pyEDM Version 1.0.1 November 28, 2019

pyEDM is a Python package interface to the cppEDM C++ library of empirical dynamic modeling (EDM) algorithms. It returns Pandas DataFrame objects, or Python dictionaries of Pandas DataFrames.

Table of Contents

University of California at San Deigo
Scripps Institute of Oceangraphy
Sugihara Lab

Joseph Park, Cameron Smith

Introduction

pyEDM is a Python interface to the C++ library cppEDM. Input and output objects are based on Pandas DataFrame objects. Core algorithms are listed in table 1.

Algorithm	API Interface	Reference
Simplex projection	`Simplex()`	Sugihara and May (1990)
Sequential Locally Weighted Global Linear Maps (S-map)	`SMap()`	Sugihara (1994)
Predictions from multivariate embeddings	`Simplex(), SMap()`	Dixon et. al. (1999)
Convergent cross mapping	`CCM()`	Sugihara et. al. (2012)
Multiview embedding	`Multiview()`	Ye and Sugihara (2016)

Convenience functions to prepare and evaluate data are listed in table 2.

Function	Purpose	Parameter Range
`Embed()`	Timeseries delay dimensional embedding	User defined
`MakeBlock()`	Timeseries delay dimensional embedding	User defined
`EmbedDimension()`	Evaluate prediction skill vs. embedding dimension	E = [1, 10]
`PredictInterval()`	Evaluate prediction skill vs. forecast interval	Tp = [1, 10]
`PredictNonlinear()`	Evaluate prediction skill vs. SMap nonlinear localisation	θ = 0.01, 0.1, 0.3, 0.5, 0.75, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9
`ComputeError()`	Pearson ρ, RMSE, MAE	
`Examples()`	Example function calls and plots	

Installation

There are two methods to install pyEDM:
1) Python Package Index and pip, which is only supported for certain OSX and Windows platforms
2) Download, build and install package.

Python Package Index (PyPI)

Certain Mac OSX and Windows platforms are supported with prebuilt binary distributions and can be installed using the Python pip module. The module is located at pypi.org/project/pyEDM/.

Installation can be executed as:
```
python -m pip install pyEDM --trusted-host pypi.org --trusted-host
files.pythonhosted.org pyEDM
```

Manual Compilation

Unfortunately, we do not have the resources to provide pre-built binary distributions for all computer platforms. In this case the user is required to first build the cppEDM library on their machine, and then install the Python package using pip. On OSX and Linux this requires gcc, on Windows, Microsoft Visual Studio Compiler (MSVC) which can be obtained from Build Tools for Visual Studio 2019. Only the Windows SDK is needed. Note that the LAPACK library is required to build cppEDM.

OSX and Linux

1) Download pyEDM: git clone https://github.com/SugiharaLab/pyEDM

2) Build cppEDM library:
```
cd pyEDM/cppEDM/src
make
```

3) Build and install package:
```
cd ../..
python -m pip install . --user --trusted-host pypi.org
```

Windows

1) Download pyEDM: git clone https://github.com/SugiharaLab/pyEDM

2) Build cppEDM library:
```
cd pyEDM/cppEDM/src
nmake /f makefile.windows
```

3) Build and install package:
```
cd ../..
python -m pip install . --user --trusted-host pypi.org
```

Usage

```
>>> import pyEDM
>>> pyEDM.Examples()
```

See the Examples section below.

All data input files are assumed to be in csv format. The files are assumed to have a single line header with column names. If column names are not detected in the header line, then column names are created as V1, V2... **It is required that the first column be a vector of times or time indices.**

Parameters

API parameter names and purpose are listed in table 3.

Parameter	Type	Default	Purpose
pathIn	string	"./"	Input data file path
dataFile	string	""	Data file name
dataFrame	Pandas DataFrame	None	Input DataFrame
pathOut	string	"./"	Output file path
predictFile	string	""	Prediction output file
lib	string	""	library start : stop row indices
pred	string	""	prediction start : stop row indices
E	int	0	Data dimension
Tp	int	0	Prediction interval
knn	int	0	Number nearest neighbors
tau	int	1	Embedding delay
theta	float	0	SMap localisation
exclusionRadius	int	0	Prediction vector exclusion radius
columns	string	""	Column names or indices for prediction
target	string	""	Target library column name or index
embedded	bool	false	Is data an embedding?
const_pred	bool	false	Include non projected forecast data
verbose	bool	false	Echo messages
smapFile	string	""	SMap coefficient output file
multiview	int	0	Number of ensembles, 0 = sqrt(N)
libSizes_str	string	""	CCM library sizes
sample	int	0	CCM number of random samples
random	bool	true	CCM use random samples?
replacement	bool	false	CCM sample with replacement?
seed	unsigned	0	RNG seed, 0 = random seed

Application Programming Interface (API)

Embed

Create a data block of Takens (1981) time-delay embedding from each of the columns in the csv file or dataFrame. The `columns` parameter can be a list of column names, or a list of column indices. If `columns` is a list of indices, then column names are created as V1, V2...

Note: The returned DataFrame will have `tau*(E-1)` fewer rows than the input data from the removal of partial vectors as a result of the embedding.

Note: The returned DataFrame will not have the time column.

```
//------------------------------------------------------------------
//
//------------------------------------------------------------------
DataFrame Embed ( path      = "./",
                  dataFile  = "",
                  dataFrame = None,
                  E         = 0,
                  tau       = 1,
                  columns   = "",
                  verbose   = False )


//------------------------------------------------------------------
//
//------------------------------------------------------------------
DataFrame MakeBlock ( dataFrame,
                      E           = 0,
                      tau         = 1,
                      columnNames = "",
                      verbose     = False )
```

Simplex

Simplex projection of the input data file or DataFrame. The returned `DataFrame` has 3 columns "Time", "Observations", "Predictions". nan values are inserted where there is no observation or prediction. See the Parameters table for parameter definitions.

`lib` and `pred` specify [start stop] row indices of the input data for the library and predictions.

If `embedded` is `false` the data columns are embedded to dimension `E` with delay `tau`. If `embedded` is `true` the data columns are assumed to be a multivariable data block.

If `knn` is not specified, it is set equal to `E+1`.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame Simplex( pathIn          = "./",
                   dataFile        = "",
                   dataFrame       = None,
                   pathOut         = "./",
                   predictFile     = "",
                   lib             = "",
                   pred            = "",
                   E               = 0,
                   Tp              = 1,
                   knn             = 0,
                   tau             = 1,
                   exclusionRadius = 0,
                   columns         = "",
                   target          = "",
                   embedded        = False,
                   verbose         = False
                   const_pred      = False,
                   showPlot        = False )
```

SMap

SMap projection of the input data file or DataFrame. See the Parameters table for parameter definitions.

`SMap()` returns a `dict` with two DataFrames:

```
dict { predictions  : DataFrame,
       coefficients : DataFrame
}
```

The `predictions DataFrame` has 3 columns "Time", "Observations", "Predictions". nan values are inserted where there is no observation or prediction. If `predictFile` is provided the `predictions` will be written to it in csv format.

The `coefficients DataFrame` will have `E+2` columns. The first column is the "Time" vector, the remaining `E+1` columns are the SMap SVD fit coefficients.

`lib` and `pred` specify [start, stop] row indices of the input data for the library and predictions.

If `embedded` is `false` the data columns are embedded to dimension `E` with delay `tau`. If `embedded` is true the data columns are assumed to be a multivariable data block. If `smapFile` is provided the `coefficients` will be written to it in csv format.

If `knn` is not specified, it is set equal to the library size. If `knn` is specified, it must be greater than `E`.

```
//-----------------------------------------------------------------
//
//-----------------------------------------------------------------
dict SMap( pathIn          = "./",
           dataFile        = "",
           dataFrame       = None,
           pathOut         = "./",
           predictFile     = "",
           lib             = "",
           pred            = "",
           E               = 0,
           Tp              = 1,
           knn             = 0,
           tau             = 1,
           theta           = 0,
           exclusionRadius = 0,
           columns         = "",
           target          = "",
           smapFile        = "",
           derivatives     = "",   // Not implemented
           embedded        = False,
           verbose         = False,
           const_pred      = False,
           showPlot        = False )
```

CCM

Convergent cross mapping via Simplex of the first vector specified in `columns` against `target`. The data cannot be multivariable, the first vector in `columns` is time-delay embedded to dimension `E`. See the Parameters table for parameter definitions.

The returned `DataFrame` has 3 columns. The first column is "LibSize", the second and third columns are Pearson correlation coefficients for "column : target" and "target : column" cross mapping.

`libSizes` specifies a string with "start stop increment" row values, i.e. "10 80 10" will evaluate library sizes from 10 to 80 in increments of 10.

If `random` is `true`, `sample` observations are radomly selected from the subset of each library size. If `seed=0`, then a random seed is generated for the random number generator. Otherwise, `seed` is used to initialise the random number generator.

If `random` is false, `sample` is ignored and contiguous library rows up to the current library size are used.

Note: Cross mappings are performed between `column : target`, and `target : column`. The default is to do this in separate threads. Threading can be disabled in the makefile by removing `-DCCM_THREADED`.

Note: The entire library size is used in the Simplex prediction at each library subset size.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame CCM( pathIn       = "./",
               dataFile     = "",
               dataFrame    = None,
               pathOut      = "./",
               predictFile  = "",
               E            = 0,
               Tp           = 0,
               knn          = 0,
               tau          = 1,
               columns      = "",
               target       = "",
               libSizes     = "",
               sample       = 0,
               random       = True,
               repalcement  = False,
               seed         = 0,     // seed=0: use RNG
               verbose      = False,
               showPlot     = False );
```

Multiview

Multiview embedding and forecasting of the input data file or DataFrame. See the Parameters table for parameter definitions.

`Multiview()` returns a `dict`:

```
dict { View        : DataFrame,
       Predictions : DataFrame
}
```

The `Predictions DataFrame` has 3 columns "Time", "Observations", "Predictions". nan values are inserted where there is no observation or prediction. If `predictFile` is provided the `Predictions` will be written to it in csv format.

The `View DataFrame` will have `E+3` columns. The first `E` columns are the the column indices in the input data `DataFrame` that are embedded and applied to Simplex prediction. The last three columns are "rho", "MAE", "RMSE" corresponding to the prediction Pearson correlation, maximum absolute error and root mean square error.

`lib` and `pred` specify [start, stop] row indices of the input data for the library and predictions.

If `multiview` is not specified it is set to sqrt(C) where C is the number of E-dimensional combinations out of all available data vectors.

If `knn` is not specified, it is set equal to `E+1`.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
dict Multiview( pathIn          = "./",
                dataFile        = "",
                dataFrame       = None,
                pathOut         = "./",
                predictFile     = "",
                lib             = "",
                pred            = "",
                E               = 0,
                Tp              = 1,
                knn             = 0,
                tau             = 1,
                columns         = "",
                target          = "",
                multiview       = 0,
                exclusionRadius = 0,
                verbose         = False,
                numThreads      = 4,
                showPlot        = False )
```

EmbedDimension

Evaluate Simplex prediction skill for embedding dimensions from 1 to 10. The returned `DataFrame` has columns "E" and "rho". See the Parameters table for parameter definitions.

Note: `nThreads` defines the number of worker threads for the 10 embeddings. The maximum number of threads is 10.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame EmbedDimension( pathIn      = "./",
                          dataFile    = "",
                          dataFrame   = None,
                          pathOut     = "./",
                          predictFile = "",
                          lib         = "",
                          pred        = "",
                          maxE        = 10,
                          Tp          = 1,
                          tau         = 1,
                          columns     = "",
                          target      = "",
                          embedded    = False,
                          verbose     = False,
                          numThreads  = 4,
                          showPlot    = False )
```

PredictInterval

Evaluate Simplex prediction skill for forecast intervals from 1 to 10. The returned `DataFrame` has columns "Tp" and "rho". See the Parameters table for parameter definitions.

Note: `nThreads` defines the number of worker threads for the 10 prediction interval forecasts. The maximum number of threads is 10.

```
//----------------------------------------------------------------
// Overload 1: Explicit data file path/name
//----------------------------------------------------------------
DataFrame PredictInterval( pathIn      = "./",
                           dataFile    = "",
                           dataFrame   = None,
                           pathOut     = "./",
                           predictFile = "",
                           lib         = "",
                           pred        = "",
                           maxTp       = 10,
                           E           = 0,
                           tau         = 1,
                           columns     = "",
                           target      = "",
                           embedded    = False,
                           verbose     = False,
                           numThreads  = 4,
                           showPlot    = False );
```

PredictNonlinear

Evaluate SMap prediction skill for localisation parameter θ (default from 0.01 to 9). The returned `DataFrame` has columns "theta" and "rho". See the Parameters table for parameter definitions.

Note: `nThreads` defines the number of worker threads for the θ value forecasts.

```
//----------------------------------------------------------------
//
//----------------------------------------------------------------
DataFrame PredictNonlinear( pathIn      = "./",
                            dataFile    = "",
                            dataFrame   = None,
                            pathOut     = "./",
                            predictFile = "",
                            lib         = "",
                            pred        = "",
                            theta       = "",
                            E           = 0,
                            Tp          = 1,
                            tau         = 1,
                            columns     = "",
                            target      = "",
                            embedded    = False,
                            verbose     = False,
                            numThreads  = 4,
                            showPlot    = False );
```

ComputeError

Compute Pearson correlation coefficient, maximum absolute error (MAE) and root mean square error (RMSE) between two vectors.

`ComputeError()` returns a dict:

```
dict { rho  :  double,
       RMSE :  double,
       MAE  :  double
}

//-----------------------------------------------------------------
//-----------------------------------------------------------------
dict ComputeError( obsIn, predIn )
```

Application Notes

All data input files are assumed to be in csv format. The files are assumed to have a single line header with column names. If column names are not detected in the header line, then column names are created as V1, V2... **It is required that the first column be a vector of times or time indices.**

`SMap()` should be called with `DataFrame` that have columns explicity corresponding to dimensions E. This means that if a multivariate data set is used, it should Not be called with an embedding from `Embed()` since `Embed()` will add lagged coordinates for each variable. These extra columns will then not correspond to the intended dimensions in the matrix inversion and prediction reconstruction. In this case, use the `embedded` parameter set to `true` so that the columns selected correspond to the proper dimension.

Examples

```
from pyEDM import *

df = EmbedDimension( pathIn = "./pyEDM/data/", dataFile = "TentMap_rEDM.csv",
                     lib = "1 100", pred = "201 500",
                     columns = "TentMap", showPlot = True )

df = PredictInterval( pathIn = "./pyEDM/data/", dataFile = "TentMap_rEDM.csv",
                      lib = "1 100", pred = "201 500",  E = 2,
                      columns = "TentMap", showPlot = True )

df = PredictNonlinear( pathIn = "./pyEDM/data/",
                       dataFile = "TentMapNoise_rEDM.csv",
                       lib = "1 100", pred = "201 500", E = 2,
                       columns = "TentMap", showPlot = True )

df = Simplex( pathIn = "./pyEDM/data/", dataFile = "block_3sp.csv",
              lib = "1 99", pred = "100 198", E = 3,
              columns = "x_t y_t z_t", target = "x_t",
              embedded = True, showPlot = True )

df = Simplex( pathIn = "./pyEDM/data/", dataFile = "block_3sp.csv",
              lib = "1 99", pred = "100 195", E = 3,
              columns = "x_t", target = "x_t", showPlot = True )

M = Multiview( pathIn = "./pyEDM/data/", dataFile = "block_3sp.csv",
               lib = "1 100", pred = "101 198", E = 3,
               columns = "x_t y_t z_t", target = "x_t", showPlot = True )

S = SMap( pathIn = "./pyEDM/data/", dataFile = "circle.csv",
          lib = "1 100", pred = "101 198", E = 2, theta = 4,
          columns = "x y", target = "x", embedded = True, showPlot = True )

df = CCM( pathIn = "./pyEDM/data/", dataFile = "sardine_anchovy_sst.csv",
          E = 3, Tp = 0, columns = "anchovy", target = "np_sst",
          libSizes = "5 75 5", sample = 100, showPlot = True )
```

References

Dixon, P. A., M. Milicich, and G. Sugihara, 1999. Episodic fluctuations in larval supply. Science 283:1528–1530.

Sugihara G. and May R. 1990. Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series. Nature, 344:734–741.

Sugihara G. 1994. Nonlinear forecasting for the classification of natural time series. Philosophical Transactions: Physical Sciences and Engineering, 348 (1688) : 477–495.

Sugihara G., May R., Ye H., Hsieh C., Deyle E., Fogarty M., Munch S., 2012. Detecting Causality in Complex Ecosystems. Science 338:496-500.

Takens, F. Detecting strange attractors in turbulence. Lect. Notes Math. 898, 366–381 (1981).

Ye H., and G. Sugihara, 2016. Information leverage in interconnected ecosystems: Overcoming the curse of dimensionality. Science 353:922–925.