TSSEARCH Documentation

Release 0.1.3

Fraunhofer AICOS

Apr 20, 2022

Contents

Time Series Subsequence Search Python package (TSSEARCH for short) is a Python package that assists researchers in exploratory analysis for query search and time series segmentation without requiring significant programming effort. It contains curated routines for query and subsequence search. TSSEARCH installation is straightforward and goes along with startup code examples. Our goal is to provide the tools to get faster insights for your time series.

CHAPTER 1

Highlights

- **Search**: we provide methods for time series query search and segmentation
- **Weights**: the relative contribution of each point of the query to the overall distance can be expressed using a user-defined weight vector
- **Visualization**: we provide visualizations to present the results of the segmentation and query search
- **Unit tested**: we provide unit tests for each distance
- **Easily extended**: adding new distances is easy, and we encourage you to contribute with your custom distances or search methods

CHAPTER 2

Contents

In development

2.1 Lockstep Distances

Distance measures that compare the i-th point of one time series to the i-th point of another are named as lock-step measures (e.g., Euclidean distance and the other Lp norms). A linear interpolation is computed to the longer series to ensure it has the same length as the shorter series.

2.2 Elastic Distances

Distance measures that perform a non-linear mapping to align the time series and allow flexible comparison of one-to-many or one-to-none points (e.g., Dynamic Time Warping, Longest Common Subsequence). These measures produce elastic adjustment to compensate for potential localized misalignment.

2.2.1 Dynamic Time Warping (DTW)

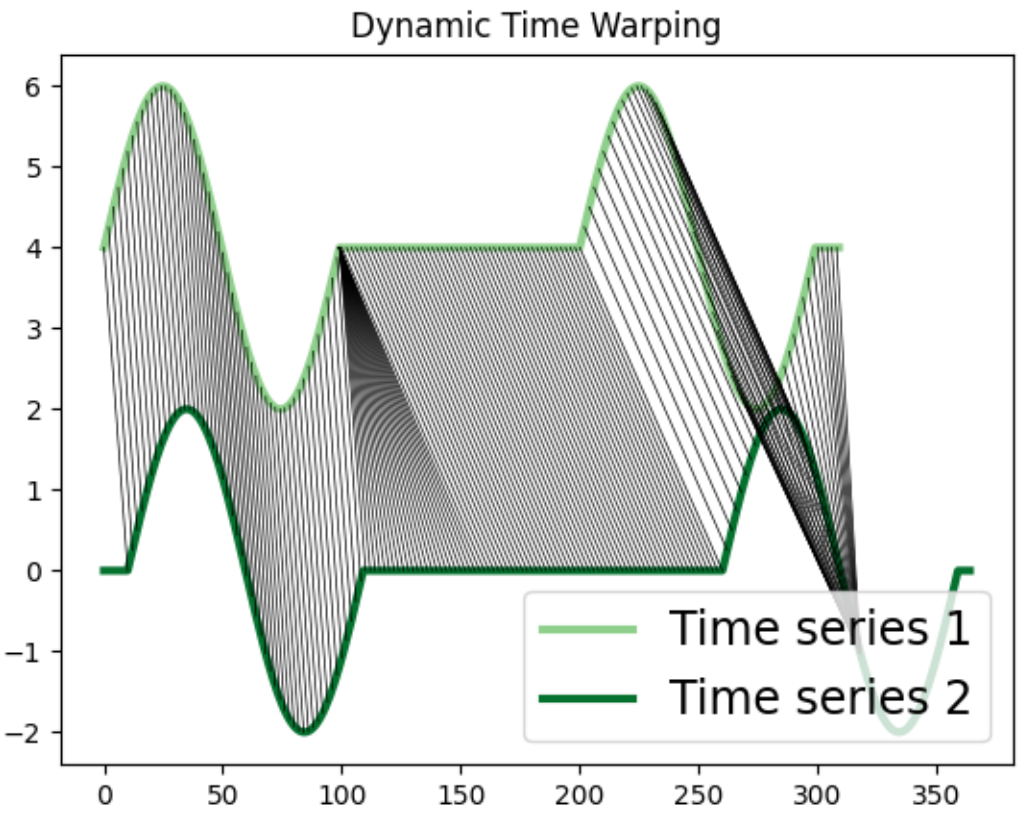
The DTW algorithm computes the stretch of the time axis which optimally maps between two time series. It measures the remaining cumulative distance after the alignment and the pairwise correspondence between each sample.

```
import numpy as np
import matplotlib.pyplot as plt
from tssearch.search.query_search import time_series_search
from tssearch.utils.visualisation import plot_alignment

# generates signals
freq = 2
amp = 2
time = np.linspace(0, 2, 100)
```

(continues on next page)

(continued from previous page)

```
ts1 = np.concatenate([amp * np.sin(np.pi * time), np.zeros(100), amp * np.sin(np.pi *␣
↪time), np.zeros(10)])
ts2 = np.concatenate([np.zeros(10), amp * np.sin(np.pi * time), np.zeros(150), amp *␣
↪np.sin(np.pi * time), np.zeros(5)])

dict_distances = {
    "elastic": {"Dynamic Time Warping": {
        "multivariate": "yes",
        "description": "",
        "function": "dtw",
        "parameters": {"dtw_type": "dtw", "alpha": 1},
        "use": "yes"}
    }
}

result = time_series_search(dict_distances, ts1, ts2, output=("number", 1))

plt.figure()
plt.title("Dynamic Time Warping")
plot_alignment(ts1, ts2, result["Dynamic Time Warping"]["path"][0])
plt.legend(fontsize=17, loc="lower right")
```



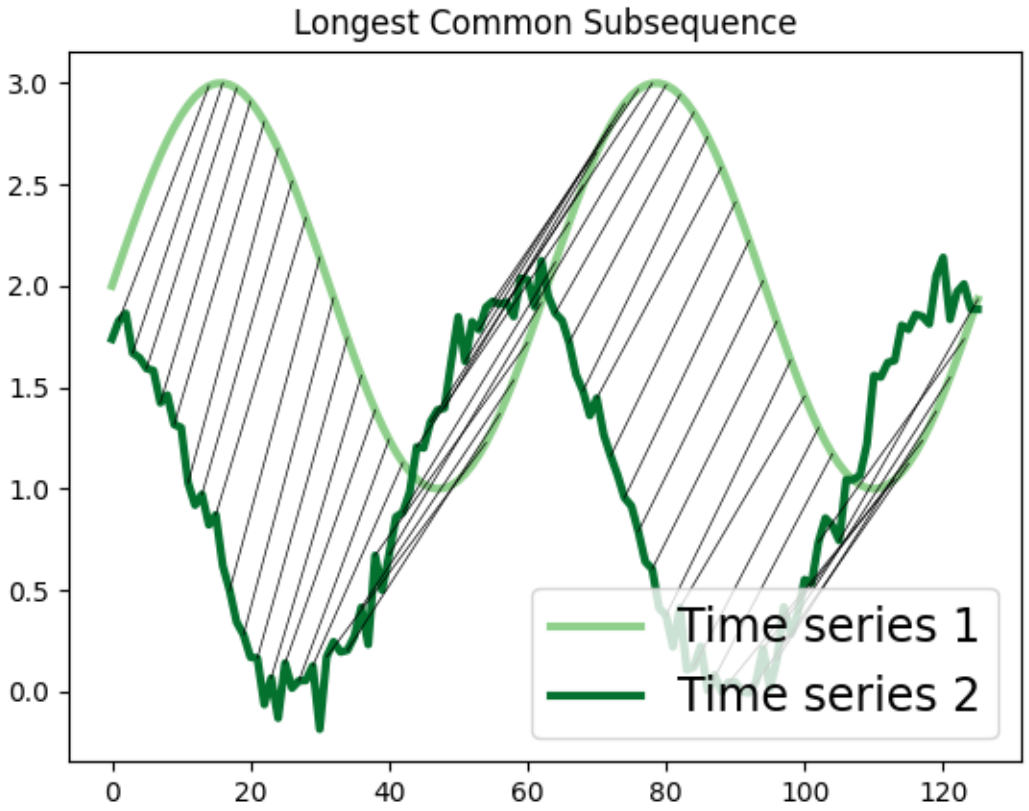
2.2.2 Longest Common Subsequence (LCSS)

The Longest Common Subsequence (LCSS) measures the similarity between two time series whose lengths might be different. Since it is formulated based on edit distances, gaps or unmatched regions are permitted and they are penalized with a value proportional to their length. It can be useful to identify similarities between time series whose lengths differ greatly or have noise[1].

In the example below, we compute the LCSS alignment between two time series, one of them with added noise.

```
import numpy as np
import matplotlib.pyplot as plt
from tssearch.search.query_search import time_series_search
from tssearch.utils.visualisation import plot_alignment

ts1 = np.sin(np.arange(0, 4*np.pi, 0.1))
noise = np.random.normal(0, 0.1, ts1.shape)
ts2 = 1 + np.sin(np.arange(0, 4*np.pi, 0.1) + 2) + noise

ts1 = ts1.reshape(-1, 1)
ts2 = ts2.reshape(-1, 1)

dict_distances = {
    "elastic": {"Longest Common Subsequence": {
        "multivariate": "yes",
        "description": "",
        "function": "lcss",
        "parameters": {"eps": 1, "report": "distance"},
        "use": "yes"}
    }
}

result = time_series_search(dict_distances, ts1, ts2, output=("number", 1))

plt.figure()
plt.title("Longest Common Subsequence")
plot_alignment(ts1, ts2, result["Longest Common Subsequence"]["path"][0])
```

[1] M. Vlachos, G. Kollios and D. Gunopulos, "Discovering similar multidimensional trajectories," Proceedings 18th International Conference on Data Engineering, 2002, pp. 673-684, doi: 10.1109/ICDE.2002.994784.



2.2.3 Time Warp Edit Distance (TWED)

Time warp edit distance (TWED) uses sequences' samples indexes/timestamps difference to linearly penalize the matching of samples for which indexes/timestamps values are too far and to favor the matching samples for which indexes/timestamps values are closed. Contrarily to other elastic measures, TWED entails a time shift tolerance controlled by the stiffness parameter of the measure. Moreover, it involves a second parameter defining a constant penalty for insert or delete operations. If stiffness > 0, TWED is a distance (i.e., verifies the triangle inequality) in both space and time[2].

TWED has been used in time series classification assessing classification performance while varying TWED input parameters[2],[3]. In the example, we calculate TWED between two time series varying its parameters.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from tssearch.distances.compute_distance import time_series_distance
```

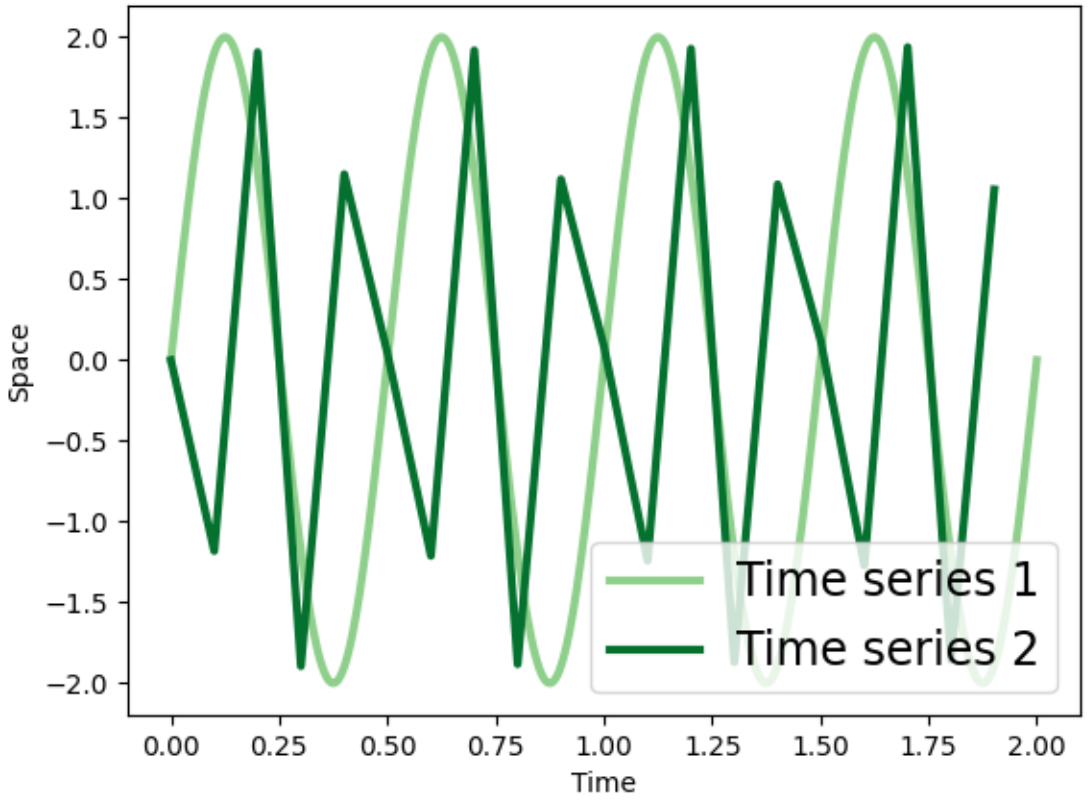
(continues on next page)

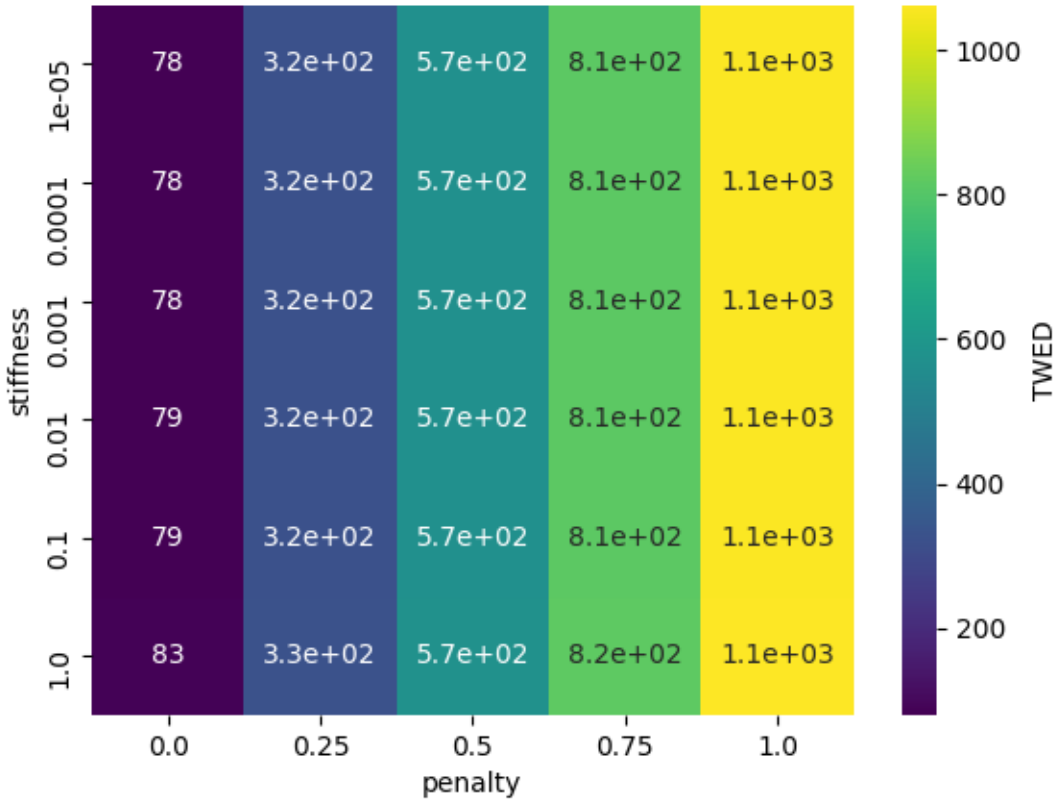
[2] P. Marteau, "Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 2, pp. 306-318, Feb. 2009, doi: 10.1109/TPAMI.2008.76.

[3] Joan Serrà, Josep Ll. Arcos, An empirical evaluation of similarity measures for time series classification, Knowledge-Based Systems, Volume 67, 2014, Pages 305-314, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2014.04.035.

(continued from previous page)

```
# generates signals
freq = 2
amp = 2
time = np.linspace(0, 2, 1000)
ts1 = amp * np.sin(2 * np.pi * freq * time)
ts2 = amp * np.sin(6 * np.pi * freq * time)[::50]

# visualize original and downsampled sequence
plt.figure()
plt.plot(time, ts1, color=sns.color_palette("Greens")[2], label="Time series 1", lw=3.
↪)
plt.plot(time[::50], ts2, color=sns.color_palette("Greens")[5], label="Time series 2",
↪  lw=3.)
plt.ylabel('Space')
plt.xlabel('Time')
plt.legend(fontsize=17, loc="lower right")

stiffness = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]
penalty = [0, .25, .5, .75, 1.0]

distance = list()
for s in stiffness:
    for p in penalty:
        # calculate distances
        dict_distances = {
                "elastic": {"Time Warp Edit Distance": {"multivariate": "no",
                                                        "description": "",
                                                        "function": "twed",
                                                        "parameters": {"nu": s, "lmbda
↪": p, "p": 2, "time": "true"},
                                                        "use": "yes"}}}

        distance.append({'stiffness': s,
                         'penalty': p,
                         'distance': time_series_distance(dict_distances,
                                                          ts1, ts2,
                                                          time, time[::50]).
↪values[0][0]})

df = pd.DataFrame(distance)
df_pivot = df.pivot("stiffness", "penalty", "distance")

plt.figure()
sns.heatmap(df_pivot, annot=True, cbar_kws={'label': "TWED"}, cmap="viridis")
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Space
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Time
Time series 1
Time series 2



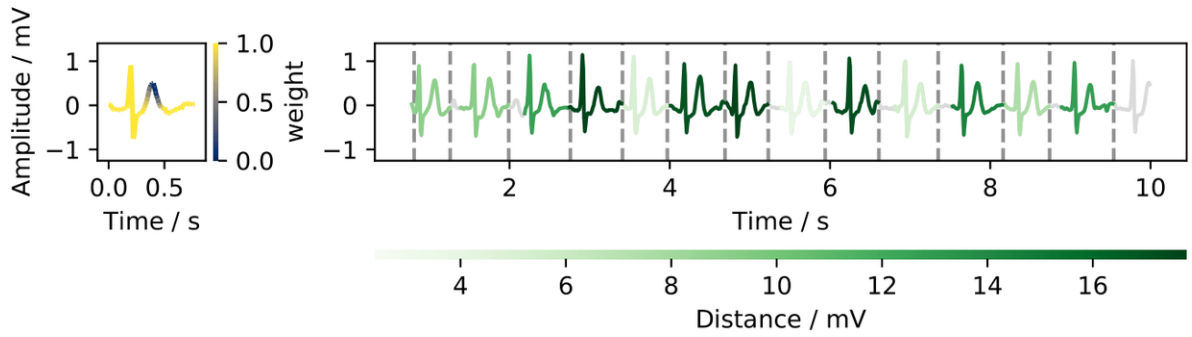
2.3 Segmentation and Search

2.3.1 Segmentation

The *time_series_segmentation* locates the time instants between consecutive query repetitions on a longer and repetitive sequence. You will need to define the distance used for segmentation and provide a query and a sequence as inputs to *time_series_segmentation*, as follows:

```
import tssearch

data = tssearch.load_ecg_example()
cfg = tssearch.get_distance_dict(["Dynamic Time Warping"])

out = tssearch.time_series_segmentation(cfg, data["query"], data["sequence"], data[
↪"weight"])
```

In the code above a ten-second segment from an electrocardiography record is used to define the query and the sequence and the DTW is defined as the distance for the segmentation. Then, the segmentation is calculated and the output is assigned to a variable. The method receives as inputs the configuration file, the query, and the sequence. Additionally, an optional vector input that assigns weights for each time instance of the query is also given as input.



In this example, the specified weights vector assigned less contribution to the second local maxima of the ECG (T wave).

If you are interested in further characterizing each subsequence, this could be accomplished using the distances values calculated for each segment and/or using TSFEL to extract temporal, statistical, and spectral features as data representations for classification algorithms.

2.3.2 Search

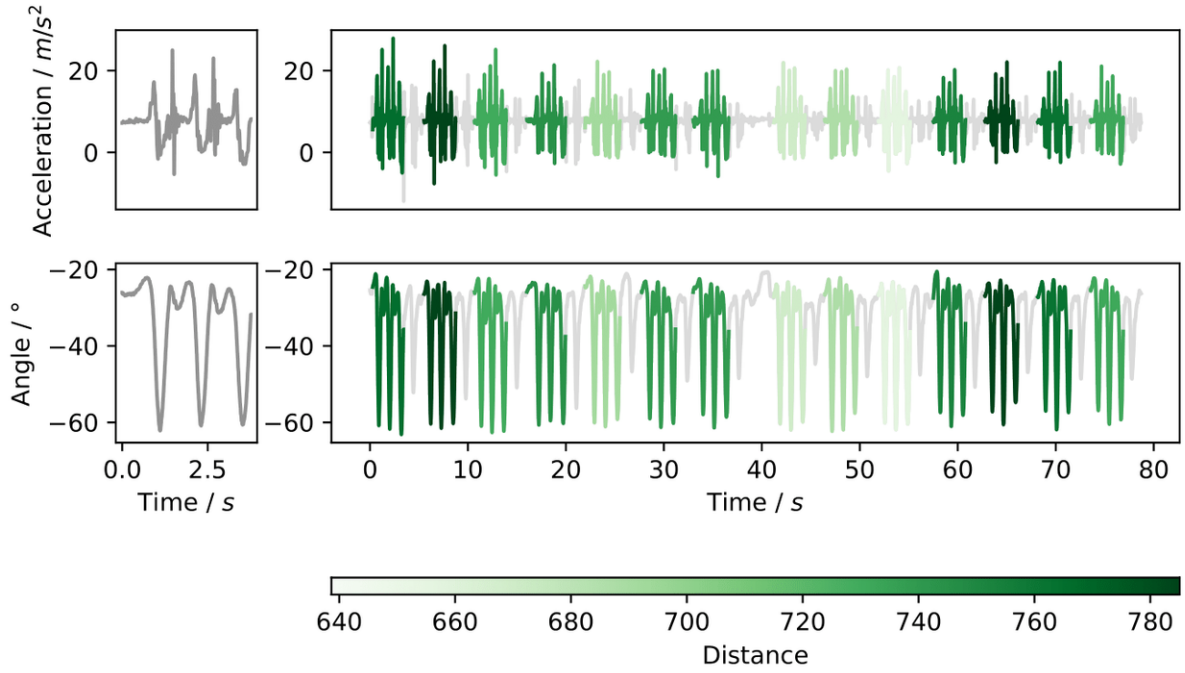
The *time_series_search* method locates the k-best occurrences of a given query on a longer sequence based on a distance measurement. By default, k is set to retrieve the maximum number of matches. The user can also explicitly define the value of k to retrieve the k-best occurrences.

An illustrative example is provided below:

```
import tssearch
import numpy as np

query = np.loadtxt("query.txt")
sequence = np.loadtxt("sequence.txt")

cfg = tssearch.get_distance_dict(["Dynamic Time Warping"])
cfg['elastic']['Dynamic Time Warping']['parameters']['alpha'] = 0.5

out = tssearch.time_series_search(cfg, query, sequence)
```

In the above code, the DTW with an additional parameter α that weights the contribution between the cost in the amplitude and its first derivative is defined. Then, the query search is calculated, and the output is assigned to a variable. The method receives as inputs the configuration file, the query, and the sequence. Since the number of matches is not defined, the method retrieves the maximum number of matches.

To illustrate this example, a wearable sensor-based human activity dataset with multidimensional data was used and the following visualization was obtained:



2.4 Module Reference

2.4.1 Search

query_search

tssearch.search.query_search.**time_series_search**(*dict_distances*, *query*, *sequence*, *tq=None*, *ts=None*, *weight=None*, *output=('number', 1)*)

Time series search method locates the k-best occurrences of a given query on a more extended sequence based on a distance measurement.

Parameters

- **dict_distances** (*dict*) – Configuration file with distances.
- **query** (*nd-array*) – Query time series.
- **sequence** (*nd-array*) – Sequence time series.
- **tq** (*nd-array*) – Time stamp time series query.
- **ts** (*nd-array*) – Time stamp time series sequence.
- **weight** (*nd-array (Default: None)*) – query weight values.
- **output** (*tuple*) – number of occurrences.

Returns **distance_results** – time instants, optimal alignment path and distance for each occurrence per distance.

Return type dict

segmentation

tssearch.search.segmentation.**time_series_segmentation**(*dict_distances*, *query*, *sequence*, *tq=None*, *ts=None*, *weight=None*)

Time series segmentation locates the time instants between consecutive query repetitions on a more extended and repetitive sequence.

Parameters

- **dict_distances** (*dict*) – Configuration file with distances
- **query** (*nd-array*) – Query time series.
- **sequence** (*nd-array*) – Sequence time series.
- **tq** (*nd-array*) – Time stamp time series query.
- **ts** (*nd-array*) – Time stamp time series sequence.
- **weight** (*nd-array (Default: None)*) – query weight values

Returns **segment_results** – Segmented time instants for each given distances

Return type dict

utils

tssearch.search.search_utils.**elastic_search**(*dict_distances*, *query*, *sequence*, *tq=None*, *ts=None*, *weight=None*)

Query search for elastic measures

Parameters

- **dict_distances** (*dict*) – Configuration file with distances
- **query** (*nd-array*) – Query time series.
- **sequence** (*nd-array*) – Sequence time series.
- **tq** (*nd-array*) – Time stamp time series query.
- **ts** (*nd-array*) – Time stamp time series sequence.
- **weight** (*nd-array (Default: None)*) – query weight values

Returns

- **distance** (*nd-array*) – distance value between query and sequence
- **ac** (*nd-array*) – accumulated cost matrix

tssearch.search.search_utils.**lockstep_search**(*dict_distances*, *query*, *sequence*, *weight*)

Query search for lockstep measures

Parameters

- **dict_distances** (*dict*) – Configuration file with distances
- **query** (*nd-array*) – Query time series.
- **sequence** (*nd-array*) – Sequence time series.
- **weight** (*nd-array (Default: None)*) – query weight values

Returns **res** – distance value between query and sequence

Return type nd-array

tssearch.search.search_utils.**start_sequences_index**(*distance*, *output=('number', 1)*, *overlap=1.0*)

Method to retrieve the k-best occurrences from a given vector distance

Parameters

- **distance** (*nd-array*) – distance values
- **output** (*tuple*) – number of occurrences
- **overlap** (*float*) – minimum distance between occurrences

Returns **id_s** – indexes of k-best occurrences

Return type nd-array

2.4.2 Distances

Lockstep Distances

tssearch.distances.lockstep_distances.**braycurtis_distance**(*x*, *y*, *weight=None*)

Computes the Braycurtis distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Braycurtis distance value.

Return type float

tssearch.distances.lockstep_distances.**canberra_distance**(*x*, *y*, *weight=None*)

Computes the Canberra distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Canberra distance value.

Return type float

tssearch.distances.lockstep_distances.**chebyshev_distance**(*x*, *y*, *weight=None*)

Computes the Chebyshev distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.

- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Chebyshev distance value.

Return type float

tssearch.distances.lockstep_distances.**correlation_distance**(*x*, *y*, *weight=None*)
Computes the correlation distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Correlation distance value.

Return type float

tssearch.distances.lockstep_distances.**cosine_distance**(*x*, *y*, *weight=None*)
Computes the cosine distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Cosine distance value.

Return type float

tssearch.distances.lockstep_distances.**euclidean_distance**(*x*, *y*, *weight=None*)
Computes the Euclidean distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Euclidean distance value.

Return type float

tssearch.distances.lockstep_distances.**hamming_distance**(*x*, *y*, *weight=None*)
Computes the Hamming distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Hamming distance value.

Return type float

tssearch.distances.lockstep_distances.**mahalanobis_distance**(*x*, *y*, *weight=None*)
Computes the Mahalanobis distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Mahalanobis distance value.

Return type float

tssearch.distances.lockstep_distances.**manhattan_distance**(*x*, *y*, *weight=None*)
Computes the Manhattan distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Manhattan distance value.

Return type float

tssearch.distances.lockstep_distances.**minkowski_distance**(*x*, *y*, *weight=None*, *p=3*)
Computes the Minkowski distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.
- **p** (*int*) – Lp norm distance degree.

Returns Minkowski distance value.

Return type float

tssearch.distances.lockstep_distances.**pearson_correlation**(*x*, *y*, *beta=None*)
Computes the Pearson correlation between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **beta** (*float*) – Beta coefficient.

Returns Pearson correlation value.

Return type float

tssearch.distances.lockstep_distances.**short_time_series_distance**(*x*, *y*, *tx=None*, *ty=None*)

Computes the short time series distance (STS) between two time series.

Reference: Möller-Levet, C. S., Klawonn, F., Cho, K., and Wolkenhauer, O. (2003).

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **tx** (*nd-array*) – Sampling index of time series x.
- **ty** (*nd-array*) – Sampling index of time series y.

Returns Short time series distance value.

Return type float

tssearch.distances.lockstep_distances.**sqeuclidean_distance**(*x*, *y*, *weight=None*)

Computes the squared Euclidean distance between two time series.

If the time series do not have the same length, an interpolation is performed.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.
- **weight** (*nd-array (Default: None)*) – query weight values.

Returns Squared Euclidean distance value.

Return type float

Elastic Distances

tssearch.distances.elastic_distances.**dlp**(*x*, *y*, *p=2*)

Computes Lp norm distance between two time series.

Parameters

- **x** (*nd-array*) – Time series x (query).
- **y** (*nd-array*) – Time series y.
- **p** (*int*) – Lp norm distance degree for local cost computation.

Returns

Return type The Lp distance.

tssearch.distances.elastic_distances.**dtw**(*x*, *y*, *weight=None*, ***kwargs*)

Computes Dynamic Time Warping (DTW) of two time series.

Parameters

- **x** (*nd-array*) – Time series x (query).
- **y** (*nd-array*) – Time series y.

- **dist** (*function*) – The distance used as a local cost measure. None defaults to the squared euclidean distance.
- ****kwargs** –
- **below** (*See*) –
- **do_sign_norm (bool) --** (***) – If `True` the signals will be normalized before computing the DTW, (default: `False`)
- **do_dist_norm (bool) --** (***) – If `True` the DTW distance will be normalized by dividing the summation of the path dimension. (default: `True`)
- **window (String) --** (***) – Selects the global window constrains. Available options are `None` and `sakoe-chiba`. (default: `None`)
- **factor (Float) --** (***) – Selects the global constrain factor. (default: `min(xl, yl) * .50`)

Returns

- **d** (*float*) – The DTW distance.
- **ac** (*nd-array*) – The accumulated cost matrix.
- **path** (*nd-array*) – The optimal warping path between the two sequences.

tssearch.distances.elastic_distances.**lcss**(*x*, *y*, *eps=1*, ***kwargs*)

Computes the Longest Common Subsequence (LCSS) distance between two numeric time series.

Parameters

- **x** (*nd-array*) – Time series x (query).
- **y** (*nd-array*) – Time series y.
- **eps** (*float*) – Amplitude matching threshold.
- ****kwargs** –
- **below** (*See*) –
- **window (String) --** (***) – Selects the global window constrains. Available options are `None` and `sakoe-chiba`. (default: `None`)

Returns

- **d** (*float*) – The LCSS distance.
- **ac** (*nd-array*) – The similarity matrix.
- **path** (*nd-array*) – The optimal path between the two sequences.

tssearch.distances.elastic_distances.**twed**(*x*, *y*, *tx*, *ty*, *nu=0.001*, *lmbda=1.0*, *p=2*, *report='distance'*)

Computes Time Warp Edit Distance (TWED) of two time series.

Reference : Marteau, P.; F. (2009). "Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (2): 306–318. arXiv:cs/0703033 http://people.irisa.fr/Pierre-Francois.Marteau/

Parameters

- **x** (*nd-array*) – Time series x (query).
- **y** (*nd-array*) – Time series y.
- **tx** (*nd-array*) – Time stamp time series x.

- **ty** (*nd-array*) – Time stamp time series y.
- **nu** (*int*) –

  Stiffness parameter (nu >= 0) nu = 0, TWED distance measure on amplitude. nu > 0, TWED distance measure on amplitude x time.
- **lmbda** (*int*) – Penalty for deletion operation (lmbda >= 0).
- **p** (*int*) – Lp norm distance degree for local cost computation.
- **report** (*str*) – distance, cost matrix, path.

Returns

- **d** (*float*) – The TWED distance.
- **ac** (*nd-array*) – The accumulated cost matrix.
- **path** (*nd-array*) – The optimal warping path between the two sequences.

Time Distances

tssearch.distances.time_distances.**tam**(*x*, *y*)

Calculates the Time Alignment Measurement (TAM) based on an optimal warping path between two time series.

Reference: Folgado et. al, Time Alignment Measurement for Time Series, 2016.

Parameters

- **x** (*nd-array*) – Time series x.
- **y** (*nd-array*) – Time series y.

Returns

- In case `report=instants` the number of indexes in advance, delay and phase
- *will be returned.*
- For `report=ratios`, the ratio of advance, delay and phase.
- will be returned. In case `report=distance`, only the TAM will be returned.

2.5 Authors

This package is being developed and maintained by Fraunhofer AICOS.

Fraunhofer PORTUGAL

TSSEARCH was written in collaboration with Cognitive Systems Lab of University of Bremen.

2.5.1 Lead Development Team

- Duarte Folgado (duarte.folgado@fraunhofer.pt)
- Hugo Gamboa (hugo.gamboa@fraunhofer.pt)
- Marília Barandas (marilia.barandas@fraunhofer.pt)
- Maria Lua Nunes (maria.nunes@fraunhofer.pt)

- Margarida Antunes (maria.antunes@fraunhofer.pt)

2.5.2 Contributors

- Hui Liu
- Tanja Schultz
- Yale Hartmann

2.6 Changelog

2.6.1 Version 0.1.3

- Fixed a bug on the setup.py to correctly build the PyPI package
- Removed novainstrumentation from dependencies
- Fixed a bug on TWED distance (#7)

2.6.2 Version 0.1.0

- Release of TSSEARCH with documentation

2.7 License

```
BSD 3-Clause License

Copyright (c) 2022, Associação Fraunhofer Portugal Research
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
```

(continues on next page)

(continued from previous page)

```
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER 3

Installation

This packages is available on PyPI:

```
$ pip install tssearch
```

CHAPTER 4

Get started

The code below segments a 10 s electrocardiography record:

```
import tssearch

# Load the query, (optional) weight vector and sequence
data = tssearch.load_ecg_example()

# Selects the Dynamic Time Warping (DTW) as the distance for the segmentation
cfg = tssearch.get_distance_dict(["Dynamic Time Warping"])

# Performs the segmentation
out = tssearch.time_series_segmentation(cfg, data['query'], data['sequence'], data[
↪'weight'])
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index