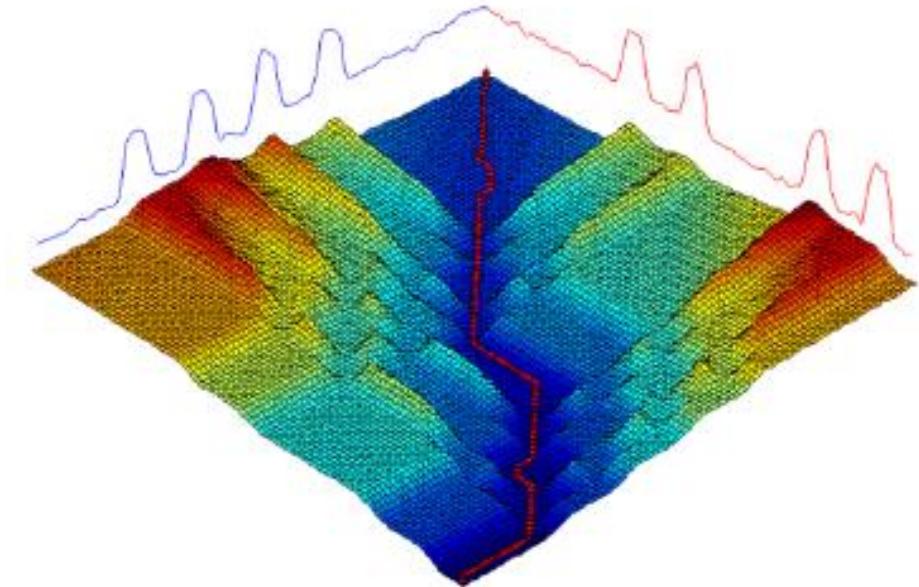


Let's do the time warp again: time series machine learning with distance functions



Tony Bagnall

A.J.Bagnall@soton.ac.uk
School of Electronics and Computer Science
University of Southampton

Chris Holder

c.holder@uea.ac.uk
School of Computing Sciences
University of East Anglia



<https://www.aeon-toolkit.org/>

```
(venv) PS C:\Code\aeon> pip install aeon
```

Talk Structure

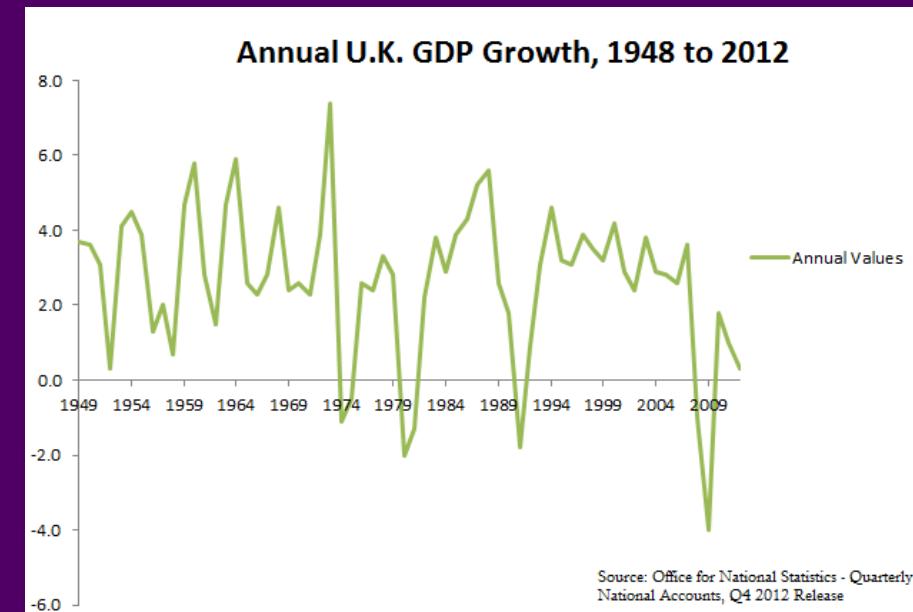
- Motivation and background: time series machine learning
- Time series distance functions
- Using aeon distance functions
- Distance based clustering

Some images are taken from talks by Eamonn Keogh, with permission

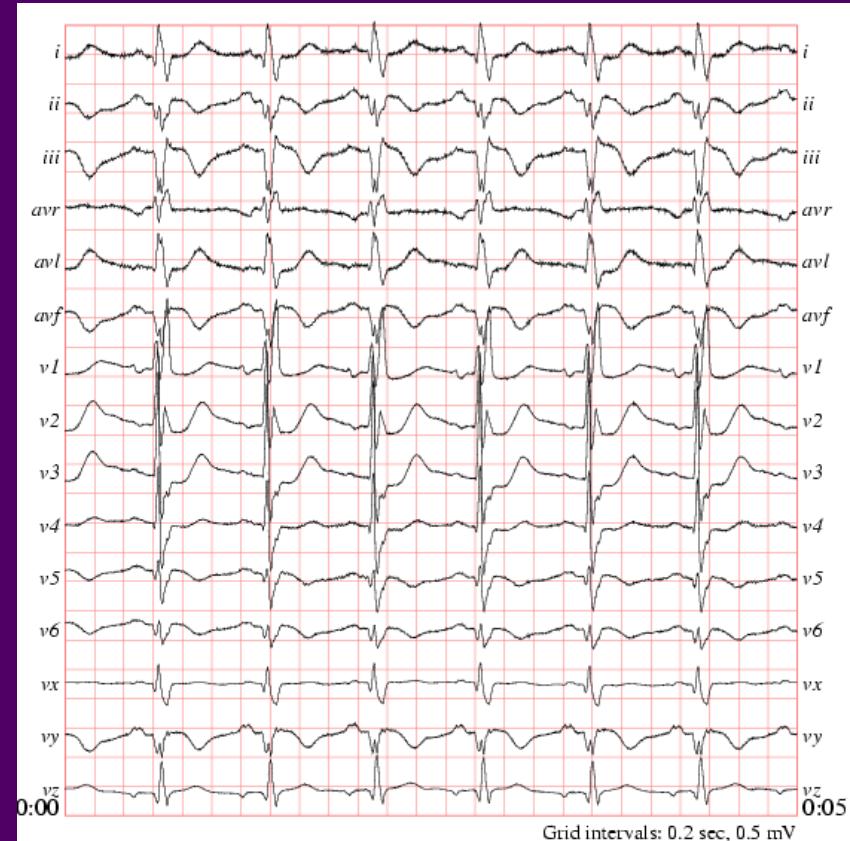
What is a time series?

A time series is an **ordered** list of observations of **real valued** variable

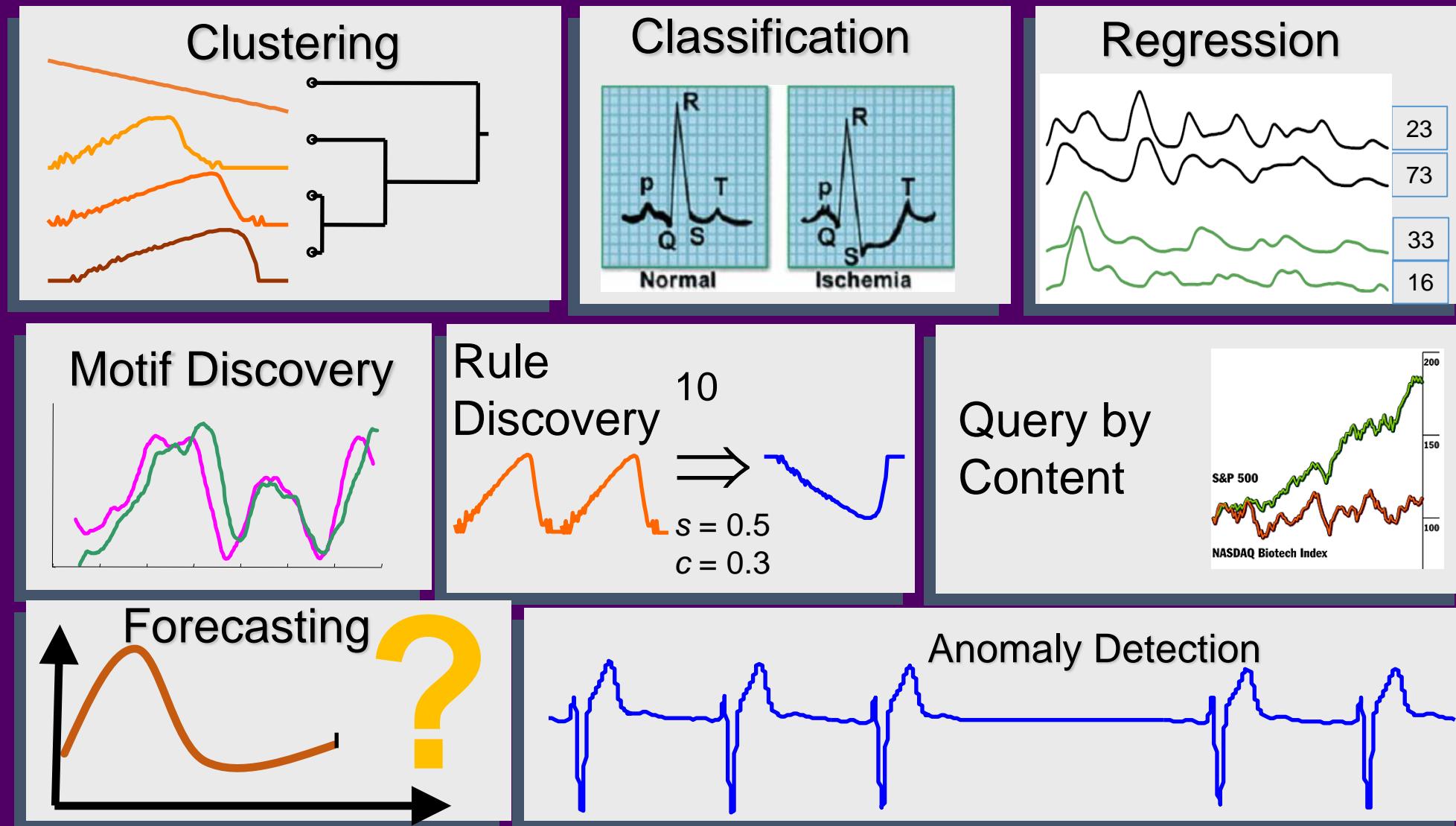
If each observation is a scalar, we call it a univariate time series



If each observation is a vector of values we call it a multivariate time series



Time Series Machine Learning Tasks

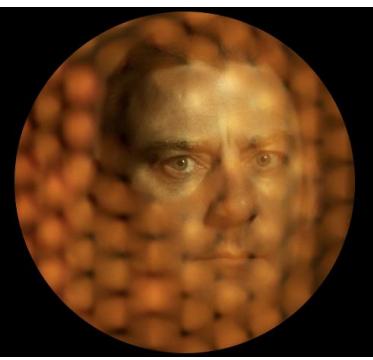


Time Series Machine Learning Repository

www.timeseriesclassification.com

Introduced in 2002 and expanded several times since, the archive datasets have been used in thousands of papers

Expanded in 2018 to 128 datasets
Multivariate Archive introduced in 2018 with 30 datasets



Time Series Classification Home Datasets Algorithms Results Researchers Code Bibliography UEA Papers ▾

Welcome to the Time Series Machine Learning Website

This site contains data, reference results and links to code for Time Series Classification (TSC), Time Series Clustering (TSCL) and Time Series Extrinsic Regression (TSER)

The classification data form part of the [UCR time series archive](#). Regression data are part of the [TSER repository](#). We thank everyone else involved in donating to and maintaining these archives.

The scikit-learn compatible [aeon toolkit](#) contains the state of the art algorithms for time series classification. All of the datasets and results stored here are directly accessible in code using aeon.

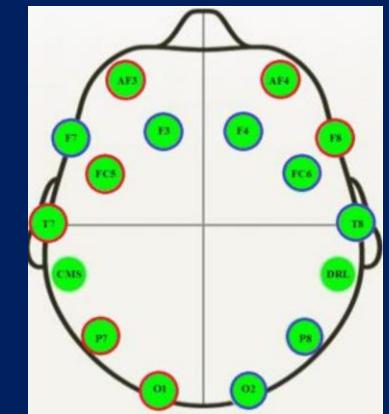
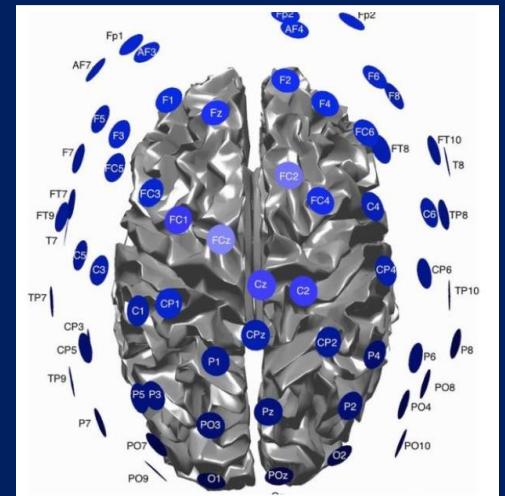
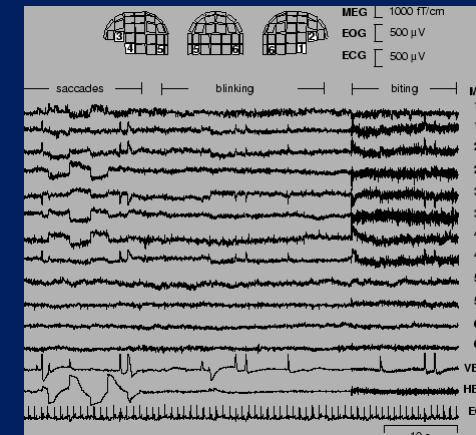
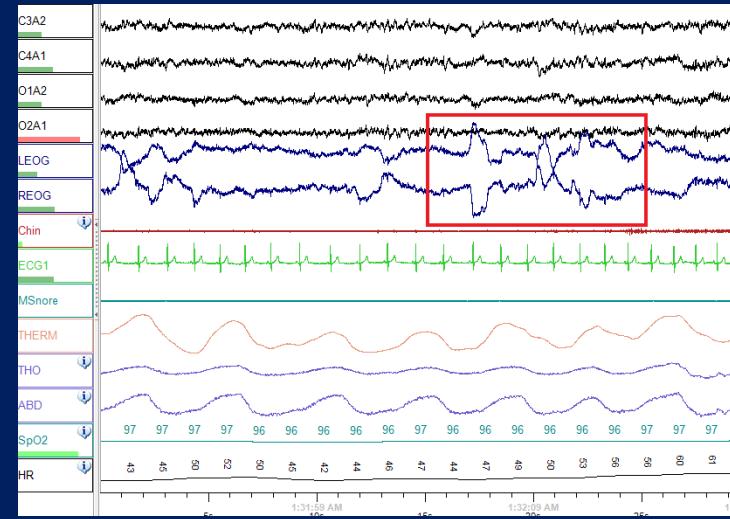
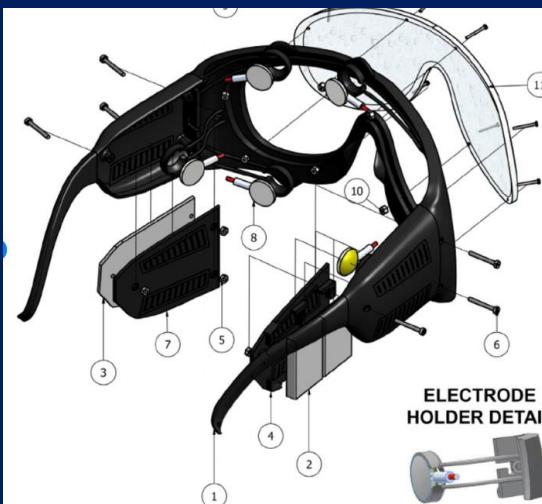
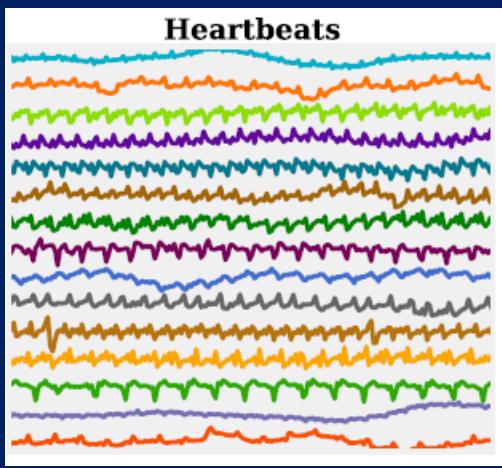
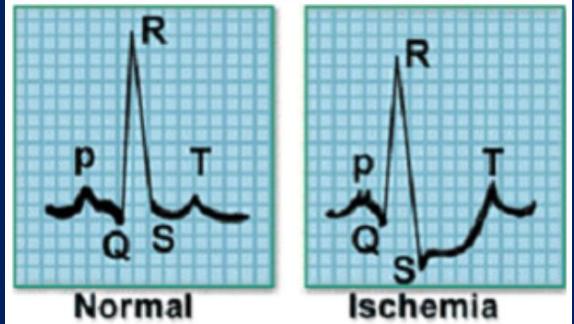
Checkout our [GitHub](#), join the [aeon slack](#) and follow aeon [on twitter](#).

The Advanced Analytics and Learning on Temporal Data (AALTD) workshop is back for the eighth time on 18th Sept in Turin. The workshop is part of the ECML/PKDD conference and selected papers will be published in LNCS. See, for example, [2022](#) and [2021](#) versions.

We would like to thank everyone who donates and helps maintain these archives

Example Data: Biomedical signals

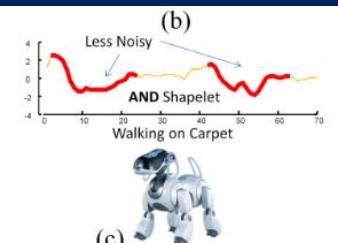
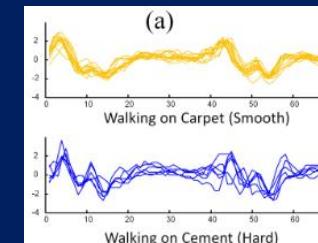
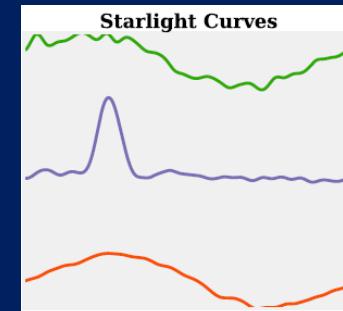
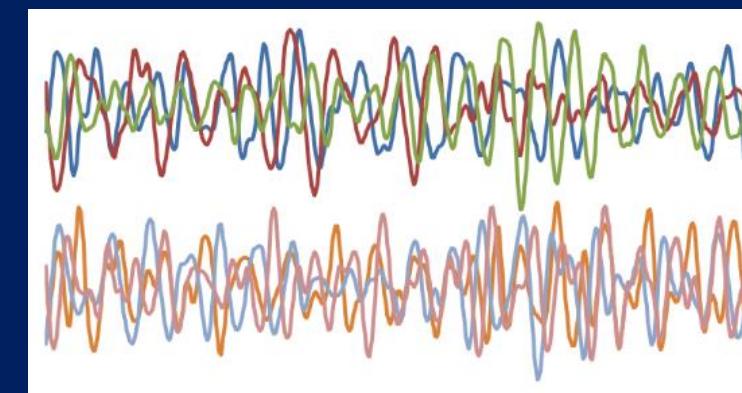
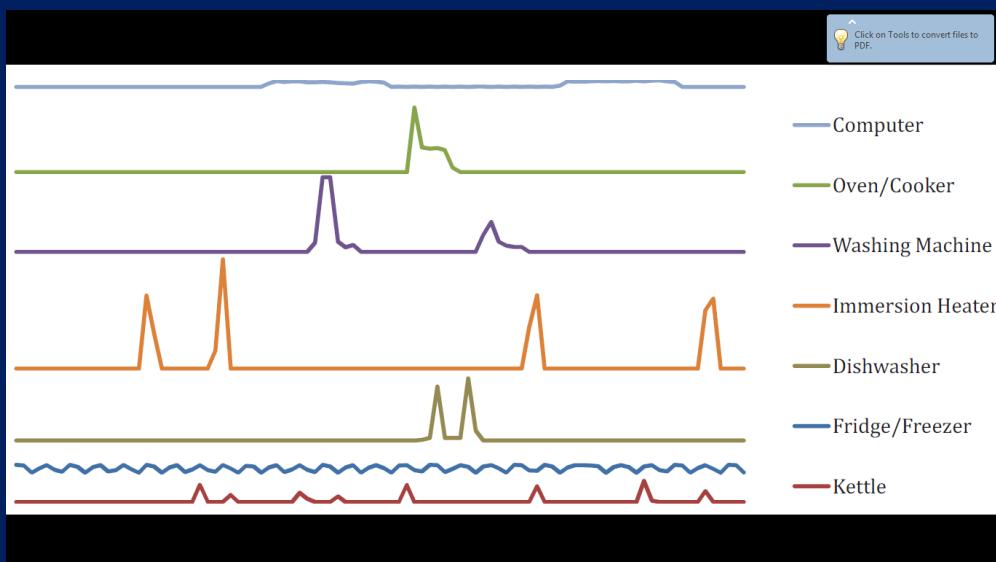
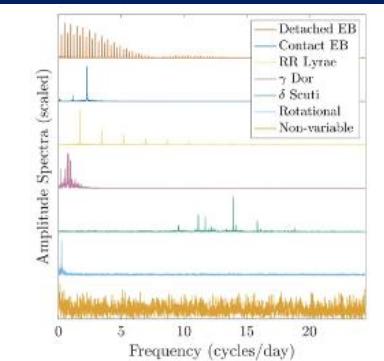
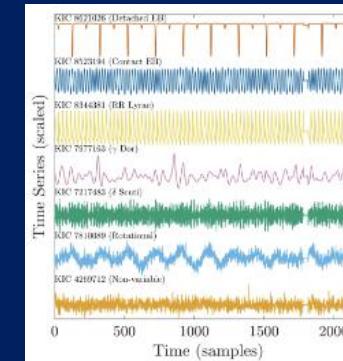
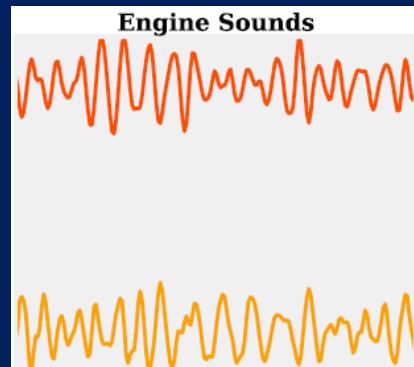
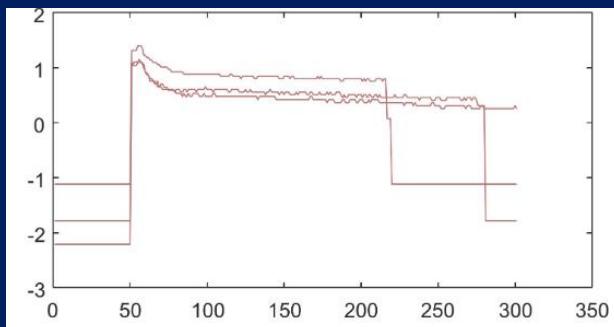
ECG, EEG, MEG, EOG



Example Data: Sensor Data

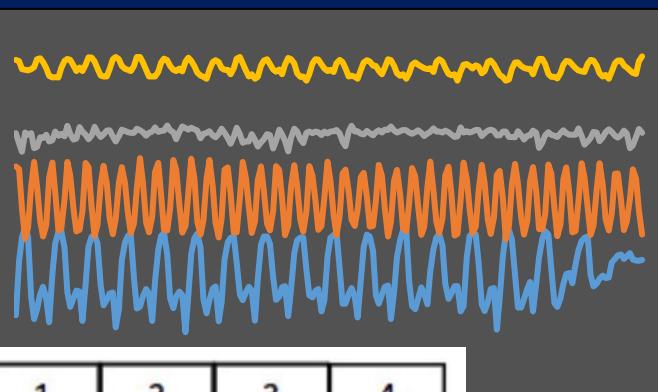
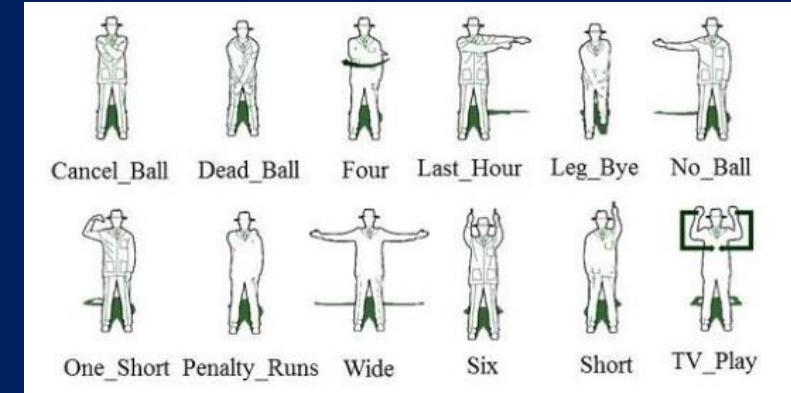
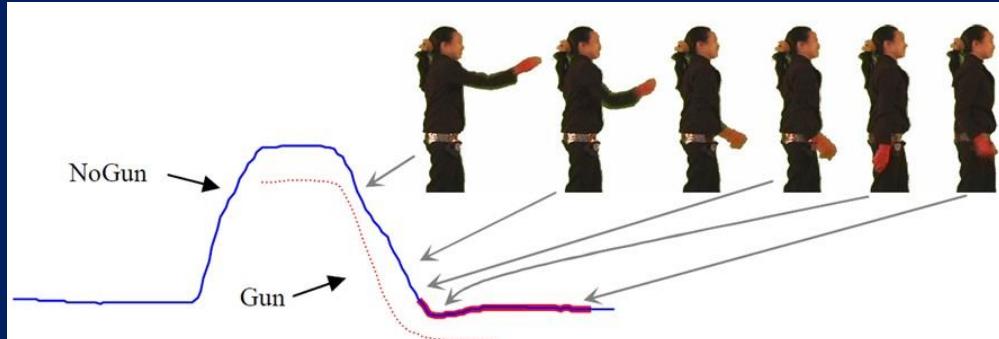
Car engines, light curves, lightning, electric devices

Insect wing beats, Car Engines, Phonemes (sound), Worm Motion



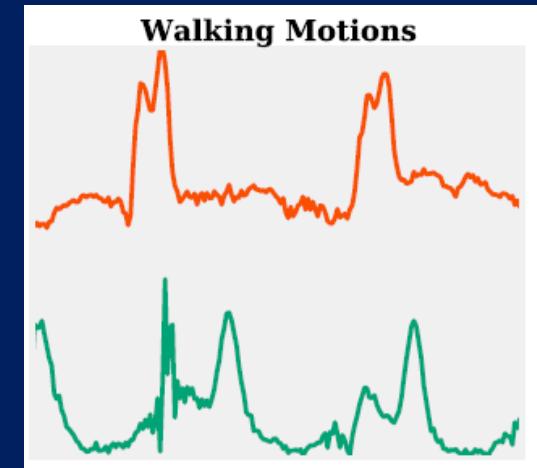
Example Data: Human Activity Recognition

Gestures (Uwave), Cricket hand signals, Gun Point, Asphalt road condition, inline skating



1	2	3	4
>	↔	→→	←→
5	6	7	8
↑↓	↓↑	Q	Q

Measuring how similar time series are is a primitive operation for time series ML



Similarity between things

similarity can be subjective and hard to define

Sometimes it's easy to describe similarity:

Sometimes similarity is more difficult to describe...

Similarity is measured by a **distance function** that takes two objects and returns a scalar, the inverse of which measures similarity

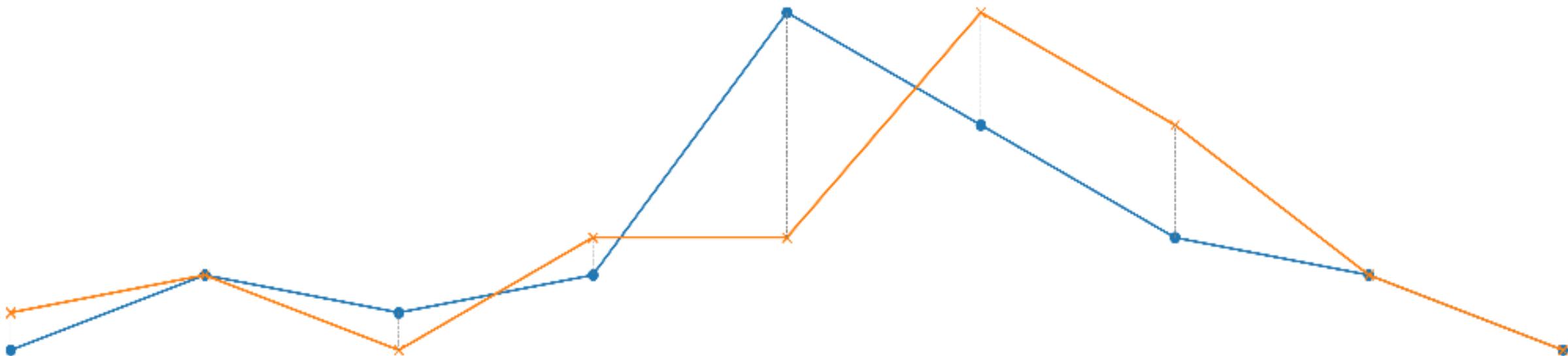


Similarity between time series

Given two time series: The Euclidean Distance is defined as

$$\begin{aligned}\mathbf{x} &= x_1 \dots x_n \\ \mathbf{y} &= y_1 \dots y_n\end{aligned}$$

$$D(x, y) \equiv \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$



Other distances for continuous variables

The Minkowski metric (a generalisation of Euclidean Distance)

$$d(a, b) = \sqrt[n]{\sum_{i=1}^n (a_i - b_i)^n}$$

If n=1, this is referred to as the Manhattan metric.

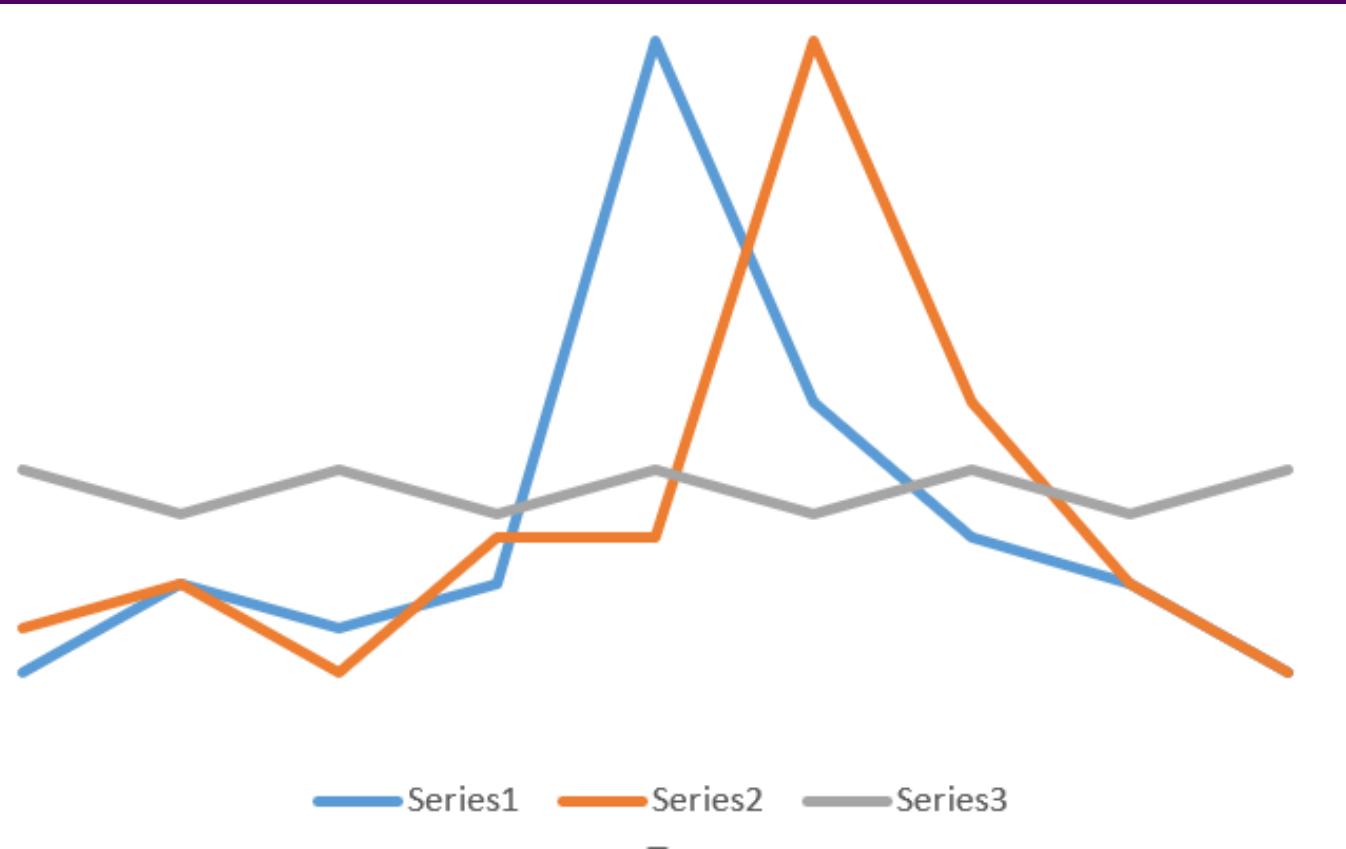
$$d(a, b) = \sum |a_i - b_i|$$

Correlation Distance

$$d(a, b) = \frac{\sum (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum (a_i - \bar{a})^2} \sum (b_i - \bar{b})^2}$$

You could shuffle indexes and all of these distances would be the same

Need for elastic distances



Is series 1 more
like series 2 or
series 3?

```
from aeon.distances import euclidean_distance
x = euclidean_distance(first, second)
print(x * x)
```

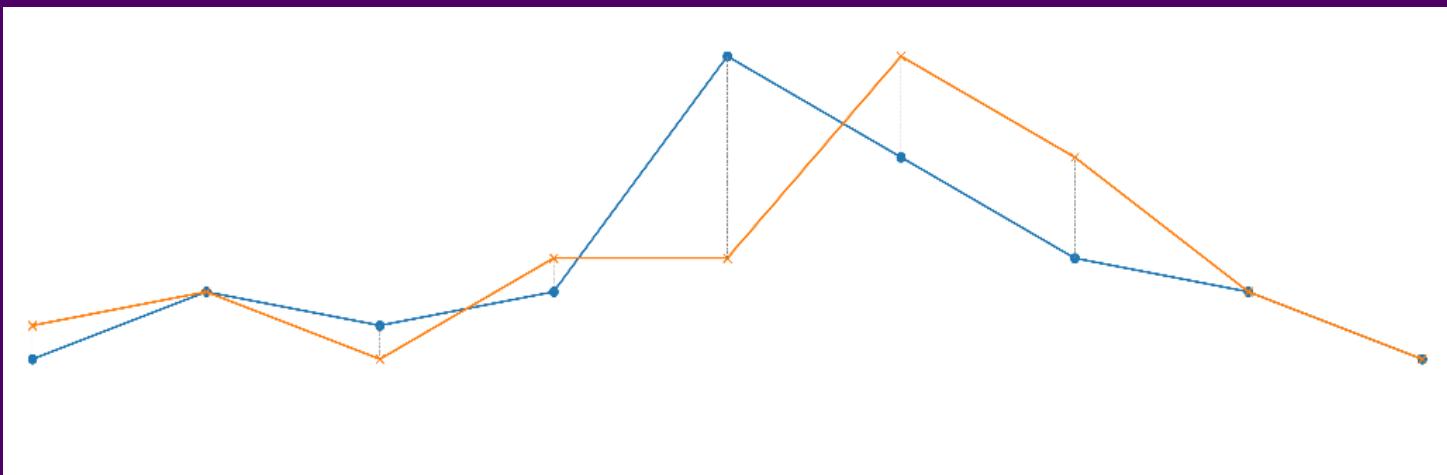
$$\text{ed_dist}(s1, s2) = 14.03 \quad \text{ed_dist}(s1, s3) = 12.58$$

Need for elastic distances

	2	3	1	4	4	15	7	3	1
1	1	4	0	9	9	196	36	4	0
3	1	0	4	1	1	144	16	0	4
2	0	1	1	4	4	169	25	1	1
3	1	0	4	1	1	144	16	0	4
15	169	144	196	121	121	0	64	144	196
7	25	16	36	9	9	64	0	16	36
4	4	1	9	0	0	121	9	1	9
3	1	0	4	1	1	144	16	0	4
1	1	4	0	9	9	196	36	4	0

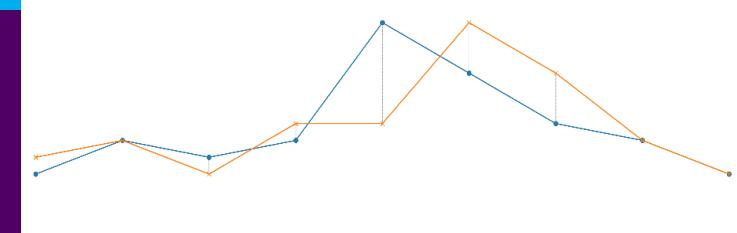


Pointwise distance
between series,
called M



```
from aeon.distances import euclidean_distance
x = euclidean_distance(first, second)
print(x * x)
```

Need for elastic distances



The series are similar if we think the offset is not important.

This path has a distance of 5, compared to 57 for squared Euclidean

What if we could “warp” the series to realign the peaks?

This is dynamic time warping

	2	3	1	4	4	15	7	3	1
1	1	4	0	9	9	196	36	4	0
3	1	0	4	1	1	144	16	0	4
2	0	1	1	4	4	169	25	1	1
3	1	0	4	1	1	144	16	0	4
15	169	144	196	121	121	0	64	144	196
7	25	16	36	9	9	64	0	16	36
4	4	1	9	0	0	121	9	1	9
3	1	0	4	1	1	144	16	0	4
1	1	4	0	9	9	196	36	4	0

Dynamic time warping

DTW finds the path through the pointwise distance matrix that minimizes the total distance

```
1: Let  $C$  be an  $(m + 1) \times (m + 1)$  matrix initialised to zero, indexed from zero.  
2: for  $i \leftarrow 1$  to  $m$  do  
3:   for  $j \leftarrow 1$  to  $m$  do  
4:      $C_{i,j} \leftarrow M_{i,j} + \min(C_{i-1,j-1}, C_{i-1,j}, C_{i,j-1})$   
return  $C_{m,m}$ 
```

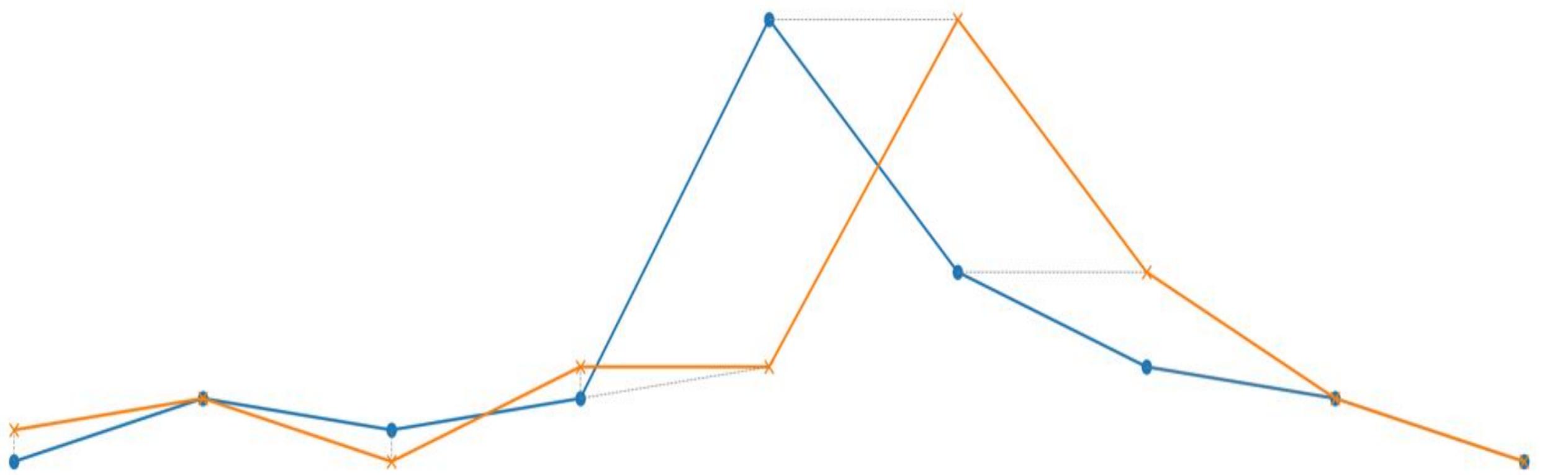
	2	3	1	4	4
1	1	4	0	9	9
3	1	0	4	1	1
2	0	1	1	4	4
3	1	0	4	1	1
15	169	144	196	121	169

M=

C=

	2	3	1	4	4
1	1	5	5	14	23
3	1	1	5	6	7
2	1	2	3	7	10
3	1	2	6	4	5
15	170	145	198	122	173

$$\text{dtw_dist}(s1, s2) = 5.0 \quad \text{dtw_dist}(s1, s3) = 151.0$$



```
from aeon.distances import dtw_distance, dtw_alignment_path
x = dtw_distance(first, second)
path = dtw_alignment_path(first, second)
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6), (7, 7), (8, 8)], 5.0
```

DTW is very popular

<https://medium.com/trusted-data-science-haleon/unleashing-the-power-of-real-time-machine-learning-to-accelerate-the-production-of-toothpaste-in-70ab7c3cf1f1>

Data Science Model

For step detection (the main steps are mentioned in “Toothpaste production process overview and problem formulation” part) in the toothpaste production process, an approximate **Dynamic Time** Warping method was used (fastDTW). **Dynamic Time** Warping is a widely used algorithm in time series analysis to measure the similarity between two temporal sequences that may vary in time or speed.

Edit Distances

Edit distances are used with discrete series to measure similarity

How similar are the names “Peter” and “Piotr”?

Assume the following cost function

<i>Substitution</i>	1 Unit
<i>Insertion</i>	1 Unit
<i>Deletion</i>	1 Unit

$D(\text{Peter}, \text{Piotr})$ is 3

Peter
Substitution (i for e)
Piter
Insertion (o)
Pioter
Deletion (e)
Piotr

Edit Distances for Time Series

- There are numerous edit based distances for time series
- They move off the diagonal when points are considered different (by some threshold)
- They apply an explicit distance penalty for moving off the diagonal
- We can relate these to DTW through looking at the options slightly differently

DTW as match/insert/delete

```
1: Let  $C$  be an  $(m + 1) \times (m + 1)$  matrix initialised to zero, indexed from zero.  
2: for  $i \leftarrow 1$  to  $m$  do  
3:   for  $j \leftarrow 1$  to  $m$  do  
4:      $C_{i,j} \leftarrow M_{i,j} + \min(C_{i-1,j-1}, C_{i-1,j}, C_{i,j-1})$   
return  $C_{m,m}$ 
```

```
1: Let  $C$  be an  $(m + 1) \times (m + 1)$  matrix initialised to zero,  
2: for  $i \leftarrow 1$  to  $m$  do  
3:   for  $j \leftarrow 1$  to  $m$  do  
4:      $match \leftarrow M_{i,j} + C_{i-1,j-1}$   
5:      $insert \leftarrow M_{i,j} + C_{i-1,j}$   
6:      $delete \leftarrow M_{i,j} + C_{i,j-1}$  ←  
7:      $C_{i,j} \leftarrow \min(match, insert, delete)$   
return  $C_{m,m}$ 
```

Edit distance
approaches apply a
penalty for insert/delete

Edit Distance with Real Penalty

DTW

```
1: Let  $C$  be an  $(m + 1) \times (m + 1)$  matrix initialised to zero,  
2: for  $i \leftarrow 1$  to  $m$  do  
3:   for  $j \leftarrow 1$  to  $m$  do  
4:      $match \leftarrow M_{i,j} + C_{i-1,j-1}$   
5:      $insert \leftarrow M_{i,j} + C_{i-1,j}$   
6:      $delete \leftarrow M_{i,j} + C_{i,j-1}$   
7:      $C_{i,j} \leftarrow \min(match, insert, delete)$   
return  $C_{m,m}$ 
```

ERP

```
1: Let  $C$  be an  $(m + 1) \times (m + 1)$  matrix initialised to zero.  
2: for  $i \leftarrow 1$  to  $m$  do  
3:   for  $j \leftarrow 1$  to  $m$  do  
4:      $match \leftarrow M_{i,j} + C_{i-1,j-1}$   
5:      $insert \leftarrow d(a_i, g) + C_{i-1,j}$   
6:      $delete \leftarrow d(g, b_j) + C_{i,j-1}$   
7:      $C_{i,j} \leftarrow \min(match, insert, delete)$   
return  $C_{m,m}$ 
```

Pointwise
distance to a
constant
instead

Move/Split/Merge

Move is match, split is insert, merge is delete

```
1: Let  $C$  be an  $(m + 1) \times (m + 1)$  matrix initialised to zero,  
2: for  $i \leftarrow 1$  to  $m$  do  
3:   for  $j \leftarrow 1$  to  $m$  do  
4:      $match \leftarrow M_{i,j} + C_{i-1,j-1}$   
5:      $insert \leftarrow cost(a_i, a_{i-1}, b_j, c) + C_{i-1,j}$   
6:      $delete \leftarrow cost(b_j, b_{j-1}, a_i, c) + C_{i,j-1}$   
7:      $C_{i,j} \leftarrow \min(match, insert, delete)$   
return  $C_{m,m}$ 
```

Cost
dependent on
three values

$$cost(x, y, z, c) = \begin{cases} c & \text{if } y \leq x \leq z \text{ or } y \geq x \geq z \\ c + \min(|x - y|, |x - z|) & \text{otherwise.} \end{cases}$$

MSM

MSM Cost function

DTW uses distance between points on insert/delete

ERP uses distance to a constant on insert/delete

MSM combines these

$$cost(x, y, z, c) = \begin{cases} c & \text{if } y \leq x \leq z \text{ or } y \geq x \geq z \\ c + \min(|x - y|, |x - z|) & \text{otherwise.} \end{cases}$$

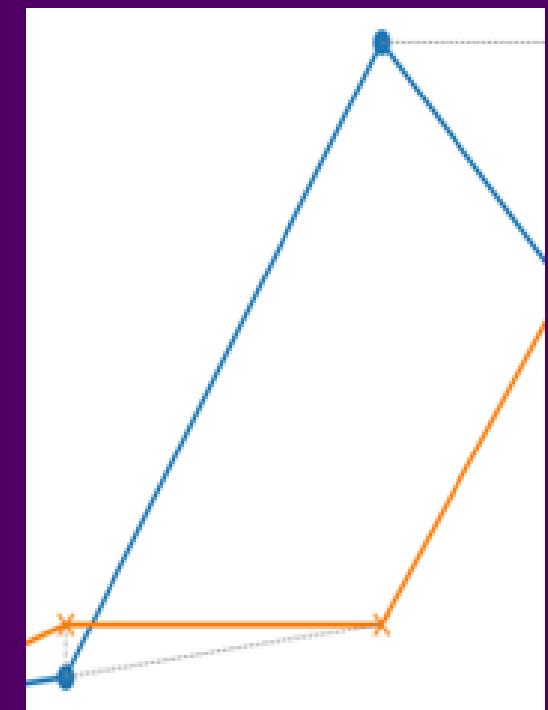
Suppose $a_4=3, a_5=15, b_4=4, b_5=4$

$$\begin{aligned} match &\leftarrow M_{i,j} + C_{i-1,j-1} \\ insert &\leftarrow cost(a_i, a_{i-1}, b_j, c) \\ delete &\leftarrow cost(b_j, b_{j-1}, a_i, c) \end{aligned}$$

Cost match(5,5) = $|15-4| = 9$

Cost insert(5,5) = $\text{cost}(15,3,4) = 1+9=10$

Cost delete(5,5) = $\text{cost}(4,4,15) = 1$



Distance functions in aeon

https://www.aeon-toolkit.org/en/latest/api_reference/distances.html

The screenshot shows a web browser displaying the aeon toolkit's API reference. The left sidebar contains navigation links: Search, USING AEON (Installation, Getting Started, API Reference), and Examples. The main content area is titled "dtw_distance". It includes the function signature: `dtw_distance(x: ndarray, y: ndarray, window: float = None, itakura_max_slope: float = None) → float`, a "[source]" link, and a description: "Compute the DTW distance between two time series." Below this, there is a detailed explanation of DTW and its mathematical formulation. A "A warping path" section is also present. At the bottom right, there is a mathematical expression: $P = \langle (e_1, f_1), (e_2, f_2), \dots, (e_s, f_s) \rangle$. The top right corner of the main content area has three small icons: a gear, a sun, and a list.

Distance functions in aeon

Name	Distance function
dtw	aeon.distance.dtw_distance
ddtw	aeon.distance.ddtw_distance
wdtw	aeon.distance.wdtw_distance
wddtw	aeon.distance.wddtw_distance
erp	aeon.distance.erp_distance
edr	aeon.distance.edr_distance
msm	aeon.distance.msm_distance
twe	aeon.distance.twe_distance
lcss	aeon.distance.lcss_distance
euclidean	aeon.distance.euclidean_distance
squared euclidean	aeon.distance.squared_distance
manhattan	aeon.distance.manhattan_distance

Distance function support

Name	Univariate	Multivariate	Unequal length
dtw	✓	✓	✓
ddtw	✓	✓	✓
wdtw	✓	✓	✓
wddtw	✓	✓	✓
erp	✓	✓	✓
edr	✓	✓	✓
msm	✓	✓	✓
twe	✓	✓	✓
lcss	✓	✓	✓
euclidean	✓	✓	✓
squared euclidean	✓	✓	✓
manhattan	✓	✓	✓

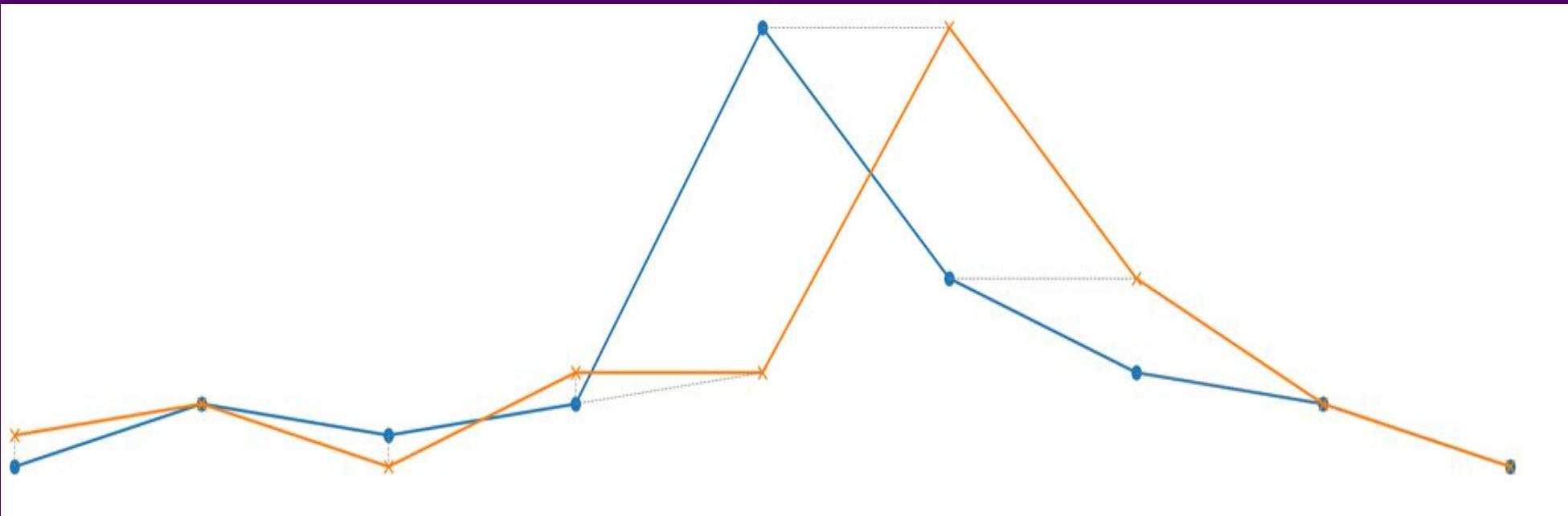
Calling distance functions

```
1 import numpy as np
2 from aeon.distances import dtw_distance, distance
3
4 x = np.array([1, 2, 3, 4, 5])
5 y = np.array([6, 7, 8, 9, 10])
6
7 # Calling a distance directly
8 dtw_distance(x, y)
9 # >> 108.0
10
11 # Calling a distance using utility function
12 # Any value in the table above is a valid metric string
13 distance(x, y, metric='dtw')
14 # >> 108.0
```

Passing parameters to distance functions

```
1 import numpy as np
2 from aeon.distances import msm_distance, distance
3
4 x = np.array([1, 2, 3, 4, 5])
5 y = np.array([6, 7, 8, 9, 10])
6
7 # Calling a distance directly
8 msm_distance(x, y, window=0.2)
9 # >> 2.0
10
11 # Calling a distance using utility function
12 distance(x, y, metric='msm', window=0.2)
13 # >> 2.0
```

Alignment path



```
([(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6), (7, 7), (8, 8)], 5.0)
```

Alignment paths in aeon

Name	Distance function
dtw	aeon.distance.dtw_alignment_path
ddtw	aeon.distance.ddtw_alignment_path
wdtw	aeon.distance.wdtw_alignment_path
wddtw	aeon.distance.wddtw_alignment_path
erp	aeon.distance.erp_alignment_path
edr	aeon.distance.edr_alignment_path
msm	aeon.distance.msm_alignment_path
twe	aeon.distance.twe_alignment_path
lcss	aeon.distance.lcss_alignment_path

Distance function support

Name	Univariate	Multivariate	Unequal length
dtw	✓	✓	✓
ddtw	✓	✓	✓
wdtw	✓	✓	✓
wddtw	✓	✓	✓
erp	✓	✓	✓
edr	✓	✓	✓
msm	✓	✓	✓
twe	✓	✓	✓
lcss	✓	✓	✓

Calling alignment paths

```
1 import numpy as np
2 from aeon.distances import msm_alignment_path, alignment_path
3
4 x = np.array([1, 2, 3, 4, 5])
5 y = np.array([6, 7, 8, 9, 10])
6
7 # Calling a alignment path directly
8 msm_alignment_path(x, y)
9 #>>([(0, 0), (1, 1), (2, 2), (3, 3), (4, 4)], 16.0)
10
11 # Using utility function
12 alignment_path(x, y, metric='msm')
13 #>>([(0, 0), (1, 1), (2, 2), (3, 3), (4, 4)], 16.0)
```

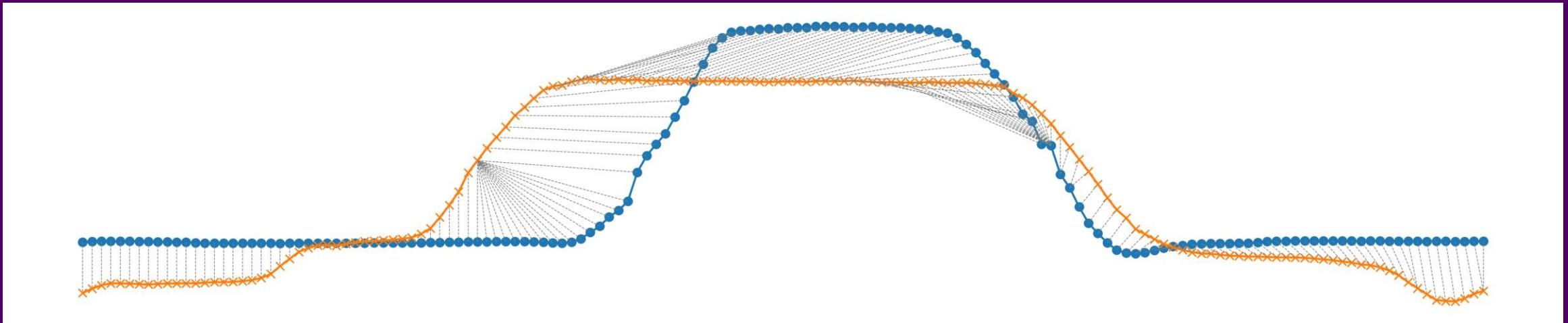
Passing parameters to alignment paths

```
1 import numpy as np
2 from aeon.distances import msm_alignment_path, alignment_path
3
4 x = np.array([1, 2, 3, 4, 5])
5 y = np.array([6, 7, 8, 9, 10])
6
7 # Calling a alignment path directly
8 msm_alignment_path(x, y, window=0.2)
9 #>>([(0, 0), (0, 1), (0, 2), (1, 3), (2, 4), (3, 4), (4, 4)], 2.0)
10
11 # Using utility function
12 alignment_path(x, y, metric='msm', window=0.2)
13 #>>([(0, 0), (0, 1), (0, 2), (1, 3), (2, 4), (3, 4), (4, 4)], 2.0)
```

Plot alignment path

```
● ● ●  
1 import matplotlib.pyplot as plt  
2 from aeon.datasets import load_gunpoint as load_data  
3 from aeon.distances import alignment_path  
4  
5 # Load data  
6 X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")  
7 X_test, y_test = load_data(split="TEST", return_type="numpy2D")  
8  
9 # Choose two random time series  
10 x = X_train[0]  
11 y = X_train[22]  
12  
13 # Compute path  
14 msm_alignment_path = alignment_path(x, y, metric='msm')  
15  
16 # Plot using function available in notebook  
17 plt = plot_alignment(x, y, msm_alignment_path[0])  
18 plt.show()
```

Plot alignment path



Pairwise distance functions in aeon

Pairwise distance functions in aeon

Name	Distance function
dtw	aeon.distance.dtw_pairwise_distance
ddtw	aeon.distance.ddtw_pairwise_distance
wdtw	aeon.distance.wdtw_pairwise_distance
wddtw	aeon.distance.wddtw_pairwise_distance
erp	aeon.distance.erp_pairwise_distance
edr	aeon.distance.edr_pairwise_distance
msm	aeon.distance.msm_pairwise_distance
twe	aeon.distance.twe_pairwise_distance
lcss	aeon.distance.lcss_pairwise_distance
euclidean	aeon.distance.euclidean_pairwise_distance
squared euclidean	aeon.distance.squared_pairwise_distance

Pairwise distance function support

Name	Univariate	Multivariate	Unequal length
dtw	✓	✓	✗
ddtw	✓	✓	✗
wdtw	✓	✓	✗
wddtw	✓	✓	✗
erp	✓	✓	✗
edr	✓	✓	✗
msm	✓	✓	✗
twe	✓	✓	✗
lcss	✓	✓	✗
euclidean	✓	✓	✗
squared euclidean	✓	✓	✗

Calling pairwise distance functions

```
1 import numpy as np
2 from aeon.distances import twe_pairwise_distance, pairwise_distance
3
4 X = np.array([[1, 2, 3, 4, 5],
5               [6, 7, 8, 9, 10],
6               [11, 12, 13, 14, 15]]))
7
8 # Calling a distance directly
9 twe_pairwise_distance(X)
10 # >> array([[ 0.    , 21.008, 26.008],
11 #                 [21.008,  0.    , 21.008],
12 #                 [26.008, 21.008,  0.    ]])
13
14 # Using utility function
15 pairwise_distance(X, metric='twe')
16 # >> array([[ 0.    , 21.008, 26.008],
17 #                 [21.008,  0.    , 21.008],
18 #                 [26.008, 21.008,  0.    ]])
```

Calling pairwise distance functions

```
● ● ●

1 import numpy as np
2 from aeon.distances import erp_pairwise_distance, pairwise_distance
3
4 X = np.array([[1, 2, 3, 4, 5],
5                 [6, 7, 8, 9, 10],
6                 [11, 12, 13, 14, 15]]))
7
8 y = np.array([[16, 17, 18, 19, 20],
9                 [21, 22, 23, 24, 25]]))
10
11 # Calling a distance directly
12 erp_pairwise_distance(X, y)
13 # >> [[ 75. 100.]
14 #       [ 50.  75.]
15 #       [ 25.  50.]]
16
17 # Using utility function
18 pairwise_distance(X, y, metric='erp')
19 # >> [[ 75. 100.]
20 #       [ 50.  75.]
21 #       [ 25.  50.]]
```

Passing parameters to pairwise distances

```
1 import numpy as np
2 from aeon.distances import wdtw_pairwise_distance, pairwise_distance
3
4 X = np.array([[1, 2, 3, 4, 5],
5               [6, 7, 8, 9, 10],
6               [11, 12, 13, 14, 15]])
7
8 y = np.array([[16, 17, 18, 19, 20],
9               [21, 22, 23, 24, 25]])
10
11 # Calling a distance directly
12 wdtw_pairwise_distance(X, y, window=0.2)
13 # >> array([[527.38945495, 937.58125325],
14 #              [234.39531331, 527.38945495],
15 #              [ 54.24009352, 234.39531331]])
16
17 # Using utility function
18 pairwise_distance(X, y, metric='wdtw', window=0.2)
19 # >> array([[527.38945495, 937.58125325],
20 #              [234.39531331, 527.38945495],
21 #              [ 54.24009352, 234.39531331]])
```

Distance functions with sklearn

Classification - precomputed

```
1 from sklearn.svm import SVC
2 from aeon.distances import pairwise_distance
3 from aeon.datasets import load_gunpoint as load_data
4
5 # Load data
6 X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
7 X_test, y_test = load_data(split="TEST", return_type="numpy2D")
8
9 # Create model with precomputed
10 model_precomputed = SVC(kernel="precomputed")
11
12 # Compute pairwise matrix
13 train_pw_distance = pairwise_distance(X_train, metric="msm")
14 test_pw_distance = pairwise_distance(X_test, X_train, metric="msm")
15
16 # Fit model using precomputed
17 model_precomputed.fit(train_pw_distance, y_train)
18
19 # Score models on training data
20 model_precomputed.score(test_pw_distance, y_test)
21 # >> 0.386
```

Classification – pass distance function

```
● ● ●  
1 from sklearn.svm import SVC  
2 from aeon.distances import msm_pairwise_distance  
3 from aeon.datasets import load_gunpoint as load_data  
4  
5 # Load data  
6 X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")  
7 X_test, y_test = load_data(split="TEST", return_type="numpy2D")  
8  
9 # Create model with precomputed  
10 model_distance = SVC(kernel=msm_pairwise_distance)  
11  
12 # Fit model using precomputed  
13 model_distance.fit(X_train, y_train)  
14  
15 # Score models on training data  
16 model_distance.score(X_test, y_test)  
17 # >> 0.386
```

Clustering - precomputed



```
1 from sklearn.cluster import AgglomerativeClustering
2 from aeon.distances import pairwise_distance
3 from aeon.datasets import load_gunpoint as load_data
4
5 # Load data
6 X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
7 X_test, y_test = load_data(split="TEST", return_type="numpy2D")
8
9 # Create model with precomputed
10 model_precomputed = AgglomerativeClustering(metric="precomputed", linkage="complete")
11
12 # Compute pairwise
13 train_pw_distance = pairwise_distance(X_train, metric="twe")
14
15 # Fit model using precomputed
16 model_precomputed.fit(train_pw_distance)
17
18 # Show cluster assignments
19 model_precomputed.labels_
20 # >> [1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 0
21 # 1 0 0 1 1 1 1 0 1 1 1 1 0]
```

Clustering - pass distance function



```
1 from sklearn.cluster import AgglomerativeClustering
2 from aeon.distances import twe_pairwise_distance
3 from aeon.datasets import load_gunpoint as load_data
4
5 # Load data
6 X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
7 X_test, y_test = load_data(split="TEST", return_type="numpy2D")
8
9 # Create model with precomputed
10 model_distance = AgglomerativeClustering(metric=twe_pairwise_distance, linkage="complete")
11
12 # Fit model using precomputed
13 model_distance.fit(X_train)
14
15 # Show cluster assignments
16 model_distance.labels_
17 # >> [1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 1 0
18 # 1 0 0 1 1 1 1 0 1 1 1 1 0]
```

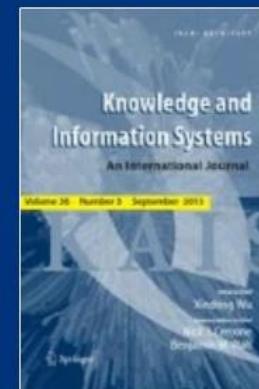
Applications: Time Series Clustering (TSCL)

<https://link.springer.com/article/10.1007/s10115-023-01952-0>

[Home](#) > [Knowledge and Information Systems](#) > Article

A review and evaluation of elastic distance functions for time series clustering

Review | [Open Access](#) | Published: 07 September 2023 | (2023)



[**Knowledge and Information Systems**](#)

[Aims and scope](#) →

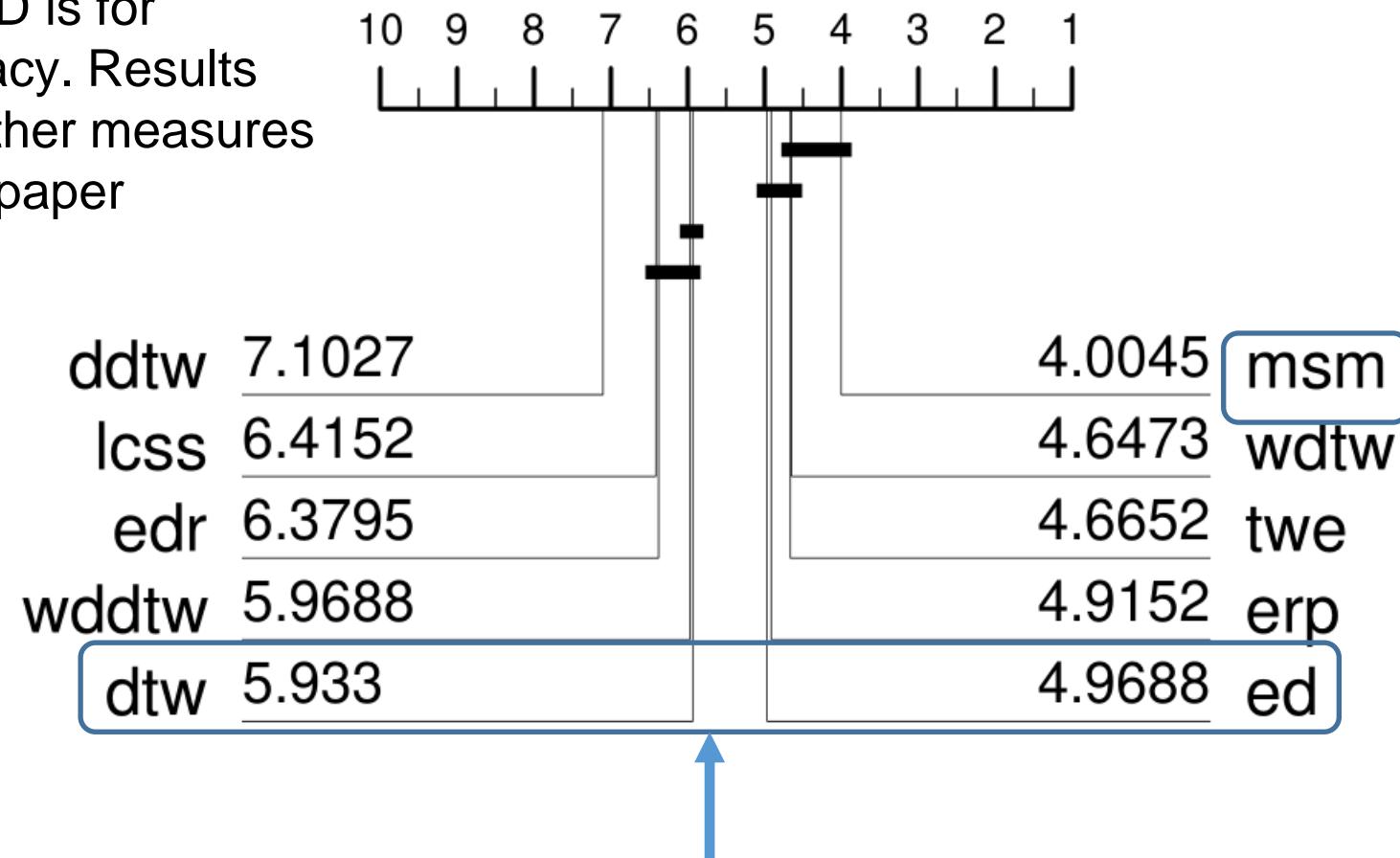
[Submit manuscript](#) →

[Download PDF](#) ↓

You have full access to this [open access](#) article

Comparison of different distance functions for k-means TSCL

The CD is for accuracy. Results with other measures in the paper

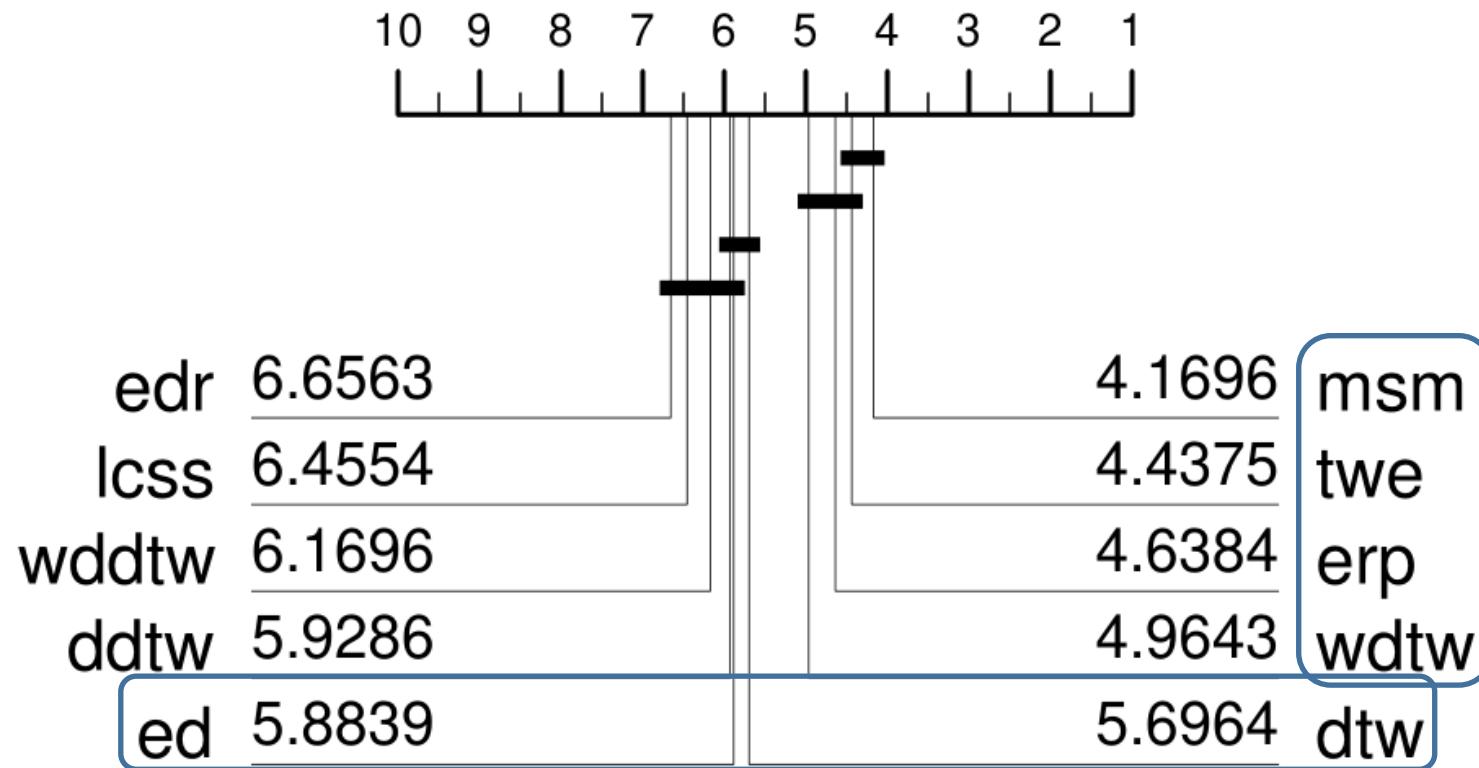


MSM only algorithm
significantly better
than Euclidean
distance

DTW significantly worse than Euclidean distance!

Comparison of different distance functions K-medoids TSCL

K-means averages to find centroids. An alternative is to use medoids as centres (members of the cluster)



DTW not worse than Euclidean distance using medoids

Distances with explicit warp penalty all better than Euclidean distance

Barycentre Averaging (DBA) [4] for K-Means

K-means averages to find centroids. An alternative is to warp cluster members onto each other.

Algorithm 7 DTW Barycentre Averaging(\mathbf{c} , the initial average sequence, $\mathbf{X_p}$, p time series to average.

- 1: Let dtw_path be a function that returns the a list of tuples that contain the indexes of the warping path between two time series.
- 2: Let W be a list of empty lists, where W_i stores the values in $\mathbf{X_p}$ of points warped onto centre point c_i .
- 3: **for** $x \in \mathbf{X_p}$ **do**
- 4: $P \leftarrow dtw_path(x, \mathbf{c})$
- 5: **for** $(i, j) \in P$ **do**
- 6: $W_i \leftarrow W_i \cup x_j$
- 7: **for** $i \leftarrow 1$ to m **do**
- 8: $c_i \leftarrow mean(W_i)$
- return** c

Find centroids by averaging
over realigned values



Comparison of DBA, k-means and k-medoids



Clear top clique of three algorithms

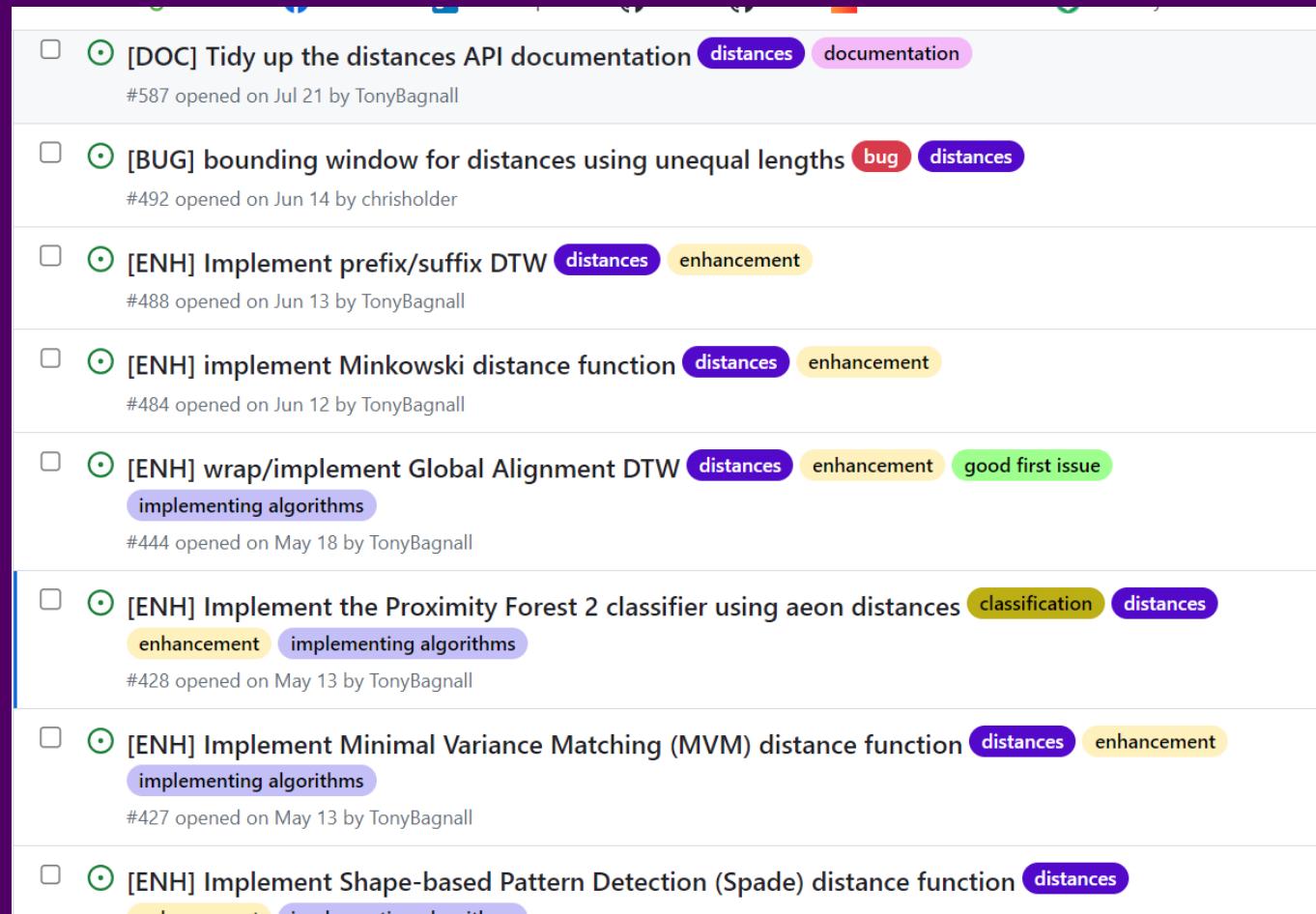
kmeans-dtw 6.4099
kmmedoids-ed 5.6036
kmmedoids-dtw 5.2973
dtw-dba 5.1306

4.0721 kmmedoids-msm
4.2477 kmeans-msm
4.2883 kmmedoids-twe
4.8243 kmeans-twe
5.1261 kmeans-ed

dba significantly improves DTW

Thank you for listening, any questions?

<https://github.com/aeon-toolkit/aeon/issues?q=is%3Aopen+is%3Aissue+label%3Adistances>



A screenshot of a GitHub issues page showing a list of open issues related to 'distances'. The issues are categorized by type: [DOC], [BUG], [ENH]. Each issue includes a title, a 'distances' label, and a brief description. The issues are listed in chronological order from top to bottom.

- [DOC] Tidy up the distances API documentation #587 opened on Jul 21 by TonyBagnall
- [BUG] bounding window for distances using unequal lengths #492 opened on Jun 14 by chrisholder
- [ENH] Implement prefix/suffix DTW #488 opened on Jun 13 by TonyBagnall
- [ENH] implement Minkowski distance function #484 opened on Jun 12 by TonyBagnall
- [ENH] wrap/implement Global Alignment DTW #444 opened on May 18 by TonyBagnall
- [ENH] Implement the Proximity Forest 2 classifier using aeon distances #428 opened on May 13 by TonyBagnall
- [ENH] Implement Minimal Variance Matching (MVM) distance function #427 opened on May 13 by TonyBagnall
- [ENH] Implement Shape-based Pattern Detection (Spade) distance function #426 opened on May 13 by TonyBagnall

We welcome new contributors, there are plenty of distance based issues if you want to get involved