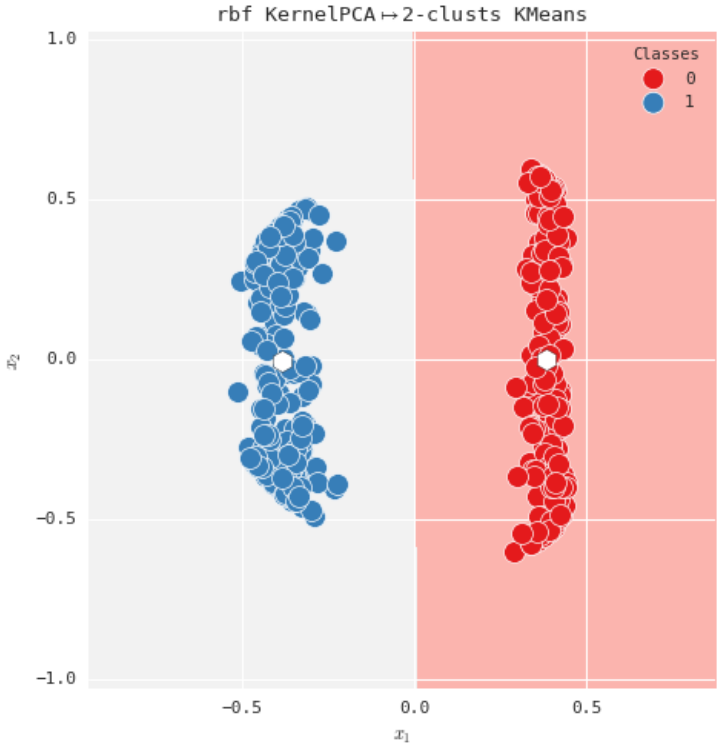
1.1.5 Results analysis

The `ade_analysis.py` script provides useful summaries and graphs from the results of the experiment. This script accepts as only parameter a result directory already created:

```
$ ade_analysis.py result-dir
```

The script produces a set of textual and graphical results. An output example obtained by one of the implemented pipelines is represented below.





You can reproduce the example above specifying `data_source.load('circles')` in the configuration file.

1.1.6 Example dataset

An example dataset can be dowloaded here. The dataset is a random extraction of 801 samples (with dimension 20531) measuring RNA-Seq gene expression of patients affected by 5 different types of tumor: breast invasive carcinoma (BRCA), kidney renal clear cell carcinoma (KIRC), colon (COAD), lung (LUAD) and prostate adenocarcinoma (PRAD). The full dataset is maintained by The Cancer Genome Atlas Pan-Cancer Project [1] and we refer to the original repository for furher details.

1.1.7 Reference

[1] Weinstein, John N., et al. "The cancer genome atlas pan-cancer analysis project." Nature genetics 45.10 (2013): 1113-1120.

1.2 API

1.2.1 Pipeline utilities

adenine.core.define_pipeline.**parse_imputing**(*key*, *content*)

Parse the options of the imputing step.

This function parses the imputing step coded as dictionary in the ade_config file.

Parameters **key** : class or str, like {'Impute', 'None'}

The type of selected imputing step. In case in which key is a *class*, it must contain both a *fit* and *transform* method.

content : dict

A dictionary containing parameters for each imputing class. Each parameter can be a list; in this case for each combination of parameters a different pipeline will be created.

Returns **tpl** : tuple

A tuple made like ('imputing_name', imputing_obj, 'imputing'), where imputing_obj is an sklearn 'transforms' (i.e. it has bot a fit and transform method).

adenine.core.define_pipeline.**parse_preproc**(*key*, *content*)

Parse the options of the preprocessing step.

This function parses the preprocessing step coded as dictionary in the ade_config file.

Parameters **key** : class or str, like {'None', 'Recenter', 'Standardize', 'Normalize',

'MinMax'}

The selected preprocessing algorithm. In case in which key is a *class*, it must contain both a *fit* and *transform* method.

content : dict

A dictionary containing parameters for each preprocessing class. Each parameter can be a list; in this case for each combination of parameters a different pipeline will be created.

Returns **tpl** : tuple

A tuple made like ('preproc_name', preproc_obj, 'preproc'), where preproc_obj is an sklearn 'transforms' (i.e. it has bot a fit and transform method).

adenine.core.define_pipeline.**parse_dimred**(*key*, *content*)

Parse the options of the dimensionality reduction step.

This function does the same as parse_preproc but works on the dimensionality reduction & manifold learning options.

Parameters **key** : class or str, like {'None', 'PCA', 'KernelPCA', 'Isomap', 'LLE', 'SE', 'MDS', 'tSNE', 'RBM'}

The selected dimensionality reduction algorithm. In case in which key is a *class*, it must contain both a *fit* and *transform* method.

content : dict

A dictionary containing parameters for each dimensionality reduction class. Each parameter can be a list; in this case for each combination of parameters a different pipeline will be created.

Returns **tpl** : tuple

A tuple made like ('dimres_name', dimred_obj, 'dimred'), where dimred_obj is a sklearn 'transforms' (i.e. it has bot a .fit and .transform method).

adenine.core.define_pipeline.**parse_clustering**(*key*, *content*)

Parse the options of the clustering step.

This function does the same as parse_preproc but works on the clustering options.

Parameters **key** : class or str, like {'KMeans', 'AP', 'MS', 'Spectral', 'Hierarchical'}

The selected clustering algorithm. In case in which key is a *class*, it must contain a *fit* method.

content : dict

A dictionary containing parameters for each clustering class. Each parameter can be a list; in this case for each combination of parameters a different pipeline will be created.

Returns **tpl** : tuple

A tuple made like ('clust_name', clust_obj, 'clustering'), where clust_obj implements the *fit* method.

adenine.core.define_pipeline.**parse_steps**(*steps*, *max_n_pipes=200*)

Parse the steps and create the pipelines.

This function parses the steps coded as dictionaries in the ade_config files and creates a sklearn pipeline objects for each combination of imputing -> preprocessing -> dimensionality reduction -> clustering algorithms.

A typical step may be of the following form: stepX = {'Algorithm': [On/Off flag, {'parameter1', [list of params]}]}

where On/Off flag = {True, False} and 'list of params' allows to specify multiple params. In case in which the 'list of params' is actually a list, multiple pipelines are created for each combination of parameters.

Parameters **steps** : list of dictionaries

A list of (usually 4) dictionaries that contains the details of the pipelines to implement.

max_n_pipes : int, optional, default: 200

The maximum number of combinations allowed. This avoids a too expensive computation.

Returns **pipes** : list of sklearn.pipeline.Pipeline

The returned list must contain every possible combination of imputing -> preprocessing -> dimensionality reduction -> clustering algorithms (up to max_n_pipes).

adenine.core.pipelines.**create**(*pdef*)

Scikit-learn Pipelines objects creation (deprecated).

This function creates a list of sklearn Pipeline objects starting from the list of list of tuples given in input that could be created using the adenine.core.define_pipeline module.

Parameters **pdef** : list of list of tuples

This arguments contains the specification needed by sklearn in order to create a working Pipeline object.

Returns **pipes** : list of sklearn.pipeline.Pipeline objects

The list of Piplines, each of them can be fitted and trasformed with some data.

adenine.core.pipelines.**which_level**(*label*)

Define the step level according to the input step label [DEPRECATED].

This function return the level (i.e.: imputing, preproc, dimred, clustring, None) according to the step label provided as input.

Parameters **label** : string

This is the step level as it is reported in the ade_config file.

Returns **level** : {imputing, preproc, dimred, clustering, None}

The appropriate level of the input step.

adenine.core.pipelines.**evaluate**(*level*, *step*, *X*)

Transform or predict according to the input level.

This function uses the transform or the predict method on the input sklearn-like step according to its level (i.e. imputing, preproc, dimred, clustering, none).

Parameters **level** : {'imputing', 'preproc', 'dimred', 'clustering', 'None'}

The step level.

step : sklearn-like object

This might be an Imputer, or a PCA, or a KMeans (and so on...) sklearn-like object.

X : array of float, shape

The input data matrix.

Returns **res** : array of float

A matrix projection in case of dimred, a label vector in case of clustering, and so on.

adenine.core.pipelines.**pipe_worker**(*pipe_id*, *pipe*, *pipes_dump*, *X*)

Parallel pipelines execution.

Parameters **pipe_id** : string

Pipeline identifier.

pipe : list of tuples

Tuple containing a label and a sklearn Pipeline object.

pipes_dump : multiprocessing.Manager.dict

Dictionary containing the results of the parallel execution.

X : array of float, shape

The input data matrix.

Adenine analyzer module.

adenine.core.analyze_results.**est_clst_perf**(*root*, *data_in*, *labels=None*, *t_labels=None*, *model=None*, *metric='euclidean'*)

Estimate the clustering performance.

This estimates the clustering performance by means of several indexes. Results are saved in a tree-like structure in the root folder.

Parameters **root** : string

The root path for the output creation.

data_in : array of float, shape

The low space embedding estimated by the dimensinality reduction and manifold learning algorithm.

labels : array of float, shape

The label assignment performed by the clustering algorithm.

t_labels : array of float, shape

The true label vector; None if missing.

model : sklearn or sklearn-like object

An instance of the class that evaluates a step. In particular this must be a clustering model provided with the **clusters_centers_** attribute (e.g. KMeans).

metric : string

The metric used during the clustering algorithms.

adenine.core.analyze_results.**make_df_clst_perf**(*root*)

Summarize all the clustering performance estimations.

Given the output file produced by est_clst_perf(), this function groups all of them together in friendly text and latex files, and saves the two files produced in a tree-like structure in the root folder.

Parameters **root** : string

The root path for the output creation.

adenine.core.analyze_results.**get_step_attributes**(*step*, *pos*)

Get the attributes of the input step.

This function returns the attributes (i.e. level, name, outcome) of the input step. This comes handy when dealing with steps with more than one parameter (e.g. KernelPCA 'poly' or 'rbf').

Parameters **step** : list

A step coded by ade_run.py as [name, level, param, data_out, data_in, mdl obj, voronoi_mdl_obj]

pos : int

The position of the step inside the pipeline.

Returns **name** : string

A unique name for the step (e.g. KernelPCA_rbf).

level : {imputing, preproc, dimred, clustering}

The step level.

data_out : array of float, shape

Where n_out is n_dimensions for dimensionality reduction step, or 1 for clustering.

data_in : array of float, shape

Where n_in is n_dimensions for preprocessing/imputing/dimensionality reduction step, or n_dim for clustering (because the data have already been dimensionality reduced).

param : dictionary

The parameters of the sklearn object implementing the algorithm.

mdl_obj : sklearn or sklearn-like object

This is an instance of the class that evaluates a step.

adenine.core.analyze_results.**analysis_worker**(*elem*, *root*, *y*, *feat_names*, *index*, *lock*)

Parallel pipelines analysis.

Parameters **elem** : list

The first two element of this list are the pipe_id and all the data of that pipeline.

root : string

The root path for the output creation.

y : array of float, shape

The label vector; None if missing.

feat_names : array of integers (or strings), shape

The feature names; a range of numbers if missing.

index : list of integers (or strings)

This is the samples identifier, if provided as first column (or row) of of the input file. Otherwise it is just an incremental range of size n_samples.

lock : multiprocessing.synchronize.Lock

Obtained by multiprocessing.Lock(). Needed for optional creation of directories.

1.2.2 Input Data

This module is just a wrapper for some sklearn.datasets functions.

adenine.utils.data_source.**generate_gauss**(*mu=None*, *std=None*, *n_sample=None*)

Create a Gaussian dataset.

Generates a dataset with n_sample * n_class examples and n_dim dimensions.

Parameters **mu** : array of float, shape

The mean of each class.

std : array of float, shape

The standard deviation of each Gaussian distribution.

n_sample : int

Number of point per class.

adenine.utils.data_source.**load_custom**(*x_filename*, *y_filename*, *samples_on='rows'*, ***kwargs*)

Load a custom dataset.

This function loads the data matrix and the label vector returning a unique sklearn-like object dataSetObj.

Parameters **x_filename** : string

The data matrix file name.

y_filename : string

The label vector file name.

samples_on : string

This can be either in ['row', 'rows'] if the samples lie on the row of the input data matrix, or viceversa in ['col', 'cols'] the other way around.

kwargs : dict

Arguments of pandas.read_csv function.

Returns **data** : sklearn.datasets.base.Bunch

An instance of the sklearn.datasets.base.Bunch class, the meaningful attributes are .data, the data matrix, and .target, the label vector.

adenine.utils.data_source.**load**(*opt='custom'*, *x_filename=None*, *y_filename=None*, *n_samples=0*, *samples_on='rows'*, ***kwargs*)

Load a specified dataset.

This function can be used either to load one of the standard scikit-learn datasets or a different dataset saved as X.npy Y.npy in the working directory.

Parameters **opt** : {'iris', 'digits', 'diabetes', 'boston', 'circles', 'moons',

'custom'}, default: 'custom'

Name of a predefined dataset to be loaded.

x_filename : string, default

The data matrix file name.

y_filename : string, default

The label vector file name.

n_samples : int

The number of samples to be loaded. This comes handy when dealing with large datasets. When n_samples is less than the actual size of the dataset this function performs a random subsampling that is stratified w.r.t. the labels (if provided).

samples_on : string

This can be either in ['row', 'rows'] if the samples lie on the row of the input data matrix, or viceversa in ['col', 'cols'] the other way around.

data_sep : string

The data separator. For instance comma, tab, blank space, etc.

Returns **X** : array of float, shape

The input data matrix.

y : array of float, shape

The label vector; np.nan if missing.

feature_names : array of integers (or strings), shape

The feature names; a range of number if missing.

index : list of integers (or strings)

This is the samples identifier, if provided as first column (or row) of of the input file. Otherwise it is just an incremental range of size n_samples.

1.2.3 Plotting functions

Adenine plotting module.

`adenine.core.plotting.`**`silhouette`**(*root*, *data_in*, *labels*, *model=None*)

Generate and save the silhouette plot of data_in w.r.t labels.

This function generates the silhouette plot representing how data are correctly clustered, based on labels. The plots will be saved into the root folder in a tree-like structure.

Parameters **root** : string

The root path for the output creation

data_in : array of float, shape

The low space embedding estimated by the dimensionality reduction and manifold learning algorithm.

labels : array of float, shape

The label vector. It can contain true or estimated labels.

model : sklearn or sklearn-like object

An instance of the class that evaluates a step.

`adenine.core.plotting.`**`scatter`**(*root*, *data_in*, *labels=None*, *true_labels=False*, *model=None*)

Generate the scatter plot of the dimensionality reduced data set.

This function generates the scatter plot representing the dimensionality reduced data set. The plots will be saved into the root folder in a tree-like structure.

Parameters **root** : string

The root path for the output creation

data_in : array of float, shape

The low space embedding estimated by the dimensinality reduction and manifold learning algorithm.

labels : array of float, shape

The label vector. It can contain true or estimated labels.

true_labels : boolean

Identify if labels contains true or estimated labels.

model : sklearn or sklearn-like object

An instance of the class that evaluates a step. In particular this must be a clustering model provided with the **clusters_centers_** attribute (e.g. KMeans).

adenine.core.plotting.**voronoi**(*root*, *data_in*, *labels=None*, *true_labels=False*, *model=None*)

Generate the Voronoi tessellation obtained from the clustering algorithm.

This function generates the Voronoi tessellation obtained from the clustering algorithm applied on the data projected on a two-dimensional embedding. The plots will be saved into the appropriate folder of the tree-like structure created into the root folder.

Parameters **root** : string

The root path for the output creation

data_in : array of float, shape

The low space embedding estimated by the dimensinality reduction and manifold learning algorithm.

labels : array of int, shape

The result of the clustering step.

true_labels : boolean [deprecated]

Identify if labels contains true or estimated labels.

model : sklearn or sklearn-like object

An instance of the class that evaluates a step. In particular this must be a clustering model provided with the **clusters_centers_** attribute (e.g. KMeans).

adenine.core.plotting.**tree**(*root*, *data_in*, *labels=None*, *index=None*, *model=None*)

Generate the tree structure obtained from the clustering algorithm.

This function generates the tree obtained from the clustering algorithm applied on the data. The plots will be saved into the appropriate folder of the tree-like structure created into the root folder.

Parameters **root** : string

The root path for the output creation

data_in : array of float, shape

The low space embedding estimated by the dimensinality reduction and manifold learning algorithm.

labels : array of int, shape

The result of the clustering step.

index : list of integers (or strings)

This is the samples identifier, if provided as first column (or row) of of the input file. Otherwise it is just an incremental range of size n_samples.

model : sklearn or sklearn-like object

An instance of the class that evaluates a step. In particular this must be a clustering model provided with the **clusters_centers_** attribute (e.g. KMeans).

adenine.core.plotting.**dendrogram**(*root*, *data_in*, *labels=None*, *index=None*, *model=None*, *n_max=150*)

Generate and save the dendrogram obtained from the clustering algorithm.

This function generates the dendrogram obtained from the clustering algorithm applied on the data. The plots will be saved into the appropriate folder of the tree-like structure created into the root folder. The row colors of the heatmap are the either true or estimated data labels.

Parameters **root** : string

The root path for the output creation

data_in : array of float, shape

The low space embedding estimated by the dimensinality reduction and manifold learning algorithm.

labels : array of int, shape

The result of the clustering step.

index : list of integers (or strings)

This is the samples identifier, if provided as first column (or row) of of the input file. Otherwise it is just an incremental range of size n_samples.

model : sklearn or sklearn-like object

An instance of the class that evaluates a step. In particular this must be a clustering model provided with the **clusters_centers_** attribute (e.g. KMeans).

n_max : int, (INACTIVE)

The maximum number of samples to include in the dendrogram. When the number of samples is bigger than n_max, only n_max samples randomly extracted from the dataset are represented. The random extraction is performed using sklearn.model_selection.StratifiedShuffleSplit (or sklearn.cross_validation.StratifiedShuffleSplit for legacy reasons).

`adenine.core.plotting.`**`pcmagnitude`**(*root*, *points*, *title=''*, *ylabel=''*)

Plot the trend of principal components magnitude.

Parameters **root** : string

The root path for the output creation.

points : array of float, shape

This could be the explained variance ratio or the eigenvalues of the centered matrix, according to the PCA algorithm of choice, respectively PCA or KernelPCA.

title : string

Plot title.

ylabel : string

Y-axis label.

`adenine.core.plotting.`**`eigs`**(*root*, *affinity*, *n_clusters=0*, *title=''*, *ylabel=''*, *normalised=True*, *n_components=20*, *filename=None*, *ylim='auto'*, *rw=False*)

Plot eigenvalues of the Laplacian associated to data affinity matrix.

Parameters **root** : string

The root path for the output creation.

affinity : array of float, shape

The affinity matrix.

n_clusters : int, optional

The number of clusters.

title : string, optional

Plot title.

ylabel : string, optional

Y-axis label.

normalised : boolean, optional, default True

Choose whether to normalise the Laplacian matrix.

n_components : int, optional, default 20

Number of components to show in the plot.

filename : None or str, optional, default None

If not None, overrides default filename for saving the plot.

ylim : 'auto', None, tuple or list, optional, default 'auto'

If 'auto', choose the highest eigenvalue for the height of the plot. If None, plt.ylim is not called (matplotlib default is used). Otherwise, specify manually the desired ylim.

rw : boolean, optional, default False

Normalise the Laplacian matrix as the random walks point of view. This should be better suited with unclear data distributions.

1.2.4 Extra tools

class adenine.utils.extra.**Palette**(*name='Set1'*, *n_colors=6*)
Wrapper for seaborn palette.

adenine.utils.extra.**values_iterator**(*dictionary*)
Add support for python2 or 3 dictionary iterators.

adenine.utils.extra.**items_iterator**(*dictionary*)
Add support for python2 or 3 dictionary iterators.

adenine.utils.extra.**modified_cartesian**(**args*, ***kwargs*)
Modified Cartesian product.

This takes two (or more) lists and returns their Cartesian product. If one of two list is empty this function returns the non-empty one.

Parameters ***args** : lists, length

The group of input lists.

Returns **cp** : list

The Cartesian Product of the two (or more) nonempty input lists.

adenine.utils.extra.**make_time_flag**()
Generate a time flag.

This function simply generates a time flag using the current time.

Returns **timeFlag** : string

A unique time flag.

adenine.utils.extra.**sec_to_time**(*seconds*)
Transform seconds into a formatted time string.

Parameters **seconds** : int

Seconds to be transformed.

Returns **time** : string

A well formatted time string.

adenine.utils.extra.**get_time**()
Get time of now, in string.

adenine.utils.extra.**ensure_symmetry**(*X*)
Ensure matrix symmetry.

Parameters **X** : numpy.ndarray

Input matrix of precomputed pairwise distances.

Returns **new_X** : numpy.ndarray

Symmetric distance matrix. Values are averaged.

adenine.utils.extra.**timed**(*function*)
Decorator that measures wall time of the decored function.

adenine.utils.extra.**set_module_defaults**(*module*, *dictionary*)
Set default variables of a module, given a dictionary.

Used after the loading of the configuration file to set some defaults.

a