



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 2,2020

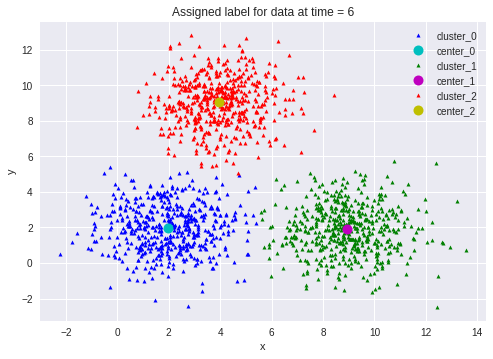
1. **Introduction** 
   1. **Introduction K-Means**

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. **Sdsad**
  2. **kjkj**

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. **as**
  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() |

**Function**

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. |
| **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. **sas**
2. **sas**
3. **sa**
4. **sa**
5. **sa**
6. **sa**
7. **sa**
8. **sa**
9. **sa**
10. **sa**
    * 1. **Dsad**
      2. **Sdsa**
      3. **Dsa**
      4. **Dsa**
      5. **Dsa**
      6. **D**
      7. **Sa**
      8. **D**
      9. **Asd**
      10. **asd**
    1. **sas**
    2. **sas**
    3. **sa**
    4. **sas**
    5. **as**
    6. **sas**
11. **Sa**
12. **sas**