adenine Documentation

Release 0.1.0

Samuele Fiorini

April 04, 2016

adenine is a machine learning and data mining Python pipeline that helps you to answer this tedious question: are my data relevant with the problem I'm dealing with?

The main structure of adenine can be summarized in the following 3 steps.

1. **Preprocessing:** Have you ever wondered what would have changed if only your data have been preprocessed in a different way? Or is data preprocessing is a good idea at all? adenine offers several preprocessing procedures, such as: data centering, Min-Max scaling, standardization or normalization and allows you to compare the results of the analysis conducted with different starting point.
2. **Dimensionality Reduction:** In the context of data exploration, this phase becomes particularly helpful for high dimensional data (e.g. -omics scenario). This step, generically named DR, may actually include some manifold learning (such as Isomap, Multidimensional Scaling, etc), supervised (Linear Discriminant Analysis) and unsupervised (Principal Component Analysis, kernel PCA) techniques.
3. **Clustering:** This section aims at grouping data into clusters without taking into account the class labels. Several techniques such as K-Means, Spectral or Hierarchical clustering will work on both original and dimensionality reduced data.

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

CHAPTER

ONE

USER DOCUMENTATION

1.1 Quick start tutorial

adenine may be installed using standard Python tools (with administrative or sudo permissions on GNU-Linux platforms):

```
$ pip install adenine

or

$ easy_install adenine
```

1.1.1 Installation from sources

If you like to manually install **adenine**, download the source tar.gz from our BitBucket repository. Then extract it and move into the root directory:

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

From here, you may use 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` directory with some `.npy` files which will be used to show **adenine**'s' functionalities.

adenine needs only 3 ingredients:

- A `X.npy` input matrix
- A `y.npy` output vector (optional)
- A `configuration` file

1.1.2 Input data format

Input data are assumed to be `numpy` array dumped in a `.npy` files organized with a row for each sample and a column for each feature.

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/python
# -*- coding: utf-8 -*-

from adenine.utils import data_source

# --------------------------  EXPERMIENT INFO ------------------------- #
exp_tag = 'debug_csv'
output_root_folder = 'results'
# parallel = False

# ----------------------------  INPUT DATA ---------------------------- #
# X, y, feat_names, class_names = data_source.load('iris')
# X, y, feat_names, class_names = data_source.load('gauss')
# X, y, feat_names, class_names = data_source.load('digits')
# X, y, feat_names, class_names = data_source.load('diabetes')
# X, y, feat_names, class_names = data_source.load('boston')
# X, y, feat_names, class_names = data_source.load('custom', 'X.npy', 'y.npy')
# X, y, feat_names, class_names = data_source.load('custom', 'X.csv', 'y.csv')
X, y, feat_names, class_names = data_source.load('custom', '/home/fede/src/adenine/adenine/examples/T
if not (X.T == X).all():
    X = (X.T + X) / 2.
    X = 1. - X

# -----------------------  PIPELINE DEFINITION ------------------------ #

# --- Missing Values Imputing --- #
step0 = {'Impute': [False], 'Missing': [-1], 'Replacement': ['median','mean']}

# --- Data Preprocessing --- #
step1 = {'None': [True], 'Recenter': [False], 'Standardize': [False],
         'Normalize': [False, ['l2']], 'MinMax': [False, [0,1]]}

# --- Dimensionality Reduction & Manifold Learning --- #
step2 = {'PCA': [False], 'IncrementalPCA': [False], 'RandomizedPCA': [False],
         'KernelPCA': [False, ['linear','rbf','poly']], 'Isomap': [False],
         'LLE': [False, ['standard','modified','hessian', 'ltsa']],
         'SE': [False], 'MDS': [False, ['metric','nonmetric']],
         'tSNE': [False], 'None': [True]}

# --- Clustering --- #
step3 = {'KMeans': [False, [5]],
         'KernelKMeans': [False, [3,['rbf','poly']]], #TODO
         'AP': [False, ['precomputed']], 'MS': [False],
         'Spectral': [True, [50, ['precomputed']]],
         #'Hierarchical': [False, [3, ['manhattan','euclidean'], ['ward','complete','average']]]
         'Hierarchical': [False, [3, ['precomputed']]]
         }
```

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
```

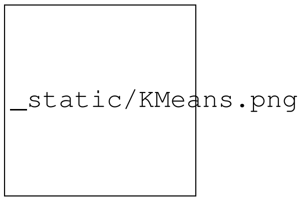
When launched, the script reads the data, then it creates and runs each pipeline saving the results in a three-like structure which has the current folder as root.

1.1.5 Results analysis

This is the last step, needed to be performed in order to get some useful summaries and plots from an already executed experiment. The `ade_analysis.py` script accepts as only parameter a result directory already created:

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

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



1.2 API

1.2.1 Pipeline utilities

class `adenine.core.define_pipeline.`**`DummyNone`**

Dummy class that does nothing.

It is a sklearn 'transforms', it implements both a fit and a transform method and it just returns the data in input. It has been created only for consistency with sklearn.

`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** : {'None', 'Recenter', 'Standardize', 'Normalize', 'MinMax'}

The type of selected preprocessing step.

content : list, len

A list containing the On/Off flag and a nested list of extra parameters (e.g. [min,max] for Min-Max scaling).

Returns **pptpl** : tuple

A tuple made like that ('PreprocName', preprocObj), where preprocObj 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 : {'None', 'PCA', 'KernelPCA', 'Isomap', 'LLE', 'SE', 'MDS', 'tSNE'}

The selected dimensionality reduction algorithm.

content : list, len

A list containing the On/Off flag and a nested list of extra parameters (e.g. ['rbf,'poly'] for KernelPCA).

Returns drtpl : tuple

A tuple made like that ('DimRedName', dimredObj), where dimredObj is an 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 : {'KMeans', 'KernelKMeans', 'AP', 'MS', 'Spectral', 'Hierarchical'}

The selected dimensionality reduction algorithm.

content : list, len

A list containing the On/Off flag and a nested list of extra parameters (e.g. ['rbf,'poly'] for KernelKMeans).

Returns cltpl : tuple

A tuple made like that ('ClusteringName', clustObj), where clustObj implements the .fit method.

`adenine.core.define_pipeline.`**`parse_steps`**`(`*steps*`)`

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 -> dimensinality reduction -> clustering algorithms.

A typical step may be of the following form: stepX = {'Algorithm': [On/Off flag, [variant0, ...]]}

where On/Off flag = {True, False} and variantX = 'string'.

Parameters steps : list of dictionaries

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

Returns pipes : list of sklearn.pipeline.Pipeline

The returned list must contain every possible combination of imputing -> preprocessing -> dimensionality reduction -> clustering algorithms. The maximum number of pipelines that could be generated is 20, even if the number of combinations is higher.

`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.

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.`**`run`**(*pipes=()*, *X=()*, *exp_tag='def_tag'*, *root=''*)

Fit and transform/predict some pipelines on some data.

This function fits each pipeline in the input list on the provided data. The results are dumped into a pkl file as a dictionary of dictionaries of the form {'pipeID': {'stepID' : [alg_name, level, params, data_out, data_in, model_obj, voronoi_suitable_object], ...}, ...}. The model_obj is the sklearn model which has been fit on the dataset, the voronoi_suitable_object is the very same model but fitted on just the first two dimensions of the dataset. If a pipeline fails for some reasons the content of the stepID key is a list of np.nan.

Parameters pipes : list of list of tuples

Each tuple contains a label and a sklearn Pipeline object.

X : array of float, shape

The input data matrix.

exp_tag : string

An intuitive tag for the current experiment.

root : string

The root folder to save the results.

parallel : bool

Run all the pipelines in parallel on the cores of your machine (if True, it requires pplus).

Returns outputFolderName : string

The path of the output folder.

`adenine.core.analyze_results.`**`make_scatter`**(*root=()*, *embedding=()*, *model_param=()*, *trueLabel=nan*)

Generate and save 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

embedding : array of float, shape

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

model_param : dictionary

The parameters of the dimensionality reduciont and manifold learning algorithm.

trueLabel : array of float, shape

The true label vector; np.nan if missing (useful for plotting reasons).

`adenine.core.analyze_results.`**`make_voronoi`**(*root=()*, *data_in=()*, *model_param=()*, *trueLabel=nan*, *labels=()*, *model=()*)

Generate and save 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.

model_param : dictionary

The parameters of the dimensionality reduciont and manifold learning algorithm.

trueLabel : array of float, shape

The true label vector; np.nan if missing (useful for plotting reasons).

labels : array of int, shape

The result of the clustering step.

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.analyze_results.`**`est_clst_perf`**(*root=()*, *data_in=()*, *label=()*, *trueLabel=nan*, *model=()*, *metric='euclidean'*)

Estimate the clustering performance.

This function estimate the clustering performance by means of several indexes. Then eventually saves the results 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.

label : array of float, shape

The label assignment performed by the clusterin algorithm.

trueLabel : array of float, shape

The true label vector; np.nan 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).

`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, results, parameters]

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.`**`make_tree`**(*root=()*, *data_in=()*, *model_param=()*, *trueLabel=nan*, *labels=()*, *model=()*)

Generate and save 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.

model_param : dictionary

The parameters of the dimensionality reduciont and manifold learning algorithm.

trueLabel : array of float, shape

The true label vector; np.nan if missing (useful for plotting reasons).

labels : array of int, shape

The result of the clustering step.

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.analyze_results.`**`make_dendrogram`**(*root=()*, *data_in=()*, *model_param=()*, *trueLabel=nan*, *labels=()*, *model=()*)

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.

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.

model_param : dictionary

The parameters of the dimensionality reduciont and manifold learning algorithm.

trueLabel : array of float, shape

The true label vector; np.nan if missing (useful for plotting reasons).

labels : array of int, shape

The result of the clustering step.

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.analyze_results.`**`make_scatterplot`**(*root=()*, *data_in=()*, *model_param=()*, *trueLabel=nan*, *labels=()*, *model=()*, *n_dimensions=2*)

Generate and save the scatter plot obtained from the clustering algorithm.

This function generates the scatter plot obtained from the clustering algorithm applied on the data projected on a two-dimensional embedding. The color of the points in the plot is consistent with the label estimated by the algorithm. 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.

model_param : dictionary

The parameters of the dimensionality reduciont and manifold learning algorithm.

trueLabel : array of float, shape

The true label vector; np.nan if missing (useful for plotting reasons).

labels : array of int, shape

The result of the clustering step.

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.analyze_results.`**`start`** (*inputDict=()*, *rootFolder=()*, *y=nan*, *feat_names=()*, *class_names=()*)

Analyze the results of ade_run.

This function analyze the dictionary generated by ade_run, generates the plots, and saves them in a tree-like folder structure in rootFolder.

Parameters **inputDict** : dictionary

The dictionary created by ade_run.py on some data.

rootFolder : string

The root path for the output creation

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 numbers if missing.

class_names : array of integers (or strings), shape

The class names; a range of numbers if missing.

1.2.2 Input Data

`adenine.utils.data_source.`**`MixGauss`** (*mu=()*, *std=()*, *n_sample=()*)

Create a Gaussian dataset.

Generates a dataset with n_sample * n_class examples and n_dim dimensions. Mu, the mean vector, is n_class x n_dim.

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`**(*fileName_X*, *fileName_y*)

Load a custom dataset.

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

Parameters **fileName_X** : string

The data matrix file name.

fileName_y : string

The label vector file name.

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'*, *fileName_X=None*, *fileName_y=None*)

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', 'blobs','custom'}, default: 'custom'

fileName_X : string, default

The data matrix file name.

fileName_y : string, default

The label vector file name.

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.

1.2.3 Extra tools

`adenine.utils.extra.`**`modified_cartesian`**(**args*)

Modified Cartesian product.

This function takes two (ore more) lists and returns their Cartesian product, if one of the 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 formatted time string

PYTHON MODULE INDEX

PYTHON MODULE INDEX