

Scikit- Splearn toolbox

Spectral learning compatible with scikit-learn

Dominique Benielli

Labex Archimède

4 octobre 2016



Table des matières

1	Introduction	2
1.1	Authors	2
2	Installation	2
2.1	Introduction	2
2.2	Installation	3
2.3	Package splearn	3
2.4	tests and test coverage	3
3	Data format and load	3
3.1	File format	4
3.2	Data input format	4
3.3	Loading an example datasets	5
3.4	Class Splearn_array data format, inherit from numpy ndarray	5
4	The estimator model : Spectral	5
4.1	Spectral init and parameter setting	5
4.2	Spectral model Fit	6
4.3	Spectral model Predict and predict_proba	7
4.3.1	The smooth_method option 'trigram'	8

4.4	Spectral model loss and score methods	9
4.5	Loss and normalize	9
4.6	Score and scoring 'perplexity'	9
5	Scikit-learn compatibility	10
5.1	check_estimator	10
5.2	cross_validation	10
5.3	GridSearch	10

1 Introduction

The goal of this toolbox is to implement a python version of Spectral learning, compatible with the well-known toolbox for statistical machine learning called Scikit-learn. This will allow the use of the tuning functions of Scikit-learn, like the cross-validation ones and the grid search. Giving the large public using Scikit-learn, we hope that will convince statistical machine learner to get interested into spectral learning. A python toolbox SP2Learning for spectral learning is already in production, this corresponds to algorithms developed in the context of the Sequence Prediction Challenge (SPiCe) but no compatibility with Scikit-learn is allowed. In this toolbox the adaptaion of scikit-learn mainly provided, with a implementation of a predictor with at least 3 methods : fit, predict, and loss. The data format is has also been a important source of modifications.

1.1 Authors

This projetct has bee developped by :

- François Denis
- Rémi Eyraud
- Denis Arrivault
- Dominique Benielli

2 Installation

2.1 Introduction

The original scikit-splearn Toolbox is developed in Python at *LabEx Archimède* <http://labex-archimede.univ-amu.fr> , as a *LIF* <http://www.lif.univ-mrs.fr> and *I2M* project, Aix-Marseille Université.

This package, as well as the splearn toolbox, is Free software, released under the BSD License.

The latest version of ****scikit-splearn**** can be downloaded from the following PyPI page <https://pypi.python.org/pypi/scikit-splearn>.

The documentation is available as a pythonhosted site <http://pythonhosted.org/scikit-splearn/>

The development is done in this gitlab project <https://gitlab.lif.univ-mrs.fr/dev/scikit-splearn> which provides the git repository managing the source code and where issues can be reported.

2.2 Installation

The package can install directly by :

```
pip install scikit-splearn
```

or after doanloading by :

```
pip install -e .
```

2.3 Package splearn

- splearn
 - splearn/datasets
 - * splearn/datasets/base.py
 - * splearn/datasets/data_sample.py
 - splearn/automaton.py
 - splearn/hankel.py
 - splearn/learning.py

2.4 tests and test coverage

```
(env3_scikit)dominique@ARCHIMEDE:~/projets/scikit-splearn$ nosetests tests
```

Name	Stmts	Miss	Cover	Missing
splearn.py	5	0	100%	
splearn/automaton.py	289	8	97%	136, 142, 147-149, 154, 247, 597
splearn/datasets.py	2	0	100%	
splearn/datasets/base.py	53	5	91%	85, 106, 154-156
splearn/datasets/data_sample.py	52	1	98%	299
splearn/hankel.py	108	10	91%	81, 88, 95, 176-181, 194-195, 200-201
splearn/spectral.py	328	33	90%	189, 213-214, 219-220, 259-273, 276-284, 332, 340, 343-344, 352
TOTAL	837	57	93%	

```
Ran 49 tests in 118.206s
```

```
OK
```

3 Data format and load

The main point for the adaptation to scikit-learn is related to the input data format.

3.1 File format

Indeed the algorithms of learning Weighted Automata (WA) use loaded data with the following 'SPiCe' or 'Pautomac' type format :

```
20000 4
7 3 0 3 1 3 1 3
2 3 3
5 3 2 0 3 0
4 3 0 1 0
7 3 3 0 0 1 3 0
2 3 1
4 3 0 3 3
5 3 0 3 1 3
23 3 1 3 0 2 0 3 1 2 0 3 0 2 0 1 0 3 0 1 0 3 3 3
....
```

where the first line contains the number of samples, and the number of letters of the alphabet , the others lines contains samples itself with in first position the length of the sample. This format is not compatible with the 2D array input format expected by scikit-learn.

3.2 Data input format

The loaded files are formatted and reshaped to be include in a 2D array, shape (n_samples, n_features). The empty spaces are replaced by -1, and so the input file follows the following format :

```
3 0 3 1 3 1 3 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 3 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 2 0 3 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 0 1 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 3 0 0 1 3 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 0 3 3 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 0 3 1 3 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 1 3 0 2 0 3 1 2 0 3 0 2 0 1 0 3 0 1 0 3 3 3
....
```

This data are analyzed by the algorithm and tranformed into dictionaries.

3.3 Loading an example datasets

Scikit-Splearn is implemented with standard datasets from a sets of strings using defined alphabet letters.

```
>>> from splearn.datasets.base import load_data_sample
>>> from splearn.tests.datasets.get_dataset_path \
    import get_dataset_path
>>> train_file = '3.pautomac_light.train'
>>> # or train_file = '4.spice.train'
>>> data = load_data_sample(adr=get_dataset_path(train_file))
>>> data.nbL
4
>>> data.nbEx
5000
>>> data.data
Splearn_array([[ 3.,  0.,  3., ..., -1., -1., -1.],
 [ 3.,  3., -1., ..., -1., -1., -1.],
 [ 3.,  2.,  0., ..., -1., -1., -1.],
 ...,
 [ 3.,  1.,  3., ..., -1., -1., -1.],
 [ 3.,  0.,  3., ..., -1., -1., -1.],
 [ 3.,  3.,  1., ..., -1., -1., -1.]])
>>>
```

while »> denotes the Python interpreter prompt. The dataset is a dictionary-like object that holds all the data and some metadata about the data, with different fields like nbL the number of letters, nbEx the number of samples, and data for the samples themselves in a plain 2D array.

3.4 Class Splearn_array data format, inherit from numpy ndarray

The loaded data are formatted in data field wich is Splearn_array, this format is inherit from numpy nd array, and contains as main object a 2D array [n_samples, n_features]. Splearn_array encapsulates also other variables : numbers nbL (numbers of letters and nbEx, numbers of samples defined above, sample, pref, suff and fact dictionnaires.

4 The estimator model : Spectral

4.1 Spectral init and parameter setting

The estimator model is named Spectral, it can be found in the splearn package and it's works as scikit estimator. The spectral estimator accepts as input differents parameters :

- *partial* boolean
 - *False* : the calculation is performed with the full length of element
 - *True* : the calculations are performed with a limited length of element (see parameter *lrows*, *lcolumns*)
- *lrows* number or list of rows (int or tuple) a list of strings or an integer indicating the max length of elements to consider if *partial=True* otherwise, based on *self.pref* if *version="classic"* or *"prefix"*, *self.fact* otherwise
- *lcolumns* number or list of columns (int or tuple) a list of strings or an integer indicating the max length of elements to consider if *partial=True* otherwise, based on *self.suff* if *version="classic"* or *"suffix"*, *self.fact* otherwise
- *sparse* (boolean) *True* for sparse calculation with a sparse Hankel, *False*
- *smooth_method* (string default value = "none") indicate the method of smoothing
 - 'trigram' the 3-Gram trigram dict is computed and used by the predict function, in this case the trigram probability is used instead of Spectral probability in negative case
 - 'none' or something else no smooth method is used in predict function.
- *rank* (int) : the ranking number
- *version* (string) : (by default = "classic") version name, version names can be ('classic', 'prefix', 'suffix', 'factor')
- *mode_quiet* : (by default value = False) if True no output information

usage in the following commands :

```
>>> from splearn.spectral import Spectral
>>> sp = Spectral(rank=5, lcolumns=6, lrows=6)
>>> sp.get_params()
{'lcolumns': 6, 'partial': True, 'rank': 5,
'sparse': True, 'version': 'classic', 'mode_quiet': False,
'smooth_method': 'none', 'lrows': 6}
>>> sp.set_params(smooth_method='trigram')
Spectral(lcolumns=6, lrows=6, mode_quiet=False, partial=True, rank=5,
smooth_method='trigram', sparse=True, version='classic')
```

4.2 Spectral model Fit

```

>>> sp.fit(data.data)
Start Hankel matrix computation
End of Hankel matrix computation
Start Building Automaton from Hankel matrix
End of Automaton computation
Spectral(lcolumns=6, lrows=6, partial=True, rank=5, smooth_method='none',
        sparse=True, version='classic')
>>> sp.set_params(mode_quiet=True)
Spectral(lcolumns=6, lrows=6, mode_quiet=True, partial=True, rank=5,
        smooth_method='none', sparse=True, version='classic')
>>>

```

The fit method calculs and instantiates a automaton, it is accessible by the following commands :

```

>>> sp.Automaton.initial
array([-0.00049249, 0.00304676, -0.04405996, -0.10765322, -0.08660063])
>>> sp.Automaton.final
array([ 0.07498848, -0.02407125, -0.44675369, 0.62799252, -0.55444987])
>>> sp.Automaton.transitions
[array([[ 0.04323057, -0.24063466, 0.35033337, -0.2795733, -0.21986786],
        [ 0.06896795, -0.30065877, 0.20745919, -0.15087461, -0.56014468],
        [ 0.02885238, -0.13799954, 0.18365036, -0.2099106, -0.14498752],
        [ 0.00612321, -0.02332236, -0.06608033, 0.10764515, -0.15109704],
        [-0.02015367, 0.08988955, -0.00553278, -0.03131386, 0.24333355])],
array([[ 0.07994532, 0.09135396, -0.30878986, 0.28080686, 0.20376293],
        [-0.09865201, -0.08073288, 0.2593956, -0.12114366, -0.11119892],
        [-0.06136638, -0.06247307, 0.12020222, 0.0025085, -0.15696747],
        [-0.00246864, -0.00898858, -0.0005062, -0.00855869, -0.05379994],
        [ 0.03049939, 0.03972289, -0.04998491, 0.00356841, 0.14193265])],
array([[ -0.06691411, -0.11421271, 0.37924968, -0.21186808, -0.24443885],
        [ 0.11687672, 0.15004827, -0.13393244, -0.00919187, 0.35004331],
        [ 0.01205358, 0.01938594, 0.04148993, -0.03541727, 0.02317928],
        [ 0.00729786, 0.00555287, -0.02248987, 0.0361802, -0.03854873],
        [-0.01069779, -0.01068881, -0.00056362, -0.025575, 0.0498595 ]]),
array([[ 0.07153358, -0.01601894, 0.07362502, -0.10483001, 0.02441764],
        [-0.05174562, -0.08994, 0.27684684, -0.23754248, 0.07353445],
        [-0.00746697, -0.04863056, -0.62933143, 0.46853876, 0.09339057],
        [-0.00695291, -0.05607908, -0.36582793, -0.01322074, 0.649387 ],
        [ 0.00240461, -0.02149357, 0.0911477, -0.38472518, 0.66182296]])]

```

4.3 Spectral model Predict and predict_proba

The predict method returns the probability of the model as a 1d array on nbEx samples, corresponding to the repective 2D X input (nb_sample, n_features). The method predict_proba gives also the same results as predict method.

```
>>> sp.predict(data.data)
array([ 4.38961058e-04, 1.10616861e-01, 1.35569353e-03, ...,
        4.66041996e-06, 4.68177275e-02, 5.24287604e-20])
>>> sp.predict_proba(data.data)
array([ 4.38961058e-04, 1.10616861e-01, 1.35569353e-03, ...,
        4.66041996e-06, 4.68177275e-02, 5.24287604e-20])
```

4.3.1 The `smooth_method` option 'trigram'

In the cas of `smooth_method='trigram'` option, the trigram dictionary is calculated, it is accessible as a attribut 'trigram' of Spectral . The attribute `trigram_index` gives the positions where the trigram dictionary is used instead of spectral prediction. The `nb_trigram` method gives the numbers of item affected by the smoothing. Values of 'trigram' dictionary are used by the predict method where the probability estimated by the Spectral model is negative.

```
>>> sp.set_params(smooth_method='trigram')
Spectral(lcolumns=6, lrows=6, partial=True, rank=5,
        smooth_method='trigram',
        sparse=True, version='classic')
>>> sp.fit(data.data)
Start Hankel matrix computation
End of Hankel matrix computation
Start Building Automaton from Hankel matrix
End of Automaton computation
Spectral(lcolumns=6, lrows=6, partial=True, rank=5,
        smooth_method='trigram',
        sparse=True, version='classic')
>>> sp.predict(data.data)
array([ 4.38961058e-04, 1.10616861e-01, 1.35569353e-03, ...,
        4.66041996e-06, 4.68177275e-02, 5.24287604e-20])
>>> sp.trigram
{(0, 1): {0: 233, 1: 309, 2: 357, 3: 742, -2: 141, -1: 1782},
 (1, 2): {0: 366, 1: 309, 2: 126, 3: 291, -2: 120, -1: 1212},
 (3, 2): {0: 292, 1: 88, 2: 45, 3: 107, -1: 667, -2: 135},
 (0, 0): {0: 177, 1: 258, 2: 80, 3: 225, -1: 984, -2: 244},
 (3, 3): {0: 1450, 1: 895, 2: 306, 3: 664, -2: 1000, -1: 4315},
 (2, 0): {0: 126, 1: 179, 2: 120, 3: 626, -2: 105, -1: 1156},
 (3, 0): {0: 571, 1: 1083, 2: 1065, 3: 2043, -1: 5517, -2: 755},
 (3, 1): {0: 359, 1: 457, 2: 567, 3: 1465, -2: 135, -1: 2983},
 (2, 2): {0: 148, 1: 21, 2: 22, 3: 73, -1: 292, -2: 28},
 (2, 1): {0: 88, 1: 232, 2: 96, 3: 202, -2: 59, -1: 677},
 (1, 1): {0: 124, 1: 605, 2: 192, 3: 562, -1: 1603, -2: 120},
 (-1, 3): {0: 1901, 1: 834, 2: 123, 3: 1872, -2: 270, -1: 5000},
```



```
(1, 3): {0: 735, 1: 339, 2: 136, 3: 1167, -1: 2971, -2: 594},
(2, 3): {0: 313, 1: 202, 2: 56, 3: 82, -1: 917, -2: 264},
(-1, -1): {3: 5000, -1: 5000},
(1, 0): {0: 110, 1: 262, 2: 127, 3: 150, -1: 804, -2: 155},
(0, 3): {0: 1118, 1: 713, 2: 46, 3: 530, -2: 637, -1: 3044},
(0, 2): {0: 350, 1: 259, 2: 99, 3: 446, -2: 238, -1: 1392}}
>>> sp.trigram_index
array([False, False, False, ..., False, False, False], dtype=bool)
>>> sp.nb_trigram()
220
```

4.4 Spectral model loss and score methods

The loss method returns the opposite of the mean/sum of the probability logarithm in the case of non-supervised learning, and the least squares in the supervised case. The score method, return the opposite of loss except in the case of supervised learning with the scoring option equal to 'perplexity'. Note that the default scoring is 'perplexity'

4.5 Loss and normalize

The boolean parameter *normalize* of the loss method (default value is True), the calculation of loss method is normalized, and not otherwise.

```
>>> sp.loss(data.data)
10.530029936056017
>>> sp.loss(data.data, normalize=False)
52650.149680280083

>>> p = sp.predict(data.data)
>>> sp.loss(data.data, p)
0.0
```

4.6 Score and scoring 'perplexity'

The score method is always normalized and return -loss method in the case of not supervised learning and also where no scoring is asked in any case.

```
>>> sp.score(data.data)
-10.530029936056017
>>> sp.score(data.data, scoring='perplexity')
-10.530029936056017
>>> sp.score(data.data, scoring='none')
-10.530029936056017
```

```
>>> sp.score(data.data, p)
1556.1778597742712
>>> sp.score(data.data, p, scoring='perplexity')
1556.1778597742712
>>> sp.score(data.data, p, scoring='none')
-0.0
```

5 Scikit-learn compatibility

5.1 check_estimator

The compatibility of splearn is validated by the check_estimator

```
>>> from sklearn.utils.estimator_checks import check_estimator
>>> check_estimator(Spectral)
Process finished with exit code 0
```

5.2 cross_validation

```
>>> sp = Spectral()
>>> sp.set_params(partial=True, version='classic', lcolumns= 6, lrows= 6, rank=5, smooth_method='trigram')
>>> from sklearn import cross_validation
>>> X_train, X_test = cross_validation.train_test_split( data.data, test_size=0.4, random_state=0)
>>> sp.fit(X_train)
>>> scores = cross_validation.cross_val_score(sp, data.data, cv=5)
>>> print(scores)
[-10.61271014 -10.71261867 -10.36931802 -11.01046119 -11.10633528]
>>> from sklearn.cross_validation import cross_val_predict
>>> predicted = cross_val_predict(sp, data.data, cv=10)
>>> print(predicted)
[ 3.63092273e-04  1.01803674e-01  1.27485779e-03 ...,  1.21643536e-06
 4.06223489e-02  2.93321679e-20]
```

The score method needs smooth_method='trigram' option if predict values are negative

5.3 GridSearch

```
>>> from sklearn.grid_search import GridSearchCV
>>> tuned_parameters = [{'version': ['classic'], 'lcolumns': [5, 6, 7],
    'lrows': [5, 6, 7]},
    {'version': ['factor'], 'lcolumns': [5, 6, 7],
    'lrows': [5, 6, 7]}]
>>> clf = GridSearchCV(sp, tuned_parameters, cv=5)
>>> clf.fit(X_train)
>>> print("Best parameters set found on development set: "), print(clf.best_params_)
Best parameters set found on development set:
{'lrows': 7, 'version': 'classic', 'lcolumns': 7}
>>> print("Grid scores on development set:")
Grid scores on development set:
>>> for params, mean_score, scores in clf.grid_scores_:
    print("%0.3f(+/-%0.03f) for %r"
        % (mean_score, scores.std() * 2, params))
Grid scores on development set:
-11.201 (+/-1.325) for {'lrows': 5, 'lcolumns': 5, 'version': 'classic'}
-11.210 (+/-1.347) for {'lrows': 6, 'lcolumns': 5, 'version': 'classic'}
-11.216 (+/-1.328) for {'lrows': 7, 'lcolumns': 5, 'version': 'classic'}
-11.202 (+/-1.326) for {'lrows': 5, 'lcolumns': 6, 'version': 'classic'}
-11.224 (+/-1.322) for {'lrows': 6, 'lcolumns': 6, 'version': 'classic'}
-11.224 (+/-1.321) for {'lrows': 7, 'lcolumns': 6, 'version': 'classic'}
-11.194 (+/-1.329) for {'lrows': 5, 'lcolumns': 7, 'version': 'classic'}
-11.221 (+/-1.328) for {'lrows': 6, 'lcolumns': 7, 'version': 'classic'}
-11.231 (+/-1.323) for {'lrows': 7, 'lcolumns': 7, 'version': 'classic'}
-10.687 (+/-1.108) for {'lrows': 5, 'lcolumns': 5, 'version': 'factor'}
-10.691 (+/-1.109) for {'lrows': 6, 'lcolumns': 5, 'version': 'factor'}
-10.695 (+/-1.110) for {'lrows': 7, 'lcolumns': 5, 'version': 'factor'}
-10.687 (+/-1.110) for {'lrows': 5, 'lcolumns': 6, 'version': 'factor'}
```

```
-10.695 (+/-1.107) for {'rows': 6, 'columns': 6, 'version': 'factor'}  
-10.695 (+/-1.109) for {'rows': 7, 'columns': 6, 'version': 'factor'}  
-10.689 (+/-1.110) for {'rows': 5, 'columns': 7, 'version': 'factor'}  
-10.693 (+/-1.109) for {'rows': 6, 'columns': 7, 'version': 'factor'}  
-10.694 (+/-1.110) for {'rows': 7, 'columns': 7, 'version': 'factor'}
```

Detailed classification report:

The model **is** trained on the full development **set**.

The scores are computed on the full evaluation **set**.