VT2024: IL2233 Lab 3
Time-Series Clustering with K-Means and SOM, and Similarity Measuring with DTW

Zhonghai Lu

May 4, 2024

1 Introduction

K-Means and Self-Organizing Map (SOM) are two representative clustering algorithms. K-means is a simple classical machine learning algorithm, while SOM is a competitive learning based neural network model. Both models belong to unsupervised learning. To facilitate the understanding and implementation of the K-means and SOM algorithms, we describe them in more detail in the following.

One fundamental concept in clustering is the measure of distance or similarity. One common metric is the Euclidean distance. However, it cannot properly reflect the distance between similar sequences with time elasticity. In this regard, one important algorithm is Dynamic Time Warping (DTW) which allows to discover similarities between temporal sequences (e.g. audio clips) with different "speed". We will experiment with DTW in contrast to the Euclidean distance.

1.1 K-Means Algorithm

The K-Means algorithm is one of the common partitioning clustering methods, and its main idea is to minimize the total distance between all objects inside a cluster while maximizing the distance between clusters. Figure 1 shows an example of a two-cluster K-means, in which the red cluster contains 7 samples around its center while the blue cluster contains the other 5 samples distant from the red cluster.

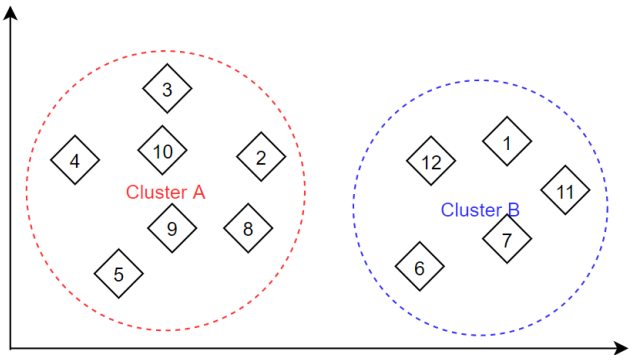

Figure 1: The K-means example with two clusters.

Suppose we have a set of time series $x_1, x_2, , x_n$, and K clusters $C_1, C_2, , C_k$. We want the distance of the time series inside the cluster close to each other, which means that we can determine a mathematical centroid of the cluster and shorten the distance of the time series to this centroid. Another target is to spread these centroids away from others to well separate the clusters. To solve this problem, the principle of the K-means algorithm is based on the *Expectation-Maximization* algorithm. It has two steps: the first step is the Expectation step, in which the time series are labeled according to the centroids; the second step is the Maximization step, in which the centroids are moved to the newly labeled time series.

To explain the steps, we define a symbol matrix τ_{nk}, if the time series x_n belongs to the cluster C_k, then $\tau_{nk} = 1$, otherwise $\tau_{nk} = 0$. In the expectation step, we can define the symbol matrix by the randomly initialized centroids, or the new centroids from the maximization step in the previous iteration. The purpose of the maximization step is to minimize the loss function with the symbol matrix τ_{nk}.

The loss function can be defined as $L = \sum_{i=1}^{k} \sum_{j=1}^{n} \tau_{ji} \times \nu(x_j, \mu_i)$, where μ_i is the centroid of the cluster C_i and $\nu(x_j, \mu_i)$ is the loss function related to the chosen distance measurement method. For example, with the Euclidean distance, we can apply the square loss function $\nu(x_j, \mu_i) = (x_j - \mu_i)^2$. Then to minimize the loss function, we take partial derivative on it by μ_i and let the result equal to zero. We have $\frac{\partial L}{\partial \mu_i} = 2 \cdot \sum_{j=1}^{n} \tau_{ji}(x_j - \mu_i) = 0$. This means we can calculate the new centroid during the maximization step: $\mu_i = \frac{\sum_{j=1}^{n} \tau_{ji} x_j}{\sum_{j=1}^{n} \tau_{ji}}$.

Each iteration of the K-means algorithm is composed of an expectation step and a maximization step. During the iterations, the cluster converges to the optimal solution. The K-means algorithm has the advantage of fast convergence speed. The clustering result is easy to compute and explain. However, it may fall into the local minimum, which depends on the selection of K and initial centroids (weights).

1.2 Self-Organizing Map (SOM)

Self-Organizing Map (SOM) is a special kind of neural network designed to solve clustering tasks, and it also intends to expose the geometry relationship of the objects in the dataset.

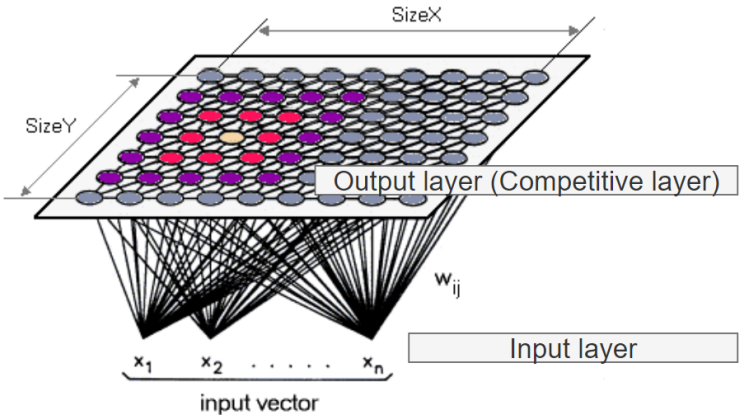

Figure 2: The two-layer structure of SOM.

As illustrated in Figure 2, the structure of SOM is a two-layer neural network, which contains the input layer and output layer (competitive layer). The input layer simulates the perception channel, through which the original time series pass. The output layer simulates the response channel. The number of neurons in this layer normally represents the number

of clusters we desire. The training process is a competitive process. Each input is assigned to the best matching unit as the winning neuron. Then we can use stochastic or batch training to update the weight of the winning neuron, and appropriately update its neighbors' weight according to the distance of the winning neuron. To define the range of neighbors, the output layer is normally designed as a two-dimensional topology structure, such as a rectangular grid, or hexagonal grid. Suppose we have an input set of time series x_i, with the length of n, and the weights connect the two layers between the ith input neuron and the jth output neuron $w_{ij}, i = 1, \cdots, n, j = 1, \cdots, k$. k is the number of expected clusters. Then the learning process can be separated as follows:

1. Initialization: The weights of the SOM are first initialized, e.g. with some small random numbers.

2. Competition: Each input will find its best matching unit using some judgment methods, in time series, i.e. the distance metric. Assume the winning unit is BMU.

3. Cooperation: The BMU decides the range of its neighbors to update their weights. Here we use one example based on Gaussian distribution. Suppose S_{ij} is the geometry distance between neuron i and j, $\sigma(t) = \sigma_0 \exp(-\frac{t}{\tau})$ is the parameter decay as iteration grows, with the initial standard deviation σ_0, total iteration number τ and current iteration number t. Then the update distribution T of node j is given by $T_{j,BMU}(t) = \exp(-\frac{D^2_{j,BMU}}{2\sigma(t)^2})$. The closer neighbor will get larger update.

4. Adaption: The weights of the neurons are updated by $\Delta w_{ij} = \eta(t) \times T_{j,BMU}(t) \times (x_i - w_{ij})$. In the equation, the learning rate is defined as $\eta(t) = \eta_0 \exp(-\frac{t}{\tau})$.

5. Iteration: Go back to the above steps from the competition, until all the iterations are done. The final winning neurons of the input time series are their clusters.

1.3 Dynamic Time Warping (DTW)

In time series analysis, dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal sequences. In contrast to the Euclidean distance, it can properly handle sequences that have different lengths and may vary in speed. For instance, it can detect similarities in a person's walking, even if the person was walking faster or slower, or if there were accelerations and decelerations during the course of walking. Similarly this goes to talking. Similarities in audio clips talking the same words but with different speeds or delays can be found using DTW, but difficult using the Euclidean distance.

2 Experimental Purpose and Setup

The laboration intends for the students to deepen their understanding of the clustering algorithms such as K-Means and SOM, and be able to implement the algorithms and use them to solve practical tasks. In addition, we have a task on the common similarity measures: DTW and the Euclidean distance. The lab has the following objectives:

- Implement K-Means and SOM algorithms in C or C++ and validate their correctness by comparing them with a corresponding high-level API library.

- Use K-Means and SOM for clustering time-series data and evaluate their performance in a comparative manner. Evaluate their execution time and accuracy for clustering applications, and discuss the suitability for deploying them on embedded microprocessors.
- Know how to calculate the Euclidean distance and the DTW distance, and be able to show the advantages of DTW in comparison with the Euclidean distance.

For the high-level API, you can use Python machine learning libraries, such as the scikit-learn package.

- sklearn K-means: `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`
- sklearn-som: `https://pypi.org/project/sklearn-som/`
- minisom: `https://github.com/JustGlowing/minisom`
- For DTW, fastdtw `https://pypi.org/project/fastdtw/`

This lab is group work. Each group consists of 2 students.

3 Tasks

To achieve the objectives, this lab comprises the following sequential tasks.

3.1 Task 1. Implement and evaluate K-means and SOM

In this task, you will implement the K-means and SOM algorithms in C/C++ and evaluate their performance. You write a function to implement each algorithm. To facilitate the testing of your algorithm functions, your functions shall have the same calling interface. In both algorithms, we use the Euclidean distance metric.

For K-means, your function has the calling template:

```
void kmeans(int *assignment, int K, int max_iter, int n_samples,
int m_features, double *data)
```

where data is the input dataset, with n_samples and m_features, K is the pre-defined number of clusters, max_iter is the number of iterations the K-means updates, and assignment is the clustering result, as a list of integer int[n_samples].

For SOM, your function has the calling template:

```
void SOM(t_pos *assignment, double *data, int n_samples,
int m_features, int height, int width, int max_iter,  float lr, float sigma)
```

where data is the input dataset, with n_samples and m_features, the output grid is a rectangle shaped in height and width, max_iter is the number of iterations the SOM updates, assignment is the clustering result, as a list of positions t_pos[n_samples], and the learning rate is lr, and the weight update is controlled by rate sigma.

To clarify your implementations steps, this task can be split into the following sub-tasks:

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.

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++.

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.

4. For C/C++ implementation of SOM, you can focus on the essential functions of competition and cooperation, and simplify the updating and neighbor selecting part.

After completing this task, hopefully you have a full understanding of how the K-means and SOM algorithms work and how they perform. Now it is time to answer the following questions:

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

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

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?

3.2 Task 2. Application of K-means and SOM for clustering

In this task, you will use K-means and SOM to do clustering and performance evaluation. You are going to use both your own C/C++ implementations of the algorithms and the function calls in the sklearn library.

To be concrete, this task may be split into the following subtasks.

1. Clustering given time-series data using K-means and SOM.

2. Evaluate the performance (clustering accuracy) of the clustering algorithms measured in RAND index.

   RAND index is a criterion for clustering quality, which compares the number of the data pairs co-assigned or separated in true clusterings according to their mutual similarity with the total number of pairs.

   $$RI = \frac{TP + TN}{TP + FP + TN + FN},$$

   where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. The similar pair co-assigned in the same cluster can be considered as a TP, and vice versa. True negative (TN) indicates non-similar samples in different clusters.

3. Compare the execution time of the K-means and SOM algorithms in your C/C++ implementation and the Python machine learning package implementation.

Specifically, complete the following tasks:

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 (Your C/C++ code)	K-means (Sklearn)	SOM (your C/C++ code)	SOM (Sklearn)
Iris				
BME				

   You can find the dataset on the website:

   Iris: `https://archive.ics.uci.edu/ml/datasets/iris`

   BME: `http://www.timeseriesclassification.com/description.php?Dataset=BME`

   You can also view the UCR Time Series Classification Archive from this link, `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/` and try some interesting datasets by yourself.

   You can read about the Rand Index here `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.rand_score.html`

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		
BME		

Table 3: Execution time comparison (SOM)

	C/C++ code	Python code
Iris		
BME		

After completing this task, you know how to apply the K-means and SOM algorithms and how they perform. Now it is time to answer the following questions:

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?

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

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

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

3.3 Task 3. Dynamic Time Warping (DTW)

In this task, you are going to calculate the Euclidean distance and the DTW distance by hand. In addition, you are going to show the advantages of the DTW in contrast to the Euclidean distance. There are two sub-tasks as follows.

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).
   - Show the distance matrix and the accumulated cost (distance) matrix when calculating the DTW.
   - Give the warping path for DTW.

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.

Answer the following questions.

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

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

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

4 Task 4. Lab completion and documentation, deliverables

After completing the lab tasks above, you are not complete yet. You need to do the following to finalize:

1. Approval: ask a lab assistant to check your results and approve your lab completion. The lab assistant will also ask all questions present in this lab manual and possible other questions regarding your implementations.

2. Write report: write a short technical document, where you write down what you have done, how you have done it, what results you have obtained, and discuss the results (often the questions in the lab manual contain discussion points, and present conclusions in your own words. Try to use figures and tables to assist your explanations and discussions.

3. Prepare deliverables. Prepare a deliverable that compresses all your source code and the result figures in one zip file. Write a short Readme.txt file to describe the zip folder structure and purpose of each file.

4. Submit report and deliverables: After (1) (2) (3), you submit your technical report together with the deliverable zip file to the Canvas course page below.

   The lab assistant will review your report and, if accepted, your lab is counted as completed. If not, comments will be given to you for revision and re-submission of your technical report.

   `https://canvas.kth.se/courses/46239`