KTH Royal Institute of Technology


Embedded Intelligence
IL2233 VT24

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

Humblet Raphaël
Yao Tianze

May 2024

Academic year 2023-2024

# Contents

# 1 Task 1: Implement and evaluate K-means and SOM

After programming the two functions, a test data set has been created.

It consists of 2 features data. The point are the following: "*[1,1], [2,2], [3,3], [4,4], [4,4], [7,7], [7,7], [8,8], [9,9], [10,10]*. This can clearly be divided into two clusters, and both programs do it correctly.

The flow charts for K-means and SOM algorithm are shown in Figure.1.

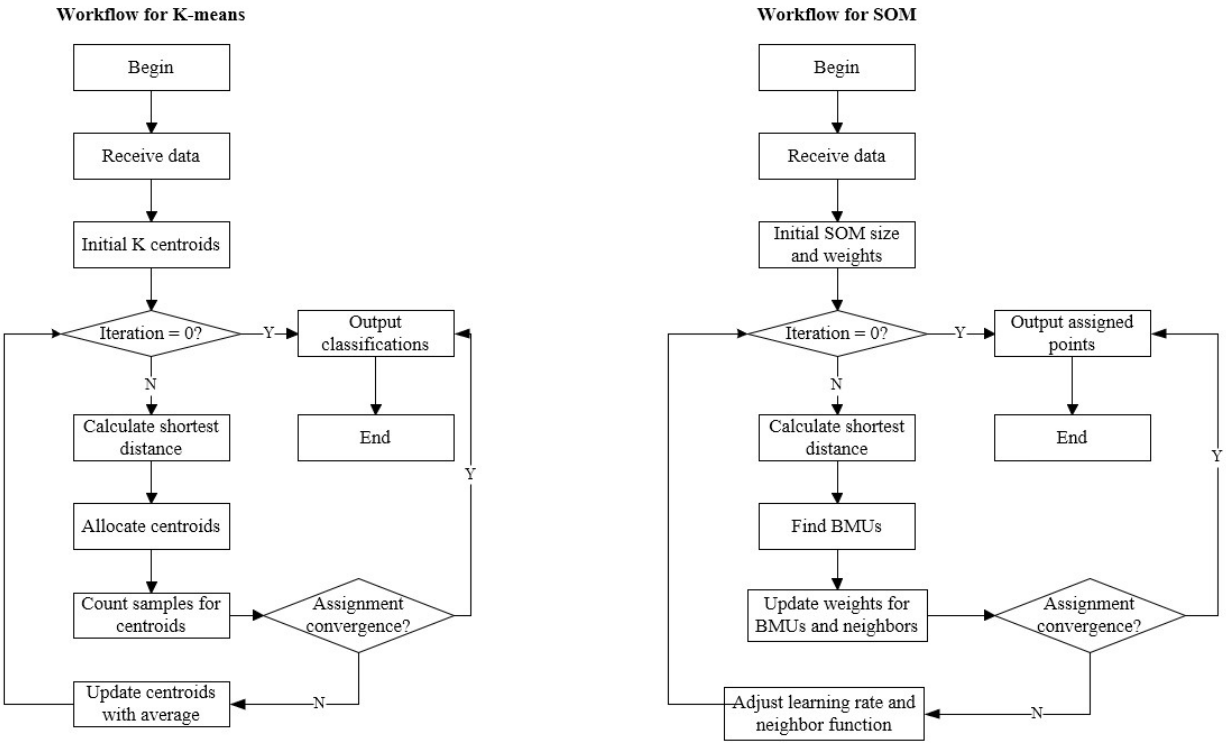


Figure 1: Flow charts for two algorithms

**In the K-means algorithm, "k" needs to be given. Why? Is it feasible to automatically search a good value for "k"?**
In the K-means algorithm, "k" represents the number of clusters you want to partition your data into. This parameter needs to be provided because it defines the number of centroids around which the clusters will be formed. Without specifying "k", the algorithm wouldn't know how many clusters to create.

It could be possible to automatically search a good value for "k" but it is not easy and it is not in the scope of the K-means algorithm.

**For the SOM algorithm, how is the learning happening? How does the learning rate affect the performance of the algorithm?**
For the SOM (Self-Organizing Map) algorithm, learning consists in a competitive learning process based on multiple steps:

1. **Initialisation:** The SOM starts with an array of neurons in a two dimensional grid. Each neuron is associated to a weight vector of the same dimensionality as the input data.

2. **Competition:** During training, for each input data point, the neuron with the weight vector closest to the input (usually measured by Euclidean distance) is determined as the winner or the Best Matching Unit (BMU).

3. **Cooperation and Adaptation:** The weights of the BMU and its neighbouring neurons are adjusted to make them more similar to the input data. The learning rate and the weight update rate is controlled by the number of iteration (negative exponential).

4. **Iteration:** Iterate 2 and 3 until max_it is reached.

The learning rate affects the performance of the algorithm depending on its value. If the rate is too high, the SOM can converge quickly but it may result in unstable learning, meaning that it may converge to suboptimal solutions or even divergence.

If it is too low, it will consume a lot of time and computations. And it risks converging towards local minimas.

In order to avoid these problems, we use an adaptive learning rate that decreases over time as the training progresses. It helps balance fast learning in the beginning and more refined adjustments later in training.

**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?**
The "neuron" in the output layer of SOM is not the same as the neuron in a MLP (Multi-Layer Perceptron).

In a SOM, the neurons in the output layer represents the regions of the input space and compete to represent input patterns.

In an MLP, the neurons in the output layer usually represents a class labels or continuous outputs and work together to produce the final output based on the input data.

# 2 Task 2: Application of K-means and SOM for clustering

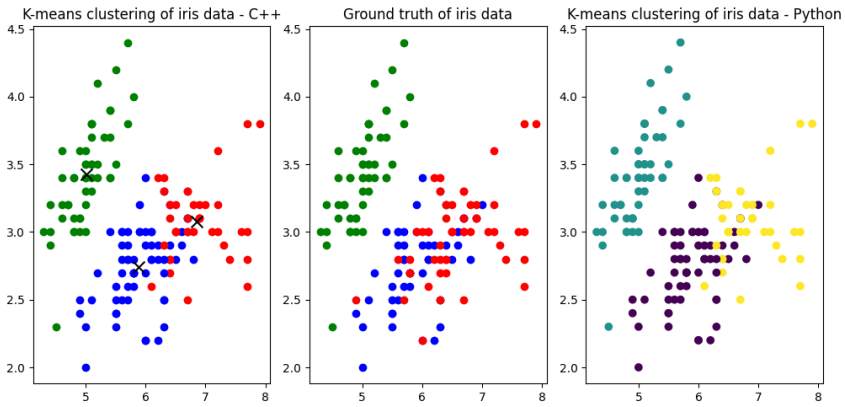


Figure 2: Iris clustering results with kmeans

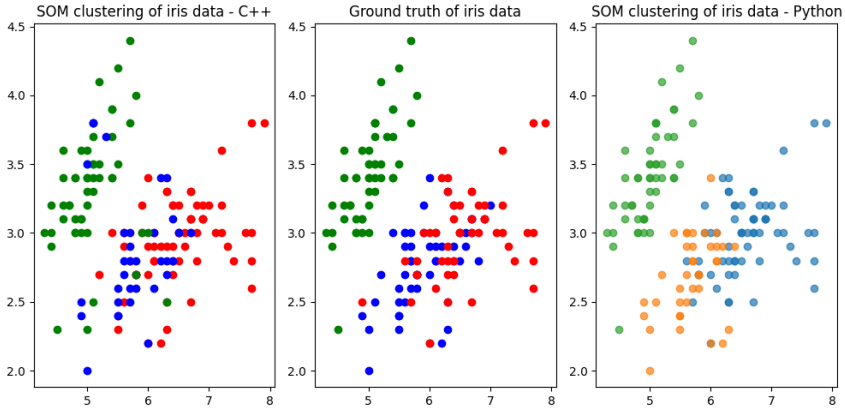


Figure 3: Iris clustering results with SOM

1. The clustering accuracy comparision is shown in Table.1. The RAND index has been computed with the function from sklearn. To calculate RAND index, we need and manage to find centroids in our results and label the points closed to each centroid. With the labels of all points, we compared it with true labels via sklearn and output the RAND index for each case.
   The clustering results for the IRIS dataset are shown in Figure 2 and 3. The

| | K-means (C++) | K-means (Sklearn) | SOM (C++) | SOM (Sklearn) |
|---|---|---|---|---|
| Iris | 0.88 | 0.88 | 0.725 | 0.89 |
| BME | 0.64 | 0.63 | 0.63 | 0.63 |

Table 1: Clustering accuracy comparision using RAND index

   results are not shown for the BME dataset as there are too many features (128) and a plot would be irrelevant.

2. The execution time for all algorithms are measured in Table.2

| | K-means (C++) | K-means (Sklearn) | SOM (C++) | SOM (Sklearn) |
|---|---|---|---|---|
| Iris | 4ms | 73ms | 42ms | 142ms |
| BME | 18ms | 194ms | 133ms | 158ms |

Table 2: Time consuming for algorithms

**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?**
As we can see, the C++ code is faster than the python code. This was expected. We can also see that the accuracy is mostly the same, proving that our implementation of kmeans and SOM is good.

**Which algorithm is more efficient? Discuss not only execution time but also memory footprint, possibly power consumption. Can we draw a general conclusion?**
From our measurement, SOM is more precise in some cases. However, if considering memory and power, SOM tends to be more consuming and inefficient compared to K-means. SOM needs a whole neural network to calculate and pass weight on each nodes, and SOM also requires more resource to map and recur. On the contrary, K-means just use recursions to find centroids, it usually can quickly finish converging. In this case, the result may be different because the neural network assigned to the input data is small, and the number of max recursion time is also different between the two algorithms.

**Which algorithm is more effective (accurate)? Can you draw a general conclusion?**
The SOM seems to be more accurate, but depending on the case. This may be because SOM is sensitive to the dimensions of data, like the number of features a sample has, and in our Iris case we use all 4 features to train. In BMG case, there are more uncommon data in samples, which is sensitive to K-means because the uncommon data can drag some centriods a lot and may cause some errors.

**What is the general challenge of the clustering problem? How can it be mitigated?**
The challenges can be large overhead to calculate great dimension size, choose appropriate samples or number of clusters to train, properly initial centroids, calculation complexity, etc. To mitigate these problems, we can choose more characteristic segments of data to train, evaluate the distribution of samples before choosing centroids, use a few dimensions to replace the whole dimensions of data and many other ways similar to these methods.

# 3 Task 3: Dynamic Time Warping (DTW)

1. First calculate the Euclidean distance, make sequence x as x = {1,2,3,2,1,0,0 }, and calculate:

$$eu\_distance(x, y) = \sqrt{\sum_{i=1}^{7} (x_i - y_i)^2} = 3.317$$

Then calculate the DTW, define a distance matrix:

$$D\_matrix = \begin{bmatrix} 0 & 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 & 0 \\ 4 & 1 & 0 & 1 & 4 \\ 9 & 4 & 1 & 4 & 9 \\ 4 & 1 & 0 & 1 & 4 \\ 0 & 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 & 0 \end{bmatrix}$$

Output the accumulated cost matrix:

$$A\_matrix = \begin{bmatrix} 0 & 1 & 5 & 6 & 6 \\ 0 & 1 & 5 & 6 & 6 \\ 4 & 1 & 1 & 2 & 6 \\ 13 & 5 & 2 & 5 & 11 \\ 17 & 6 & 2 & 3 & 7 \\ 17 & 7 & 6 & 3 & 3 \\ 17 & 8 & 10 & 4 & 3 \end{bmatrix}$$

So the DTW is $\sqrt{3} = 1.732$, with the wraping path (0, 0), (1, 1), (2, 2), (3, 2), (4, 3), (5, 4), (6, 4).

2. For the formula of Euclidean distance in this case, with n observations,

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

The two sine waves are defined as Figure.4.
And we got the Euclidean distance of 14.11.

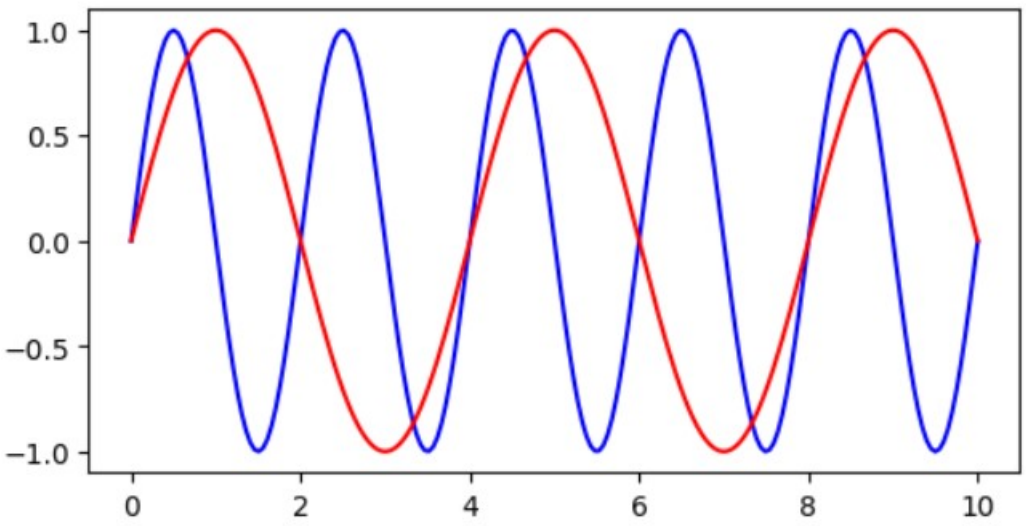


Figure 4:

For DTW calculation, first we need to create a distance matrix based on the number of observations, then use accumulation to get the cost matrix, finally

find the wraping path and use the lower right element of the cost matrix to calculate the DTW.

For Euclidean distance, this distance is very obvious to evaluate and calculate, it also has lower complexity in calcutaion and wider adaptation to different data, but this distance can't (or not suitable to) handle samples with different dimensions, it's also sensitive to uncommon data.

For DTW, it has a perfect coverage over time series and orders of data, which is very useful in matching and it's capable for samples with different dimensions. However, this algorithm is complex and time-consuming in calculation, its dynamic algorithm requires more memory and resource to run, also it is not obvious compared to Euclidean distance.