Mapping Learning Documentation

Alban Thomas

Dec 14, 2019

Table of Contents

Note: Mapping Learning (also called *maplearn*) makes **use of machine learning easy** (easier, at least). Initially designed for geographical data (cartography based on remote sensing), *Maplearn* also deals very well with classical data (*ie* tabular).

NB: information in french is available in maplearn's wiki .

Maplearn is a **free software and library**, distributed under lGPL v3 license.

Written in Python, *maplearn* can be used whichever your operation system (Linux, Mac, Windows).

CHAPTER 1

Features

- **many algorithms** to make predictions (classification, clustering or regression)
- look for best hyper-parameters to **improve accuracy** of your results
- generalize machine learning's best practices (k-fold…)
- several preprocessing tasks available : reduction of dimensions…
- reads/writes **several file formats** (*geographic or not*)
- synthetizes results in a **standardized report**
- statiscal and more empirical **advices** will help novice users

CHAPTER 2

Install

Note: Mapping Learning is still in an early stage of development . Will you dare installing an *alpha* version of a software? Mapping Learning is worth it…

2.1 Installing

If you do not understand the below lines, check *2. Before installing*.

2.1.1 Anaconda (easiest)

Just want to enjoy *Mapping Learning*? Type:

```
conda install -c sympythy maplearn
```

2.1.2 PIP

You can also use PIP, but you may have to deal with some tricky dependancies (*ie* GDAL):

```
pip install maplearn
```

2.1.3 From source code (for developpers/curious)

Want to contribute to the source code of Mapping Learning ? Or just curious ? You can also install *maplearn* from source code:

```
git clone https://bitbucket.org/thomas_a/maplearn.git
cd maplearn
pip install .
```

Then, you will be able to get last changes (in *maplearn* folder):

```
# get source code updates
git pull
# upgrade maplearn
pip install .
```

2.2 2. Before installing

Mapping Learning is based on Python and you need a way to install properly its dependancies (either Anaconda or PIP).

2.2.1 Using conda (Windows, Mac or Linux)

First, install an Anaconda distribution:

- either Anaconda (complete): https://www.anaconda.com/distribution/
- or Miniconda (lighter): https://docs.conda.io/en/latest/miniconda.html

Nb: if you wonder which one you should use, have a look on https://docs.conda.io/projects/conda/en/latest/user-guide/install/download.html#anaconda-or-miniconda

Then, start "Anaconda prompt" and type:

```
conda install -c sympythy maplearn
```

2.2.2 Using system packages and PIP (Linux, Debian-based distributions)

First, install pip and some dependancies (Blas, Lapack..):

```
apt-get install python3-pip
apt-get install libblas-dev liblapack-dev libatlas-base-dev gfortran
```

Then, you can also install some Python librairies:

```
apt-get install python3-numpy python3-scipy python3-pandas python3-gdal
apt-get install python3-sklearn python3-seaborn python3-markdown
```

NB: you can install Python librairies thanks to system packages (apt) or Pip. Using System packages should be easier and faster. PIP will bring libraries in latest version.

Finally, you can install *maplearn*:

```
pip install maplearn
```

CHAPTER 3

Use

Mapping Learning aims lets you choose how to use it:

- as a *"classical"* software (**GUI** - Graphical User Interface)
- typing in a terminal, with a **CLI** (Command Line Interface)
- as a Python library

Note:

- Whichever interface you choose, **configuration is the same**. Check `maplearn.app.config` for more details.
- Machine Learning concepts are explained in `maplearn.ml`

This page describes how to use Mapping Learning as an application, whereas every other pages will help you to develop your own scripts using one or several of its module(s).

3.1 Mapping Learning's GUI

The **Graphical User Interface** (GUI) aims to help you to:

1. **understand** how to do machine learning well
2. **properly configure** the application and get results

To run *maplearn* with its GUI, type in a terminal:

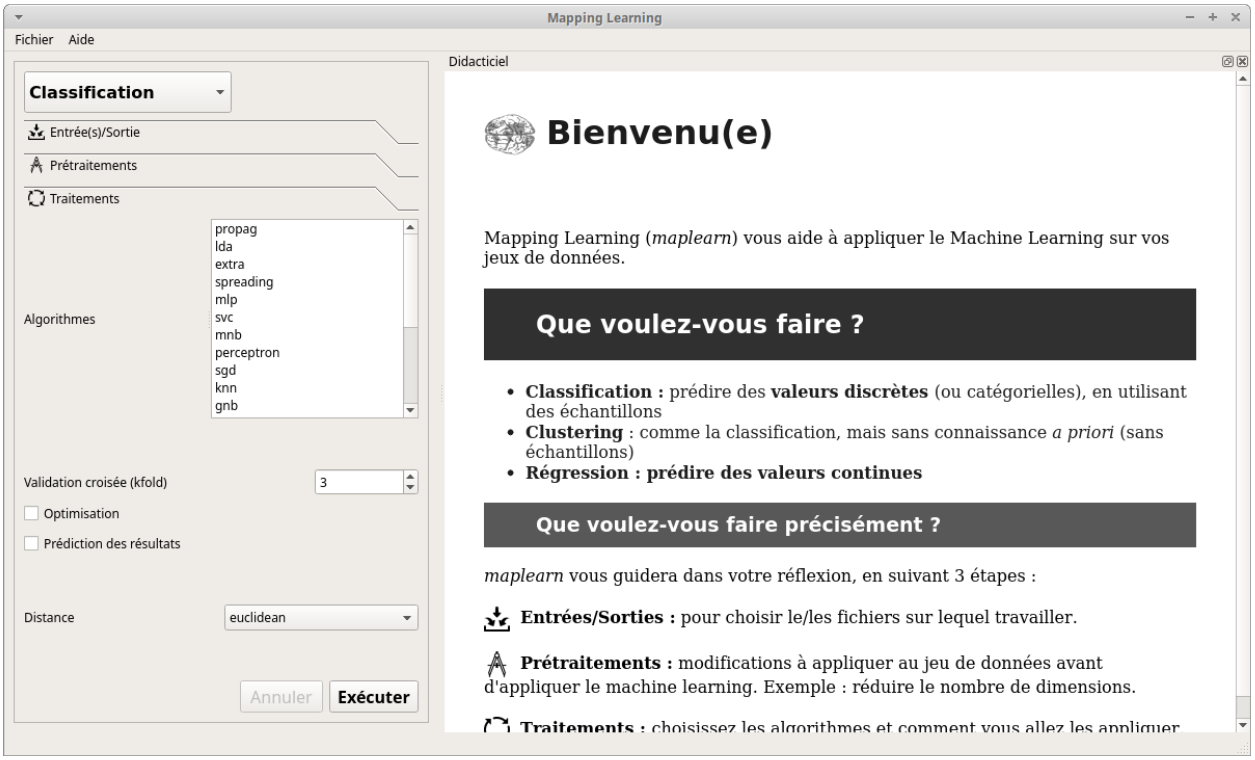
```
maplearn_gui
```

NB: this command calls the Python script "run_gui.py".

Structure:

- On the left, you can set paramaters

- On the right, you can read help about these parameters



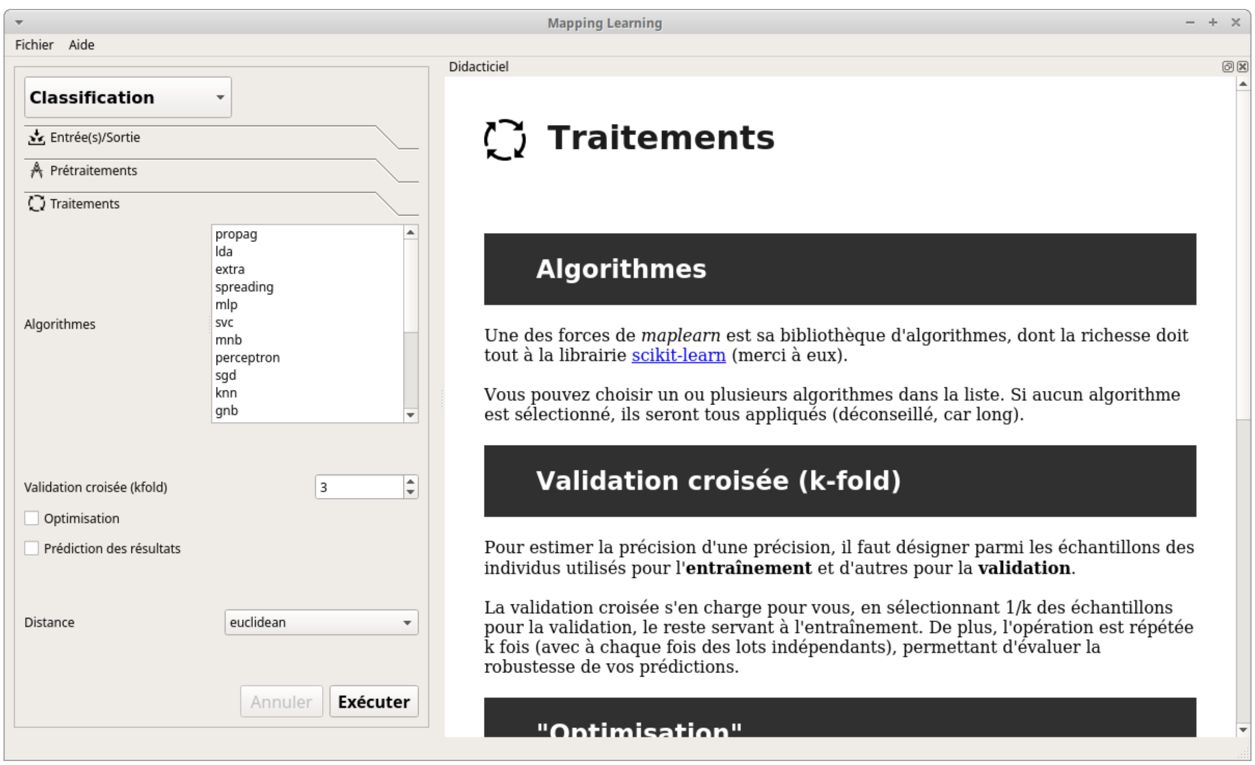
NB: For now, the GUI is only available in French but its translation (at least in English) is considered.

The interface will accompany you through the 3 steps necessary to the configuration.

1. **Input/Output**

2. **Preprocessing**

3. **Processing**



After having defined all the necessary parameters, all you have to do is click on "Executer" and be a little patient. . .

`maplearn.run_gui.main()`

Run Mapping Learning with its GUI (Graphical User Interface)

3.2 Mapping Learning's CLI

The CLI (Commande Line Interface) is one of the main entry to play with Mapping Learning. Just specify a well-formatted configuration file and run.

The syntax of configuration file is described on https://bitbucket.org/thomas_a/maplearn/wiki/configuration. A few examples of configuration are also available in "examples" sub-folder.

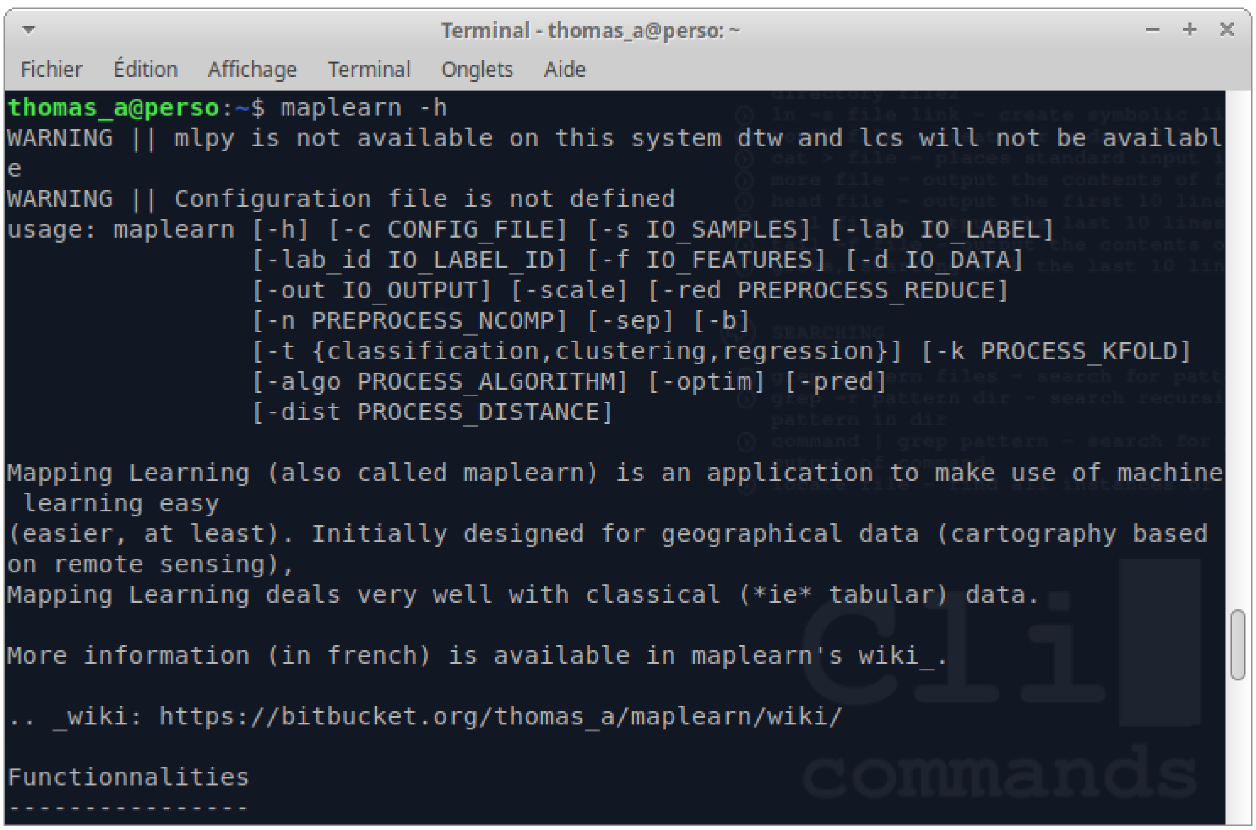
Example:

```
maplearn -c /path/to/configfile
```

With its CLI you can call Maplearn in an **automated** way through planified tasks for instance. You can also check easily the effect of a (few) parameter(s).

To get available parameters, type:

```
maplearn --help
# or
maplearn -h
```



Now you can easy change the value of a parameter without creating a new configuration file:

Example:

```
# Changing the number of k-folds (to 5):
maplearn -c /path/to/configfile -k 5
```

Be careful about the output folder, or every new run of maplearn will replace previous results. Don't worry: there is a parameter for that.

```
maplearn -c /path/to/configfile -k 5 -out /path/to/new/directory
```

`maplearn.run.`**`run`**`()`
Run Mapping Learning using the previously loaded configuration

Mapping Learning : examples

This script is a good way to discover Mapping Learning possibilities. Using configuration files available in "examples" subfolder and datasets included in "datasets" sub-folder, you will see what can do this application.

Example:

- Asks the user to choose which examples(s) he wants to test:

```
maplearn_example
```

- Execute 3rd example (CLI way):

```
maplearn_example 3
```

- Launch every available examples (takes some minutes…):

```
maplearn_example all
```

NB: "maplearn_example" calls the code inside "run_example.py"

`maplearn.run_example.`**`main`**`()`
Main script to run included examples

`maplearn.run_example.`**`run_cfg`**`(`*cfg_file*, *path*`)`
Run one of available examples in "examples" folder based on a configuration file

Args:

- cfg_file (str) : path to the configuration file
- path (str) : path to run.py script (that launches the application)

`maplearn.run_example.`**`run_example`**`(`**ex*`)`
Run one or several examples based on their number

Arg:

- number (str) : identifies the example to run

3.3 Output

Note: When processing is done, *maplearn* will show you the results in a standardized report (HTML page), describing:

- the dataset used
- any pre-treatment(s)
- applied algorithm(s)
- statistical results (in the form of graphs and tables)

- a synthesis comparing the result of the different algorithms

---

For your convenience, an example of output (in french) is available on this link .

CHAPTER 4

Source code

Note: Mapping Learning is written in Python and uses major Open Source libraries, like scikit-learn (Machine Learning algorithms), numpy and pandas to manipulate scientific data and Gdal to handle geographic data.

4.1 Modules

Mapping Learning consists of 4 modules. The first 3 modules allow you to start from your files and to obtain predictions (based on machine learning), and vice versa. The fourth (*app*) is the "conductor", who drives the other parts of the code.

1. `maplearn.ml` : machine learning processing (and preprocessing)
2. `maplearn.datahandler` : to get/export a dataset usable in machine learning and the corresponding files
3. `maplearn.filehandler` : read/write a file
4. `maplearn.app` : application modules (configuration, . . .)

4.1.1 maplearn.ml package

Machine Learning

What is Machine Learning?

From Wikipedia: "Machine learning algorithms build a mathematical **model** based on **sample** data, known as "training data", in order to make **predictions** or decisions without being explicitly programmed to perform the task."

So, we use Machine Learning to predict results about unknown data:

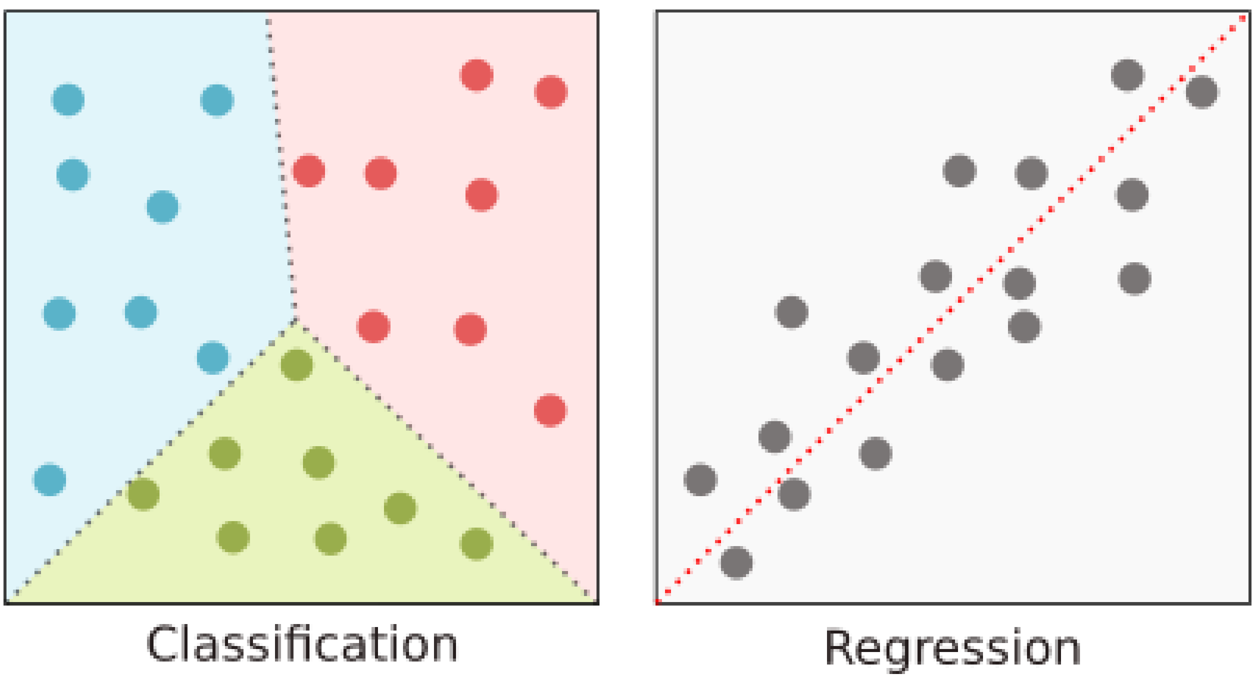
- Is this new email a spam?
- Is this an image of a cat or a dog?
- How many people are going to buy my new product?
- *Applications are infinite…*

To answer these questions, we will use mathematical models (the cloud in the above figure) that need to be trained (or **fitted**) prior to make **predictions**.

What to predict?

Depending on the nature of the values to be predicted, we will talk about:

- **classification** when the values are discrete (also called categorical)
- **regression** when the values are continuous



Classification and regression both needs some samples for training, they belong to *supervised learning*. If you do not have samples, then you should consider *unsupervised classification*, also called **clustering**.

Note: On the other hand, a regression can't be made without samples.

Maplearn: machine Learning modules

In *maplearn*, machine learning is empowered by **scikit-learn**. One reason is its great documentation. Have a look to go further.

Maplearn provides 3 modules corresponding to each of these tasks:

1. **Classification**
2. **Clustering**
3. **Regression**

Two other modules are linked to these tasks:

- **Confusion**: confusion matrix (used to evaluate classifications)
- **Distance**: computes distance using different formulas

Another task that can accomplish machine learning is to reduce the number of dimensions (also called *features*)

- **Reduction**: dimensionnality reduction

The last submodule is needed for programmation but should not be used itself:

- *Machine*: abstract class of a machine learning processor, one or more algorithms can be applied

Submodules

maplearn.ml.classification module

Classification

Classification methods are used to generate a map with each pixel assigned to a class based on its multispectral composition. The classes are determined based on the spectral composition of training areas defined by the user.

Classification is supervised and need samples to fit on. The output will be be a matrix with integer values.

Example:

```
>>> from maplearn.datahandler.loader import Loader
>>> from maplearn.datahandler.packdata import PackData
>>> loader = Loader('iris')
>>> data = PackData(X=loader.X, Y=loader.Y, data=loader.aData)
>>> lst_algos = ['knn', 'lda', 'rforest']
>>> dir_out = os.path.join('maplean_path', 'tmp')
>>> clf = Classification(data=data, dirOut=dir_out, algorithm=lst_algos)
>>> clf.run()
```

class maplearn.ml.classification.**Classification**(*data=None*, *algorithm=None*, ***kwargs*)

Bases: *maplearn.ml.machine.Machine*

Apply supervised classification onto a dataset:

- samples needed for fitting
- data to predict

Args:

- data (PackData): data to play with

- algorithm (list or str): name of an algorithm or list of algorithm(s)
- **kwargs: other parameters like kfold

export_tree(*out_file=None*)
Exports a decision tree

Args:

- out_file (str): path to the output file

fit_1(*algo*, *verbose=True*)
Fits a classifier using cross-validation

Arg:

- algo (str): name of the classifier

load(*data*)
Loads necessary data for supervised classification:

- samples (X and Y): necessary for fitting
- other (unknwon) data to predict, after fitting

Args:

- data (PackData)

optimize(*algo*)
Optimize parameters of a classifier

Args:

- algo (str): name of the classifier to use

predict_1(*algo*, *proba=True*, *verbose=True*)
Predict classes using a fitted algorithm applied to unknown data

Args:

- algo (str): name of the algorithme to apply
- proba (bool): should probabilities be added to result

run(*predict=False*, *verbose=True*)
Applies every classifiers specified in 'algorithm' property

Args: predict (bool): should be the classifier only fitted or also used to predict?

maplearn.ml.classification.**lcs_kernel**(*x*, *y*)
Custom kernel based on LCS (Longest Common Substring)

Args:

- x and y (matrices)

Returns: matrix of float values

maplearn.ml.classification.**skreport_md**(*report*)
Convert a classification report given by scikit-learn into a markdown table TODO: replaced by a pandas dataframe

Arg:

- report (str): classification report

Returns: str_table: a table formatted as markdown

maplearn.ml.classification.**svm_kernel**(*x*, *y*)
Custom Kernel based on DTW

Args:

- x and y (matrices)

Returns: matrix of float values

maplearn.ml.clustering module

Clustering (unsupervised classification)

A clustering algorithm groups the given samples, each represented as a vector x in the N-dimensional feature space, into a set of clusters according to their spatial distribution in the N-D space. Clustering is an unsupervised classification as no a priori knowledge (such as samples of known classes) is assumed to be available.

Clustering is unsupervised and does not need samples for fitting. The output will be a matrix with integer values.

Example:

```
>>> from maplearn.datahandler.loader import Loader
>>> from maplearn.datahandler.packdata import PackData
>>> loader = Loader('iris')
>>> data = PackData(X=loader.X, Y=loader.Y, data=loader.aData)
>>> lst_algos = ['mkmeans', 'birch']
>>> dir_out = os.path.join('maplean_path', 'tmp')
>>> cls = Clustering(data=data, dirOut=dir_out, algorithm='mkmeans')
>>> cls.run()
```

class maplearn.ml.clustering.**Clustering**(*data=None*, *algorithm=None*, ***kwargs*)
Bases: *maplearn.ml.machine.Machine*

Apply one or several methods of clustering onto a dataset

Args:

- data (PackData): dataset to play with
- algorithm (str or list): name of algorithm(s) to use
- **kwargs: more parameters about clustering. The 'metric' to use, the number of clusters expected ('n_clusters')

fit_1(*algo*, *verbose=True*)
Fits one clustering algorithm

Arg:

- algo (str): name of the algorithm to fit

load(*data*)
Loads necessary data for clustering: no samples are needed.

Arg:

- data (PackData): data to play with

predict_1(*algo*, *export=False*, *verbose=True*)
Makes clustering prediction using one algorithme

Args:

- algo (str): name of the algorithm to use
- export (bool): should the result be exported?

maplearn.ml.confusion module

Confusion matrix

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of a classificarion algorithm (see 'classification' class).

Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class. The name stems from the fact that it makes it easy to see if the system is confusing two classes.

Example:

```
>>> import numpy as np
>>> # creates 2 vectors representing labels
>>> y_true = np.random.randint(0, 15, 100)
>>> y_pred = np.random.randint(0, 15, 100)
>>> cm = Confusion(y_true, y_pred)
>>> cm.calcul_matrice()
>>> cm.calcul_kappa()
>>> print(cm)
```

class maplearn.ml.confusion.**Confusion**(*y_sample*, *y_predit*, *fTxt=None*, *fPlot=None*)

Bases: `object`

Computes confusion matrix based on 2 vectors of labels:

1. labels of known samples
2. predicted labels

Args:

- y_sample (vector): vector with known labels
- y_predit (vector): vector with predicted labels
- fTxt (str): path to the text file to write confusion matrix into
- fPlot (str): id. with chart

Attributes:

- y_sample (vector): true labels (ground data)
- y_predit (vector): corresponding predicted labels
- cm (matrix): confusion matrix filled with integer values
- kappa (float): kappa index
- score (float): precision score

TODO:

- y_sample and y_predit should be renamed y_true and y_pred

calcul_matrice()

Computes a confusion matrix and display the result

Returns:

- matrix (integer): confusion matrix
- float: kappa index

export (*fTxt=None*, *fPlot=None*, *title=None*)

Saves confusion matrix in:

- a text file
- a graphic file

Args:

- fTxt (str): path to the output text file
- fPlot (str): path to the output graphic file
- title (str): title of the chart

kappa

Computes kappa index based on 2 vectors

Returns:

- float: kappa index

`maplearn.ml.confusion.`**confusion_cl** (*cm*, *labels*, *os1*, *os2*)

Computes confusion between 2 given classes (expressed in percentage) based on a confusion matrix

Args:

- cm (matrix): confusion matrix
- labels (array): vector of labels
- os1 and os2 (int): codes of th classes

Returns:

- float: confusion percentage between 2 classes

maplearn.ml.distance module

Distance

Computes pairwise distance between 2 matrices, using several metric (euclidean is the default)

Example:

```
>>> import numpy as np
>>> y1 = np.random.random(50)
>>> y2 = np.random.random(50)
>>> dist = Distance(y1, y2)
>>> dist.run()
```

class `maplearn.ml.distance.`**Distance** (*x=None*, *y=None*)

Bases: `object`

Computes pairwise distance between 2 matrices (x and y)

Args:

- x (matrix)
- y (matrix)

compare (*x=None*, *y=None*, *methods=[]*)
Compare pairwise distances got with different metrics

Args:

- x and y (matrices)
- methods (list): list of metrics used to compute pairwise distance. if empty, every available metrics will be compared

dtw (*x=None*, *y=None*)
Dynamic Time-Warping distance

lcs (*x=None*, *y=None*, *eps=10*, *delta=3*)
Distance based on Longest Common Subsequence

run (*x=None*, *y=None*, *meth='euclidean'*)
Distance calculation according to a specified method

Args:

- x (matrix)
- y (matrix)
- meth (str): name of the metric distance to use

Returns: matrix of pairwise distance values

simplex (*x=None*, *y=None*, *sigma=50*)
Simplex distance

maplearn.ml.reduction module

Dimensionnality reduction

The number of dimensions are reduced by selecting some of the features (like in kbest approach) or transforming them (like in PCA…). This reduction is applied to samples and the data to predict in further step.

Several approaches are available, which are listed in the class attribute "ALG_ALGOS".

class maplearn.ml.reduction.**Reduction** (*data=None*, *algorithm=None*, ***kwargs*)
Bases: `maplearn.ml.machine.Machine`

This class reduces the number of dimensions by selecting some of the features or transforming them (like in PCA…). This reduction is applied to samples and the data to predict in further step.

Args:

- data (PackData): dataset to reduced
- algorithm (list): list of algorithm(s) to apply on dataset
- **kwargs: parameters about the reduction (numberof components) or the dataset (like features)

Attributes:

- attributes inherited from Machine classe
- ncomp (int): number of components expected

fit_1 (*algo*)
Fits one reduction algorithm to the dataset

Args:

- algo (str): name of the algorithm to fit

load(*data*)
Loads dataset to reduce

Args:

- data (PackData): dataset to load

predict_1(*algo*)
Applies chosen way of reduction to the dataset

Args: algo (str): name of the algorithm to apply

run(*predict=True*, *ncomp=None*)
Executes reduction of dimensions (fits and applies)

Args:

- **predict (bool): should apply the reduction or just fit the** algorithm ?
- ncomp (int): number of dimensions expected

Returns:

- array: reduced features data
- array: reduced samples features
- list: liste of features

maplearn.ml.regression module

Regression

In statistical modeling, regression analysis is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables.

Regression analysis is supervised and need samples for fitting. The output will be a matrix with float values.

Example:

```
>>> from maplearn.datahandler.loader import Loader
>>> from maplearn.datahandler.packdata import PackData
>>> from maplearn.ml.regression import Regression
>>> loader = Loader('boston')
>>> data = PackData(X=loader.X, Y=loader.Y, data=loader.aData)
>>> reg = Regression(data=data, dirOut=os.path.join('maplearn_path', 'tmp'))
>>> reg.fit_1(self.__algo)
```

class maplearn.ml.regression.**Regression**(*data=None*, *algorithm=None*, ***kwargs*)
Bases: *maplearn.ml.machine.Machine*

Applies regression using 1 or several algorithm(s) onto a specified dataset

Args:

- data (PackData): dataset to play with
- algorithm (list or str): name of the algorithm(s) to use
- **kwargs: more parameters like k-fold

Attributes and properties are inherited from *Machine* class

fit_1(*algo*)
Fits one regression algorithm

Arg:

- algo (str): name of the algorithm to fit

load(*data*)
Loads necessary data for regression, with samples (labels are float values).

Arg:

- data (PackData): data to play with

Returns:

- int: did data load correctly (returns 0) or not (<> 0) ?

TODO:

- checks a few things when loading…

optimize(*algo*)
Optimize parameters of a regression algorithm

Args:

- algo (str): name of the regressor to use

predict_1(*algo*, *proba=False*)
Predicts Y using one regressor (specified by algo)

Args:

- algo (str): key of the regressor to use
- proba (bool): should probabilities (if available) given by algorithm be added to result?

run(*predict=False*)
Applies every regressors specified in 'algorithm' property

Args:

- **predict (bool): should be the regressor only fitted or also used** to predict?

maplearn.ml.machine module

Machine Learning class

Fits and predict result using one or several machine learning algorithm(s).

This is an abstract class that should not be used directly. Use instead one one of the these classes:

- **Classification**: supervised classification
- **Clustering**: unsupervised classification
- **Regression**: regression
- **Reduction**: to reduce dimensions of a dataset

class maplearn.ml.machine.**Machine**(*data=None*, *algorithm=None*, ***kwargs*)
Bases: `object`

Class to apply one or several machine learning algorithm(s) on a given dataset.

Args:

- data (PackData): data to use with machine learning algorithm(s)
- algorithm (list or str): algorithm(s) to use

Attributes:

- algo (str): key code of the currently used algorithm
- result (dataframe): result(s) predicted by algorithm(s)
- proba (dataframe): probabilities produced by some algorithm(s)

Properties:

- algorithm (list): machine learning algorithm(s) to use

`ALL_ALGOS = {}`

`algorithm`
Gets list of algorithm that will be used when running the class

`fit_1`(*algo*)
Fits an algorithm to dataset

`load`(*data*)
Loads necessary data to machine learning algorithm(s)

Args:

- data (PackData): dataset used by machine learning algorithm(s)

`predict_1`(*algo*, *export=False*)
Predict a result using a given algorithm

Args:

- algo (str): key name identifying the algorithm to use
- export (bool): should the algorithm be used to predict results

`run`(*predict=False*)
Apply machine learning task(s) using every specified algorithm(s)

Args:

- predict (boolean): should machine learning algorithm(s) be used to predict results (or just be fitted to samples) ?

4.1.2 maplearn.datahandler package

Data handlers

Interim classes between file(s) and dataset

- **packdata**: creates a dataset with samples and data
- **labels**: labels associated to features (in samples)
- **loader**: loads data from a file or known datasets
- **writer**: writes data into a file
- **signature**: graphs a dataset
- **plotter**: generic class to make charts

Submodules

maplearn.datahandler.packdata module

Machine Learning dataset

A machine learning dataset is classically a table where:

- columns are all **variables** that can be used by machine learning algorithms
- lines correspond to the **individuals**

Variables

The variables fall into two categories:

1. the variables for which you have information: these are the **predictors** (or *features*)
2. the variable to predict, also called *label*

Individuals

- The individuals for whom you know the label are called **samples**.
- The others are just called **data**

class maplearn.datahandler.packdata.**PackData**(*X=None*, *Y=None*, *data=None*, ***kwargs*)

Bases: `object`

PackData: a container for datasets

A PackData contains:

- **samples (Y and X) to fit algorithm(s)**
  - Y: a vector with samples' labels
  - X: a matrix with samples' features
- **data**: 2d matrix with features to use for prediction

PackData checks if samples are compatible with data (same features…) and is compatible with Machine Learning algorithm(s).

Example:

```
>>> import numpy as np
>>> data = np.random.random((10, 5))
>>> x = np.random.random((10, 5))
>>> y = np.random.randint(1, 10, size=10)
>>> ds = PackData(x, y, data)
>>> print(ds)
```

Args:

- X (array): 2d matrix with features of samples
- Y (array): vector with labels of samples
- data (array): 2d matrix with features
- **kwargs: other parameters about dataset (features, na…)

Attributes:

- not_nas: vector with non-NA indexes

X
X (array): 2d matrix with features of samples

Y
Y (array): vector with labels of samples

balance (*seuil=None*)
Balance samples and remove some individuals within the biggest classes.

Args:

- seuil (int): max number of samples inside a class

classes
dict: labels classes and associated number of individuals

data
data (array): 2d matrix with features

features
list: list of features of the dataset

load (*X=None*, *Y=None*, *data=None*, *features=None*)
Loads data to the packdata

Args:

- X (array): 2d matrix with features of samples
- Y (array): vector with labels of samples
- data (array): 2d matrix with features
- features (list): list of features

plot (*prefix='sig'*)
Plots the dataset (signature): * one chart for the whole samples * one chart per samples' class

Args:

- prefix (str): prefix of output files to save charts in

reduit (*meth='lda'*, *ncomp=None*)
Reduces number of dimensions of data and X

Args:

- meth (str): reduction method to apply
- ncomp (int): number of dimensions expected

scale ()
Normalizes data and X matrices

separability (*metric='euclidean'*)
Performs separability analysis between samples

Arg:

- metric (str): name of the distance used

maplearn.datahandler.labels module

Labels

This class handles labels associated to features in samples:

- counts how many samples for each class

class `maplearn.datahandler.labels.`**`Labels`**(*Y*, *codes=None*, *output=None*)

Bases: `object`

Samples labels used in PackData class

Args:

- Y (array): vector with samples' labels
- codes (dict): dictionnary with labels code and associated description

Attributes:

- summary ()
- dct_codes (dict): dictionnary with labels code and associated description

Property:

- Y (array): vector containing labels of samples (codes)

`Y`

Samples (as a vector)

`convert`()

Conversion between codes

`count`()

Summarizes labels of each class (how many samples for each class)

`libelle2code`()

Converts labels' names into corresponding codes

maplearn.datahandler.loader module

Loads data from a file

This class aims to feed a PackData. It gathers data from one or more files or uses known datasets stored in a library

class `maplearn.datahandler.loader.`**`Loader`**(*source*, ***kwargs*)

Bases: `object`

Loads data from a file or a known dataset

Args:

- source (str): path the file to load or name of a dataset ("iris" for example)
- **kwargs: other attributes to drive loading (handles NA, labels…)

Attributes:

- src (dct): informations about the source (type, path…)
- X: samples' features
- Y: samples' labels
- aData:
- matrix: (needed when loading from a raster file)
- features
- nomenclature

Examples:

- Loading data from a know dataset:

```
>>> ldr = Loader('iris')
>>> print(ldr)
>>> print(ldr.X, ldr.Y)
>>> print(ldr.data)
```

- Loading data from a file (here a shapefile):

```
>>> ldr = Loader(os.path.join('maplearn_path', 'datasets',
                              'ex1.xlsx'))
>>> print(ldr)
>>> print(ldr.X, ldr.Y)
```

X
Matrix of values corresponding to samples

Y
Vector of labels describing samples. Values to be predicted by machine learning algorithm

aData
Data to predict

df
Dataframe loaded

features
List of features that contains the dataset

matrix
Data served as a matrix. Needed when loading data from an image

nomenclature
Legends of labels. Dictionnary combining labels codes and their corresponding names

run(***kwargs*)
Gets samples (X with features and Y containing labels)

Args:

- ****kwargs:**
  - features (list): features to load
  - label (str): column with class labels (description)
  - label_id (str): column with labels codes

maplearn.datahandler.writer module

Writes data into a file

This class is to be used with PackData. It puts data into one file (different formats are useable).

class maplearn.datahandler.writer.**Writer**(*path=None*, ***kwargs*)
Bases: `object`

Writes data in a file (different formats available)

Args:

- path (str): path towards the file to write data into
- ****kwargs:**
  - origin (str): path to the original file used as a model

path

run (*data*, *path=None*, *na=None*, *dtype=None*)
Writes data into a file

Args:

- data (pandas dataframe): dataset to write
- path (str): path towards the file to write data into
- na : value used as a code for "NoData"
- dtype (np.dtype): desired data type

maplearn.datahandler.signature module

Signature

This class makes charts about a dataset:

- spectral signature
- temporal signature

Example:

```
>>> from maplearn.datahandler.loader import Loader
>>> from maplearn.datahandler.signature import Signature
>>> ldr = Loader('iris')
>>> sig = Signature()
>>> sig.plot(ldr.X, title='test')
```

class maplearn.datahandler.signature.**Signature** (*data*, *features=None*, *model='boxplot'*, *output=None*)

Bases: object

Makes charts about a dataset:

- one global graph
- one graph per class in samples (if samples are available)

Args:

- data (array or DataFrame): data to plot
- features (list): name of columns
- model (str): how to plot signature (plot or boxplot)
- ouput (str): path to the output directory where will be saved plots

plot (*title='Signature du jeu de donnees'*, *file=None*)
Plots (spectral) signature of data as boxplots or points depending of the number of features

Args:

- title (str): title to add to the plot

- file (str): name of the output file

plot_class (*data_class*, *label=''*, *file=None*)
Plots the signature of one class above the whole dataset

Args:

- data_class (dataframe): data of one class
- label (str): label of the class to plot
- file (str): path to the file to save the chart in

maplearn.datahandler.plotter module

4.1.3 maplearn.filehandler package

File handlers

Read/write data from different kind of files

- **Csv**: tabular data as a text file
- **Excel**: tabular data as a Microsoft Excel file
- **Shapefile**: geographical vector file
- **ImageGeo**: geographical raster file
- *FileHandler*: abstract class to handle files

Submodules

maplearn.filehandler.csv module

CSV file reader and writer

With this class, you can read a text file or write a new one with your own dataset (Pandas Dataframe).

Examples:

- Read an existing file

```
>>> exch = Csv(os.path.join('maplearn path', 'datasets', 'ex1.xlsx'))
>>> exch.read()
>>> print(exch.data)
```

- Write a new Excel File from scratch

```
>>> exc = Excel(None)
>>> out_file = os.path.join('maplearn path', 'tmp', 'scratch.xlsx')
>>> df = pd.DataFrame({'A' : 1,
                       'B' : pd.Timestamp('20130102'),
                       'C' : pd.Series(2,index=list(range(4))),
                       'D' : np.array([3] * 4,dtype='int64')})
exc.write(path=out_file, data=df)
```

class maplearn.filehandler.csv.**Csv** (*path*)
Bases: *maplearn.filehandler.filehandler.FileHandler*

Handler to read and write attributes in a text file. It inherits from the abstract class *FileHandler*.

Attributes:

- FileHandler's attributes

Args:

- path (str): path to the Csv file to open

open_()
Opens the CSV file specified in dsn['path']

read()
Reads the content of the CSV file

write(*path=None*, *data=None*, *overwrite=True*, ***kwargs*)
Write specified attributes in a text File

Args:

- path (str): path to the Excel to create and write
- data (pandas DataFrame): dataset to write in the Excel file
- overwrite (bool): should the output Excel file be overwritten ?

maplearn.filehandler.excel module

Excel file reader and writer

With this class, you can read an Excel file or write a new one with your own dataset (Pandas Dataframe).

Examples:

- Read an existing Excel file

```
>>> exch = Excel(os.path.join('maplearn path', 'datasets', 'ex1.xlsx'))
>>> exch.read()
>>> print(exch.data)
```

- Write a new Excel File from scratch

```
>>> exc = Excel(None)
>>> out_file = os.path.join('maplearn path', 'tmp', 'scratch.xlsx')
>>> df = pd.DataFrame({'A' : 1,
                       'B' : pd.Timestamp('20130102'),
                       'C' : pd.Series(2,index=list(range(4))),
                       'D' : np.array([3] * 4,dtype='int64')})
exc.write(path=out_file, data=df)
```

class maplearn.filehandler.excel.**Excel**(*path*, *sheet=None*)
Bases: *maplearn.filehandler.filehandler.FileHandler*

Handler to read and write attributes in an Excel file. It inherits from the abstract class *FileHandler*.

Attributes:

- FileHandler's attributes

Args:

- path (str): path to the Excel file to open
- sheet (str): name of the sheet to open

open_()
Opens the Excel file specified in dsn['path']

read()
Reads the content of the opened Excel file

write(*path=None*, *data=None*, *overwrite=True*, ***kwargs*)
Write specified attributes in an Excel File

Args:

- path (str): path to the Excel to create and write
- data (pandas DataFrame): dataset to write in the Excel file
- overwrite (bool): should the output Excel file be overwritten ?

maplearn.filehandler.imagegeo module

Geographic Images (raster)

This class handles raster data with geographic dimension (projection system, bounding box expressed with coordinates).

A raster data relies on:

- a matrix of pixels (data)
- geographic data (where to put this matrix on earth)

Example:

```
>>> img = ImageGeo(os.path.join('maplearn_path', 'datasets',
                                'landsat_rennes.tif'))
>>> img.read()
>>> print(img.data)
```

class maplearn.filehandler.imagegeo.**ImageGeo**(*path=None*, *fmt='GTiff'*)
Bases: *maplearn.filehandler.filehandler.FileHandler*

Handler of geographical rasters

Args:

- path (str): path to the raster file to read
- **fmt (str): format of the raster file ('GTiff'... see GDAL** documentation)

Attributes:

- Several attributes are inherited from *FileHandler* class

data
The dataset read from a file or to write in a file

data_2_img(*data*, *overwrite=False*, *na=None*)
Transforms a data set (dataframe) into a matrix in order to export it as an image (inverse operation to __img_2_data () method).

Args:

- data (dataframe): the dataset to transform
- overwrite (bool): should the result *data* property ?

Returns: matrix: transformed dataset

img_2_data()

Transforms the data set in order to make it easier to handle in following steps.

Converts the data set (matrix) into to 2 dimensions dataframes (where 1 line = 1 individual and 1 column = 1 feature)

Returns: dataframe: transformed dataset (2 dimensions)

init_data(*dims*, *dtype=None*)

Creates an empty matrix with specified dimension

Args:

- dims (list): dimensions of the image to create
- dtype (str): numerical type of pixels

open_()

Opens the Geographical Image to get information about projection system…

pixel2xy(*j*, *i*)

Computes the geographic coordinate (X,Y) corresponding to the specified position in an image (column, row)

It does the inverse calculation of xy2pixel, and uses a gdal geomatrix

Source: http://geospatialpython.com/2011/02/clip-raster-using-shapefile.html

Args:

- j (int): column position
- i (int): row position

Returns: list: geographical coordinate of the pixel (lon and lat)

read(*dtype=None*)

Reads the raster file and puts the matrix in *data* property

Args:

- dtype (str): type values stored in pixels (int, float…)

set_geo(*transf=None*, *prj=None*)

Sets geographical dimension of a raster:

- the projection system
- the bounding box, whose coordinates are compatible with the given

projection system

Args:

- prj (str): projection system
- transf (list): affine function to translate an image

Definition of 'transf' (to translate an image to the right place): [0] = top left x (x Origin) [1] = w-e pixel resolution (pixel Width) [2] = rotation, 0 if image is "north up" [3] = top left y (y Origin) [4] = rotation, 0 if image is "north up" [5] = n-s pixel resolution (pixel Height)

TODO []

- Check compatibility between bounding box and image size

- Adds EPSG code corresponding to *prj* in __geo

write (*path=None*, *data=None*, *overwrite=True*, ***kwargs*)
Writes a data in a raster file

Args:

- path (str): raster file to write data into
- data (array): data to write
- overwrite (bool): should the raster file be overwritten?

xy2pixel (*lon*, *lat*)
Computes the position in an image (column, row), given a geographic coordinate

Uses a gdal geomatrix (gdal.GetGeoTransform()) to calculate the pixel location of a geospatial coordinate (http://geospatialpython.com/2011/02/clip-raster-using-shapefile.html)

Args:

- lon (float): longitude (X)
- lat (float): latitude (Y)

Returns: list with the position in the image (column, row)

maplearn.filehandler.shapefile module

Shapefile reader and writer

With this class, you can read a shapefile or more precisely get attributes from a shapefile. You can also write a new shapefile using geometry from an original shapefile and adding the attributes you want.

Examples:

```
>>> shp = Shapefile(os.path.join('maplearn path', 'datasets',
                                 'echantillon.shp'))
>>> shp.read()
>>> print(shp.data)
```

TODO: Guess character encoding in shapefile's attributes

class maplearn.filehandler.shapefile.**Shapefile** (*path*)
Bases: *maplearn.filehandler.filehandler.FileHandler*

Handler to read and write attributes in a shapefile. It inherits from the abstract class *FileHandler*.

Attributes:

- FileHandler's attributes
- str_type (str): kind of geometry (polygon, point…)
- lst_flds (list): list of fields in dataset

open_ ()
Opens the shapefile and put in __ds attribute, so attributes can then be read

read ()
Reads attributes associated to entities in the shapefile

Returns: Pandas Dataframe: data (attributes) available in the shapefile

write (*path=None*, *data=None*, *overwrite=True*, ***kwargs*)
Write attributes (and only attributes) in a new shapefile, using geometries of an original shapefile.

Args:

- path (str): path to the shapefile to create and write
- data (pandas DataFrame): dataset to write in the shapefile
- overwrite (bool): should the output shapefile be overwritten ?

maplearn.filehandler.filehandler module

Handling files (abstract class)

This class is to handle generic files. FileHandler is not supposed to be called directly. Use rather one of the classes that inherits from it (ImageGeo, Excel, Shapefile…).

class maplearn.filehandler.filehandler.**FileHandler** (*path=None*, ***kwargs*)
Bases: object

Reads data from a generic file or write data into it.

Attributes:

- **_drv (object): driver to communicate with a file (necessary for some** formats)
- **_data (numpy array or pandas dataframe): dataset got from a file or** to write into it. See *data* property.
- opened (bool): is the file opened or not ?

Args:

- path (str): path the file to read data from
- **kwargs: additional settings to specify how to load data from file

data
The dataset read from a file or to write in a file

dsn
Dictionnary containing informations about data source. For example, *path* contains the path of the file to get data from. Other items can exist, which are specific to the data type (raster, vector or tabular, geographical or not…)

open_()
Opens a file prior to write in it

read()
Reads the dataset from the file mentioned during initialization

write (*path=None*, *data=None*, *overwrite=True*, ***kwargs*)
Writes data in a file

Args:

- path (str): path to the file to write into
- data (numpy array or pandas dataframe): the data to write
- overwrite (bool): should the file be overwritten if it exists ?

4.1.4 maplearn.app package

Application modules

Modules necessary to Mapping Learning when it is used as an application :

- **config**: configuration
- **main**: the main class that uses other classes to process your data
- **reporting**: a module to format results in an html output

Submodules

maplearn.app.config module

Mapping Learning Configuration

The configuration contains 3 mandatory parts :

- **Inputs/outputs** [io]: which file(s) and how to work with them, where to save results . . .)
- **Preprocessing** [preprocess]: what to do before training the algorithm(s)?
- **Processing** [process]: which kind of processing? Regression, supervised or unsupervised classification (clustering)? Which algorithm(s)?

An optional part, [metadata] permits to include some information about your work in the output report.

Input/Output [io]

Mapping Learning allows you to work on many formats (csv, excel, tiff, shp. . .), but also in many ways. You can choose:

- to use *samples*, a dataset without knowledge (*data*), or both
- the variable(s) (*features*) to use
- to use directly the values of the variable to be predicted (*label*) or some codes corresponding to these values (*label_id*)

NB: don't forget to check where will be saved your results (*output*).

```
[io]
# [txt] path to the samples used to train algorithm(s)
samples=
# [optional:txt] name of the column with class ID (as numbers)
label_id=
# [optional:txt] name of column with class description (described as
#                 strings)
label=
# [optional:txt] list of features to use (separated with ',')
features=
# [optional:txt] path to the dataset to predict with
data=
# [txt] path to the output folder (which will contain the results)
output=
```

Preprocessing [preprocess]

Maplearn is not intended to perform all the necessary manipulations to your dataset to make it usable by machine learning. Nevertheless, some preprocessing tools are available, that will modify the values of the data (*scale*), the features (*reduce* and *ncomp*), the samples (*balance*). Finally, *separability* permits to estimate the chances of getting good results with your samples.

NB: check `maplearn.datahandler.packdata` *to see how dataset should be structured for machine learning use.*

```
[preprocess]
# [optional:boolean] center/reduce? [true/false]
scale=
# [optional:boolean] make number of individuals about similar between
#                    classes? [true/false]
balance=
# [optional:txt] name of the method to reduce dimensions of the dataset
#                [one between pca, lda, kbest, rfe, kernel_pca]
reduce=
# [optional:number] number of expected dimensions after reduction
ncomp=
# [optional:boolean] check separability between classes? [true/false]
separability=
```

Processing [process]

Note: Here we are finally at the most interesting part: what do you want to predict? Continuous numbers (temperature, . . .) or discrete values (social class, land use. . .)? In any case, *maplearn* will allow you to use lots of algorithms, and will help you obtain the most accurate predictions.

This *process* part will allow you to define:

- type of prediction (*type*)
- algorithm(s) to apply (*algorithm*)
- if you want to try to improve the accuracy (*optimize*)
- how to use your samples (*kfold*)
- should we predict?

Note: This question may seem absurd but it is prudent not to predict results immediately. If your dataset is large and you do not know exactly which algorithm(s) are relevant, then you can focus first on the statistical results.

```
[process]
# [txt] which kind of process? [classification, clustering ou regression]
type=classification
# [optional:txt] how to measure distance?
distance=euclidean
# [optional:txt] algorithm(s) to use (if several, separated with ',')
algorithm=
# [optional:number] how many folds to use in cross-validation?
```

(continues on next page)

(continued from previous page)

```
kfold=
# [optional:boolean] look for best hyperparameters? [true/false]
optimize=
# [optional:boolean] should predict results (exports)? [true/false]
predict=
```

Metadata [metadata]

```
[metadata]
# [optional:txt] give a title to your work
title =
# [optional:txt] describe your work (please avoid special characters)
description =
# [optional:txt] name of the author(s)
author =
```

class maplearn.app.config.**Config**(*file_config*)

Bases: `object`

This class is the medium between a configuration file and the applicaton. It is able to load and check a configuration file (Yaml) and rewrite a new configuration file (that can be re-used by Mappling Learning later).

Config checks that application will be able to run properly using a given configuration:

- input files exists?
- value of parameters belong to expected type
- …

Args: config_file (str) : path to a configuration file

The class attributes described below reflects the sections in configuration file.

Properties:

- io (dict): input/output. path to samples, dataset files and output. list of features to use…
- codes (dict): label codes and corresponding names
- preprocess (dict) : which preprocessing step(s) to apply
- process (dict) : which processes to apply (list of algorihms…)

check()

Check that parameters stored in attributes are correct

Returns: int : number of issues detected

codes

Dictionnary describing label codes and the name of classes

io

Input/Output property

preprocess

Dictionnary of preprocess parameters

process

Dictionnary of process parameters

read()
Load parameters from configuration file and put them in corresponding class attributes

Returns: int : number of issues got when reading the file

write(*fichier=None*)
Write a new configuration file feeded by class attributes content.

Args: fichier (str) : path to configuration file to write

maplearn.app.config.**splitter**(*text*)
Splits a character string based on several separators and remove useless empty characters.

Args: text (str) : character string to split

Returns: list: list of stripped character strings, None elsewhere

maplearn.app.main module

Main class *(one class to rule the others)*

This class is the engine powering Mapping Learning. It uses every other classes to load data, apply preprocesses and finally process the dataset, using one or several algorithm(s). The results are synthetized and compared.

The class can apply classification, clustering and regression processes.

Examples:

```
>>> from maplearn.app.main import Main
```

- Apply 2 different classifications on a known dataset

```
>>> ben = Main('.', type='classification', algorithm=['knn', 'lda'])
>>> ben.load('iris')
>>> ben.preprocess()
>>> ben.process(True)
```

- Apply every available clustering algorithm(s) on the same dataset

```
>>> ben = Main('.', type='clustering')
>>> ben.load('iris')
>>> ben.preprocess()
>>> ben.process(False) # do not predict results
```

- Apply regression on another known dataset

```
>>> ben = Main('.', type='regression', algorithm='lm')
>>> ben.load('boston')
>>> ben.preprocess()
>>> ben.process(False) # do not predict results
```

class maplearn.app.main.**Main**(*dossier*, ***kwargs*)
Bases: object

Realizes every steps from loading dataset to processing

Args:

- dossier (str): output path where will be stored every results
- **kwargs: parameters data and processing to apply on it

Attributes:

- dataset (PackData): dataset to play with

load (*source*, ***kwargs*)
Loads samples (labels with associated features) used for training algorithm(s)

Args:

- source (str): file to load or name of an available datasets
- **kwargs: parameters to specify how to use datasets (which features to use…)

load_data (*source*, *label_id=None*, *label=None*, *features=None*)
Load dataset to predict with previously trained algorithm(s)

Args:

- source (str): path to load or name of an available dataset
- label_id (optional[str]): column used to identify labels
- label (optional[str]): column with labels' names
- features (list): columns to use as features. Every available columns are used if None

preprocess (***kwargs*)
Apply preprocessings tasks asked by user and give the dataset to the Machine Learning processor

Args: **kwargs: available preprocessing tasks (scaling dataset, reducing number of features…)

process (*predict=False*, *optimize=False*, *proba=True*)
Apply algorithm(s) to dataset

Args:

- predict (bool): should the algorithm(s) be only fitted on samples or also predict results ?
- optimize (bool): should maplearn look for best hyperparameters for the algorithm(s) ?
- proba (bool): should maplearn try to get probabilities associated to predictions ?

maplearn.app.reporting module

CHAPTER 5

Contribute

Note: open source

Mapping Learning is a **free, open-source application**, distributed under the lGPL v3 license. Feel free to contribute !

You do not have to know how to code, you can contribute by:

- Using it
- Tracing the issues or proposing improvements on the bug tracker
- Improving documentation

Feel free to contact me (alban.thomas@univ-rennes2.fr).

5.1 Documentation

The documentation is built from the source code (using sphinx) and is available in PDF, epub and HTML. Up-to-date documentation is available at https://maplearn.readthedocs.io/en/latest/ .

5.2 Source code

Maplearn is written in **Python**. The source code is available at https://bitbucket.org/thomas_a/maplearn/src/master/. You can simply download a copy from this link but using git you can easily get updates.

```
git clone https://bitbucket.org/thomas_a/maplearn.git
# then, to get updates
git pull
```

5.3 Philosophy

Wondering what you can expect from *Mapping Learning* software? The few points below give the "philosophy" of the software:

- Mapping Learning should be able to be **used as you wish** (*freedom*)

Mapping Learning is usable whatever your environment (Windows, Linux or Mac) and the way you want (graphical or online interface of commands, or even write a Python script).

- Mapping Learning should help you to **learn machine learning** (*knowledge base*)

We learn from our mistakes. Mapping Learning will not prevent you from making meaningless predictions but must help you to realize you are doing it wrong (through advice, warnings . . .).

- Mapping Learning should help you to **understand your data** (*visualization*)

Data visualization really matters. Mapping Learning will integrate all possible means (not just graphics) to better understand your data and results.

- Mapping Learning should be **useful to everyone** (*openness*)

Mapping Learning was initially dedicated to remote sensing, but the applications of machine learning are much larger. *Maplearn* allows you to use your data whether they are geographic or not (text files, Excel, and more to come).

- Mapping Learning should be **up to date**

Machine Learning evolves quickly and Mapping Learning will try to give you access to the latest algorithms.

- Mapping Learning is about machine learning and **only machine learning**

Mapping Learning is not a GIS or data manipulation software (ETL). Very good software already exists.

5.4 Thanks

Rennes 2 University

LETG - UMR6554

AFPy - Association Francophone Python

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index