IL2233

Lab 3: Time-Series Clustering with K-Means and SOM, and Similarity Measuring with DTW

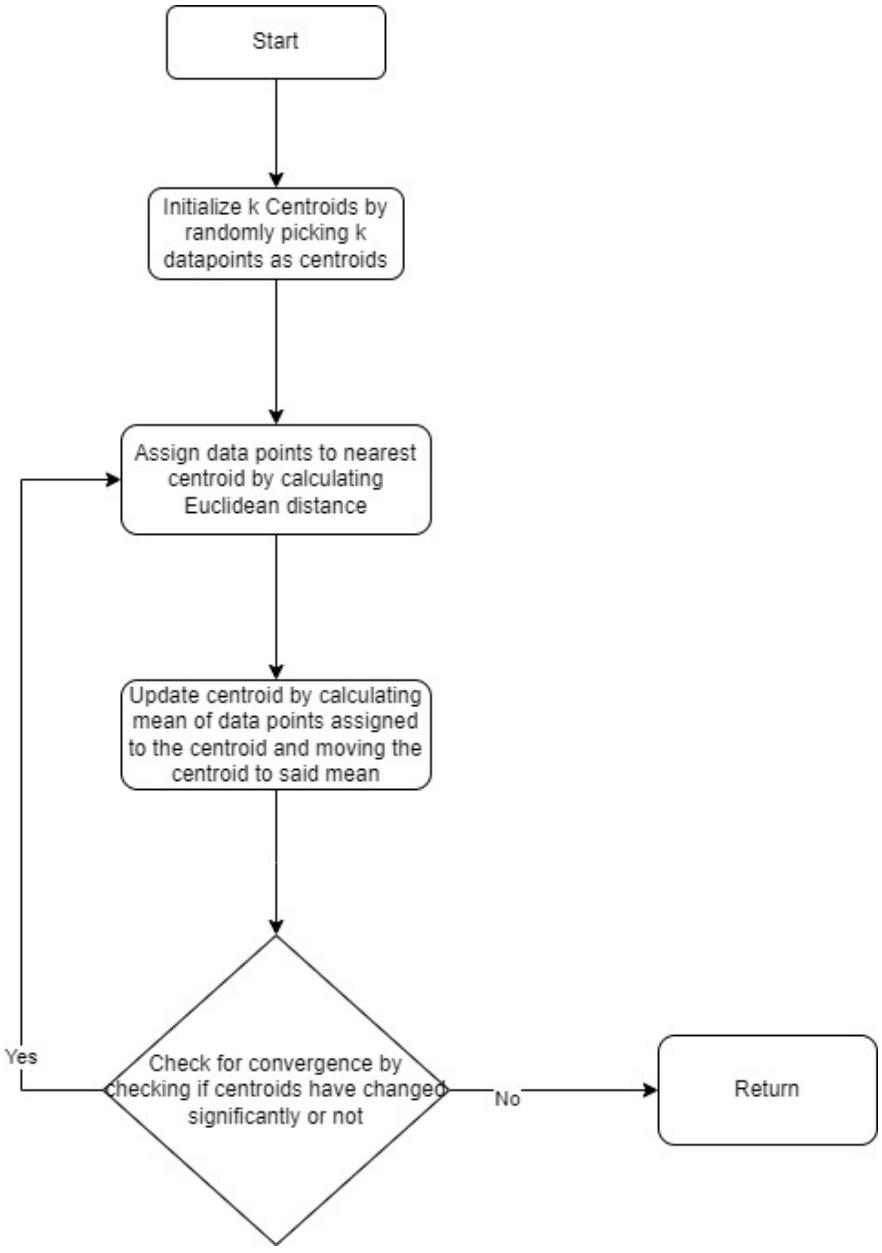
Shawn Nagar, Leon Moll

May 17, 2024

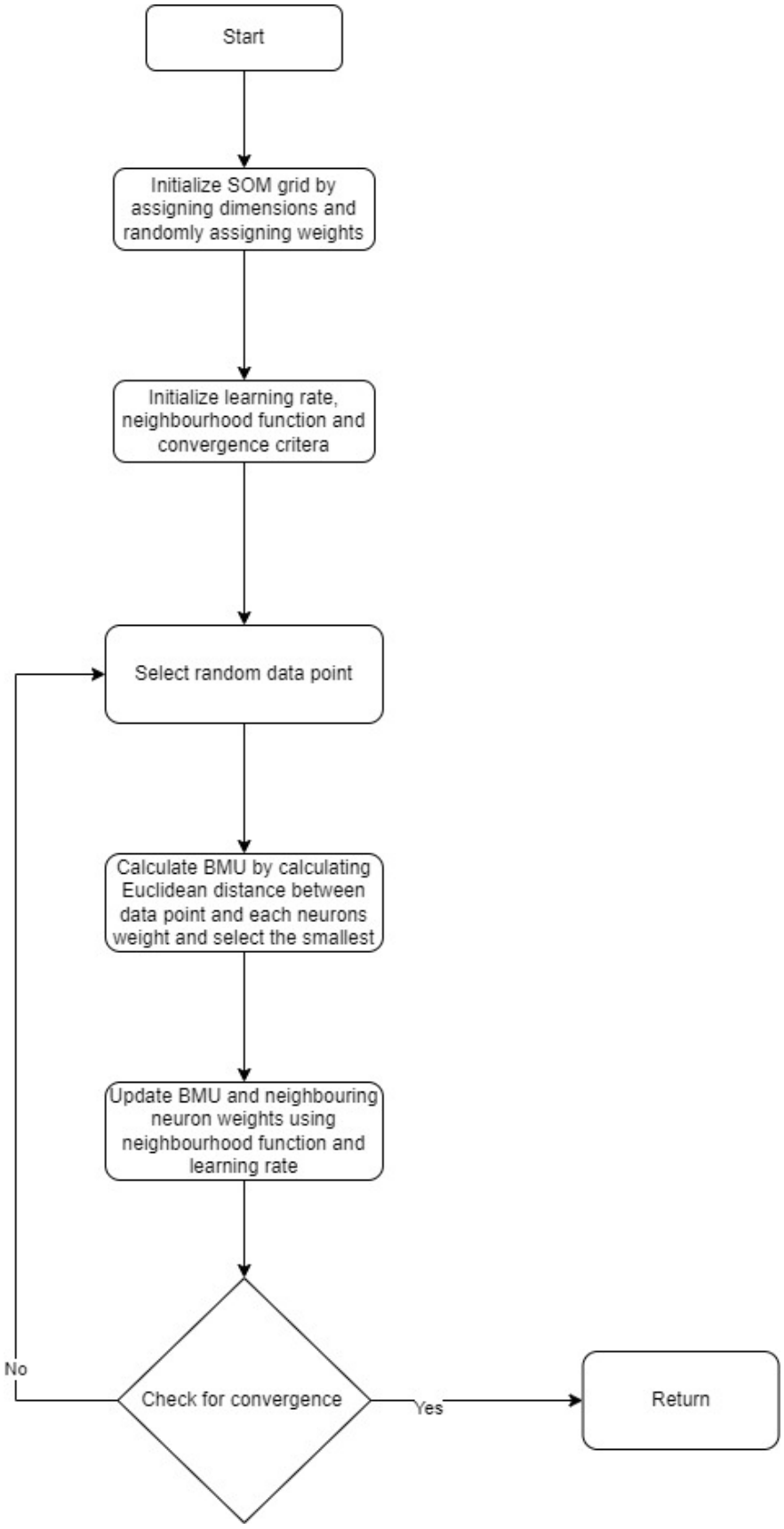
1 Implement and evaluate K-means and SOM

1.1 FlowCharts

1.1.1 K-Means



1.1.2 SOM


Start
Initialize SOM grid by assigning dimensions and randomly assigning weights
Initialize learning rate, neighbourhood function and convergence critera
Select random data point
Calculate BMU by calculating Euclidean distance between data point and each neurons weight and select the smallest
Update BMU and neighbouring neuron weights using neighbourhood function and learning rate
Check for convergence
No
Yes
Return

1.2 Questions

In the K-means algorithm, "k" needs to be given. Why? Is it feasible to automatically search a good value for "k"?

K refers to the number of centroids used for clustering the data. There is no universal rule for finding the ideal value of K as that depends heavily on the type and characteristics of the data being clustered.

For the SOM algorithm, how is the learning happening? How does the learning rate affect the performance of the algorithm?

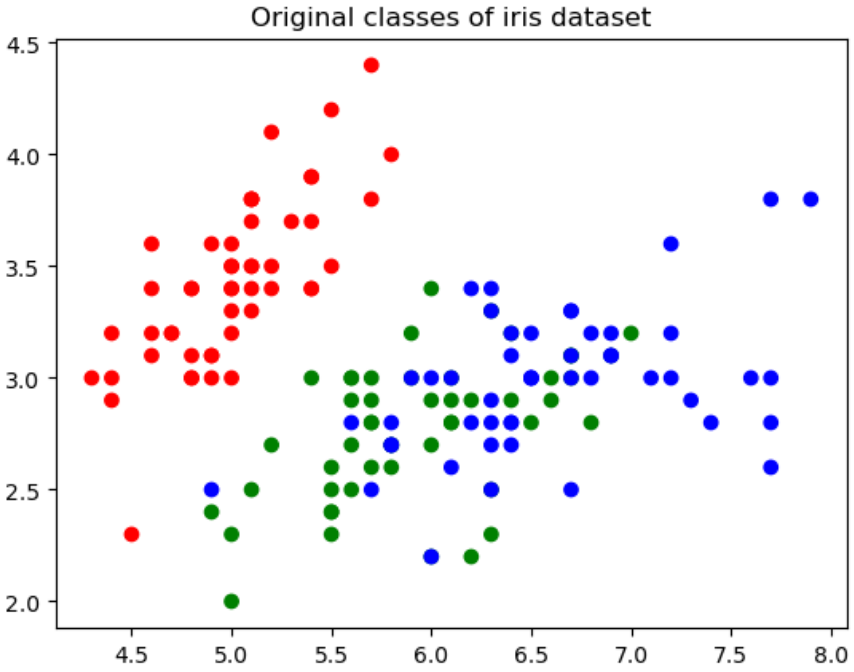
SOM stands for Self-Organizing Map. Learning occurs through an iterative process where the weights of the neurons in the map are updated based on their proximity to the input data points. Learning occurs by finding the Best Matching Unit (BMU) by calculating the difference in the weight vectors of neurons and the input vector. Weight of the BMU and neighbouring neurons is updated. Learning rate and neighbourhood functions are updated. The process then repeats. Learning rate affects how significantly the weights are updated. A large learning rate results in larger changes to weights of neurons per iteration. A smaller rate allows for fine tuning. Thus a decaying learning rate is often used.

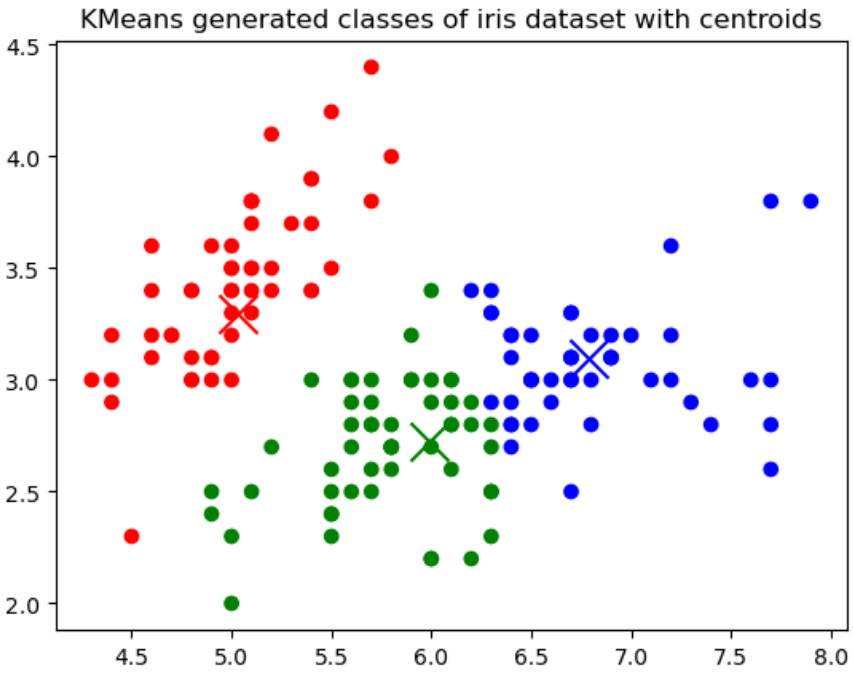
For the SOM algorithm, is the "neuron" in the output layer the same as the neuron in an MLP? If different, what are the differences?

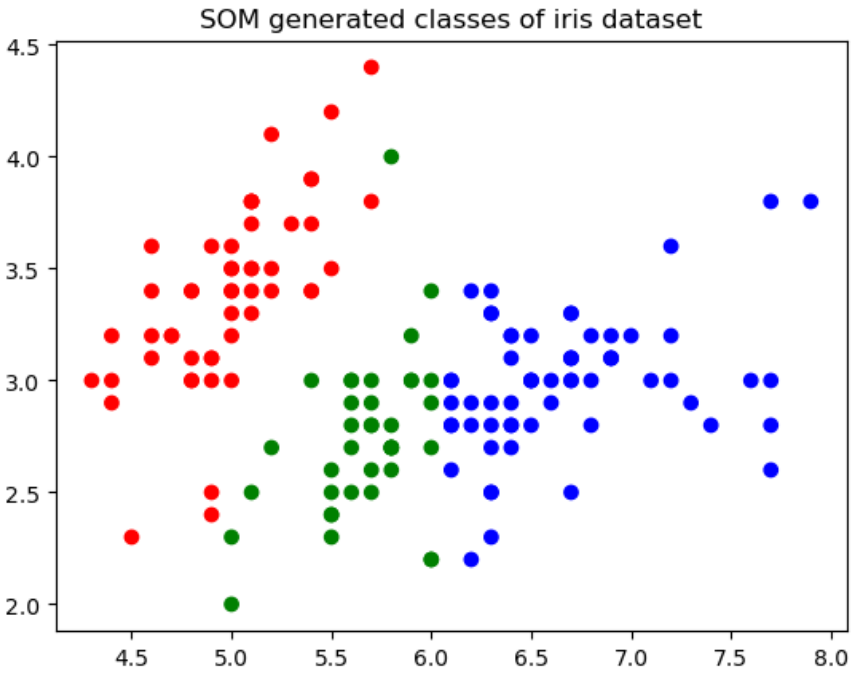
No, they are not the same. In a SOM, Neurons refer to units on a grid. Their configuration on this 2D grid is used to match the data set. Neurons are defined by a weight vector which is of the same dimensions as the input vector. Neurons on the output layer produce the BMU's position on the grid.

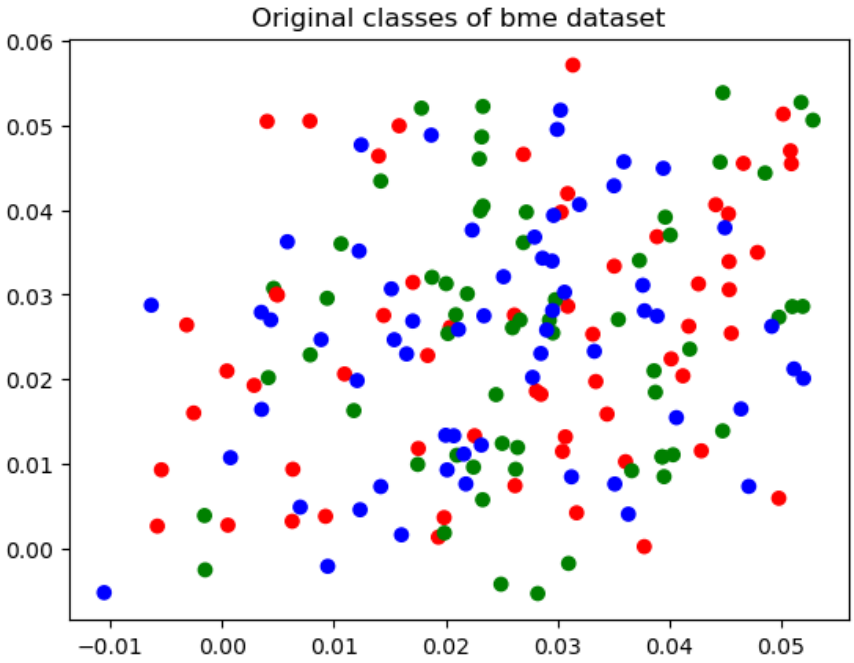
In a MLP, Neurons sit on layers. Multiple neurons sit on each layer. The MLP consists of an input layer, hidden internal layers and an output layer linked by connections between neurons. Neurons refer to computation units which take weighted inputs, perform computations and produce outputs for the next layer. Neurons on the output layer produce the classifications / predictions of the input data set.

2 Application of K-means and SOM for clustering









For the x and y coordinates we picked the first and second entry of the vectors in the Iris dataset. For BME this does not result in good graphs as BME has 128 entries for each vector. This is why we only show one plot for the BME dataset.

Table 1: Comparison of RAND indexes

	K-means (C)	K-means (Sklearn)	SOM (C)	SOM (Sklearn)
Iris	0.797	0.880	0.486	0.774
BME	0.579	0.563	0.136	0.450

We notice a way worse score for the BME dataset. This is probably due to the high dimensionality compared to the Iris dataset (4 vs 128). With more dimensions the room becomes more empty and the distance between points become less meaningfull. Also not all features contribute equally to distinguishing clusters. For 128 features its harder to distinguish between meaningfull, irrelevant and redundant features.

Table 2: Comparison of runtimes in s

K-means	C	Python
Iris	0.035	0.036
BME	1.17	0.56

Table 3: Comparison of runtimes in s

SOM	C	Python
Iris	0.074	0.039
BME	2.31	0.080

Regarding timing of the c implementation of both algorithms, given that c code is a much lower level implementation than python, our expectation was that it will have significantly lower execution time. However, we found that python was significantly faster. We reason that this occurred due to our specific implementation of C code. It is probably inefficient.

We note multiple for-loops that traverse the entire data set which adds significant overhead. We also note that we utilize heap memory allocation of arrays which is runtime allocation and adds further overhead.

Possible solutions to improve timing maybe adjusting compiler optimizations and combining multiple for-loops.

2.1 Questions

Are your C/C++ implementation getting the same clustering results and accuracy as the Python functions in the sklearn library? Can you validate the correctness of your program with the sklearn library?

The results for K-means are in a range of 10 % compared to the results from the Python functions. This is good enough to validate the correctness of the C-programm. The accuracy for Iris is worse where on the other hand the accuracy for BME is better than the results from Python. The results from the C-code for SOM are not as good as the results from Sklearn.

It was difficult to distinguish the correct size of the output matrix to only get 3 output classes.

Which algorithm is more efficient? Discuss not only execution time but also memory footprint, possibly power consumption. Can we draw a general conclusion?

Execution time: K-means is faster for both datasets in C and Python. This makes sense as it has a lower computational complexity.
Memory footprint: K-means only needs to store the centroid positions and cluster assignments. SOM has a much larger memory footprint as it needs to store the entire map structure.
Power consumption: K-means has a lower computational complexity resulting in a smaller power consumption.
All in all K-means is the more efficient algorithm.

What is the general challenge of the clustering problem? How can it be mitigated?

The general challenge is to find meaningful groupings within data. This data can be highly dimensional, noisy and can contain complex patterns. Some ways to mitigate that can be:

- Reduce the dimensionality of the data
- Choosing the right number of clusters
- Remove outliers/noise

3 Dynamic Time Warping (DTW

3.1 Hand-calculation

I filled up x with zeros for the Euclidean distance calculation:

$$\sqrt{(1-1)^2+(2-1)^2+(3-3)^2+(2-4)^2+(1-3)^2+(0-3)^2+(0-1)^2} \approx 4,36$$

Distance matrix:

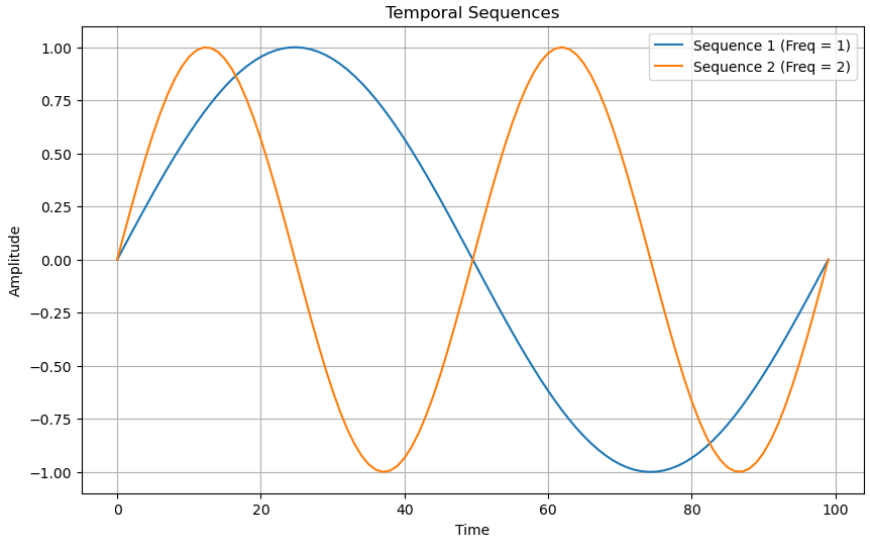
$$\text{Distance matrix} = \begin{bmatrix} 0 & 1 & 4 & 9 & 4 & 9 & 16 \\ 1 & 0 & 0 & 1 & 0 & 1 & 4 \\ 4 & 1 & 0 & 1 & 0 & 1 & 4 \\ 1 & 0 & 0 & 1 & 0 & 1 & 4 \\ 0 & 1 & 4 & 9 & 4 & 9 & 16 \\ g < & & & & & & \end{bmatrix}$$

$$\text{Accumulated cost matrix} = \begin{bmatrix} 0 & 1 & 5 & 14 & 18 & 27 & 43 \\ 1 & 1 & 1 & 2 & 2 & 3 & 7 \\ 5 & 2 & 1 & 2 & 2 & 3 & 7 \\ 6 & 2 & 1 & 2 & 2 & 3 & 7 \\ 7 & 3 & 2 & 3 & 4 & 5 & 9 \end{bmatrix}$$

Warping path:

$$(1,1) \to (2,2) \to (3,3) \to (4,4) \to (5,5) \to (5,6) \to (5,7) = 9$$

3.2 Temporal sequences



Euclidean distance ≈ 9.95
DTW distance ≈ 33.42

3.3 Questions

Give the formulas for calculating the Euclidean distance of two series x and y, both with n observations

$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

Describe the DTW algorithm steps for calculating the DTW distance

1. Create distance matrix (m x n) where each entry (i,j) is the Euclidian distance between the vector elements n_i and m_j

2. Create accumulated cost matrix (m x n). Initialize the top-left cell to the distance value from the distance matrix
3. Starting from top-left cell, calculate the accumulated cost at each cell by taking the minimum of the three neighboring cells (above, left, and diagonal) while adding the distance value from the distance matrix
4. Find a path to towards the bottom right while selecting the neighboring cell with minimum accumulated cost
5. The DTW distance is the accumulated cost at the bottom right cell

Compare the strength and weaknesses of DTW and the Euclidean distance.

DTW Strengths:

- Handles non linear alignments to find similarities in data although they are not perfectly aligned
- Flexible algning sequences with different lengths
- Robust to noise

DTW Weaknesses:

- Has higher computational complexity compared to Euclidean distance
- Needs lots of memory for storing distances for all pairs in time series

Euclidean strengths:

- Low computational complexity
- Easier to comprehend

Euclidean weaknesses:

- Sequences of different lengths have to be extended
- Does not account for time shifts
- Not suitable for time-series with varying speeds