VT2023: IL2233 Lab 3

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

Conglei Xiang & Yuqi Sun

May 18, 2023

Task 1

1. Draw a flow chart for each algorithm. The flow chart defines the data flow and control flow from the beginning to the end of algorithm execution.

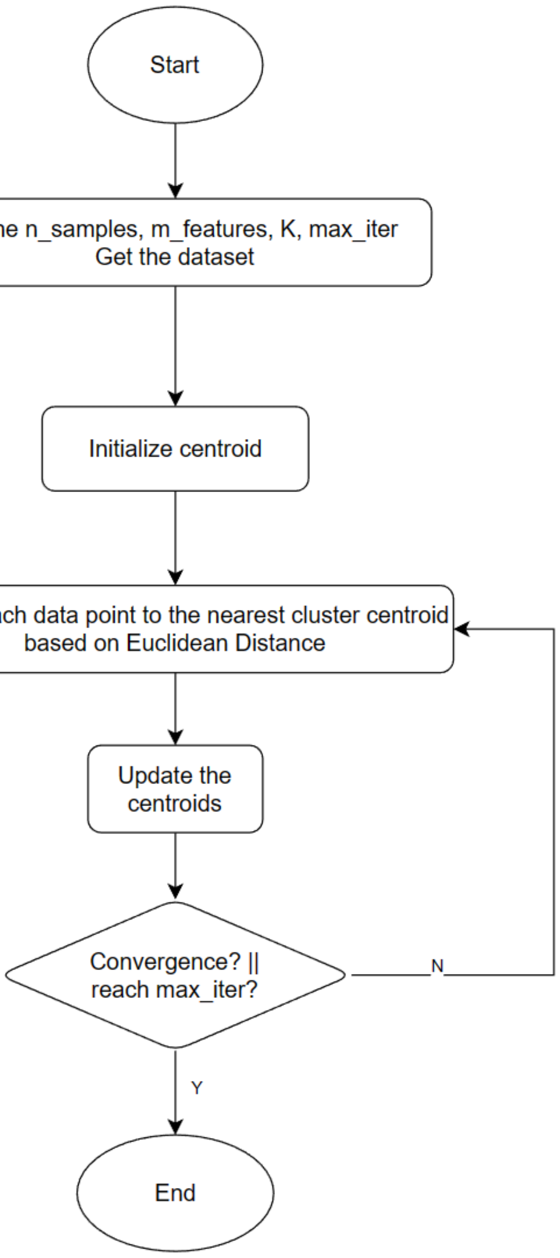

Start
Define n_samples, m_features, K, max_iter
Get the dataset
Initialize centroid
Assign each data point to the nearest cluster centroid based on Euclidean Distance
Update the centroids
Convergence? || reach max_iter?
N
Y
End

Figure 1 Flow chart of K-means algorithm

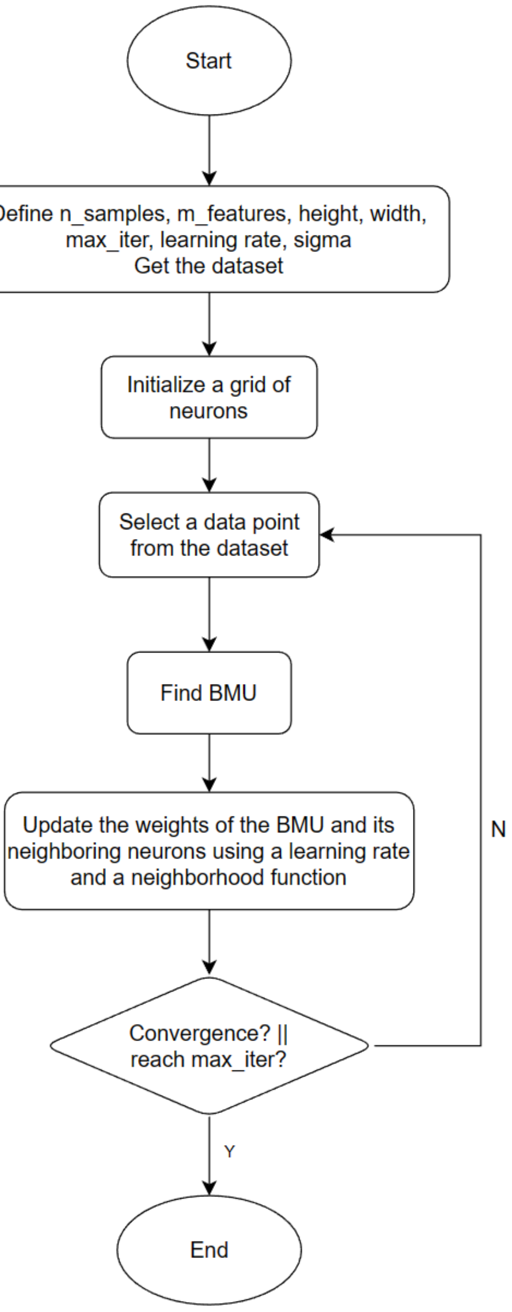

Start
Define n_samples, m_features, height, width, max_iter, learning rate, sigma
Get the dataset
Initialize a grid of neurons
Select a data point from the dataset
Find BMU
Update the weights of the BMU and its neighboring neurons using a learning rate and a neighborhood function
Convergence? || reach max_iter?
N
Y
End

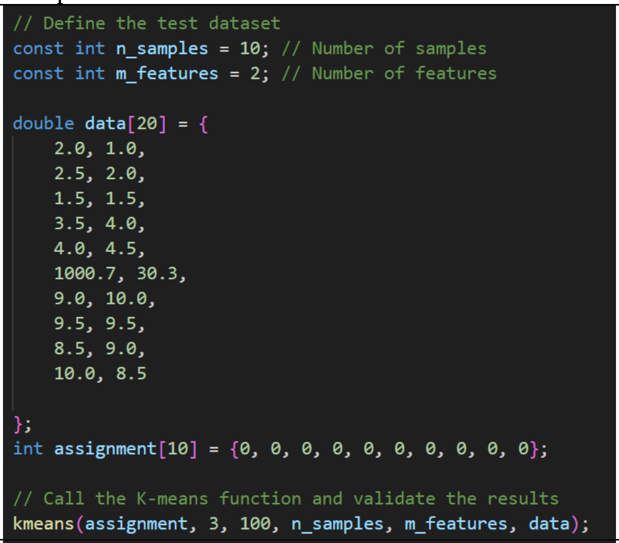
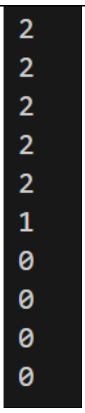
Figure 2 Flow chart of SOM algorithm

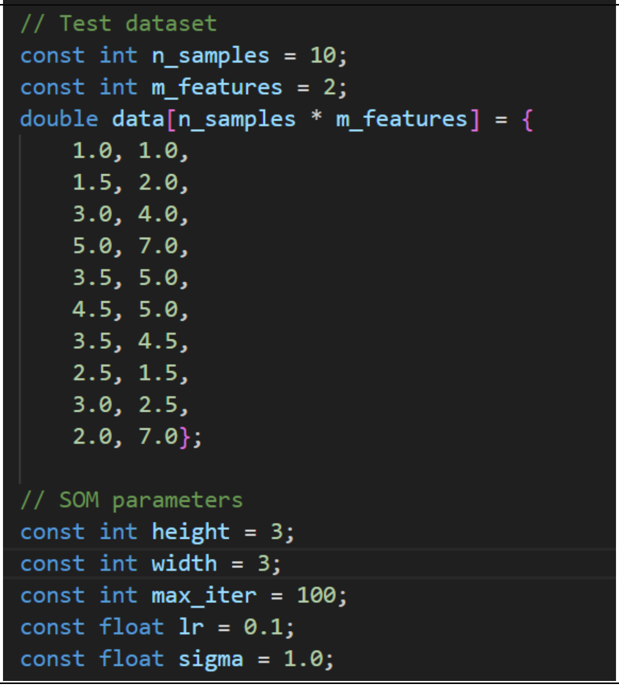
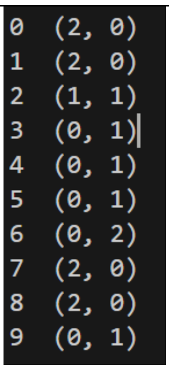
2. Follow the flow chart to implement the two algorithms in C/C++. It can be a good practice to write pseudo-code (which is language independent) before writing your implementation code in C/C++.

The code can be found in the file *'k_means_1.cpp'* and *'som_1.cpp'*.

3. Design your own test patterns to validate the correctness of your program. By test patterns, it means small but clear data points which allow you to reason about the correctness of your algorithms.

	Test pattern	Test result

K-means	 `// Define the test dataset`
	`const int n_samples = 10; // Number of samples`
	`const int m_features = 2; // Number of features`
	
	`double data[20] = {`
	`    2.0, 1.0,`
	`    2.5, 2.0,`
	`    1.5, 1.5,`
	`    3.5, 4.0,`
	`    4.0, 4.5,`
	`    1000.7, 30.3,`
	`    9.0, 10.0,`
	`    9.5, 9.5,`
	`    8.5, 9.0,`
	`    10.0, 8.5`
	
	`};`
	`int assignment[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};`
	
	`// Call the K-means function and validate the results`
	`kmeans(assignment, 3, 100, n_samples, m_features, data);`
		 `2`
		`2`
		`2`
		`2`
		`2`
		`1`
		`0`
		`0`
		`0`
		`0`

SOM	 `// Test dataset`
	`const int n_samples = 10;`
	`const int m_features = 2;`
	`double data[n_samples * m_features] = {`
	`    1.0, 1.0,`
	`    1.5, 2.0,`
	`    3.0, 4.0,`
	`    5.0, 7.0,`
	`    3.5, 5.0,`
	`    4.5, 5.0,`
	`    3.5, 4.5,`
	`    2.5, 1.5,`
	`    3.0, 2.5,`
	`    2.0, 7.0};`
	
	`// SOM parameters`
	`const int height = 3;`
	`const int width = 3;`
	`const int max_iter = 100;`
	`const float lr = 0.1;`
	`const float sigma = 1.0;`
		 `0 (2, 0)`
		`1 (2, 0)`
		`2 (1, 1)`
		`3 (0, 1)`
		`4 (0, 1)`
		`5 (0, 1)`
		`6 (0, 2)`
		`7 (2, 0)`
		`8 (2, 0)`
		`9 (0, 1)`

[Questions & Answers]

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

Because it determines the number of clusters the algorithm will attempt to find in the data. Each cluster represents a group of data points that are similar to each other and dissimilar to data points in other clusters. There is no universally optimal method to automatically search for the best value of "k" as it depends on the specific characteristics of the data and the problem domain, but there are some techniques that can assist in making an informed decision.

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

In the Self-Organizing Map (SOM) algorithm, learning occurs through the adjustment of the weight vectors associated with each neuron in the SOM grid. The learning process involves iteratively presenting input samples to the SOM and updating the weights to reflect the topological relationships and data distribution. The learning rate is a crucial parameter in the SOM algorithm. A higher learning rate results in larger weight updates, while a lower learning rate leads to smaller updates. Thus, if the learning rate is too high, it may cause overshooting and instability in the training process. On the other hand, a lower learning rate can result in slower convergence but potentially more stable and accurate clustering in the long run.

3. 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?

They are different. The neurons in the SOM grid act as prototypes to represent the data distribution, while MLP neurons perform computations and transformations for learning complex mappings between inputs and outputs.

Task 2

1. Use the given datasets (Iris, BME) to test your algorithms, and draw graphs in Python to illustrate the clustering results. One is the classical dataset, Iris dataset. The other is a simulated time-series dataset, BME. Note that the original use of the datasets is for classification, you need to concatenate both the training set and testing set as the input data for this task.

List the clustering accuracy results for each dataset for each algorithm. (In Rand index score) in table 1.

Table 1: Clustering accuracy comparison using RAND index

	K-means (C++ code)	K-means (Sklearn)	SOM (C++ code)	SOM (Sklearn)
Iris	0.868	0.880	0.556	0.559
BME	0.614	0.612	0.689	0.678

2. Write a small program to measure the execution time of your algorithms for clustering a given dataset. Do this for your C/C++ implementation and Python library. Write down the execution time in Tables 2 and 3.

Table 2: Execution time comparison (K-means)

	C/C++ code	Python code
Iris	1.518s	0.242s
BME	1.861s	0.241s

Table 3: Execution time comparison (SOM)

	C/C++ code	Python code
Iris	1.518s	0.051s
BME	1.482s	0.017s

[Questions & Answers]

1. 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?

Yes, the C++ implementation are getting similar clustering results and accuracy as the Python functions in the sklearn library. From Table 1, it can be observed that the RAND index score is really similar to each other for the same algorithm dealing with the same dataset. For Iris dataset, RAND index score is a bit higher using Python sklearn library. For BME dataset, the performance of using C++ algorithm is even a bit better than using Python sklearn library.

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

When comparing the efficiency of algorithms, we need to consider factors such as execution time, memory footprint, and possibly power consumption.

Execution time	K-means generally has a lower computational complexity compared to SOM. In terms of execution time, K-means can be faster than SOM, especially for large datasets and a large number of clusters.
Memory footprint	K-means typically requires less memory as it only needs to store the centroid positions and cluster assignments. On the other hand, SOM needs to store the entire map or grid structure, which can consume more memory, especially for large maps.
Power consumption	Since K-means generally requires fewer iterations and computations, it may consume less power compared to SOM.

Considering the execution time, memory footprint, and possibly power consumption, K-means is generally more efficient than SOM.

3. Which algorithm is more effective (accurate)? Can you draw a general conclusion?

The effectiveness or accuracy of clustering algorithms can depend on various factors such as the nature of the data, the choice of distance measure, and the appropriateness of the algorithm for the specific problem.
K-means is a hard clustering algorithm that assigns each data point to one cluster. It assumes that clusters are spherical and have equal variance. K-means can work well when the underlying clusters have these characteristics, and the data is well-separated.
SOM, on the other hand, is a self-organizing map that can capture complex data patterns and topology. It is particularly effective for visualizing high-dimensional data and preserving the neighborhood relationships of data points. SOM can handle non-linear relationships and is robust to noise.
The effectiveness or accuracy of K-means and SOM can vary depending on the dataset and the specific problem. In general, there is no one-size-fits-all conclusion on which algorithm is more effective. It is important to evaluate and compare the performance of both algorithms on specific datasets and problem domains.

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

Challenges of the clustering problem and mitigation:
The clustering problem poses several challenges, including the following:
a) Determining the optimal number of clusters: Selecting the appropriate number of clusters is often subjective and requires domain knowledge. It can impact the quality of clustering results.
b) Handling high-dimensional data: Clustering high-dimensional data can be challenging due to the curse of dimensionality, where the data becomes sparse and distances lose meaning. Dimensionality reduction techniques can be used to mitigate this challenge.
c) Dealing with different cluster shapes and densities: Clustering algorithms like K-means assume spherical clusters with equal variance. However, real-world data may have clusters of different shapes and densities. Using algorithms that can handle non-linear relationships and adapting distance measures (e.g., using distance metrics based on density or connectivity) can help overcome this challenge.
d) Handling noisy or outlier data: Noisy or outlier data can negatively impact clustering results. Preprocessing steps such as outlier detection and removal or robust distance measures can mitigate the effects of noisy data.
Mitigating these challenges often involves a combination of preprocessing techniques, appropriate algorithm selection, parameter tuning, and evaluation measures tailored to the specific problem domain.
In summary, the clustering problem requires careful consideration of algorithm choice, parameter selection, and evaluation methods to achieve accurate and meaningful results. The effectiveness and efficiency of clustering algorithms can vary depending on the specific problem, dataset, and constraints. It is important to evaluate and compare different algorithms to find the most suitable approach for a given task.

Task 3

1. Given two short sequences, x and y.

$$x = \{1, 2, 3, 2, 1\}$$
$$y = \{1, 1, 3, 4, 3, 1, 1\}$$

• Give a hand-calculation of the Euclidean distance: eu_distance(x,y) and the DTW distance: dtw_distance(x,y).

$$eu_{distance(x,y)} = \sqrt{(1-1)^2 + (2-1)^2 + (3-3)^2 + (2-4)^2 + (1-3)^2} = 3$$

$$dtw_{distance(x,y)} = 3$$

• Show the distance matrix and the accumulated cost (distance) matrix when calculating the DTW.

	1	**2**	**3**	**2**	**1**
1	0	1	3	4	4
1	0	1	3	4	4
3	2	1	1	2	4

4	5	3	2	3	5
3	7	4	2	3	5
1	7	5	4	3	3
1	7	6	6	4	3

• Give the warping path for DTW.

(0,0) -> (0,1) -> (1,2) -> (2,2) -> (2,3) -> (2,4) -> (3,5) -> (4,6)

2. Design two temporal sequences by yourself, e.g., using two frequency-customized sine waves, to compare and show the differences of the Euclidean distance and the DTW distance.

Sequence 1:

$\sin(4\pi t)$ Frequency: 2 Hz Amplitude: 1

Sequence 2:

$\frac{1}{2}\sin(6\pi t)$ Frequency: 3 Hz Amplitude: 0.5

Assume a time interval of 0.1 seconds.

Sequence 1:

$t = 0.0$ s, value $= \sin(2\pi * 2 * 0) * 1 = 0.0$

$t = 0.1$ s, value $= \sin(2\pi * 2 * 0.1) * 1 = 0.588$

$t = 0.2$ s, value $= \sin(2\pi * 2 * 0.2) * 1 = 0.951$

$t = 0.3$ s, value $= \sin(2\pi * 2 * 0.3) * 1 = 0.951$

$t = 0.4$ s, value $= \sin(2\pi * 2 * 0.4) * 1 = 0.588$

$t = 0.5$ s, value $= \sin(2\pi * 2 * 0.5) * 1 = 0.0$

Sequence 2:

$t = 0.0$ s, value $= \sin(2\pi * 3 * 0) * 0.5 = 0.0$

$t = 0.1$ s, value $= \sin(2\pi * 3 * 0.1) * 0.5 = 0.433$

$t = 0.2$ s, value $= \sin(2\pi * 3 * 0.2) * 0.5 = 0.75$

$t = 0.3$ s, value $= \sin(2\pi * 3 * 0.3) * 0.5 = 0.933$

$t = 0.4$ s, value $= \sin(2\pi * 3 * 0.4) * 0.5 = 0.933$

$t = 0.5$ s, value $= \sin(2\pi * 3 * 0.5) * 0.5 = 0.75$

Euclidean distance:

$$eu_{distance(x,y)}$$

$$= \sqrt{(0)^2 + (0.588 - 0.433)^2 + (0.951 - 0.75)^2 + (0.951 - 0.933)^2 + (0.588 - 0.933)^2 + (0 - 0.75)^2}$$

$$= \sqrt{0.746275}$$

$$\approx 0.864$$

DTW distance:

	0	**0.433**	**0.75**	**0.933**	**0.933**	**0.75**
0	0	0.433	1.183	2.116	3.049	3.799
0.588	0.588	0.155	0.317	0.662	1.007	1.169

0.951	1.539	0.673	0.356	0.335	0.353	0.554
0.951	2.49	1.191	0.557	0.353	0.353	0.554
0.588	3.078	1.346	0.719	0.698	0.698	0.515
0	3.078	1.779	1.469	1.631	1.631	1.265

$dtw_{distance(x,y)} = 1.265$

Answer the following questions.

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

$$eu_{distance(x,y)} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + \cdots + (x_n - y_n)^2}$$

2. Describe the DTW algorithm steps for calculating the DTW distance.

Step 1: Create a distance matrix D of size (n x m), where n is the length of series x and m is the length of series y. Initialize all entries of the matrix to positive infinity.

Step 2: Calculate the distance between each pair of elements in series x and y using a distance measure such as Euclidean distance and populate the distance matrix D accordingly.

Step 3: Create an accumulated cost matrix C of the same size as D. Initialize the top-left cell of C to the distance value from D.

Step 4: Starting from the top-left cell of C, calculate the accumulated cost at each cell by taking the minimum of the three neighboring cells (above, left, and diagonal) in C, and adding the corresponding distance value from D.

Step 5: Traverse the accumulated cost matrix C from the bottom-right cell towards the top-left cell, selecting the neighboring cell with the minimum accumulated cost at each step. This path represents the optimal warping path.

Step 6: The DTW distance is the accumulated cost at the bottom-right cell of C.

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

DTW:

Strengths:

Can handle sequences of different lengths.

Robust to temporal distortions, shifts, and speed variations.

Allows warping or stretching of sequences to find optimal alignments.

Widely used in time series analysis and pattern recognition tasks.

Weaknesses:

Computationally more expensive compared to Euclidean distance.

Sensitivity to noise and outliers in the data.

May require parameter tuning for optimal alignment results.

Difficult to interpret and visualize the warping path in complex cases.

Euclidean distance:

Strengths:

Simple and straightforward to compute.

Suitable for comparing sequences of equal lengths.

Less computationally intensive compared to DTW.

Intuitive and easy to interpret.

Weaknesses:

Cannot handle sequences of different lengths without preprocessing.

Insensitive to temporal variations or time shifts.

Not suitable for comparing sequences with speed variations or shape distortions.

Limited applicability to time series analysis tasks.

Overall, DTW is advantageous when dealing with time series data that exhibit temporal variations, speed differences, or shape distortions. It offers more flexibility and robustness in alignment and similarity comparisons. However, it comes at the cost of increased computational complexity and sensitivity to noise. Euclidean distance, on the other hand, is simpler and faster but lacks the ability to handle sequences with varying lengths or temporal distortions. The choice between DTW and Euclidean distance depends on the specific characteristics and requirements of the data and the analysis task at hand.