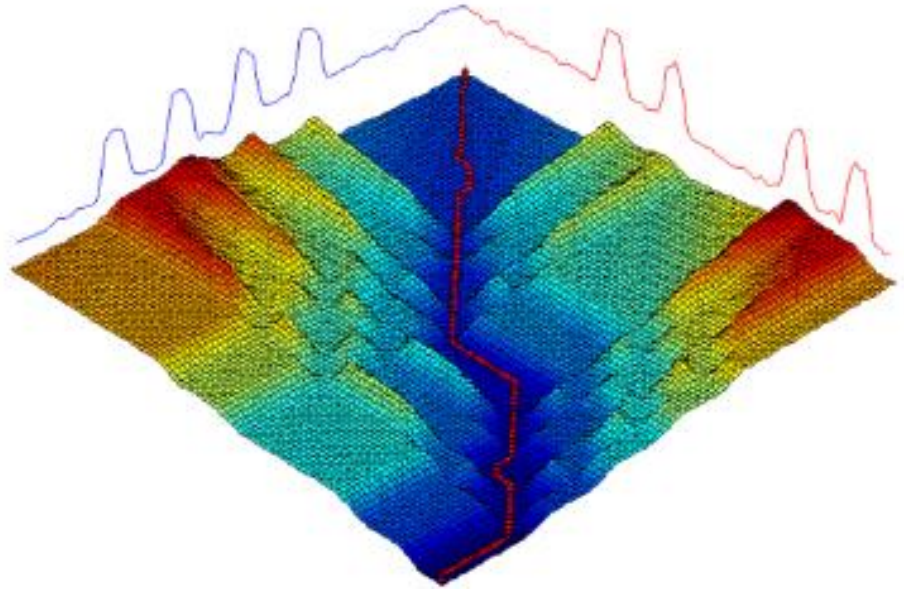
Let’s do the time warp again: time series machine learning with distance functions



Engineering and Physical Sciences Research Council

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

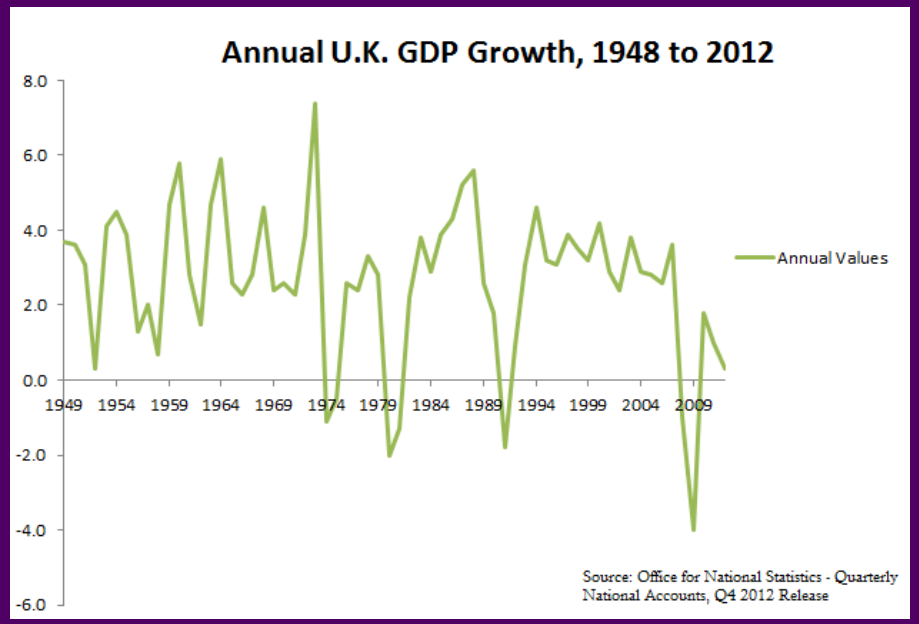
- Motivation and background: time series machine learning
- Time series distance functions
- Using aeon distance functions
- Distance based clustering

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

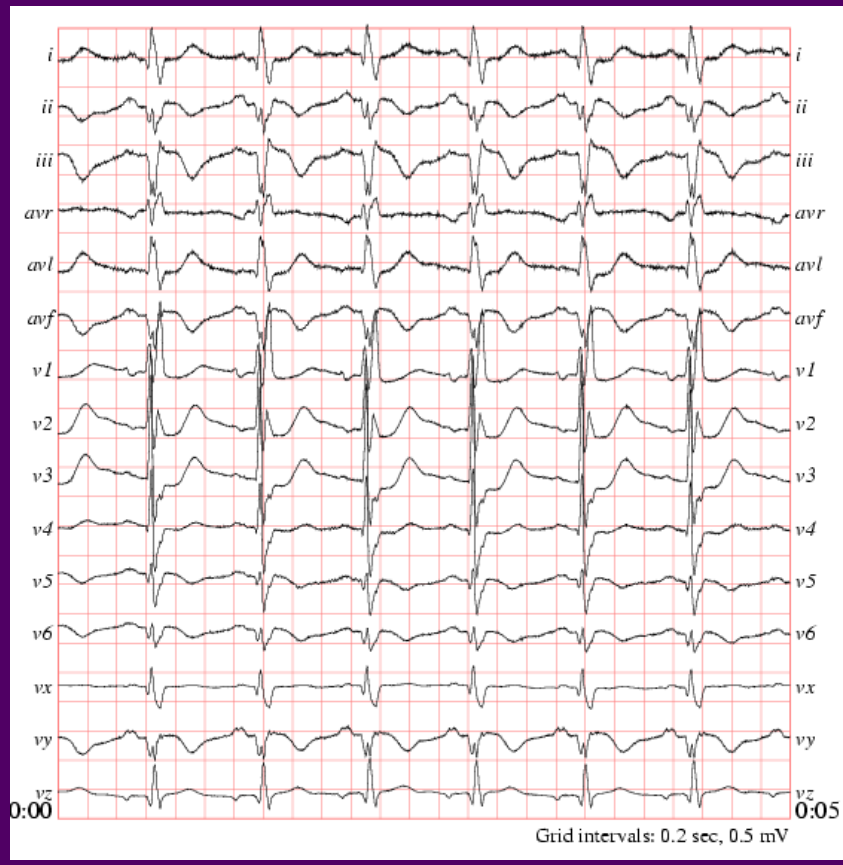
What is a time series?

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

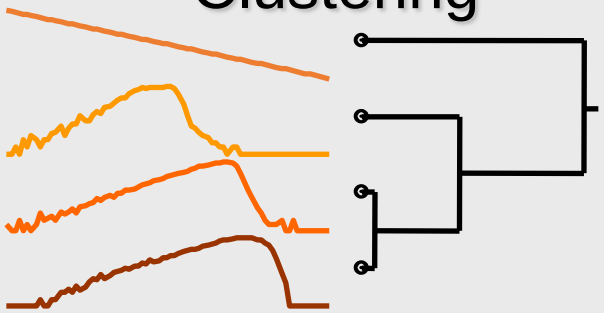


If each observation is a vector of values we call it a multivariate time series

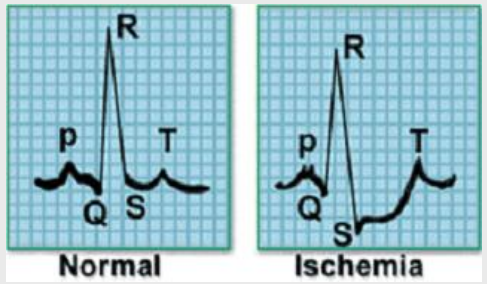


Time Series Machine Learning Tasks

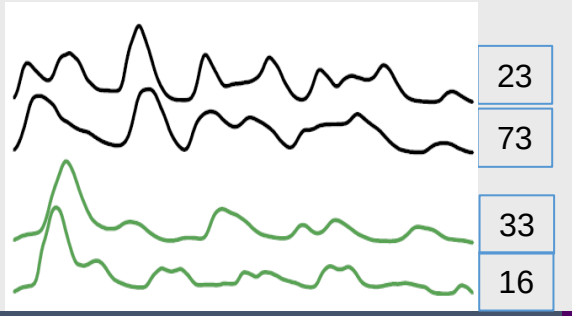
Clustering



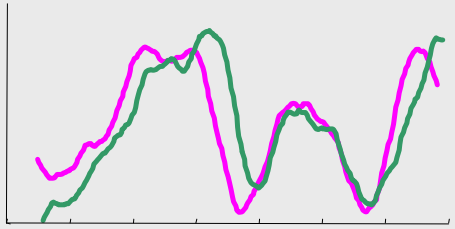
Classification



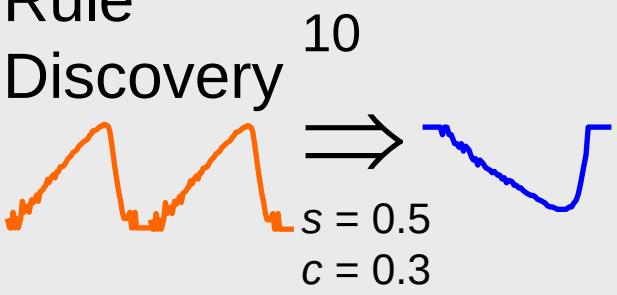
Regression



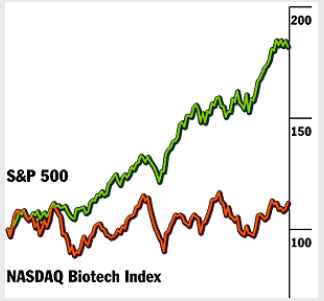
Motif Discovery



Rule Discovery



Query by Content



Forecasting

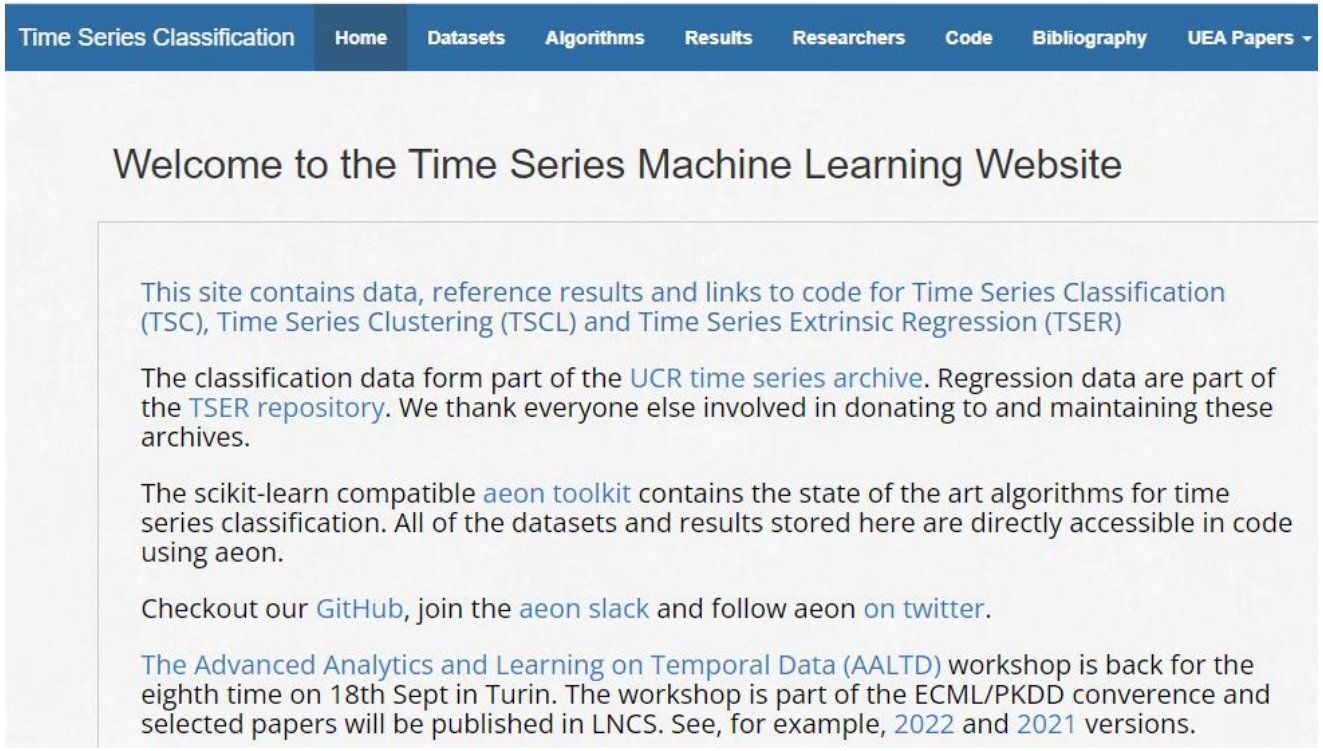
Anomaly Detection

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

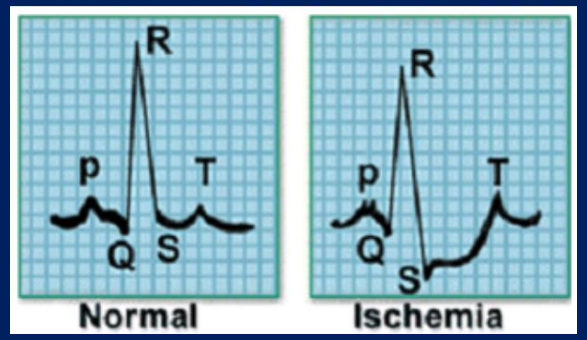
UNIVERSITY OF CALIFORNIA UCRIVERSIDE

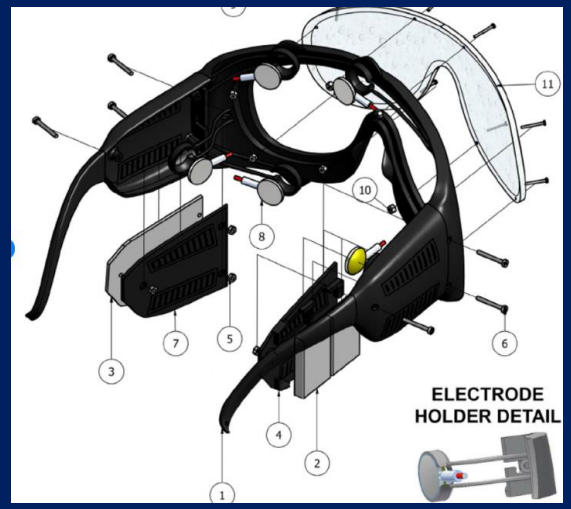


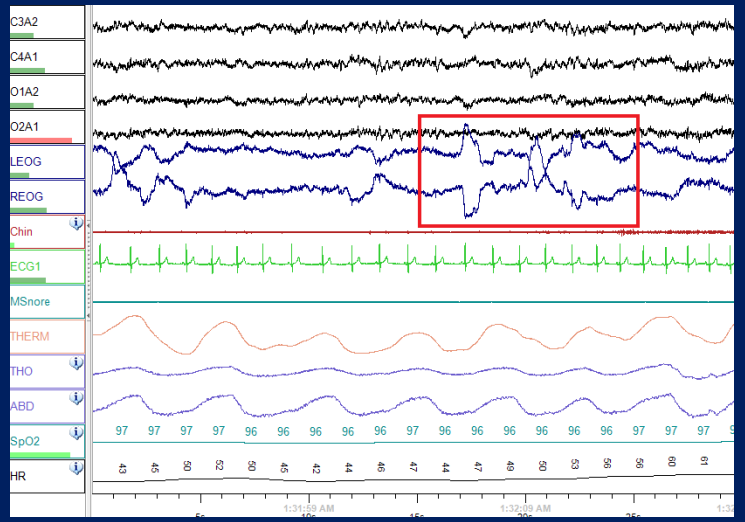
We would like to thank everyone who donates and helps maintain these archives

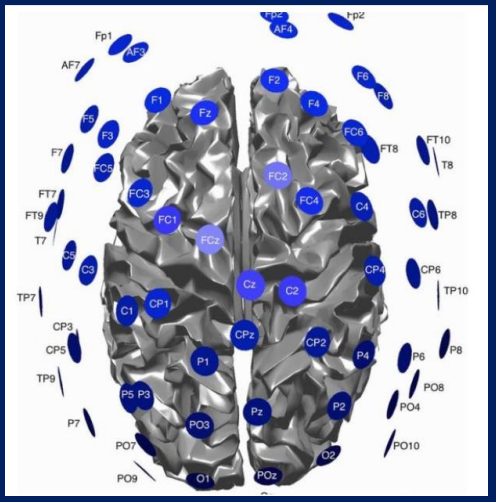
Example Data: Biomedical signals

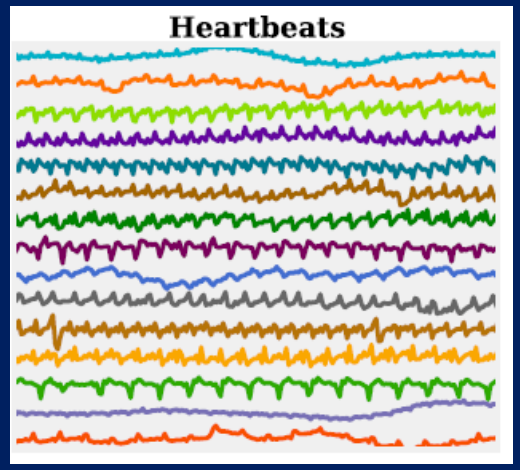
ECG, EEG, MEG, EOG

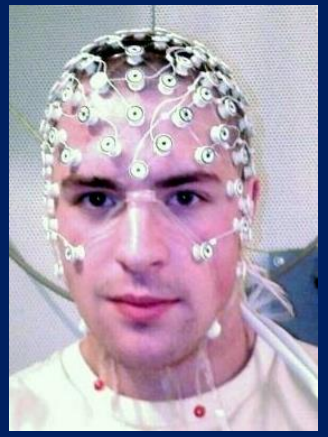


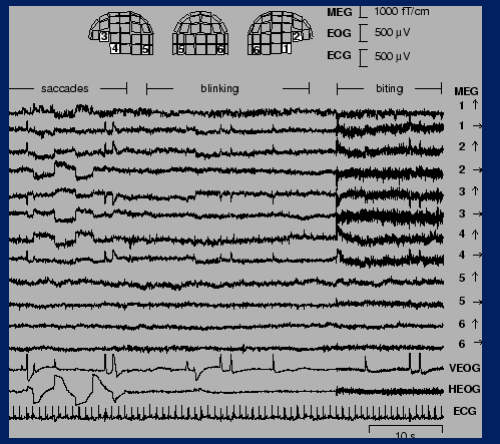


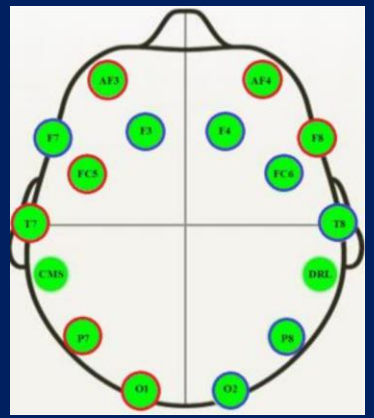








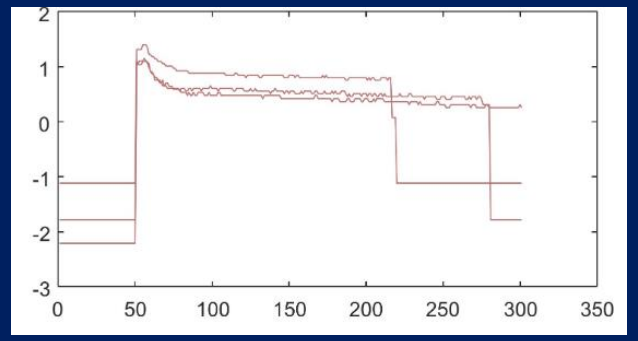


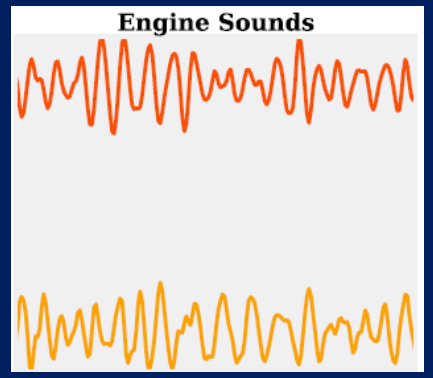


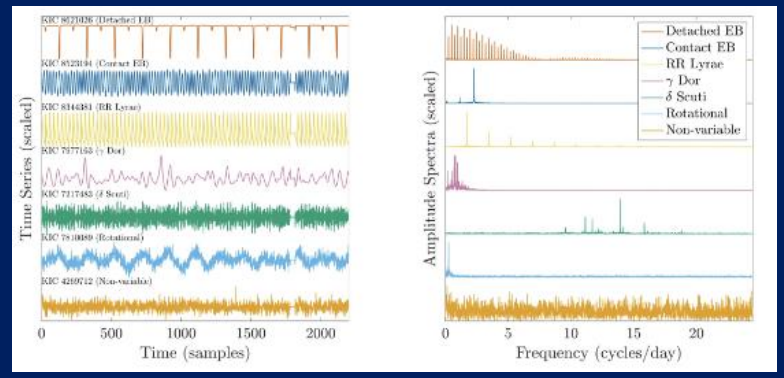
Example Data: Sensor Data

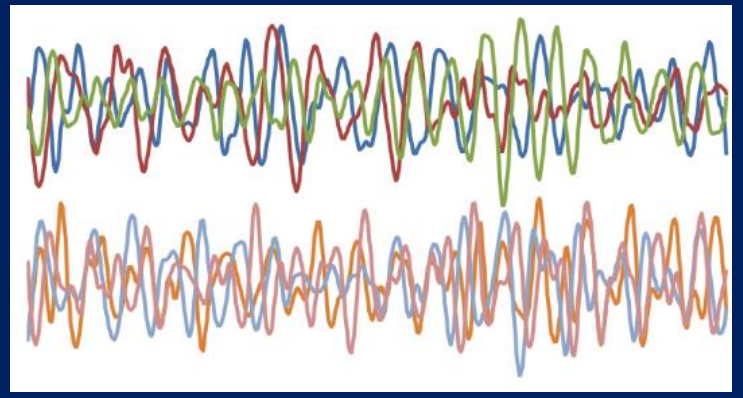
Car engines, light curves, lightning, electric devices

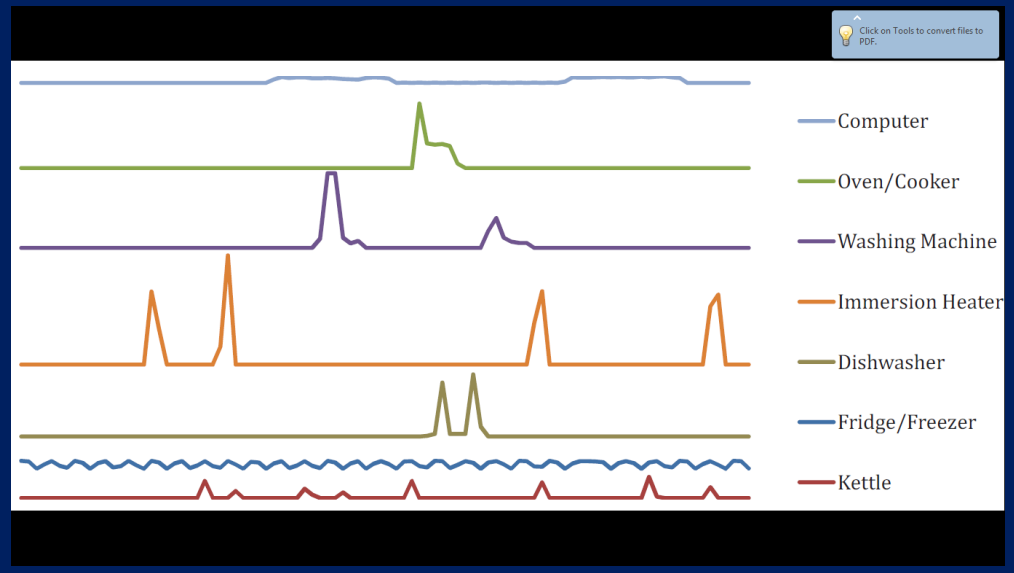
Insect wing beats, Car Engines, Phonemes (sound), Worm Motion

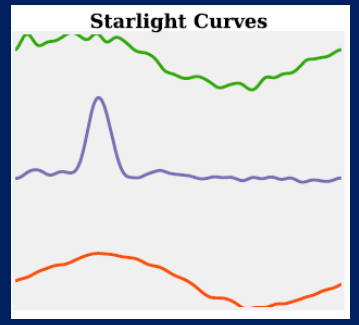


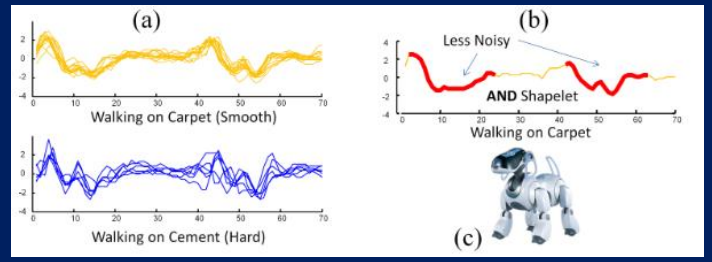






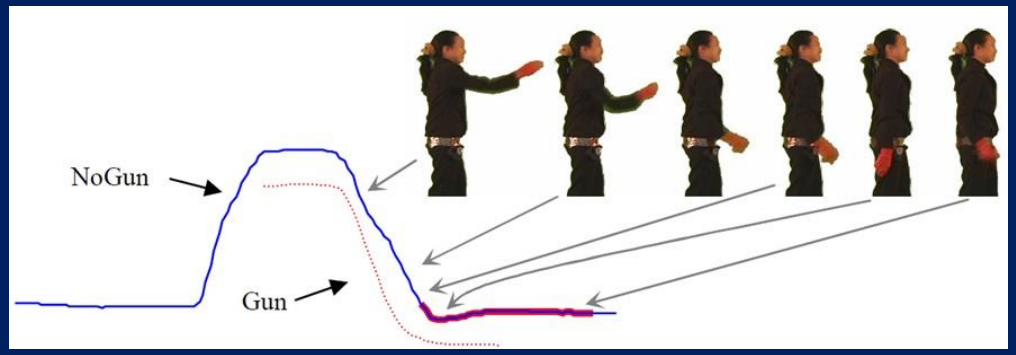


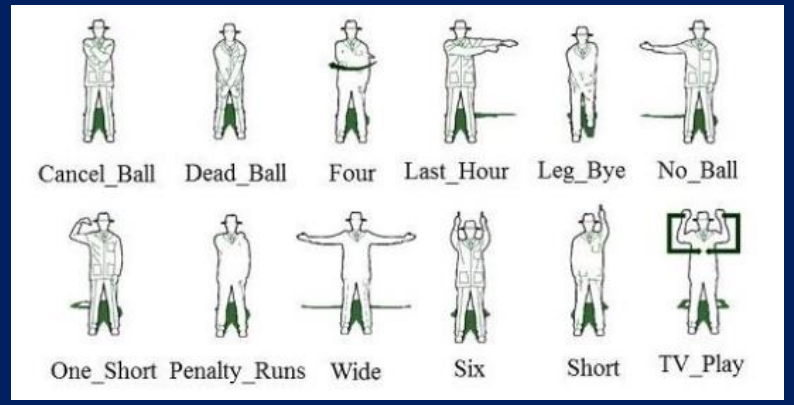


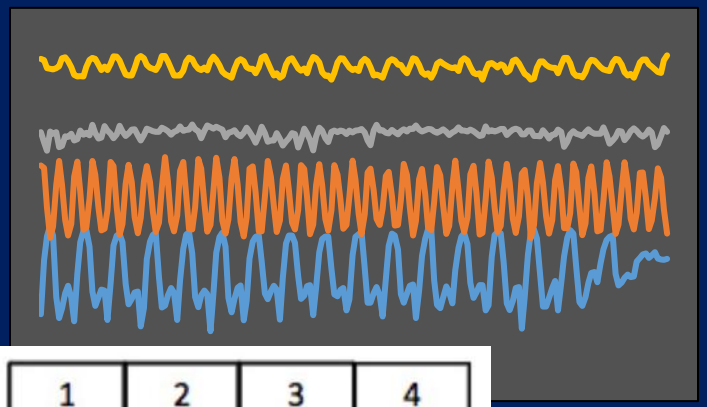


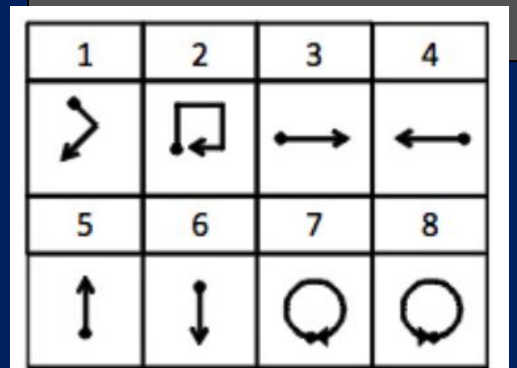
Example Data: Human Activity Recognition

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







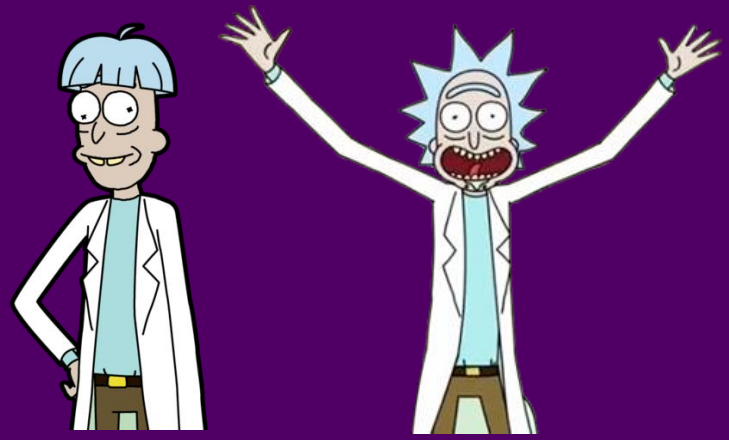


Measuring how similar time series are is a primitive operation for time series ML

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

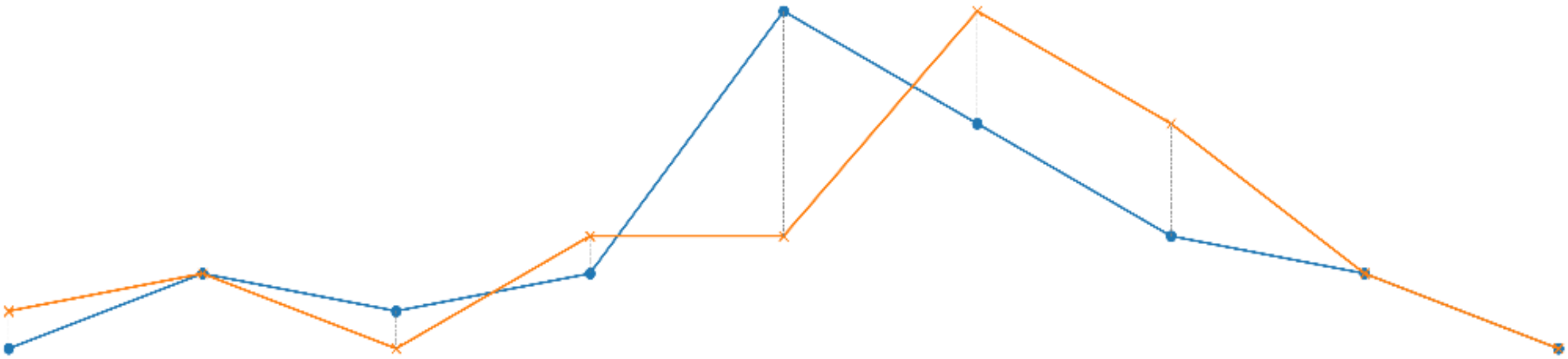
Given two time series:

$\boldsymbol{x} = x_1 \ldots x_n$

$\boldsymbol{y} = y_1 \ldots y_n$

The Euclidean Distance is defined as

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



Other distances for continuous variables

The Minkowski metric (a generalisation of Euclidean Distance)

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

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

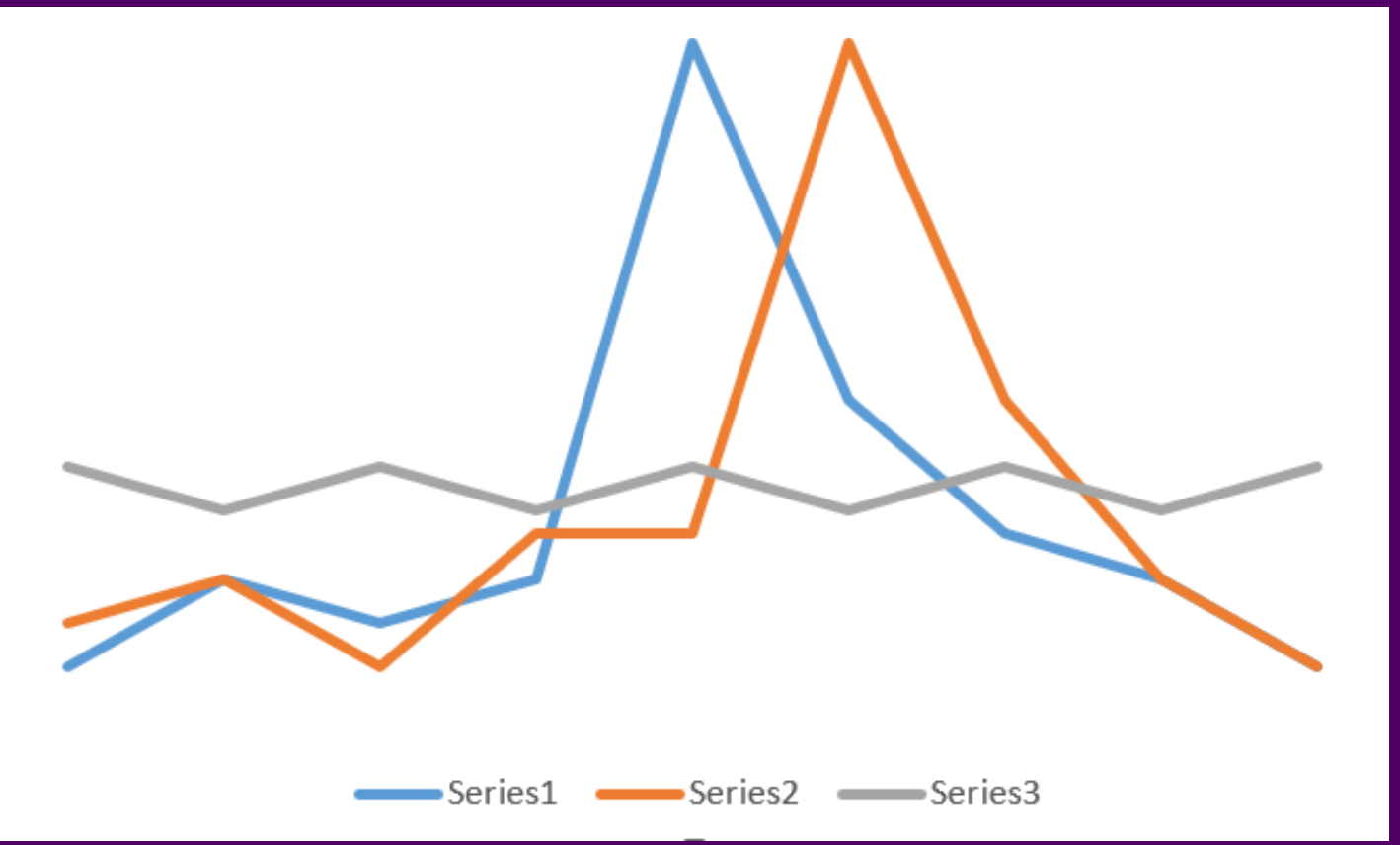
$$d(a,b) = \sum |a_i - b_i|$$

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?

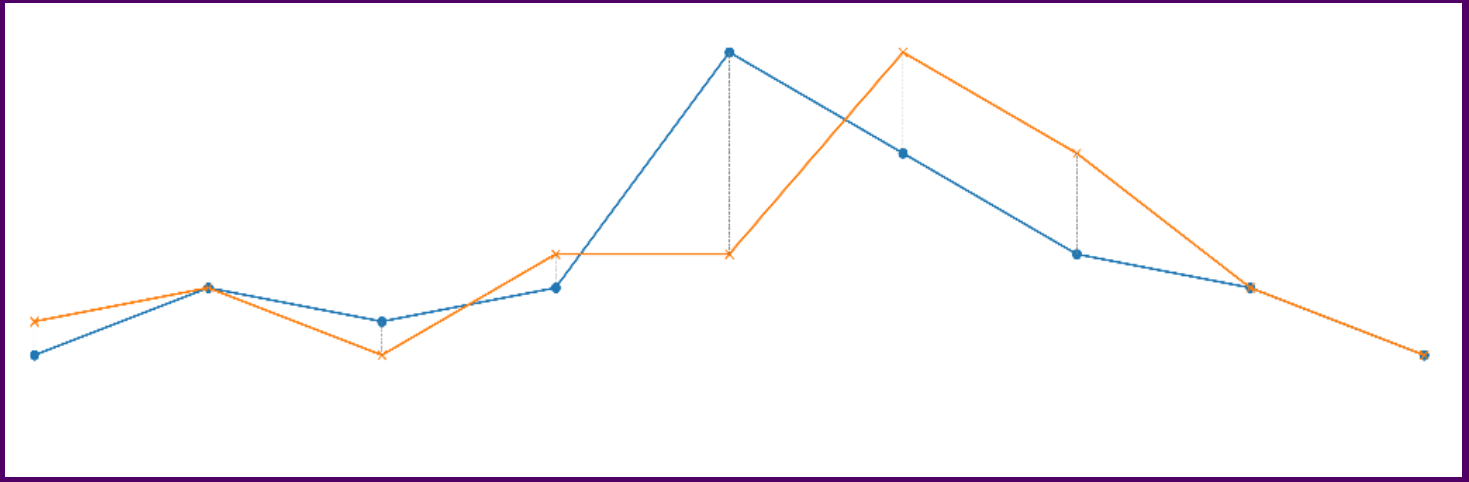
```
from aeon.distances import euclidean_distance
x = euclidean_distance(first, second)
print(x * x)
```

ed_dist(s1, s2) = 14.03 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

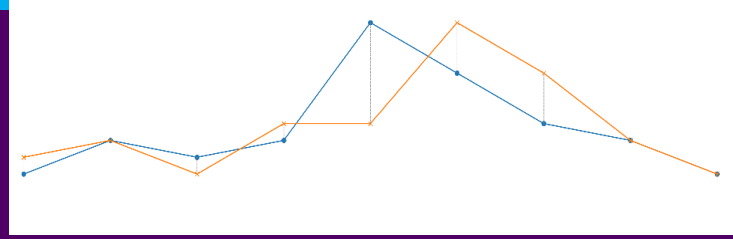
Pointwise distance between series, called M



tances.py | plot.png | _data_loaders.py | test_contir

```
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.

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

This is dynamic time warping

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

	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 $C_{m,m}$

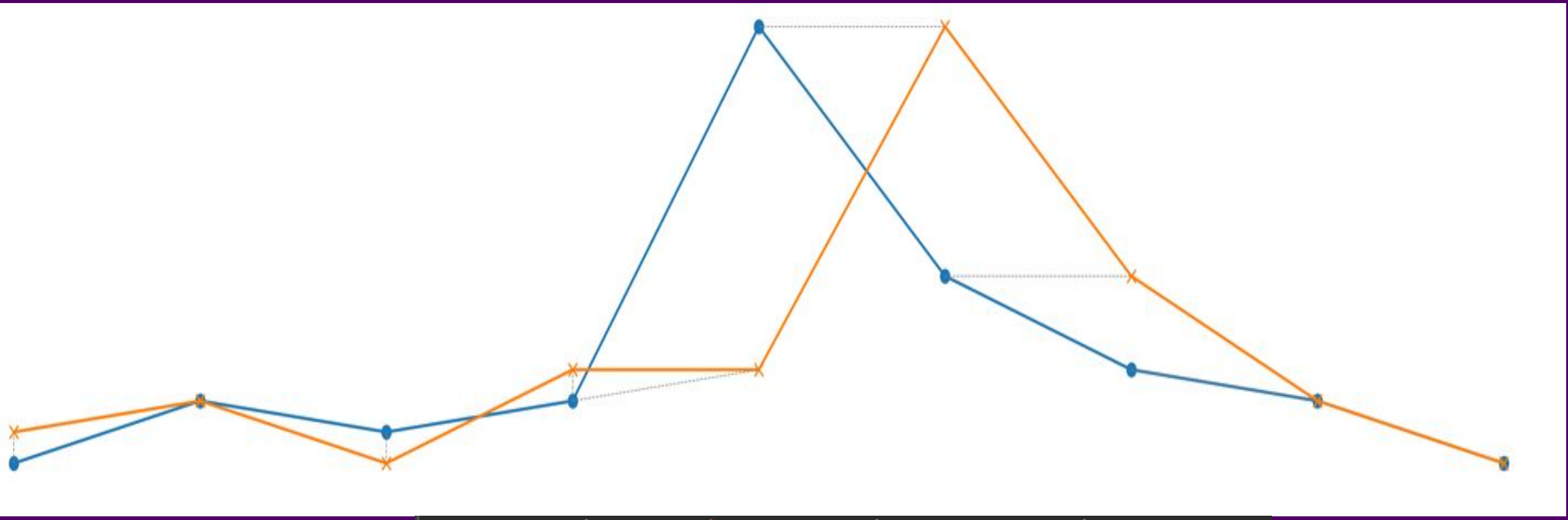
M=

	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

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

dtw_dist(s1, s2) = 5.0 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

Substitution	1 Unit
Insertion	1 Unit
Deletion	1 Unit

D(**`Peter,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) × (m + 1) matrix initialised to zero, indexed from zero.
2: for i ← 1 to m do
3:     for j ← 1 to m do
4:         C_{i,j} ← 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) × (m + 1) matrix initialised to zero,
2: for i ← 1 to m do
3:     for j ← 1 to m do
4:         match ← M_{i,j} + C_{i−1,j−1}
5:         insert ← M_{i,j} + C_{i−1,j}
6:         delete ← M_{i,j} + C_{i,j−1}
7:         C_{i,j} ← min(match, insert, delete)
   return C_{m,m}
```

Edit distance approaches apply a penalty for insert/delete

Edit Distance with Real Penalty

DTW

Let C be an $(m+1) \times (m+1)$ matrix initialised to zero,
for $i \leftarrow 1$ to m **do**
    for $j \leftarrow 1$ to m **do**
        $match \leftarrow M_{i,j} + C_{i-1,j-1}$
        $insert \leftarrow M_{i,j} + C_{i-1,j}$
        $delete \leftarrow M_{i,j} + C_{i,j-1}$
        $C_{i,j} \leftarrow \min(match, insert, delete)$
return $C_{m,m}$

ERP

Let C be an $(m+1) \times (m+1)$ matrix initialised to zero
for $i \leftarrow 1$ to m **do**
    for $j \leftarrow 1$ to m **do**
        $match \leftarrow M_{i,j} + C_{i-1,j-1}$
        $insert \leftarrow d(a_i, g) + C_{i-1,j}$
        $delete \leftarrow d(g, b_j) + C_{i,j-1}$
        $C_{i,j} \leftarrow \min(match, insert, delete)$
return $C_{m,m}$

Pointwise distance to a constant instead

Move/Split/Merge

Move is match, split is insert, merge is delete

MSM

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

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

MSM Cost function

DTW uses distance between points on insert/delete
ERP uses distance to a constant on insert/delete
MSM combines these

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

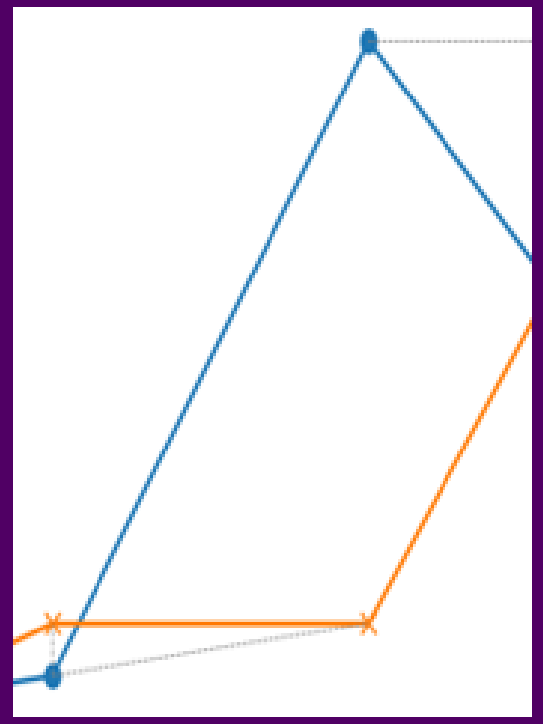
$$match \leftarrow M_{i,j} + C_{i-1,j-1}$$
$$insert \leftarrow cost(a_i, a_{i-1}, b_j, c)$$
$$delete \leftarrow cost(b_j, b_{j-1}, a_i, c)$$

Suppose a4=3, a5=15, b4=4, b5=4

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

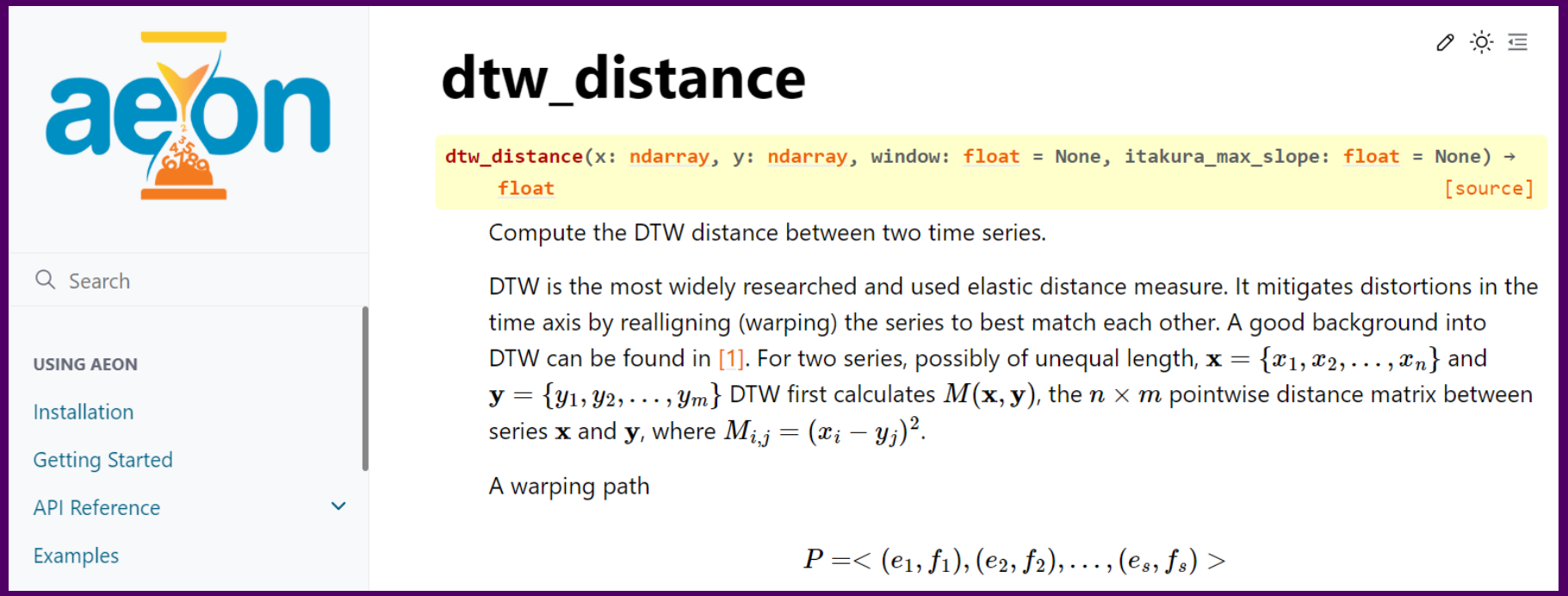
Cost insert(5,5) =cost(15,3,4) = 1+9=10

Cost delete(5,5) =cost(4,4,15) = 1



Distance functions in aeon

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



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	✓	✓	✓

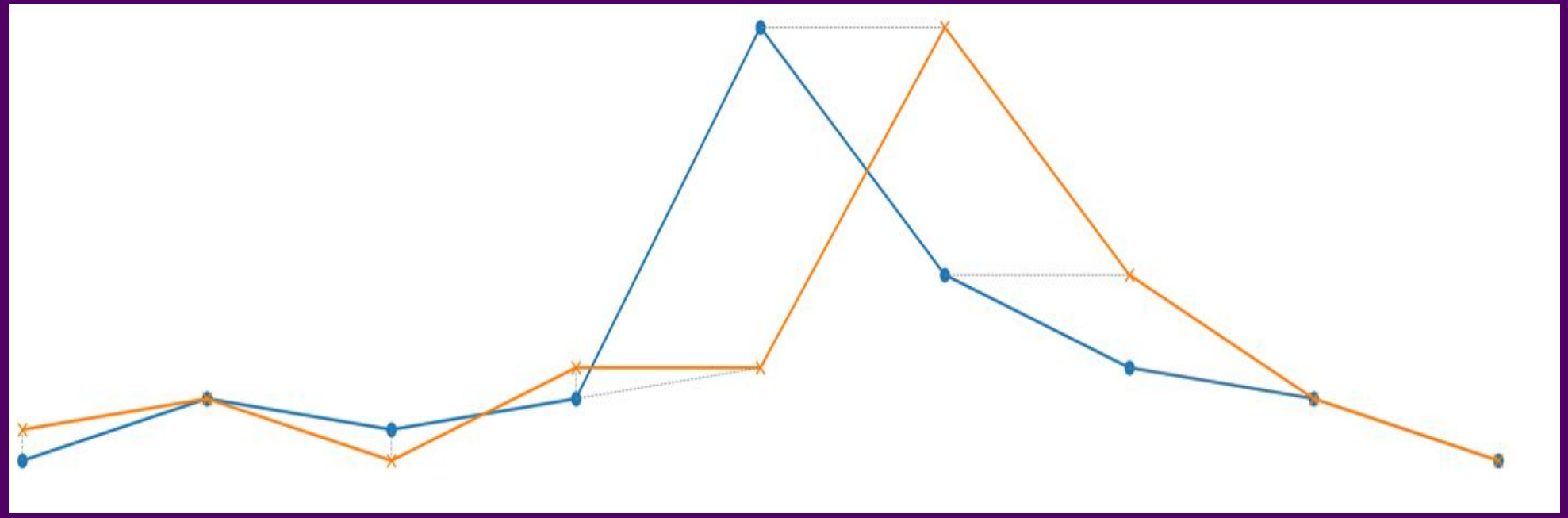
Calling distance functions

```python
import numpy as np
from aeon.distances import dtw_distance, distance

x = np.array([1, 2, 3, 4, 5])
y = np.array([6, 7, 8, 9, 10])

# Calling a distance directly
dtw_distance(x, y)
# >> 108.0

# Calling a distance using utility function
# Any value in the table above is a valid metric string
distance(x, y, metric='dtw')
# >> 108.0
```

Passing parameters to distance functions

```python
import numpy as np
from aeon.distances import msm_distance, distance

x = np.array([1, 2, 3, 4, 5])
y = np.array([6, 7, 8, 9, 10])

# Calling a distance directly
msm_distance(x, y, window=0.2)
# >> 2.0

# Calling a distance using utility function
distance(x, y, metric='msm', window=0.2)
# >> 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

```python
import numpy as np
from aeon.distances import msm_alignment_path, alignment_path

x = np.array([1, 2, 3, 4, 5])
y = np.array([6, 7, 8, 9, 10])

# Calling a alignment path directly
msm_alignment_path(x, y)
#>> ([(0, 0), (1, 1), (2, 2), (3, 3), (4, 4)], 16.0)

# Using utility function
alignment_path(x, y, metric='msm')
#>> ([(0, 0), (1, 1), (2, 2), (3, 3), (4, 4)], 16.0)
```

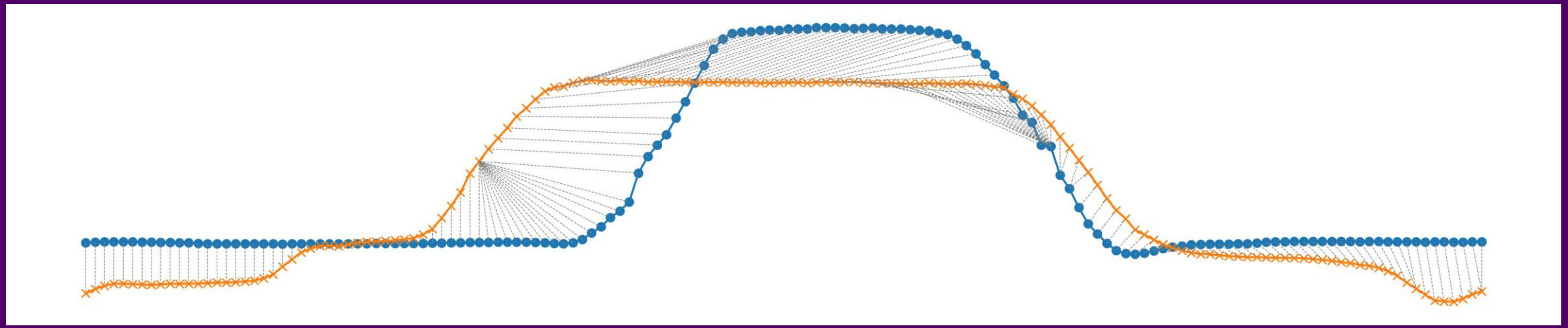
Passing parameters to alignment paths

```python
import numpy as np
from aeon.distances import msm_alignment_path, alignment_path

x = np.array([1, 2, 3, 4, 5])
y = np.array([6, 7, 8, 9, 10])

# Calling a alignment path directly
msm_alignment_path(x, y, window=0.2)
#>> ([(0, 0), (0, 1), (0, 2), (1, 3), (2, 4), (3, 4), (4, 4)], 2.0)

# Using utility function
alignment_path(x, y, metric='msm', window=0.2)
#>> ([(0, 0), (0, 1), (0, 2), (1, 3), (2, 4), (3, 4), (4, 4)], 2.0)
```

Plot alignment path

```python
import matplotlib.pyplot as plt
from aeon.datasets import load_gunpoint as load_data
from aeon.distances import alignment_path

# Load data
X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
X_test, y_test = load_data(split="TEST", return_type="numpy2D")

# Choose two random time series
x = X_train[0]
y = X_train[22]

# Compute path
msm_alignment_path = alignment_path(x, y, metric='msm')

#  Plot using function avaliable in notebook
plt = plot_alignment(x, y, msm_alignment_path[0])
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

```python
import numpy as np
from aeon.distances import twe_pairwise_distance, pairwise_distance

X = np.array([[1, 2, 3, 4, 5],
              [6, 7, 8, 9, 10],
              [11, 12, 13, 14, 15]])

# Calling a distance directly
twe_pairwise_distance(X)
# >> array([[ 0.    , 21.008, 26.008],
#           [21.008,  0.    , 21.008],
#           [26.008, 21.008,  0.    ]])

# Using utility function
pairwise_distance(X, metric='twe')
# >> array([[ 0.    , 21.008, 26.008],
#           [21.008,  0.    , 21.008],
#           [26.008, 21.008,  0.    ]])
```

Calling pairwise distance functions

```python
import numpy as np
from aeon.distances import erp_pairwise_distance, pairwise_distance

X = np.array([[1, 2, 3, 4, 5],
              [6, 7, 8, 9, 10],
              [11, 12, 13, 14, 15]])

y = np.array([[16, 17, 18, 19, 20],
              [21, 22, 23, 24, 25]])

# Calling a distance directly
erp_pairwise_distance(X, y)
# >> [[ 75. 100.]
#     [ 50.  75.]
#     [ 25.  50.]]

# Using utility function
pairwise_distance(X, y, metric='erp')
# >> [[ 75. 100.]
#     [ 50.  75.]
#     [ 25.  50.]]
```

Passing parameters to pairwise distances

```python
import numpy as np
from aeon.distances import wdtw_pairwise_distance, pairwise_distance

X = np.array([[1, 2, 3, 4, 5],
              [6, 7, 8, 9, 10],
              [11, 12, 13, 14, 15]])

y = np.array([[16, 17, 18, 19, 20],
              [21, 22, 23, 24, 25]])

# Calling a distance directly
wdtw_pairwise_distance(X, y, window=0.2)
# >> array([[527.38945495, 937.58125325],
#           [234.39531331, 527.38945495],
#           [ 54.24009352, 234.39531331]])

# Using utility function
pairwise_distance(X, y, metric='wdtw', window=0.2)
# >> array([[527.38945495, 937.58125325],
#           [234.39531331, 527.38945495],
#           [ 54.24009352, 234.39531331]])
```

Distance functions with sklearn

Classification - precomputed

```python
from sklearn.svm import SVC
from aeon.distances import pairwise_distance
from aeon.datasets import load_gunpoint as load_data

# Load data
X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
X_test, y_test = load_data(split="TEST", return_type="numpy2D")

# Create model with precomputed
model_precomputed = SVC(kernel="precomputed")

# Compute pairwise matrix
train_pw_distance = pairwise_distance(X_train, metric="msm")
test_pw_distance = pairwise_distance(X_test, X_train, metric="msm")

# Fit model using precomputed
model_precomputed.fit(train_pw_distance, y_train)

# Score models on training data
model_precomputed.score(test_pw_distance, y_test)
# >> 0.386
```

Classification – pass distance function

```python
from sklearn.svm import SVC
from aeon.distances import msm_pairwise_distance
from aeon.datasets import load_gunpoint as load_data

# Load data
X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
X_test, y_test = load_data(split="TEST", return_type="numpy2D")

# Create model with precomputed
model_distance = SVC(kernel=msm_pairwise_distance)

# Fit model using precomputed
model_distance.fit(X_train, y_train)

# Score models on training data
model_distance.score(X_test, y_test)
# >> 0.386
```

Clustering - precomputed

```python
from sklearn.cluster import AgglomerativeClustering
from aeon.distances import pairwise_distance
from aeon.datasets import load_gunpoint as load_data

# Load data
X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
X_test, y_test = load_data(split="TEST", return_type="numpy2D")

# Create model with precomputed
model_precomputed = AgglomerativeClustering(metric="precomputed", linkage="complete")

# Compute pairwise
train_pw_distance = pairwise_distance(X_train, metric="twe")

# Fit model using precomputed
model_precomputed.fit(train_pw_distance)

# Show cluster assignments
model_precomputed.labels_
# >> [1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 1 0
# 1 0 0 1 1 1 1 0 1 1 1 1 0]
```

Clustering - pass distance function

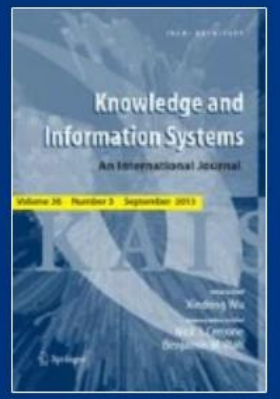
```python
from sklearn.cluster import AgglomerativeClustering
from aeon.distances import twe_pairwise_distance
from aeon.datasets import load_gunpoint as load_data

# Load data
X_train, y_train = load_data(split="TRAIN", return_type="numpy2D")
X_test, y_test = load_data(split="TEST", return_type="numpy2D")

# Create model with precomputed
model_distance = AgglomerativeClustering(metric=twe_pairwise_distance, linkage="complete")

# Fit model using precomputed
model_distance.fit(X_train)

# Show cluster assignments
model_distance.labels_
# >> [1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 1 0
# 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

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 →

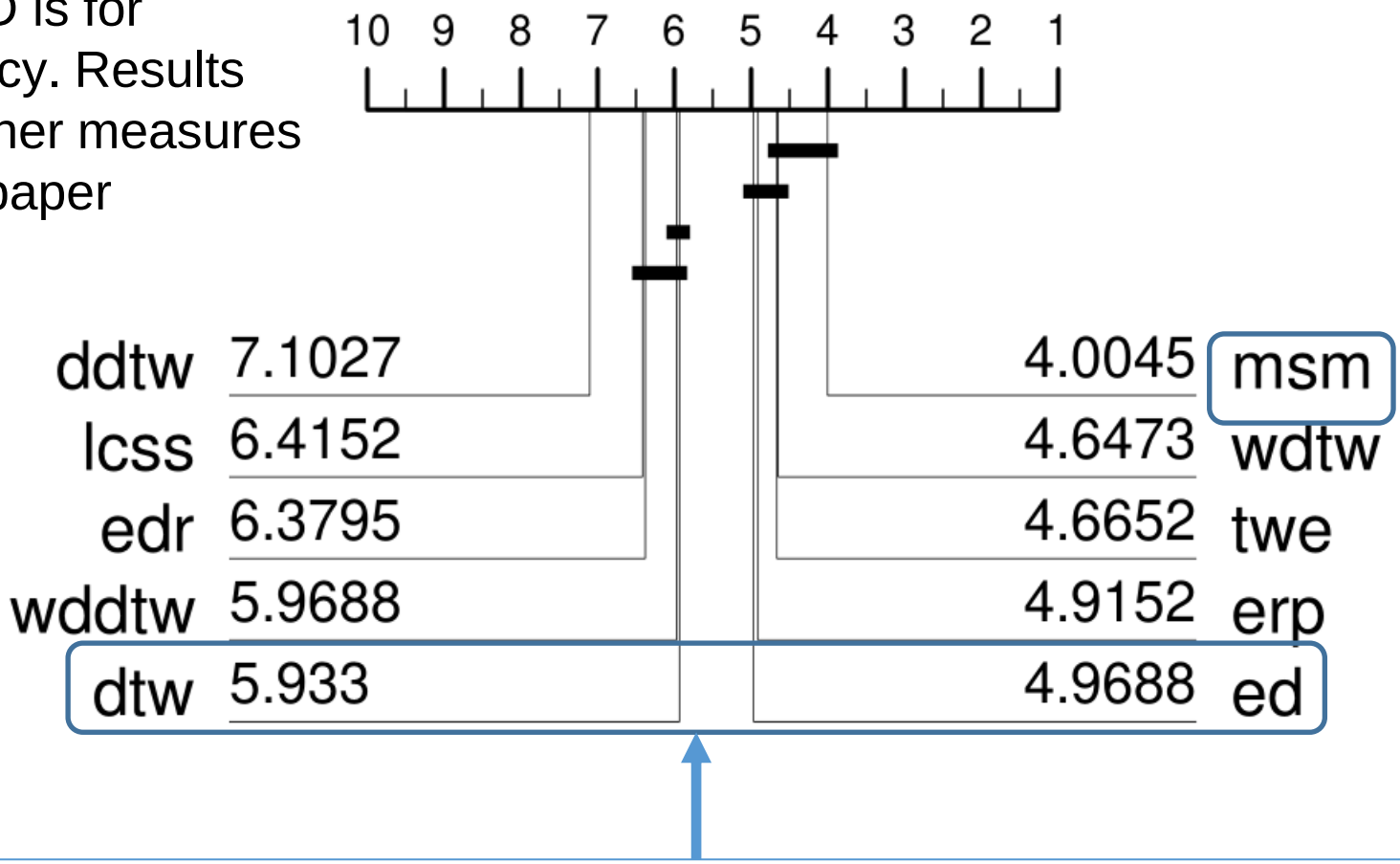
Submit manuscript →

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

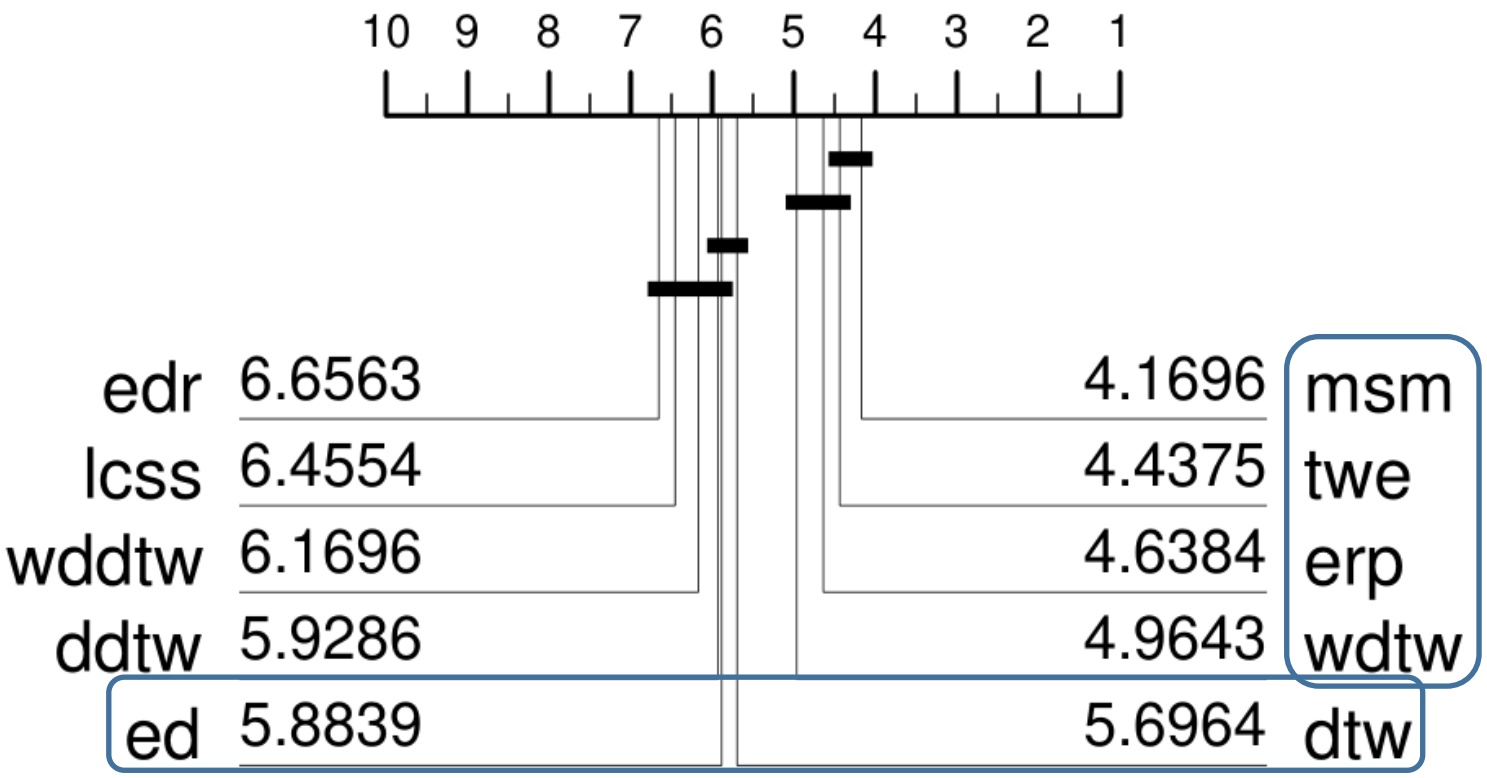


MSM only algorithm significantly better than Euclidean distance

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

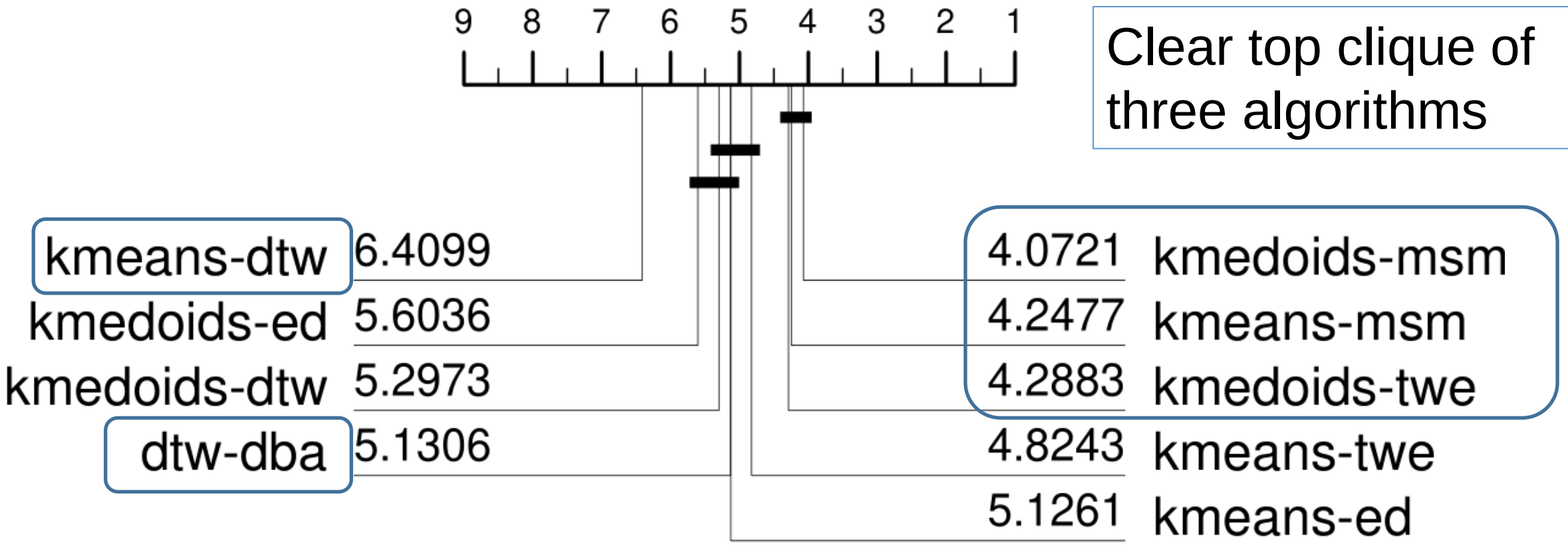
K-means averages to find centroids. An alternative is to warp cluster members onto each other.

Algorithm 7 DTW Barycentre Averaging(**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: $P \leftarrow dtw\_path(\mathbf{x}, \mathbf{c})$
5: **for** $(i, j) \in P$ **do**
6: $W_i \leftarrow W_i U x_j$
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

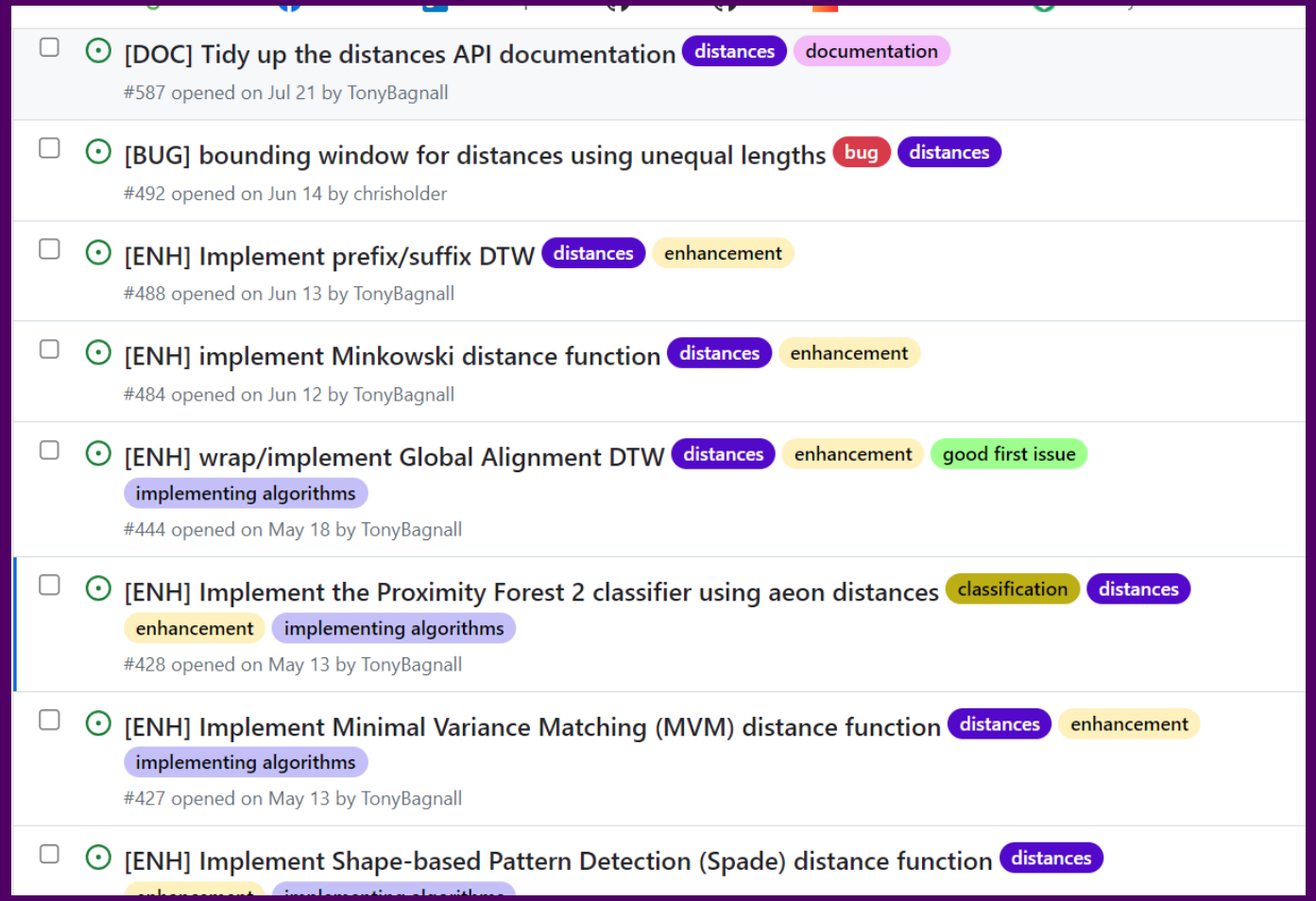


Clear top clique of three algorithms

dba significantly improves DTW

Thank you for listening, any questions?

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



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