



Project 3

Learn and implement

K-Mean Algorithm

Member

17110008 – Hồ Ngọc Đình Châu

17110195 – Nguyễn Thị Bích Nhàn

GVHD : TS.Lê Văn Vinh

Thu Duc, November 16,2020

**Contents**

[**A.** **Introduction** 3](#_Toc56399879)

[**a.** **Introduction K-Means** 3](#_Toc56399880)

[**b.** **Application in practice** 4](#_Toc56399881)

[**B.** **What is the K-Means algorithm ?** 5](#_Toc56399882)

[**C.** **The idea of the K-Means algorithm** 5](#_Toc56399883)

[**a.** **Step by step** 5](#_Toc56399884)

[**b.** **Attention** 6](#_Toc56399885)

[**i.** **Choosing the number of clusters ?** 6](#_Toc56399886)

[**ii.** **Initialize k initial positions.** 6](#_Toc56399887)

[**iii.** **On the problem of stopping (convergence).** 6](#_Toc56399888)

[**D.** **Implement K-Mean Algorithm** 7](#_Toc56399889)

[**a.** **Issue** 7](#_Toc56399890)

[**b.** **Library** 7](#_Toc56399891)

[**i.** **Numpy** 7](#_Toc56399892)

[**ii.** **Matplotlib** 7](#_Toc56399893)

[**iii.** **Scipy** 8](#_Toc56399894)

[**E.** **Code** 8](#_Toc56399895)

[**a.** **Overview** 8](#_Toc56399896)

[**b.** **Analysis** 11](#_Toc56399897)

[**i.** **Library declaration** 11](#_Toc56399898)

[**ii.** **Initialization** 11](#_Toc56399899)

[**1.** **numpy.random.multivariate\_normal (mean, cov[, size])** 12](#_Toc56399900)

[**2.** **numpy.concatenate((a1, a2, ...), axis=0, out=None)** 13](#_Toc56399901)

[**iii.** **Function** 15](#_Toc56399902)

[**a.** **def kmeans\_init\_centers(X, n\_cluster)** 15](#_Toc56399903)

[**i.** **numpy.random.choice(a, size=None, replace=True, p=None)** 16](#_Toc56399904)

[**b.** **def kmeans\_predict\_labels(X, centers)** 17](#_Toc56399905)

[**i.** **cdist(XA,XB)** 17](#_Toc56399906)

[**ii.** **numpy.argmin(a, axis=None, out=None)** 18](#_Toc56399907)

[**c.** **def kmeans\_update\_centers(X, labels, n\_cluster)** 19](#_Toc56399908)

[**i.** **numpy.zeros(shape, dtype=float, order='C')** 19](#_Toc56399909)

[**ii.** **numpy.mean(a,axis=None,dtype=None,out=None,keepdims=<no value>)** 20](#_Toc56399910)

[**d.** **def kmeans\_has\_converged(centers, new\_centers)** 21](#_Toc56399911)

[**e.** **def kmeans\_visualize(X, centers, labels, n\_cluster, title)** 21](#_Toc56399912)

[**iv.** **Summary** 22](#_Toc56399913)

[**v.** **Main** 23](#_Toc56399914)

[**a.** **Code** 24](#_Toc56399915)

[**b.** **Runtime** 25](#_Toc56399916)

[**F.** **Tài liệu kham thảo** 32](#_Toc56399917)

**Image in report**

[Figure 1 : Create data 15](#_Toc56399918)

[Figure 2 : Initial coordinates of k 17](#_Toc56399919)

[Figure 3 : Init centers in the first run 24](#_Toc56399920)

[Figure 4 : Data and cluster in the first time 26](#_Toc56399921)

[Figure 5 : Classify data in clusters 26](#_Toc56399922)

[Figure 6 : Update the new location of the clusters 27](#_Toc56399923)

[Figure 7 : Update the new location of the clusters at time = 2 28](#_Toc56399924)

[Figure 8 : Update the new location of the clusters at time = 3 29](#_Toc56399925)

[Figure 9 : Update the new location of the clusters at time = 4 30](#_Toc56399926)

[Figure 10 : Update the new location of the clusters at time = 5 31](#_Toc56399927)

[Figure 11 : Update the new location of the clusters at tme = 6 32](#_Toc56399928)

**Teacher review**

…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

Sign

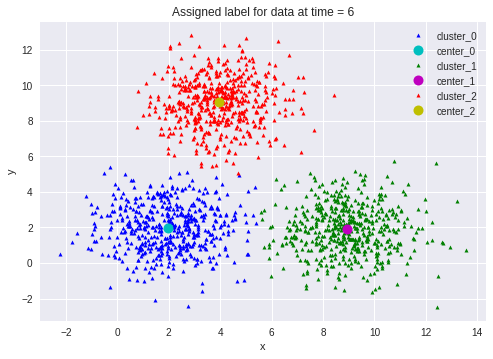
1. **Introduction** 
   1. **Introduction K-Means**

K-means is a simple clustering algorithm of the unsupervised learning type (i.e. unlabeled data) and is used to solve the clustering problem.

The idea of the k-means clustering algorithm is to divide 1 set of data into different clusters. In which the number of clusters is given to as k.

Clustering work is established based on the principle: Data points in the same cluster must have the same certain properties. That is, the points in the same cluster must be related to each other. For a computer, the points in that cluster will be data points that are close together.

The k-means clustering algorithm is often used in search engine applications, customer segmentation, data statistics, ...



* 1. **Application in practice**

Because of the basic ability of k-Means Clustering to break the original data into small groups, all operate on the algorithm without requiring any user's knowledge of the collected data ( big - small, ugly - beautiful, distorted - round). It can be used to confirm hypotheses about how many groups should be divided into which groups, when the amount of data obtained is large and complex. Once these two parameters are specified, any new sample will easily be labeled in the correct position.

This is a flexible algorithm that can be applied to any sorting and grouping process. Some examples are as follows:

For example:

1. In banking transactions

The classification of customer data is especially important, often based on which to make general policies for the whole system or to have care policies for each customer. Some of the classifications based on user behavior are as follows:

* Classification based on payment history (spending).
* Classification based on activity on a mobile app, on a website, or on an ATM platform.
* Defines personal characteristics of customers based on their interests (through shopping history).
* Create customer profiles based on activity tracking data.

1. Classification of inventions by the department of science and technology.

* Group of inventions based on business activity.
* Group of inventions based on the manufacturing sector.

1. Sensor classification by function:

* The detection method works in the motion sensor group.
* Group photos.
* Audio file classification.
* Grouping in health monitoring.

1. **What is the K-Means algorithm ?**

The k-means clustering algorithm is a method used in clustering analysis of data. It is especially heavily used in data mining and statistics. It partitions data into k different clusters. This algorithm helps us to determine which group of our data actually belongs to.

In business models, businesses will split the customer file into different groups of objects so that you can apply specific business strategies to each audience. This helps customers to get access to products that are truly suitable for them. That fit will increase our sales. The problem is how can we split that customer file when the number of invoices is very large and we cannot sit down to analyze each customer.

* **The goal of clustering algorithms is from that huge data set.**

1. **The idea of the K-Means algorithm**
   1. **Step by step**

|  |  |
| --- | --- |
| **Step** | **Content** |
| **1** | Initialize K data points in the data set and temporarily treat it as the center of our data clusters. |
| **2** | For each data point in the data set, its cluster center will be identified as 1 of the K nearest cluster centers. |
| **3** | After all the data points are centered, recalculate the position of the cluster center to ensure that the cluster center is in the center of the cluster. |
| **4** | Steps 2 and 3 will be repeated until the position of the cluster center does not change or the center of all data points does not change |

* 1. **Attention**
     1. **Choosing the number of clusters ?**

Just choose the number of clusters k can be divided into a separate problem. There is no 1 k number that is reasonable for all problems. Can you read through your data set to determine how many clusters there might be? But you can't always do that. The only way is to try with each value k = 1,2,3,4,5,… to see how the clustering results change. Some studies show that the k change will be effective but will stop at a certain number. So you can try to see if your data is good with some k value.

* + 1. **Initialize k initial positions.**

Somehow, try to initialize these k centers of the cluster evenly distributed over the space of the data set. That can be done when you can define the space and nature of the data. But at least, the clusters that you create are not too close together, nor do they overlap.

One last way is that you will run the algorithm many times to get the best results in those runs. Provided that you initialize the mind of k random clusters.

* + 1. **On the problem of stopping (convergence).**

For complex data situations, the k-means algorithm takes a long time or never converges. That is, the fixed cluster center will never be determined to end the problem. Or run through lots of iterations. In such cases, instead of finding k fixed cluster centers, we will stop the problem when a change in a number is acceptable. That is, between two cluster center updates, the position difference between the old and new centers is less than a certain allowed delta number.

1. **Implement K-Mean Algorithm**
   1. **Issue**

There is a set of points in the oxygen coordinate space. Each point will have coordinates (x, y) specified. The problem to be solved is to divide these points into K distinct clusters.

* 1. **Library**
     1. **Numpy**

Numpy is a core Python computer science core library that supports the computation of large, multidimensional arrays with optimized functions applied to those multidimensional arrays. Numpy is especially useful when performing functions related to Linear Algebra.

* Install numpy : **pip install numpy**
  + 1. **Matplotlib**

In order to make the necessary statistical inference, it is necessary to visualize your data, and Matplotlib is one such solution for Python users. It is a very powerful graphing library useful for people working with Python and NumPy. Matplotib's most used module is Pyplot which provides an interface like MATLAB but instead, it uses Python and its open source.

General concept : A matplotlib figure can be categorized into the following sections.

* **Figure:** As a window containing everything you will draw on it.
* **Axes:** The main components of a figure are axes (the smaller frames to draw the picture on). A figure can contain one or more axes. In other words, the figure is just the container, it is the axes that are really where the drawings are drawn.
* **Axis:** They are numerical lines like objects and are responsible for creating the chart limits.
* **Artist:** Everything that you can see on the figure is an artist like Text objects, Line2D objects, collection objects. Most Artists are attached to Axes.
* Install matplotlib : **pip install matplotlib**
  + 1. **Scipy**

SciPy completes NumPy features, in order to provide algorithms for linear algebra, matrix space, signal processing, and image processing, optimization, Fourier transform, ...

* Install scipy : **pip install scipy**

1. **Code**
   1. **Overview**

|  |
| --- |
| 1. import numpy as np # thư viện tính toán toán học 2. import matplotlib.pyplot as plt # visualize data sử dụng đồ thị 3. from scipy.spatial.distance import cdist # Hỗ trợ tính khoảng cách 4. means = [[2, 2], [9, 2], [4, 9]] 5. cov = [[2, 0], [0, 2]] 6. n\_samples = 500 7. n\_cluster = 3 8. X0 = np.random.multivariate\_normal(means[0], cov, n\_samples) 9. X1 = np.random.multivariate\_normal(means[1], cov, n\_samples) 10. X2 = np.random.multivariate\_normal(means[2], cov, n\_samples) 11. X = np.concatenate((X0, X1, X2), axis = 0) 12. #plt.xlabel('x') 13. #plt.ylabel('y') 14. #plt.plot(X[:, 0], X[:, 1], 'bo', markersize=5) 15. #plt.plot() 16. #plt.show() 17. def kmeans\_init\_centers(X, n\_cluster): 18. # random k index beetween 0 and shape(X) without duplicate index. 19. # Then return X[index] as cluster 20. return X[np.random.choice(X.shape[0], n\_cluster, replace=False)] 21. def kmeans\_predict\_labels(X, centers): 22. D = cdist(X, centers) 23. # return index of the closest center 24. return np.argmin(D, axis = 1) 25. def kmeans\_update\_centers(X, labels, n\_cluster): 26. centers = np.zeros((n\_cluster, X.shape[1])) 27. for k in range(n\_cluster): 28. # collect all points assigned to the k-th cluster 29. Xk = X[labels == k, :] 30. # take average 31. centers[k,:] = np.mean(Xk, axis = 0) 32. return centers 33. def kmeans\_has\_converged(centers, new\_centers): 34. # return True if two sets of centers are the same 35. return (set([tuple(a) for a in centers]) == 36. set([tuple(a) for a in new\_centers])) 37. # Hàm này dùng để vẽ dữ liệu lên đồ thị 38. # Random color chỉ làm việc với k <= 4 39. # Nếu bạn thay đổi k > 4, hãy sửa lại phần random color nhé 40. # Chỉ sử dụng trong bài toán này thôi nhé. 41. def kmeans\_visualize(X, centers, labels, n\_cluster, title): 42. plt.xlabel('x') # label trục x 43. plt.ylabel('y') # label trục y 44. plt.title(title) # title của đồ thị 45. plt\_colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'] # danh sách các màu hỗ trợ 47. for i in range(n\_cluster): 48. data = X[labels == i] # lấy dữ liệu của cụm i 49. plt.plot(data[:, 0], data[:, 1], plt\_colors[i] + '^', markersize = 4, label = 'cluster\_' + str(i)) # Vẽ cụm i lên đồ thị 50. plt.plot(centers[i][0], centers[i][1], plt\_colors[i+4] + 'o', markersize = 10, label = 'center\_' + str(i)) # Vẽ tâm cụm i lên đồ thị 51. plt.legend() # Hiện bảng chú thích 52. plt.show() 54. def kmeans(init\_centes, init\_labels, X, n\_cluster): 55. centers = init\_centes 56. labels = init\_labels 57. times = 0 58. while True: 59. labels = kmeans\_predict\_labels(X, centers) 60. kmeans\_visualize(X, centers, labels, n\_cluster, 'Assigned label for data at time = ' + str(times + 1)) 61. new\_centers = kmeans\_update\_centers(X, labels, n\_cluster) 62. if kmeans\_has\_converged(centers, new\_centers): 63. break 64. centers = new\_centers 65. kmeans\_visualize(X, centers, labels, n\_cluster, 'Update center possition at time = ' + str(times + 1)) 66. times += 1 67. return (centers, labels, times) 68. init\_centers = kmeans\_init\_centers(X, n\_cluster) 69. print(init\_centers) # In ra tọa độ khởi tạo ban đầu của các tâm cụm 70. init\_labels = np.zeros(X.shape[0]) 71. kmeans\_visualize(X, init\_centers, init\_labels, n\_cluster, 'Init centers in the first run. Assigned all data as cluster 0') 72. centers, labels, times = kmeans(init\_centers, init\_labels, X, n\_cluster) 74. print('Done! Kmeans has converged after', times, 'times') |

1. **Analysis**
   * 1. **Library declaration**

To implement the k-means algorithm we need to add in the .py file the necessary libraries such as:

* **Numpy** : helps us to calculate
* **Matplotlib** : helps us to draw diagrams based on data
* **Scripy** : helps us to draw diagrams based on data

|  |
| --- |
| 1. import numpy as np # thư viện tính toán toán học 2. import matplotlib.pyplot as plt # visualize data sử dụng đồ thị 3. from scipy.spatial.distance import cdist # Hỗ trợ tính khoảng cách |

* + 1. **Initialization**

|  |  |
| --- | --- |
| **Line** | **Content** |
| 1 | Declare points around the coordinates [2, 2], [9, 2], [4, 9] |
| 2 | Covariance matrix of the distribution. It must be symmetric and positive-semidefinite for proper sampling. |
| 3 | Create 500 points around a cluster |
| 4 | Create cluster center number (default 3 ) |
| 5 🡪 7 | For each cluster center, we will initialize 500 data points around it. Here, respectively, X0, X1, X2. |
| 8 | Join a sequence of arrays along an existing axis. |
| 9 🡪 13 | Draw statistical charts |

|  |
| --- |
| 1. means = [[2, 2], [9, 2], [4, 9]] 2. cov = [[2, 0], [0, 2]] 3. n\_samples = 500 4. n\_cluster = 3 5. X0 = np.random.multivariate\_normal(means[0], cov, n\_samples) 6. X1 = np.random.multivariate\_normal(means[1], cov, n\_samples) 7. X2 = np.random.multivariate\_normal(means[2], cov, n\_samples) 8. X = np.concatenate((X0, X1, X2), axis = 0) 9. plt.xlabel('x') 10. plt.ylabel('y') 11. plt.plot(X[:, 0], X[:, 1], 'bo', markersize=5) 12. plt.plot() 13. plt.show() |

1. **numpy.random.multivariate\_normal (mean, cov[, size])**

Draw random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or “center”) and variance (standard deviation, or “width,” squared) of the one-dimensional normal distribution.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **mean** | * 1-D array\_like, of length N * Mean of the N-dimensional distribution. |
| **cov** | * 2-D array\_like, of shape (N, N) * Covariance matrix of the distribution. It must be symmetric and positive-semidefinite for proper sampling. |
| **Size** | * Int or tuple of ints, optional * Given a shape of, for example, (m,n,k), m\*n\*k samples are generated, and packed in an m-by-n-by-k arrangement. Because each sample is N-dimensional, the output shape is (m,n,k,N). If no shape is specified, a single (N-D) sample is returned. |
| **Returns** | **out** | * Ndarray * The drawn samples, of shape size, if that was provided. If not, the shape is (N,). * In other words, each entry out[i,j,...,:] is an N-dimensional value drawn from the distribution. |

1. **numpy.concatenate((a1, a2, ...), axis=0, out=None)**

Join a sequence of arrays along an existing axis.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **a1,a2** | * Sequence of array\_like * The arrays must have the same shape, except in the dimension corresponding to axis (the first, by default). |
| **axis** | * Int, optional * The axis along which the arrays will be joined. If axis is None, arrays are flattened before use. Default is 0. |
| **Out** | * ndarray, optional * If provided, the destination to place the result. The shape must be correct, matching that of what concatenate would have returned if no out argument were specified. |
| **dtype** | * str or dtype * If provided, the destination array will have this dtype. Cannot be provided together with out. |
| **casting** | * {‘no’, ‘equiv’, ‘safe’, ‘same\_kind’, ‘unsafe’}, optional * Controls what kind of data casting may occur. Defaults to ‘same\_kind’. |
| **Returns** | **res** | * Ndarray * The concatenated array. |

Here is the result

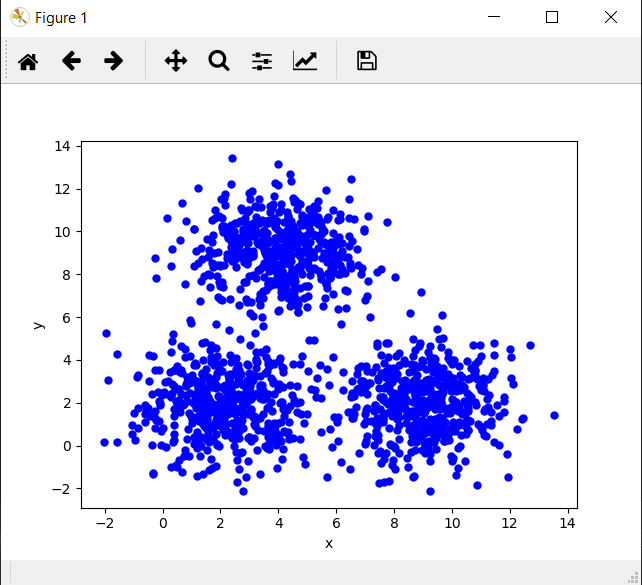


Figure 1 : Create data

* + 1. **Function**
  1. **def kmeans\_init\_centers(X, n\_cluster)**

Thus, if the k-means clustering algorithm works well, it will have to learn 3 cluster centers with coordinates close to the 3 cluster centers (2, 2), (9, 2), and (4,9). And initially, the coordinates of these centers will be taken at random.

Now, let's write the cluster-centered n\_clusters constructor

Here, I instantiate the cluster center n\_clusters by randomly fetching the data set's n\_cluster of the data point.

|  |
| --- |
| def kmeans\_init\_centers(X, n\_cluster):  # random k index beetween 0 and shape(X) without duplicate index.  # Then return X[index] as cluster  return X[np.random.choice(X.shape[0], n\_cluster, replace=False)] |

* + 1. **numpy.random.choice(a, size=None, replace=True, p=None)**

Generates a random sample from a given 1-D array

|  |  |  |
| --- | --- | --- |
| **Parameters** | **a** | * D array-like or int * If an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if a was np.arange(n) |
| **size** | * int or tuple of ints, optional * Output shape. If the given shape is, e.g., (m, n, k), then m \* n \* k samples are drawn. Default is None, in which case a single value is returned. |
| **Replace** | * boolean, optional * Whether the sample is with or without replacement |
| **p** | * 1-D array-like, optional * The probabilities associated with each entry in a. If not given the sample assumes a uniform distribution over all entries in a. |
| **Return** | **samples** | * 1-D ndarray, shape (size,) * The generated random samples |
| **Raises** | **Value error** | * If a is an int and less than zero, if a or p are not 1-dimensional, if a is an array-like of size 0, if p is not a vector of probabilities, if a and p have different lengths, or if replace=False and the sample size is greater than the population size |

Here is the result : Initial coordinates of k

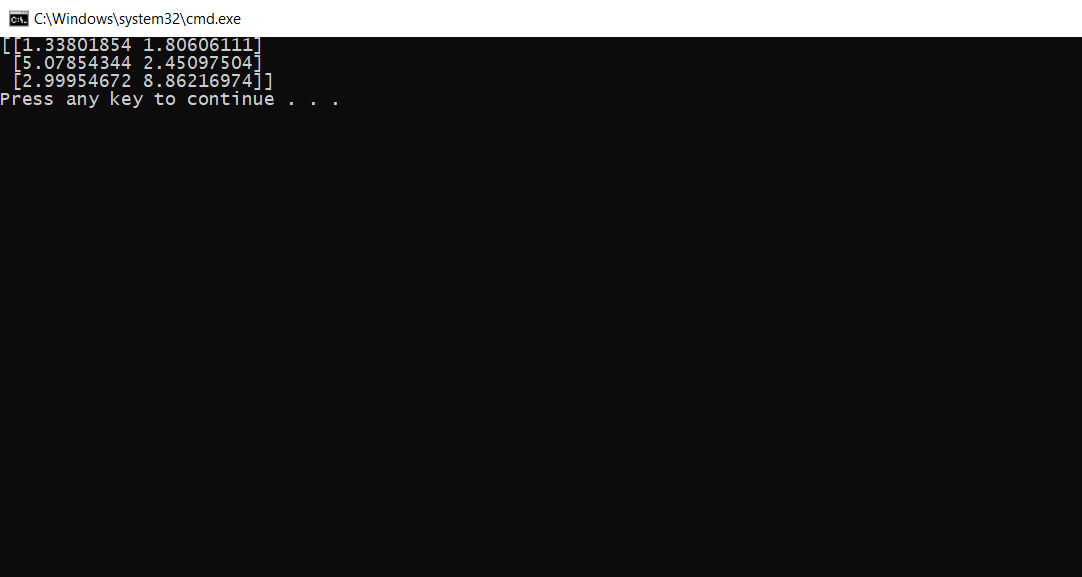


Figure 2 : Initial coordinates of k

* 1. **def kmeans\_predict\_labels(X, centers)**

For each data point in the data set, its cluster center will be 1 of the nearest cluster center n\_cluster. Calculating the distance between 2 points in this article uses Euclidean distance.

|  |
| --- |
| def kmeans\_predict\_labels(X, centers):  D = cdist(X, centers)  # trả về vị trí tâm gần nhất  return np.argmin(D, axis = 1) |

* + 1. **cdist(XA,XB)**

Compute distance between each pair of the two collections of inputs.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **XA** | * ndarray * An mA by n array of mA original observations in an n-dimensional space. Inputs are converted to float type. |
| **XB** | * ndarray * An mA by n array of mA original observations in an n-dimensional space. Inputs are converted to float type. |
| **Returns** | **Y** | * ndarray * A mA by mB distance matrix is returned. For each i and j, the metric dist(u=XA[i], v=XB[j]) is computed and stored in the ij th entry. |
| **Raises** | **ValueError** | * An exception is thrown if XA and XB do not have the same number of columns. |

* + 1. **numpy.argmin(a, axis=None, out=None)**

Returns the indices of the minimum values along an axis.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **a** | * array\_like * Input array. |
| **axis** | * int, optional * By default, the index is into the flattened array, otherwise along the specified axis. |
| **out** | * array, optional * If provided, the result will be inserted into this array. It should be of the appropriate shape and dtype. |
| **Returns** | **index\_array** | * ndarray of ints * Array of indices into the array. It has the same shape as a.shape with the dimension along axis removed. |

* 1. **def kmeans\_update\_centers(X, labels, n\_cluster)**

Recalculating the coordinates of each cluster center is done simply by taking the average of the coordinates of all data points of the cluster. After the calculation is complete, the new position of the cluster center will be in the center of its cluster.

|  |
| --- |
| def kmeans\_update\_centers(X, labels, n\_cluster):  centers = np.zeros((n\_cluster, X.shape[1]))  for k in range(n\_cluster):  # collect all points assigned to the k-th cluster  Xk = X[labels == k, :]  # take average  centers[k,:] = np.mean(Xk, axis = 0)  return centers |

* + 1. **numpy.zeros(shape, dtype=float, order='C')**

Return a new array of given shape and type, filled with zeros.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **shape** | * int or tuple of ints * Shape of the new array, e.g., (2, 3) or 2. |
| **dtype** | * data-type, optional * The desired data-type for the array, e.g., numpy.int8. Default is numpy.float64. |
| **order** | * {‘C’, ‘F’}, optional, default: ‘C’ * Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory. |
| **Returns** | **out** | * outndarray * Array of zeros with the given shape, dtype, and order. |

* + 1. **numpy.mean(a,axis=None,dtype=None,out=None,keepdims=<no value>)**

Compute the arithmetic mean along the specified axis.

Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **a** | array\_like  Array containing numbers whose mean is desired. If a is not an array, a conversion is attempted. |
| **axis** | None or int or tuple of ints, optional  Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. |
| **dtype** | data-type, optional  Type to use in computing the mean. For integer inputs, the default is float64; for floating point inputs, it is the same as the input dtype. |
| **out** | ndarray, optional  Alternate output array in which to place the result. The default is None; if provided, it must have the same shape as the expected output, but the type will be cast if necessary. See ufuncs-output-type for more details. |
| **keepdims** | bool, optional  If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.  If the default value is passed, then keepdims will not be passed through to the mean method of sub-classes of ndarray, however any non-default value will be. If the sub-class’ method does not implement keepdims any exceptions will be raised. |
| **Returns** | **m** | ndarray, see dtype parameter above  If out=None, returns a new array containing the mean values, otherwise a reference to the output array is returned. |

* 1. **def kmeans\_has\_converged(centers, new\_centers)**

If updating the position of the cluster centers did not change anything, it means we can stop and give results. That means the old coordinates and the updated coordinates of the cluster centers are the same. In this case, if you keep running, no change will be made.

|  |
| --- |
| def kmeans\_has\_converged(centers, new\_centers):  # return True if two sets of centers are the same  return (set([tuple(a) for a in centers]) ==  set([tuple(a) for a in new\_centers])) |

* 1. **def kmeans\_visualize(X, centers, labels, n\_cluster, title)**

After each center update for each data point, or update the coordinates of the centers again. Call this function to see how the mind and center of the data points change. The results will be plotted on a graph for easy viewing.

|  |
| --- |
| def kmeans\_visualize(X, centers, labels, n\_cluster, title):  plt.xlabel('x') # label trục x  plt.ylabel('y') # label trục y  plt.title(title) # title của đồ thị  plt\_colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'] # danh sách các màu hỗ trợ    for i in range(n\_cluster):  data = X[labels == i] # lấy dữ liệu của cụm i  plt.plot(data[:, 0], data[:, 1], plt\_colors[i] + '^', markersize = 4, label = 'cluster\_' + str(i)) # Vẽ cụm i lên đồ thị  plt.plot(centers[i][0], centers[i][1], plt\_colors[i+4] + 'o', markersize = 10, label = 'center\_' + str(i)) # Vẽ tâm cụm i lên đồ thị  plt.legend() # Hiện bảng chú thích  plt.show() |

* + 1. **Summary**

|  |  |
| --- | --- |
| **Function** | **Content** |
| kmeans\_init\_centers(X, n\_cluster) | Initialize the cluster center n\_clusters  🡪 Return : The array contains random coordinates of n clusters |
| kmeans\_predict\_labels(X, centers) | Identify cluster centers  🡪 Return : The data closest to the cluster center |
| kmeans\_update\_centers(X, labels, n\_cluster) | Update the location of the cluster centers  🡪 Return : New coordinates of cluster centers |
| kmeans\_has\_converged(centers, new\_centers) | Check convergence  🡪 Return : True if two sets of centers are the same |
| kmeans\_visualize(X, centers, labels, n\_cluster, title) | Draw a chart |

* + 1. **Main**

|  |
| --- |
| 1. def kmeans(init\_centes, init\_labels, X, n\_cluster): 2. centers = init\_centes 3. labels = init\_labels 4. times = 0 5. while True: 6. labels = kmeans\_predict\_labels(X, centers) 7. kmeans\_visualize(X, centers, labels, n\_cluster, 'Assigned label for data at time = ' + str(times + 1)) 8. new\_centers = kmeans\_update\_centers(X, labels, n\_cluster) 9. if kmeans\_has\_converged(centers, new\_centers): 10. break 11. centers = new\_centers 12. kmeans\_visualize(X, centers, labels, n\_cluster, 'Update center possition at time = ' + str(times + 1)) 13. times += 1 14. return (centers, labels, times) 15. init\_centers = kmeans\_init\_centers(X, n\_cluster) 16. print(init\_centers) # In ra tọa độ khởi tạo ban đầu của các tâm cụm 17. init\_labels = np.zeros(X.shape[0]) 18. kmeans\_visualize(X, init\_centers, init\_labels, n\_cluster, 'Init centers in the first run. Assigned all data as cluster 0') 19. centers, labels, times = kmeans(init\_centers, init\_labels, X, n\_cluster) 21. print('Done! Kmeans has converged after', times, 'times') |

* 1. **Code**

Step 1 : Line 16 🡪 19

Content : Print out the initial coordinates of the cluster centers.

Result :

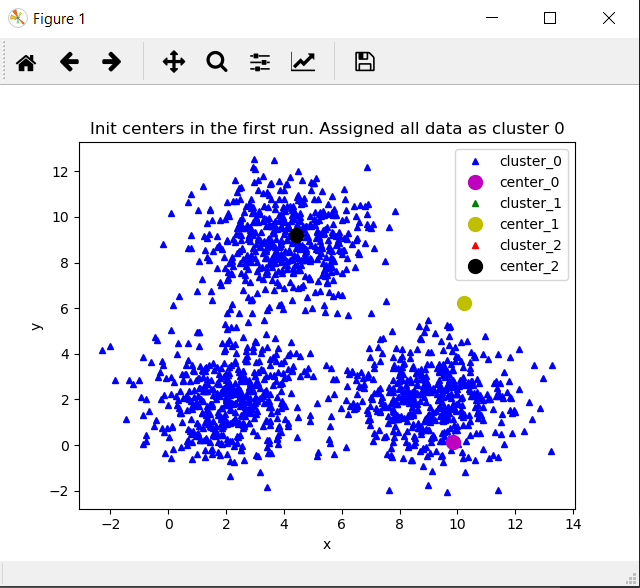


Figure 3 : Init centers in the first run

Step 2 : Line 20

Call k-mean function

|  |
| --- |
| 1. def kmeans(init\_centes, init\_labels, X, n\_cluster): 2. centers = init\_centes 3. labels = init\_labels 4. times = 0 5. while True: 6. labels = kmeans\_predict\_labels(X, centers) 7. kmeans\_visualize(X, centers, labels, n\_cluster, 'Assigned label for data at time = ' + str(times + 1)) 8. new\_centers = kmeans\_update\_centers(X, labels, n\_cluster) 9. if kmeans\_has\_converged(centers, new\_centers): 10. break 11. centers = new\_centers 12. kmeans\_visualize(X, centers, labels, n\_cluster, 'Update center possition at time = ' + str(times + 1)) 13. times += 1 14. return (centers, labels, times) |

|  |  |
| --- | --- |
| **Line** | **Content** |
| **5** | Start loop |
| **6** | Select points near corresponding cluster centers and store them in a one-dimensional array |
| **7** | Plot the graph according to the data that has been clustered |
| **8** | Create the coordinates of the new cluster center by dividing the average of the data coordinates so that the center can be in the middle |
| **9** | Check that the new cluster center coordinates are the same as the old ones. Exit the loop when 2 centers have overlapping. |
| **11 🡪13** | If there is no match, the function will call the redrawing function to update the new cluster centers |
| **End** | Continue until exiting the loop |

* 1. **Runtime**

Because it is random, all runs are not the same

Step 1 : First run – cluster center positions are randomly generated.

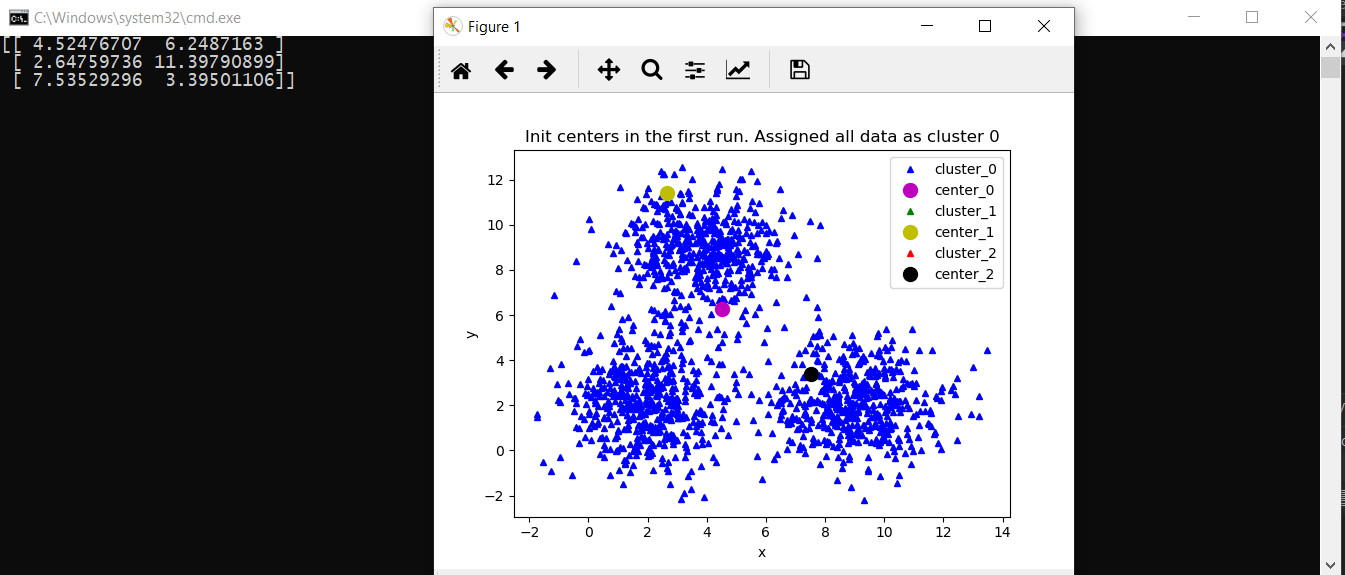


Figure 4 : Data and cluster in the first time

Step 2 : Classify data in clusters based on the distance from the points to the cluster center

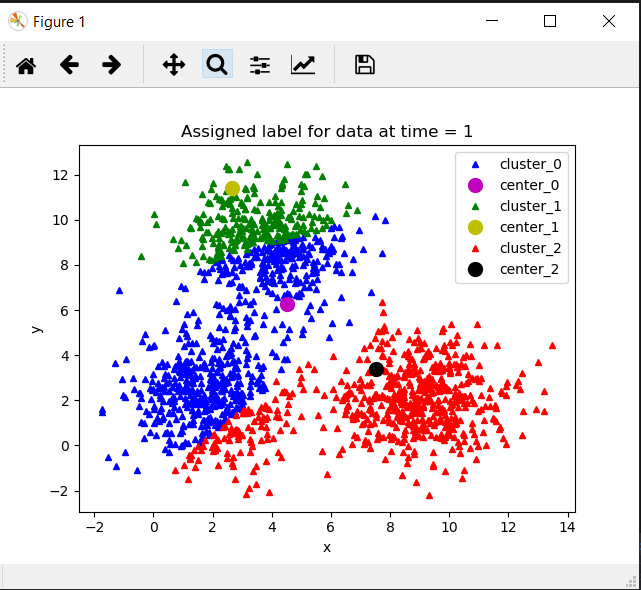


Figure 5 : Classify data in clusters

Step 3 : Update the cluster centers by dividing the average of the coordinates of the points

At the same time reclassify points near the center of the new cluster

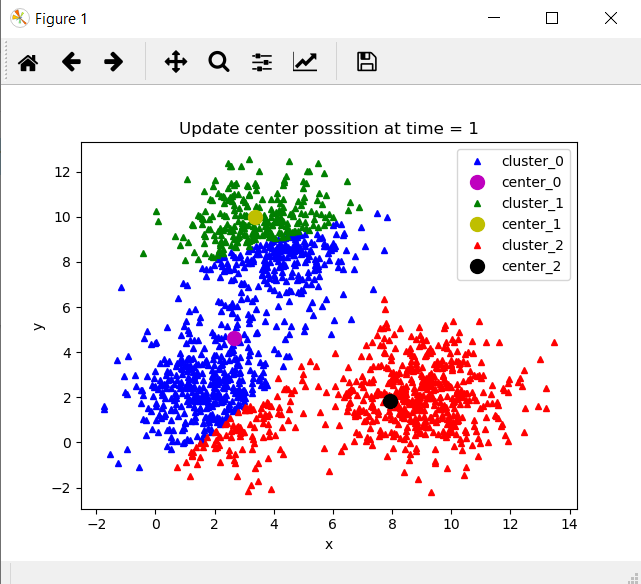


Figure 6 : Update the new location of the clusters

Step 4 : Repeat step 3 until exiting the loop

At this point, the points were divided among the cluster centers and successfully clustered

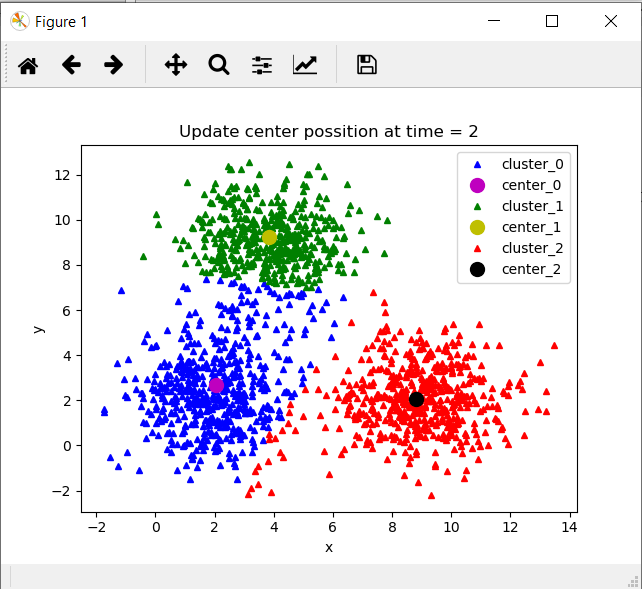


Figure 7 : Update the new location of the clusters at time = 2

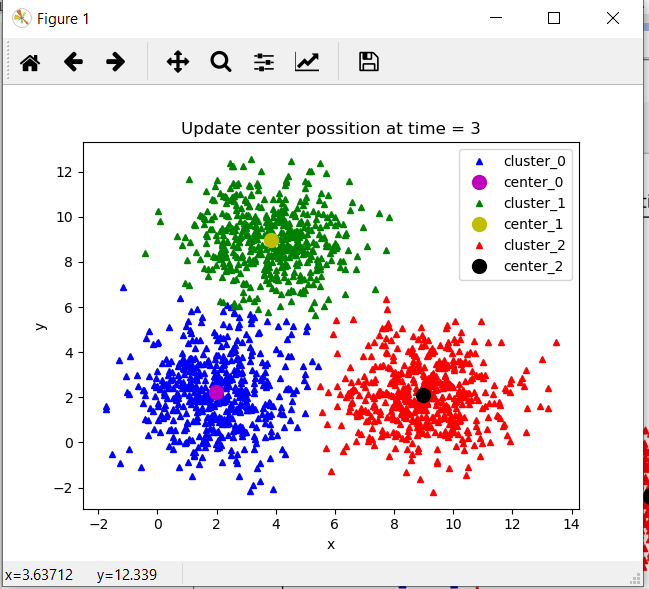


Figure 8 : Update the new location of the clusters at time = 3

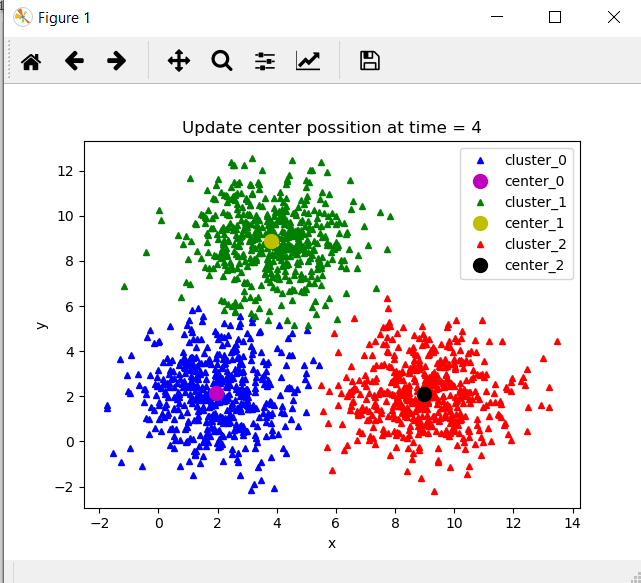


Figure 9 : Update the new location of the clusters at time = 4

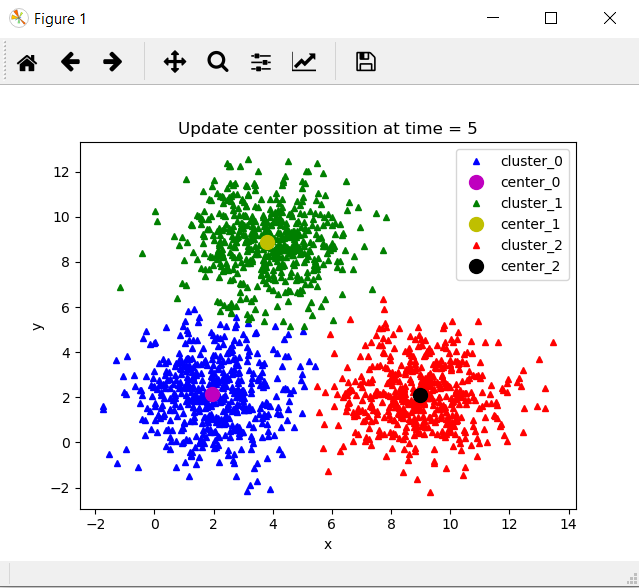


Figure 10 : Update the new location of the clusters at time = 5

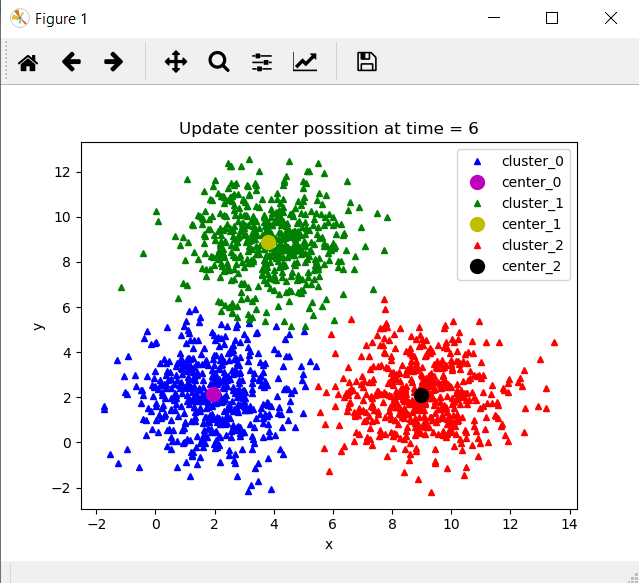


Figure 11 : Update the new location of the clusters at tme = 6

1. **Tài liệu kham thảo**

**Link web kham thảo :**

https://nguyenvanhieu.vn/thuat-toan-phan-cum-k-means/#gioi-thieu-ve-kmeans

https://machinelearningcoban.com/2017/01/01/kmeans/

<https://ml-for-vietnameses.readthedocs.io/vi/latest/k-means-clustering.html>

**Link github :**

<https://github.com/hndc9x/Project3>

**Link video kham thảo :**

<https://www.youtube.com/watch?v=_aWzGGNrcic>

<https://www.youtube.com/watch?v=4b5d3muPQmA>

Link video demo :

<https://www.youtube.com/watch?v=1XqG0kaJVHY&t=1158>s