slearn

Release 0.0.1

NLA group

Feb 26, 2022

CONTENTS:

skearn is a package linking symbolic representation (SAX, ABBA, and fABBA) with scikit-learn machine learning for time series prediction. Symbolic representations of time series have proved their usefulness in the field of time series motif discovery, clustering, classification, forecasting, anomaly detection, etc. Symbolic time series representation methods do not only reduce the dimensionality of time series but also speed up the downstream time series task. It has been demonstrated by [S. Elsworth and S. Güttel, Time series forecasting using LSTM networks: a symbolic approach, arXiv, 2020] that symbolic forecasting has greatly reduced the sensitivity of hyperparameter settings for Long Short Term Memory networks. How to appropriately deploy machine learning algorithms on the level of symbols instead of raw time series poses a challenge to the interest of applications. To boost the development of the research community on symbolic representation, we develop this Python library to simplify the process of machine learning algorithm practice on symbolic representation.

Before getting started, please install the slearn package simply by

CHAPTER

ONE

INSTALLATION GUIDE

slearn has the following dependencies for its clustering functionality:

- numpy>=1.21
- scipy>1.6.0
- pandas
- lightgbm
- scikit-learn

To install the current release via PIP use:

```
pip install slearn
```

Note

The documentation is still on going.

1.1 Symbolic sequence prediction with machine learning

1.1.1 Machine learning with symbols

Given a sequence of symbols, ask you to predict the following symbols, what will you do with machine learning? An intuitive way is to transform the symbols to numerical labels, decide the appropriate windows size for features input (lag), and then define a classification problem. slearn build a pipeline for this process, and provide user-friendly API.

First import the package:

```
from slearn import symbolicML
```

We can predict any symbolic sequence by choosing the classifiers available in scikit-learn. Currently slearn supports:

Classifiers	Parameter call
Multi-layer Perceptron	'MLPClassifier'
K-Nearest Neighbors	'KNeighborsClassifier'
Gaussian Naive Bayes	'GaussianNB'
Decision Tree	'DecisionTreeClassifier'
Support Vector Classification	'SVC'
Radial-basis Function Kernel	'RBF'
Logistic Regression	'LogisticRegression'
Quadratic Discriminant Analysis	'QuadraticDiscriminantAnalysis'
AdaBoost classifier	'AdaBoostClassifier'
Random Forest	'RandomForestClassifier'
LightGBM	'LGBM'

Now we predict a simple synthetic symbolic sequence

```
string = 'aaaabbbccd'
```

First, we define the classifier, and specify the `ws` (windows size or lag) and `classifier_name` following the above table, initialize with

```
sbml = symbolicML(classifier_name="MLPClassifier", ws=3, random_seed=0, verbose=0)
```

Then we can use the method `encode` to split the features and target for training models. The we use method `forecast` to apply forecasting:

```
pred = sbml.forecast(x, y, step=5, hidden_layer_sizes=(10,10), learning_rate_init=0.1)
```

The parameters of `x`, `y`, and `step` are fixed, the rest of parameters are depend on what classifier you specify, the parameter settings can be referred to scikit-learn library. For nerual network, you can define the parameters of `hidden_layer_sizes` and `learning_rate_init`, while for support vector machine you might define `C`.

1.1.2 Generating symbols

slearn library also contains functions for the generation of strings of tunable complexity using the LZW compressing method as base to approximate Kolmogorov complexity.

```
from slearn import *
df_strings = LZWStringLibrary(symbols=3, complexity=[3, 9])
df_strings
```

Processing: 2 of 2

	nr_symbols	LZW_complexity	length	string
0	3	3	3	BCA
1	3	9	12	ABCBBCBBABCC

Also, you can deploy RNN test on the symbols you generate:

```
df_iters = pd.DataFrame()
for i, string in enumerate(df_strings['string']):
    kwargs = df_strings.iloc[i,:-1].to_dict()
    seed_string = df_strings.iloc[i,-1]
    df_iter = RNN_Iteration(seed_string, iterations=2, architecture='LSTM', **kwargs)
    df_iter.loc[:, kwargs.keys()] = kwargs.values()
    df_iters = df_iters.append(df_iter)
df_iter.reset_index(drop=True, inplace=True)
df_iters.reset_index(drop=True, inplace=True)
print(df_iters)
```

	jw	dl	total_epochs	seq_test	seq_forecast	total_time	nr_symbols	LZ
0	1.000000	1.0	12	ABCABCABCA	ABCABCABCA	2.685486	3	
1	1.000000	1.0	14	ABCABCABCA	ABCABCABCA	2.436733	3	
2	0.657143	0.5	36	CBBCBBABCC	AABCABCABC	3.352712	3	
3	0.704762	0.4	36	CBBCBBABCC	ABCBABBBBB	3.811584	3	

1.2 Time series forecasting with symbolic representation

slearn package contains the fast symbolic representation method, namely SAX and fABBA (more methods will be included).

Summmary

You can select the available classifiers and symbolic representation method (currently we support SAX, ABBA and fABBA) for prediction. Similarly, the parameters of the chosen classifier follow the same as the scikit-learn library. We usually deploy ABBA symbolic representation, since it achieves better forecasting against SAX.

slean leverages user-friendly API, time series forecasting follows:

Step 1: Define the windows size (features size), the forecasting steps, symbolic representation method (SAX or fABBA) and classifier.

Step 2: Transform time series into symbols with user specified parameters defined for symbolic representation.

Step 3: Define the classifier parameters and forecast the future values.

Now we illustrate how to use slearn with symbolic representation to forecast time series step by step.

First of all, we set the number of symbols you would like to predict and load libraries and data..

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from slearn import *

time_series = pd.read_csv("Amazon.csv") # load the required dataset, here we use Amazon␣
↪stock daily close price.
```

(continues on next page)

(continued from previous page)

```
ts = time_series.Close.values
step = 50
```

we start off with initializing the slearn with fABBA (alternative options: `SAX` and `ABBA`) and GaussianNB classifier, setting windows size to 3 and step to 50:

```
sl = slearn(method='fABBA',  ws=3, step=step, classifier_name="GaussianNB") # step 1
```

Next we transform the time series into symbols with method `set_symbols`:

```
sl.set_symbols(series=ts, tol=0.01, alpha=0.2) # step 2
```

Then we predict the time series with method `predict`:

```
abba_nb_pred = sl.predict(var_smoothing=0.001) # step 3
```

Together, we combine the code with three classifiers:

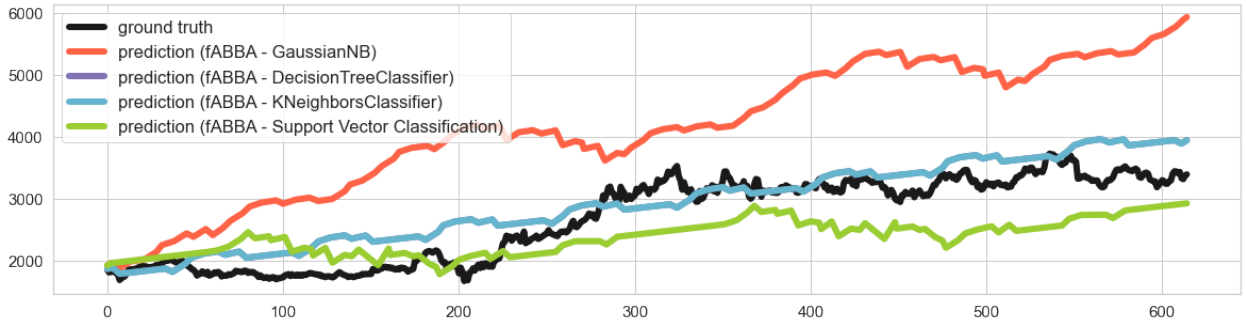
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from slearn import *
np.random.seed(0)

time_series = pd.read_csv("Amazon.csv")
ts = time_series.Close.values
length = len(ts)
train, test = ts[:round(0.9*length)], ts[round(0.9*length):]

sl = slearn(method='fABBA', ws=8, step=1000, classifier_name="GaussianNB")
sl.set_symbols(series=train, tol=0.01, alpha=0.1)
abba_nb_pred = sl.predict(var_smoothing=0.001)
sl = slearn(method='fABBA', ws=8, step=1000, classifier_name="DecisionTreeClassifier")
sl.set_symbols(series=train, tol=0.01, alpha=0.1)
abba_nn_pred = sl.predict(max_depth=10, random_state=0)
sl = slearn(method='fABBA', ws=8, step=1000, classifier_name="KNeighborsClassifier")
sl.set_symbols(series=train, tol=0.01, alpha=0.1)
abba_kn_pred = sl.predict(n_neighbors=10)
sl = slearn(method='fABBA', ws=8, step=100, classifier_name="SVC")
sl.set_symbols(series=train, tol=0.01, alpha=0.1)
abba_svc_pred = sl.predict(C=20)
min_len = np.min([len(test), len(abba_nb_pred), len(abba_nn_pred)])

plt.figure(figsize=(20, 5))
sns.set(font_scale=1.5, style="whitegrid")
sns.lineplot(data=test[:min_len], linewidth=6, color='k', label='ground truth')
sns.lineplot(data=abba_nb_pred[:min_len], linewidth=6, color='tomato', label='prediction␣
↪(ABBA - GaussianNB)')
sns.lineplot(data=abba_nn_pred[:min_len], linewidth=6, color='m', label='prediction␣
↪(ABBA - DecisionTreeClassifier)')
sns.lineplot(data=abba_nn_pred[:min_len], linewidth=6, color='c', label='prediction␣
↪(ABBA - KNeighborsClassifier)')
```

(continues on next page)

(continued from previous page)

```
sns.lineplot(data=abba_svc_pred[:min_len], linewidth=6, color='yellowgreen', label=
↪'prediction (ABBA - Support Vector Classification)')
plt.legend()
plt.tick_params(axis='both', labelsize=15)
plt.savefig('demo1.png', bbox_inches = 'tight')
plt.show()
```

The result is as plotted below:



1.3 API Reference

slearn.slearn([method, ws, step, ...])	A package linking symbolic representation with scikit-learn for time series prediction.
slearn.symbolicML([classifier_name, ws, ...])	Classifier for symbolic sequences.
slearn.SAX(*[, width, n_paa_segments, k, ...])	Modified from https://github.com/nla-group/TARZAN
slearn.fABBA([tol, alpha, sorting, scl, ...])	
slearn.ABBA([tol, k_cluster, verbose, max_len])	

1.3.1 slearn.slearn

class slearn.**slearn**(*method='fABBA'*, *ws=1*, *step=10*, *classifier_name='MLPClassifier'*, *form='numeric'*, *random_seed=0*, *verbose=1*)

A package linking symbolic representation with scikit-learn for time series prediction.

classifier_name - str, default=MLPClassifier,

optional choices = {"KNeighborsClassifier", "GaussianProcessClassifier" "QuadraticDiscriminantAnalysis", "DecisionTreeClassifier", "LogisticRegression", "AdaBoostClassifier", "RandomForestClassifier", "GaussianNB", "LGBM", "SVC", "RBF"}:

The classifier you specify for symbols prediction.

ws - int, default=3: The windows size for symbols to be the features, i.e, the dimensions of features.

step - int, default=1, The number of symbols for prediction.

method - str {'SAX', 'ABBA', 'fABBA'}: The symbolic time series representation. We use fABBA for ABBA method.

form - str, default='numeric': predict in symboli form or numerical form.

random_seed - int, default=0: The random state fixed for classifers in scikit-learn.

verbose - int, default=0: log print. Whether to print progress or other messages to stdout.

__init__(*method='fABBA'*, *ws=1*, *step=10*, *classifier_name='MLPClassifier'*, *form='numeric'*, *random_seed=0*, *verbose=1*)

Methods

__init__([method, ws, step, ...])	

construct_train(series)	Construct features and target labels for symbols.
encode(string)	Construct features and target labels for symbols and encode to numerical values.
forecast(x, y[, step, inversehash, centers])	
init_classifier()	
params_secure()	
predict(**params)	
set_symbols(series, **kwargs)	Transform time series to specified symplic representation

1.3.2 slearn.symbolicML

class slearn.**symbolicML**(*classifier_name='MLPClassifier'*, *ws=3*, *random_seed=0*, *verbose=0*)

Classifier for symbolic sequences.

classifier_name - str, default=MLPClassifier,

optional choices = {"KNeighborsClassifier", "GaussianProcessClassifier" "QuadraticDiscriminantAnalysis", "DecisionTreeClassifier", "LogisticRegression", "AdaBoostClassifier", "RandomForestClassifier", "GaussianNB", "DeepForest", "LGBM", "SVC", "RBF"}:

The classifier you specify for symbols prediction.

ws - int, default=3: The windows size for symbols to be the features, i.e, the dimensions of features. The larger the window, the more information about time series can be taken into account.

random_seed - int, default=0: The random state fixed for classifers in scikit-learn.

verbose - int, default=0: Whether to print progress messages to stdout.

__init__(*classifier_name='MLPClassifier'*, *ws=3*, *random_seed=0*, *verbose=0*)

Methods

`__init__`([classifier_name, ws, random_seed, ...])	
`construct_train`(series)	Construct features and target labels for symbols.
`encode`(string)	Construct features and target labels for symbols and encode to numerical values.
`forecast`(x, y[, step, inversehash, centers])	
`init_classifier`()	

1.3.3 slearn.SAX

class slearn.**SAX**(*, *width=2*, *n_paa_segments=None*, *k=5*, *return_list=False*, *verbose=True*)

Modified from https://github.com/nla-group/TARZAN

__init__(*, *width=2*, *n_paa_segments=None*, *k=5*, *return_list=False*, *verbose=True*)

Methods

`__init__`(*[, width, n_paa_segments, k, ...])	
`inverse_transform`(symbolic_time_series)	
`paa_mean`(ts)	
`transform`(time_series)	

1.3.4 slearn.fABBA

class slearn.**fABBA**(*tol=0.1*, *alpha=0.5*, *sorting='2-norm'*, *scl=1*, *verbose=1*, *max_len=inf*)

__init__(*tol=0.1*, *alpha=0.5*, *sorting='2-norm'*, *scl=1*, *verbose=1*, *max_len=inf*)

tol - float Control tolerence for compression, default as 0.1.

scl - int Scale for length, default as 1, means 2d-digitization, otherwise implement 1d-digitization.

verbose - int Control logs print, default as 1, print logs.

max_len - int The max length for each segment, default as np.inf.

Methods

__init__([tol, alpha, sorting, scl, ...])	Parameters tol - float Control tolerence for compression, default as 0.1. scl - int Scale for length, default as 1, means 2d-digitization, otherwise implement 1d-digitization. verbose - int Control logs print, default as 1, print logs. max_len - int The max length for each segment, default as np.inf.
digitize(pieces[, early_stopping])	Greedy 2D clustering of pieces (a Nx2 numpy array), using tolernce tol and len/inc scaling parameter scl.
fit_transform(series)	Compress and digitize the time series together.
inverse_compress(pieces, start)	Modified from ABBA package, please see ABBA package to see guidance.
inverse_digitize(strings, centers, hashmap)	
inverse_transform(strings[, start])	
quantize(pieces)	

1.3.5 slearn.ABBA

class slearn.**ABBA**(*tol=0.1*, *k_cluster=10*, *verbose=1*, *max_len=inf*)

__init__(*tol=0.1*, *k_cluster=10*, *verbose=1*, *max_len=inf*)

tol - float Control tolerence for compression, default as 0.1.

k_cluster - int Number of symbols used for digitization.

verbose - int Control logs print, default as 1, print logs.

max_len - int The max length for each segment, default as np.inf.

Methods

__init__([tol, k_cluster, verbose, max_len])	Parameters tol - float Control tolerence for compression, default as 0.1.
digitize(pieces[, early_stopping])	Greedy 2D clustering of pieces (a Nx2 numpy array), using tolernce tol and len/inc scaling parameter scl.
fit_transform(series)	Compress and digitize the time series together.
inverse_compress(pieces, start)	Modified from ABBA package, please see ABBA package to see guidance.
inverse_digitize(strings, centers, hashmap)	
inverse_transform(strings[, start])	

continues on next page

Table 6 – continued from previous page

`quantize`(pieces)

1.4 License

INDEX

Symbols

A

F

S