timeWarpOB Documentation

Release 1.1.0

Oliver Binns

September 09, 2016

Contents

timeWarpOB is a Dynamic Time Warping implementation, built in python, by Oliver Binns

Contents:

CHAPTER 1

About timeWarpOB

timeWarpOB is a python implementation of Dynamic Time Warping. Time Warping allows time-series data to be compared when one of the time series exhibits accellerations or decellerations relative to the other at different points in time.

It has applications in speech recognition (where people may pronounce sounds at different rates) and in the general analysis of time-series data (e.g. financial market indicators over time), where is can be combined with other machine learning techniques. [1]

For more details on how the algorithm works, see the the User guide section

[1] Dynamic Time Warping (Wikipedia)

CHAPTER 2

Version history

2.1 v1.1

- Bug fixes to ERP window function
- Python speed improvements to L1 distance calc
- Support for numba jit-compiler for speed improvements
- More developed test suite

2.2 v1.0.1

- Initial release of timeWarpOB

CHAPTER 3

Installation

timeWarpOB can be installed by using pip:

```
pip install timeWarpOB
```

Note that timeWarpOB requires numpy to be be installed for handling the time series and results arrays. The matplotplib module is also required for plotting the warp graphs.

Calculation speed can be substantially improved by installing the numba module, which compiles some of the inner calculation loops to give calculation performance comparable to native compiled code. However, timeWarpOB can still run without numba. To check if numba is detected by timeWarpOB, run the following in python:

```
import timeWarpOB as tw
print(tw.foundNumba)
```

You can quickly test the timeWarpOB installation by using:

```
x, y, ts = tw.tests.basic.sinCos()
tw.tests.basic.testWarp(x,y,ts)
```

This will generate two time series based on a sin and cos function and attempt to warp them. The warp statistics will be printed to the console and a plot showing the warp result will be shown.

CHAPTER 4

User guide

4.1 Background to Time Warping

Time warping is used to determine the similarity between two time series that vary in speed along the time axis. By warping the time axis on one of the series, it is possible to make the time series 'match' each other, with the amount of warping being a measure of the dis-similarity of the series. By dynamically changing the amount of warping along the time-axis, it is possible to deal with time series that accelerate or decelerate at different points in the series. Because of this, time warping has applications in many areas, including speech recognition, video analysis and understanding of market data. The output from time warping can be used as a similarity distance metric that is fed into other machine learning methods such as clustering.

Time warping techniques are based on edit distance in strings, which calculates the 'cost' of converting one string into another by making a series of insertion, deletion or replacement operations. The optimal (lowest) number of these operations can be determined using a dynamic programming algorithm [1]. The same methods can be used, but replacing the symbols in the two strings with the numeric values of a time series.

timeWarpOB implements two methods of warping the time axis, namely Dynamic Time Warping (DTW) and Edit distance with Real Penalty (ERP) [2].

Both work by taking two time series of length n and initially forming a $n \times n$ matrix measuring the pairwise Euclidean distance between every point on the two time series (the distance matrix). Both algorithms will then create a second $n \times n$ that describes the cumulative minimum 'cost' of moving through the distance matrix (the cost matrix). Depending on the method, the costs involved are calculated differently. After this cost matrix has been produced, a back tracing routine works backwards through the matrix and finds the lowest cost route. This path describes the warping of the time series with respect to each other (i.e. where one is accelerating or decelerating relative to the other)

In DTW, the algorithm will traverse the distance matrix and calculate the cumulative distance, by the minimum of moving one step in both time series (i.e. when they are in sync) or moving only one step in one of the time series (i.e. when one is accelerating relative to another. In ERP, the cumulative distance is based upon a penalty factor (g) when moving along only one of the time series. For a formal definition of these cost functions, see [2]

In order to prevent the back tracing function from selecting a circuitous route through the cost matrix and warping large parts of a time series in a way that is not realistic for the physical application, it is possible to apply a 'warp window' to the cost matrix, which sets all values a given distance from the diagonal to a very high number (or infinity), so the back trace algorithm does not go through them.

For a comparison of various time warping techniques, including some not implemented by *timeWarpOB*, see this paper: [3]. Additionally this paper [4] details the Minimum Jump cost (MJC) method.

[1] Wagner-Fischer algorithm (Wikipedia)

[2] Chen, Lei, and Raymond T Ng. 2004. "On the Marriage of Lp-Norms and Edit Distance.." Vldb, 792–803.

[3] Serrà, Joan, and Josep Lluís Arcos. 2014. "An Empirical Evaluation of Similarity Measures for Time Series Classification.." Knowl.-Based Syst. () cs.LG: 305–14. doi:10.1016/j.knosys.2014.04.035.

[4] Serrà, Joan, and Josep Lluís Arcos. 2012. "A Competitive Measure to Assess the Similarity Between Two Time Series." In Case-Based Reasoning Research and Development, edited by Belén Díaz Agudo and Ian Watson, 7466:414–27. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-32986-9_31.

NB: with very long time series, the methods used in this package may be slow. This is as for a time series of length n, an $n \times n$ matrix of values must be populated, meaning the algorithm runs on $\mathcal{O}(n^2)$. Alternatives for large time series include the FTSE method [5] (not implemented in *timeWarpOB*).

4.2 Example usage

NOTE: This section assumes you have followed the instructions in the Installation section.

First, import timeWarpOB:

```
import timeWarpOB as tw
```

Then prepare two python lists of data, along with an optional list containing timestamps If you have data loaded into a pandas dataframe called `df`, with two columns and an index of timestamps, you can convert them by:

```
a = list(df["columnA"])
b = list(df["columnB"])
ts = list(df.index)
```

Alternatively, generate a sin and cos series using the built-in test function (passing `n` for the number of half-cycles to generate):

```
a, b, ts = tw.tests.basic.sinCos(n=2)
```

The two timeseries then be warped using the `timeWarp()` function, to return a `warpObject`:

```
wo = tw.timeWarp(a,b)
```

This will return a warp object. Useful statistics about the time warping result can be found in `wo['warpStats']` and the total warp cost can be found in `wo['backTraceCost']`. More information about the warp object and the calculation of the statistics can be found in the API reference section.

To plot the results of the time warp, use:

```
tw.plotting.plotWarp(a,b,wo,ts)
```

Which will give a warping result as a matplotlib chart, containing the following items:

- The central chart displays the cost matrix as a series of grey pixels in a graph. The darker pixels represent higher costs. Time series 'a' runs along the vertical axis from bottom to top and time series 'b' runs along the horizontal axis from left to right . The diagonal line (yellow) represents the path that would be taken if the two series were perfectly in sync. The diagonal, dashed magenta lines show the extent of the warp window (if applied) and the red line shows the back-traced path through the cost matrix. When the red line deviates from the yellow diagonal line, the time series are accelerating or decelerating with respect to each other. If the red line moves closer to the top-left corner, then this indicates that time series a is moving ahead of time series b (and *vice versa*).
- To the left of the central plot is a graph of time series a (rotated, so each point aligns with its position on the central chart). Below the central plot is a graph of time series b.
- To the right of the central plot, if a graph which shows the back-traced path, but rotated by 45 degrees to facilitate better understanding of where the time series accelerate or decelerate relative to one another. A dashed cyan line is added to show the average position of the red line (which is equal to the `warpStats.avgWarp` value)
- Above the central plot, both time series are plotted against each other and red lines are added to show how point on each time series are related according to the back-traced warp path.

[5] Morse, Michael D, and Jignesh M Patel. 2007. "An Efficient and Accurate Method for Evaluating Time Series Similarity.." Sigmod. New York, New York, USA: ACM Press, 569–80. doi:10.1145/1247480.1247544.

4.3 References

CHAPTER 5

API reference

The following is a reference guide for the functions in the modules of neurOB, sorted by submodule.

5.1 timeWarpOB

The timeWarpOB module contains the main functions for time warping. The other submodules contain functions to help with plotting.

timeWarpOB.**timeWarp**(*a*, *b*, *method='DTW'*, *window=0*, *retMat=True*, ***kwargs*)

This function is the main time warping interface, and acts as a convenient wrapper to the other functions.

Parameters

- **a** (*list or numpy 1D-array*) – First time series, which will be compared against time series b
- **b** (*list or numpy 1D-array*) – Second time series (reference)
- **method** (*str*) – Time warping method {'DTW','ERP'} - see below
- **window** (*int*) – Time warping window constraint (default = 0)
- **retMat** (*bool*) – Whether to include the cost matrices in the returned object
- **ERPg** (*int*) – g-value (for method = 'ERP' only)

Returns

- **warpObj** (*dict*) – A timeWarpOB warp object, containing the following items:
- **warpObj.backTraceCost** (*float*) – The sum cost of following the backtrace through the cost matrix
- **warpObj.backTracePath** (*list*) – List or numpy array of pairs of coordinates in time-space describing the backtrace through the cost matrix
- **warpObj.cost** (*float*) – Bottom left value on the cost matrix. With no warping window, this should equal warpObj.backTraceCost
- **warpObj.costMat** (*list*) – A matrix (list of lists) or numpy array describing the cost matrix between the two time series. For a series of length n, this matrix will be of size n x n. Only output if retMat = True in the input parameters
- **warpObj.distMat** (*list*) – A matrix (list of lists) or numpy array describing the L1-distance matrix between the two time series. For a series of length n, this matrix will be of size n x n. Only output if retMat = True in the input parameters
- **warpObj.warpWindow** (*int*) – Returning the warp window parameter used (used by plotting functions)
- **warpObj.warpStats** (*dict*) – Warp statistics object, containing:

- **warpObj.warpStats.timeAhead** (*int*) – The number of periods that time series a was in sync with time series b.
- **warpObj.warpStats.timeAhead** (*int*) – The number of periods that time series a was ahead of time series b.
- **warpObj.warpStats.timeBehind** (*int*) – The number of periods that time series a was behind of time series b.
- **warpObj.warpStats.amountAhead** (*int*) – The total sum of the number of periods that time series a was leading time series b by.
- **warpObj.warpStats.amountBehind** (*int*) – The total sum of the number of periods that time series a was lagging time series b by.
- **warpObj.warpStats.avgAhead** (*int*) – Average amount of periods that a was ahead of a by, i.e. amountAhead divided by timeAhead
- **warpObj.warpStats.avgAhead** (*int*) – Average amount of periods that a was behind b by, i.e. amountBehind divided by timeBehind
- **warpObj.warpStats.avgWarp** (*int*) – Average amount of periods that a ahead or behind b by, i.e. `(amountAhead - amountBehind) / (timeAhead + timeBehind + timeSync)` This will give positive values if a is on average ahead of b and negative values is a is on average behind b.

Notes

•If lists are supplied, the output matrices will be in list form. If numpy arrays are supplied, the output will be as numpy arrays.

•Numpy arrays will calculate faster, as no internal type conversion is required.

•Time series should be of equal length. If they are not, the longer will be clipped from the end.

•ERP and DTW methods are available. For information on how they work, see the module documentation.

•Cost matricies should be returned for use by the plotting functions

`timeWarpOB.`**`L1distances`**(*a*, *b*)

Calcluates the L1 distance matrix between two time series.

Parameters

- **a** (*list*) – First time series, which will be compared against time series b
- **b** (*list*) – Second time series (reference)

Returns **distance** – A matrix (list of lists) describing the L1-distance matrix between the two time series. For a series of length n, this matrix will be of size n x n.

Return type list

`timeWarpOB.`**`ERPwarp`**(*dist*, *x*, *y*, *w=0*, *g=0*)

Calcluates the ERP cost matrix between two time series.

Parameters

- **dist** (*list*) – A matrix (list of lists) describing the L1-distance matrix between two time series. For a time series of length n, this matrix must be of size n x n.
- **x** (*list*) – First time series, which will be compared against time series y
- **y** (*list*) – Second time series (reference)
- **w** (*int*) – Time warping window constraint (default = 0)
- **g** (*int*) – ERP g-value (deafult = 0)

Returns **costMat** – A matrix (list of lists) describing the ERP cost matrix between the two time series. For a series of length n, this matrix will be of size n x n.

Return type list

timeWarpOB.**DTWwarp**(*dist*, *x*, *y*, *w=0*)

Calcluates the ERP cost matrix between two time series.

Parameters

- **dist** (*list*) – A matrix (list of lists) describing the L1-distance matrix between two time series. For a time series of length n, this matrix must be of size n x n.
- **x** (*list*) – First time series, which will be compared against time series y
- **y** (*list*) – Second time series (reference)
- **w** (*int*) – Time warping window constraint (default = 0)

Returns **costMat** – A matrix (list of lists) describing the DTW cost matrix between the two time series. For a series of length n, this matrix will be of size n x n.

Return type list

timeWarpOB.**backTrace**(*costMat*, *dist*)

Finds the optimal warping path by backtracking through the cost matrix.

Parameters

- **dist** (*list*) – A matrix (list of lists) describing the L1-distance matrix between two time series. For a time series of length n, this matrix must be of size n x n.
- **costMat** (*list*) – A matrix (list of lists) describing the time-warped cost matrix between two time series. For a time series of length n, this matrix must be of size n x n.

Returns

- **path** (*list*) – List of pairs of coordinates in time-space describing the backtrace through the cost matrix
- **backTraceCost** (*float*) – The sum cost of following the backtrace through the cost matrix
- **warpStats** (*dict*) – Warp statistics object, containing:
- **warpStats.timeAhead** (*int*) – The number of periods that time series a was in sync with time series b.
- **warpStats.timeAhead** (*int*) – The number of periods that time series a was ahead of time series b.
- **warpStats.timeBehind** (*int*) – The number of periods that time series a was behind of time series b.
- **warpStats.amountAhead** (*int*) – The total sum of the number of periods that time series a was leading time series b by.
- **warpStats.amountBehind** (*int*) – The total sum of the number of periods that time series a was lagging time series b by.
- **warpStats.avgAhead** (*int*) – Average amount of periods that a was ahead of a by, i.e. amountAhead divided by timeAhead
- **warpStats.avgAhead** (*int*) – Average amount of periods that a was behind b by, i.e. amountBehind divided by timeBehind
- **warpStats.avgWarp** (*int*) – Average amount of periods that a ahead or behind b by, i.e. `(amountAhead - amountBehind) / (timeAhead + timeBehind + timeSync)` This will give positive values if a is on average ahead of b and negative values is a is on average behind b.

5.2 timeWarpOB.plotting

The timeWarpOB.plotting module contains functions for plotting the outcome of a time warp experiment.

`timeWarpOB.plotting.`**`plotWarp`**(*a*, *b*, *warpObj*, *ts=[]*)

Plots the comprehensive results of a time warp, including the cost matrix, backtrace path, visualisation of the warping statistics and the individual time series, with warp lines.

Parameters

- **a** (*list*) – First time series, which has been warped against time series b
- **b** (*list*) – Second time series (reference)
- **warpObj** (*dict*) – A timeWarpOB warp object - see output of `timeWarpOB.timeWarp()`. NB: the object must contain the calclauted cost matrix (i.e. `retMat = True`)
- **ts** (*list*) – Optional list of timestamps for displaying on the graphs. If no timestamps are given, each time period will be given an incremented number, staring at 1.

Returns A composite plot showing the results of the time warp.

Return type matplotlib.pyplot

`timeWarpOB.plotting.`**`plotSeries`**(*a*, *b*, *warpObj*)

Plots the two time series with warp lines as a single plot

Parameters

- **a** (*list*) – First time series, which has been warped against time series b
- **b** (*list*) – Second time series (reference)
- **warpObj** (*dict*) – A timeWarpOB warp object - see output of `timeWarpOB.timeWarp()`.

Returns A single plot showing the warp lines on the two time series.

Return type matplotlib.pyplot

5.3 timeWarpOB.tests.basic

Basic tests of the timeWarpOB module functionality.

`tests.basic.`**`testWarp`**(*x*, *y*, *ts*)

Warps two time series, displays plots and prints the warp statistics.

Parameters

- **x** (*list*) – First time series, which has been warped against time series b
- **y** (*list*) – Second time series (reference)
- **ts** (*list*) – Optional list of timestamps for displaying on the graphs. If no timestamps are given, each time period will be given an incremented number, staring at 1.

Returns

- *matplotlib.pyplot* – A composite plot showing the results of the time warp.
- **warpStats** (*dict*) – Warp statistics (printed to the console only)

`tests.basic.`**`sinCos`**(*n=2*, *l=200*)

Generates a test dataset of sin and cosine over 200 points in a time series.

Parameters

- **n** (*int*) – Number of half-cycles to calculate the values over
- **l** (*int*) – Number of points in the time series

Returns

- **x** (*list*) – A list of sin(ts) values
- **y** (*list*) – A list of cos(ts) values
- **ts** (*list*) – A list of 'timestamp' values (radian angle values)

B

D

E

L

P

S

T