ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

Môn học: Học máy và Ứng dụng

Đề tài:

Giải thuật K-means với các kỹ thuật khởi tạo k trung tâm ban đầu và Ứng dụng trong bài toán phân cụm

GVHD: Assoc. Prof. Dr. Dương Tuấn Anh

Nhóm lớp: SDH-12
SVTH: Nguyễn Hoàng Phúc
1927030

TP. HỒ CHÍ MINH, THÁNG 6/2021

Mục lục

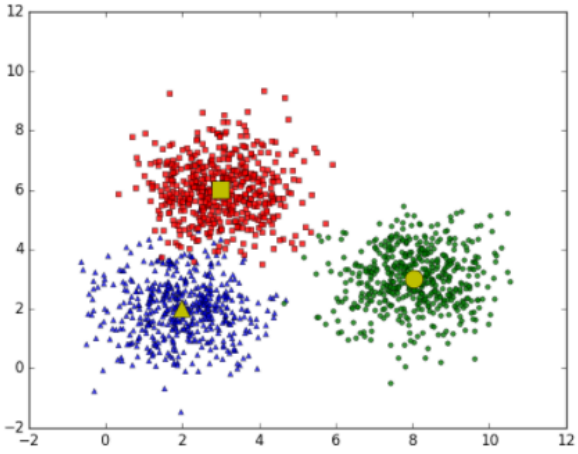
1 Tìm hiểu về giải thuật K-means

1.1 Ý tưởng giải thuật K-means

Thuật toán K-means clustering (phân cụm K-means) là một trong những thuật toán cơ bản nhất trong Unsupervised learning. Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thế nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau.

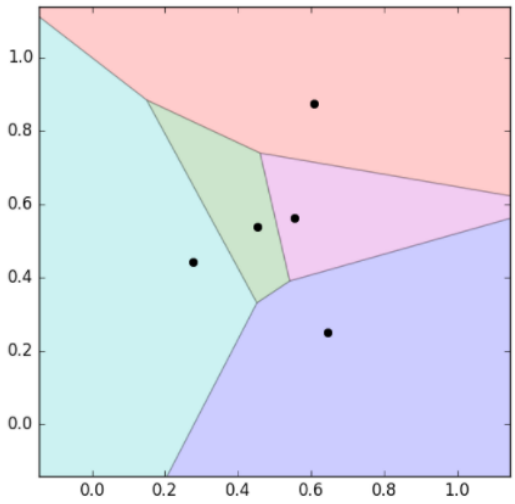
Ví dụ: Một công ty muốn tạo ra những chính sách ưu đãi cho những nhóm khách hàng khác nhau dựa trên sự tương tác giữa mỗi khách hàng với công ty đó (số năm là khách hàng; số tiền khách hàng đã chi trả cho công ty; độ tuổi; giới tính; thành phố; nghề nghiệp; . . .).
Giả sử công ty đó có rất nhiều dữ liệu của rất nhiều khách hàng nhưng chưa có cách nào chia toàn bộ khách hàng đó thành một số nhóm/cụm khác nhau. Nếu một người biết Machine Learning được đặt câu hỏi này, phương pháp đầu tiên nghĩ đến sẽ là K-means Clustering. Sau khi đã phân ra được từng nhóm, nhân viên công ty đó có thể lựa chọn ra một vài khách hàng trong mỗi nhóm để quyết định xem mỗi nhóm tương ứng với nhóm khách hàng nào. Phần việc cuối cùng này cần sự can thiệp của con người, nhưng lượng công việc đã được rút gọn đi rất nhiều.

Ý tưởng đơn giản nhất về cluster (cụm) là tập hợp các điểm ở gần nhau trong một không gian nào đó (không gian này có thể có rất nhiều chiều trong trường hợp thông tin về một điểm dữ liệu là rất lớn). Hình bên dưới là một ví dụ về 3 cụm dữ liệu).



Hình 1: Bài toán với 3 clusters

Giả sử mỗi cluster có một điểm đại diện (center) màu vàng. Và những điểm xung quanh mỗi center thuộc vào cùng nhóm với center đó. Một cách đơn giản nhất, xét một điểm bất kỳ, ta xét xem điểm đó gần với center nào nhất thì nó thuộc về cùng nhóm với center đó. Tới đây có một bài toán thú vị: Trên một vùng biển hình vuông lớn có ba đảo hình vuông, tam giác, và tròn màu vàng như hình trên. Một điểm trên biển được gọi là thuộc lãnh hải của một đảo nếu nó nằm gần đảo này hơn so với hai đảo kia . Hãy xác định ranh giới lãnh hải của các đảo.
Hình dưới đây là một hình minh họa cho việc phân chia lãnh hải nếu có 5 đảo khác nhau được biểu diễn bằng các hình tròn màu đen.



Hình 2: Phân vùng lãnh hải của mỗi đảo

Chúng ta thấy rằng đường phân định giữa các lãnh hải là các đường thẳng (chính xác hơn thì chúng là các đường trung trực của các cặp điểm gần nhau). Vì vậy, lãnh hải của một đảo sẽ là một hình đa giác. Cách phân chia này trong toán học được gọi là Voronoi Diagram.
Trong không gian ba chiều, lấy ví dụ là các hành tinh, thì lãnh không của mỗi hành tinh sẽ là một đa diện. Trong không gian nhiều chiều hơn, sẽ có siêu đa diện (hyperpolygon).

Mặt hạn chế của thuật toán K-mean tiêu chuẩn:
Một nhược điểm của thuật toán K-mean là nó nhạy cảm với việc khởi tạo k trung tâm hoặc các điểm trung bình ban đầu. Vì vậy, nếu một centroid được khởi tạo là một điểm "xa", nó có thể chỉ kết thúc không có điểm nào được liên kết với nó và đồng thời, nhiều hơn một cụm có thể kết thúc với một centroid duy nhất. Tương tự, nhiều hơn một trung tâm có thể được khởi tạo vào cùng một cụm dẫn đến phân cụm kém. Ví dụ, hãy xem xét các hình ảnh hiển thị bên dưới.

Hình 3: Khởi tạo trung tâm kém dẫn đến phân cụm kém

Để hạn chế nhược điểm trên của giải thuật K-means các nhà khoa học đã đưa ra rất nhiều kỹ thuật khởi tạo k trung tâm cụm ban đầu. Trong phạm vi bài tiểu luận, tác giả trình bày 5 kỹ thuật phổ biến cũng như ứng dụng từng kỹ thuật vào 2 bài toán cụ thể để so sánh hiệu quả của từng phương pháp.

1.2 Tìm hiểu ý tưởng các kỹ thuật khởi tạo k trung tâm ban đầu

1.2.1 Phương pháp Ngẫu nhiên

Chia tập dữ liệu thành k cụm một cách ngẫu nhiên, sau đó tìm tâm của từng cụm để khởi tạo k cụm trung tâm ban đầu. Phương pháp này được sử dụng phổ biến nhất trong việc khởi tạo k cụm ban đầu của giải thuật K-means.

```
# RANDOM K CLUSTERS
def kmeans_init_randomly_k_clusters(data, k):
    start = time.time()
    print('RANDOM K CLUSTERS')
    centroids = []
    # randomly separate rows of data as k clusters
    size_data = len(data)
    while True:
      clusters_random = np.random.randint(k, size = size_data)
      if len(set(clusters_random)) == k:
          break
    for i in range(k):
      cluster = data[clusters_random == i]
      centroids.append(np.mean(cluster, axis = 0))
    print(round(time.time() - start, 5))
    return np.array(centroids)
```

1.2.2 Phương pháp K-MEANS++

Kỹ thuật này đảm bảo khởi tạo centroid thông minh hơn và cải thiện chất lượng của phân cụm.

Thuật toán khởi tạo:

- **Bước 1:** Chọn ngẫu nhiên centroid đầu tiên từ các điểm dữ liệu.
- **Bước 2:** Đối với mỗi điểm dữ liệu, tính toán khoảng cách của nó từ trung tâm gần nhất, đã chọn trước đó.
- **Bước 3:** Chọn centroid tiếp theo từ các điểm dữ liệu sao cho xác suất chọn một điểm làm centroid tỷ lệ thuận với khoảng cách của nó từ centroid gần nhất, đã chọn trước đó. (tức là điểm có khoảng cách tối đa từ tâm gần nhất có nhiều khả năng được chọn tiếp theo làm tâm).
- **Bước 4:** Lặp lại các bước 2 và 3 cho đến khi k centroid được lấy mẫu.

Bằng cách thực hiện theo quy trình khởi tạo ở trên, chúng ta nhận ra các centroid cách xa nhau. Điều này làm tăng cơ hội ban đầu nhặt được các trung tâm nằm trong các cụm khác nhau. Ngoài ra, vì các centroid được chọn từ các điểm dữ liệu, mỗi centroid có một số điểm dữ liệu được liên kết với nó ở cuối.

```
# K-MEANS++
def kmeans_init_kmeansPlusPlus(data, k):
    start = time.time()
    print('K-MEANS++')
    # initialize the centroids list and add a randomly selected
   data point to the list
    centroids = []
    centroids.append(data[np.random.randint(data.shape[0]), :])
    if k < 4:
        plot(data, np.array(centroids))
    # compute remaining k - 1 centroids
    for c_id in range(k - 1):
        dist = []
        for centroid in centroids:
          dist.append(list(map(lambda x: math.sqrt(sum(x)), (data
   - centroid)**2)))
        dist = np.array(dist).T
        # index of the closest center
        dist = np.array(list(map(lambda x: min(x), dist)))
        # select data point with maximum distance as our next
   centroid
        next_centroid = data[np.argmax(dist)]
        centroids.append(next_centroid)
```

```
        if k < 4:
            plot(data, np.array(centroids))
    print(round(time.time() - start, 5))
    return np.array(centroids)
```

1.2.3 Phương pháp FORGY APPROACH (FA)

Được đề xuất bởi Forgy vào năm 1965. Ý tưởng của kỹ thuật này là chọn k điểm một cách ngẫu nhiên trong tập dữ liệu làm k trung tâm cụm ban đầu.Vì là chọn k điểm ngẫu nhiên nên phương pháp này dễ sinh ra rủi ro nhất (chọn các điểm biên và không có điểm nào trong tập dữ liệu được gán vào hoặc 2 tâm cụm được chọn trong cùng 1 cụm thực), nhưng lại là phương pháp khởi tạo nhanh nhất, ít chi phí tính toán nhất.

```
# FORGY APPROACH (FA)
def kmeans_init_randomly_centers(data, k):
    start = time.time()
    print('FORGY APPROACH (FA)')
    # randomly pick k rows of data as initial centers
    print(round(time.time() - start, 5))
    return data[np.random.choice(data.shape[0], k, replace=False)]
```

1.2.4 Phương pháp MACQUEEN APPROACH (MA)

Được đề xuất bởi MacQueen vào năm 1967. Ý tưởng của kỹ thuật này là chọn k điểm một cách ngẫu nhiên trong tập dữ liệu làm k trung tâm cụm ban đầu. Theo thứ tự các điểm dữ liệu trong tập dữ liệu, ta gán lần lượt vào cụm có khoảng cách từ điểm đó đến tâm của cụm gần nhất. Sau mỗi lần gán một điểm dữ liệu, ta tính lại tâm của cụm vừa được gán điểm mới. Thực hiện lần lượt đến khi tất cả các điểm dữ liệu được gán vào k cụm với k trung tâm.

```
# MACQUEEN APPROACH (MA)
def kmeans_init_MA(data, k):
    start = time.time()
    print('MACQUEEN APPROACH (MA)')
    # randomly pick k rows of data as initial centers
    choice_index = np.random.choice(data.shape[0], k, replace=
    False)
    centroids = np.array(data[choice_index], dtype='f')
    data = np.delete(data,choice_index,axis=0)
    # save current mean (centroids) of each cluster and num of
    member of k clusters
    meta_centroids = [1] * k
    for item in data:
```

```
12          d = np.array(list(map(lambda x: math.sqrt(sum(x)), (
    centroids - item)**2)))
13          # index of the closest center
14          index_centroid = np.argmin(d)
15          # update centroids and prepare for next interator
16          centroids[index_centroid] = (centroids[index_centroid] *
    meta_centroids[index_centroid] + item) / (meta_centroids[
    index_centroid] + 1)
17          meta_centroids[index_centroid] += 1
18      print(round(time.time() - start, 5))
19      return centroids
```

1.2.5 Phương pháp KAUFMAN APPROACH (KA)

Được đề xuất bởi Kaufman và Rousseeuw vào năm 1990. Trong kỹ thuật này, k trung tâm ban đầu có được bằng cách chọn từng điểm cho đến khi tìm thấy đủ k trung tâm. Trung tâm đầu tiên là medoid của bộ dữ liệu. K - 1 trung tâm còn lại được chọn theo quy tắc heuristic rằng điểm được chọn hứa hẹn có giá trị lượng giá cao hơn so với phần còn lại.

Thuật toán khởi tạo:

- **Bước 1:** Chọn trung tâm đầu tiên là medoid của tập dữ liệu.
- **Bước 2:** Với mỗi điểm ω_i chưa được chọn:
  - **Bước 2.1:** Với mỗi điểm ω_j chưa được chọn:
    Tính $C_{ij} = max(D_j - d_{ij}, 0)$, với $d_{ij} = \|\omega_i - \omega_j\|$ và $D_{ij} = min_s d_{sj}$, với s là một trung tâm cụm đã tìm được.
  - **Bước 2.2:** Tính giá trị lượng giá hứa hẹn của ω_i bằng $\Sigma_j C_{ij}$.
- **Bước 3:** Chọn ω_i chưa được chọn là trung tâm cụm và có giá trị $\Sigma_j C_{ij}$ lớn nhất.
- **Bước 4:** Nếu đã tìm đủ k trung tâm cụm thì ngừng giải thuật. Nếu chưa thì quay lại bước 2.
- **Bước 5:** Gán phần còn lại của bộ dữ liệu vào cụm gần nhất với k trung tâm cụm đã tìm được.

Giải thuật này có chi phí tính toán khá lớn. Với bộ dữ liệu lớn giải thuật có thể không chạy đến khi đạt được kết quả.

```python
# KAUFMAN APPROACH (KA)
def kmeans_init_KA(data, k):
    start = time.time()
    print('KAUFMAN APPROACH (KA)')
    dist = []
    for item in data:
        dist.append(sum(list(map(lambda x: math.sqrt(sum(x)), (data - item)**2))))
    medoid =  data[np.argmin(dist)]
    selected_seed = np.array([medoid])
    while len(selected_seed) < k:
        Cji = []
        for wi in data:
            sub_Cji = 0
            for wj in data:
                Dj = min(list(map(lambda x: math.sqrt(sum(x)), (selected_seed - wj)**2)))
                sub_Cji += max(Dj - np.linalg.norm(wi-wj), 0)
            Cji.append(sub_Cji)
        while True:
            index = np.argmax(np.array(Cji))
            prepare_selected_seed = data[index]
            if list(prepare_selected_seed) not in selected_seed.tolist():
                break
            Cji[index] = 0
        selected_seed = np.append(selected_seed, [prepare_selected_seed], axis=0)
    print(round(time.time() - start, 5))
    return selected_seed
```

2 Ứng dụng giải thuật K-means với các kỹ thuật khởi tạo k trung tâm ban đầu trong việc phân cụm tập dữ liệu IRIS FLOWER

2.1 Mô tả tập dữ liệu

Iris flower dataset là một bộ dữ liệu nhỏ đã được gắn nhãn. Bộ dữ liệu này bao gồm thông tin của ba loại hoa Iris (một loài hoa lan) khác nhau: Iris setosa, Iris virginica và Iris versicolor. Mỗi loại có 50 bông hoa được đo với dữ liệu là 4 thông tin: chiều dài, chiều rộng đài hoa (sepal), và chiều dài, chiều rộng cánh hoa (petal). Dưới đây là ví dụ về hình ảnh của ba loại hoa, mỗi điểm dữ liệu trong tập này chỉ là một vector 4 chiều.

Hình 4: Ví dụ về Iris flower dataset

Bài toán phân nhóm giả định:

Bài toán: Giả sử rằng chúng ta không biết nhãn của các dữ liệu hoa này, chúng ta muốn phân nhóm các hoa gần giống nhau về một nhóm. Với thuật toán phân nhóm K-means clustering, chúng ta sẽ giải quyết bài toán này thế nào? Đồng thời sẽ sử dụng Error Sum of Squares (Sai số SSE) để đánh giá hiệu quả phân cụm của bài toán.

Mô tả chung về dữ liệu Iris flower

```
#Code
from sklearn.datasets import load_iris

iris = load_iris()
iris_X = iris.data
iris_y = iris.target
```

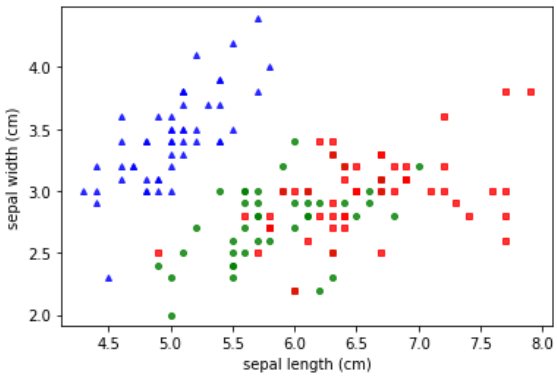
```
print('Number of classes: ' , len(np.unique(iris_y)))
print('Number of data points: ' , len(iris_y))

X0 = iris_X[iris_y == 0,:]
print('\nSamples from class 0:\n', X0[:5,:])

X1 = iris_X[iris_y == 1,:]
print('\nSamples from class 1:\n', X1[:5,:])

X2 = iris_X[iris_y == 2,:]
print('\nSamples from class 2:\n', X2[:5,:])

#Out
Number of classes:  3
Number of data points:  150

Samples from class 0:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]

Samples from class 1:
 [[7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]]

Samples from class 2:
 [[6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]]
```

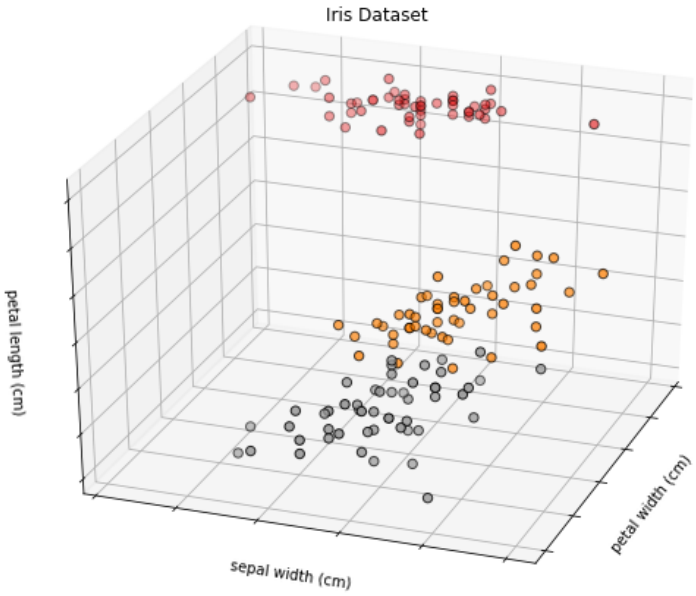
Thể hiện các điểm dữ liệu lên biểu đồ 2D và 3D

Để thể hiện tốt được sự khác biệt giữa các điểm dữ liệu 4 chiều trên biểu đồ 2D và 3D ta chọn các thuộc tính để vẽ như sau:

- Với biểu đồ 2D, ta chọn 2 thuộc tính độ rộng đài hoa và độ dài đài hoa.
- Với biểu đồ 3D, ta chọn 3 thuộc tính độ rộng cánh hoa, độ dài cánh hoa và độ rộng đài hoa .

Cách biểu diễn này cũng được áp dụng trong phần sau.





Hình 5: Biểu diễn tập dữ liệu Iris flower có nhãn ban đầu

Sai số SSE của tập dữ liệu ban đầu đo được là: 100.39574. Giá trị này giúp so sánh hiệu quả của các phương pháp phân cụm được trình bày ở phần sau.

2.2 Phân loại dữ liệu với các kỹ thuật khởi tạo k trung tâm ban đầu

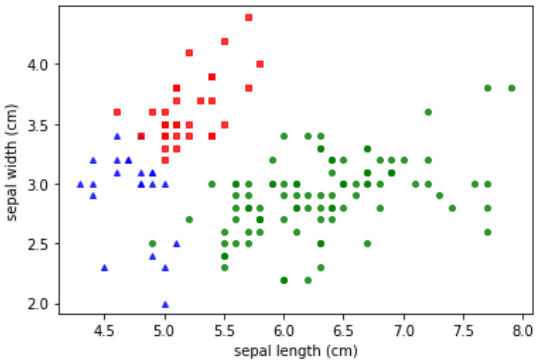
2.2.1 Phương pháp Ngẫu nhiên

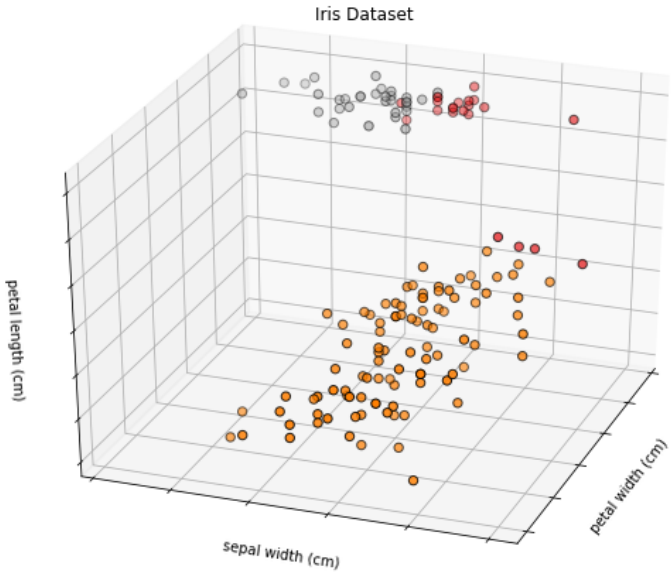
Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
RANDOM K CLUSTERS	Thời gian	0.00085	0.00053	0.00029	0.00542	0.00078	0.00568	0.00039	0.0019	0.00256	0.00337	**0.002177**
	Vòng lặp	10	11	10	11	11	11	12	8	11	10	**10.5**
	SSE	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	124.02237	97.22487	97.22487	97.22487	**99.90462**

Hình 6: Bảng kết quả với phương pháp Ngẫu nhiên

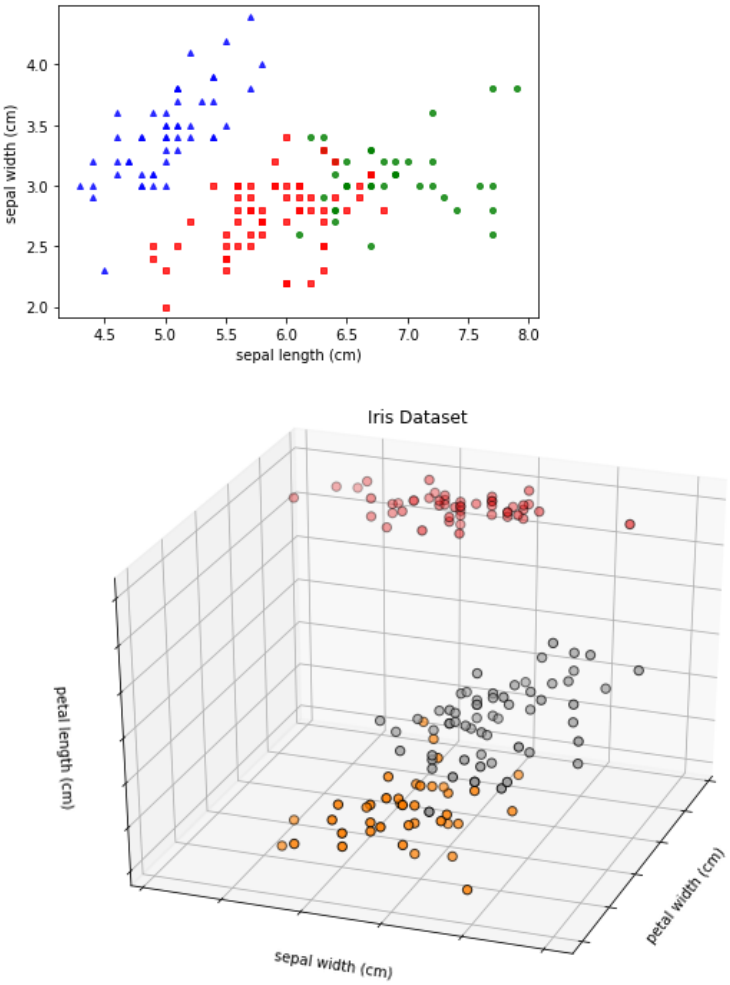
Kết quả của lần chạy thứ 7, do việc khởi tạo 3 cụm trung tâm ban đầu dẫn đến việc phân chia các cụm không tối ưu (Sai số SSE = 124.02237)





Hình 7: Lần chạy thứ 7 bằng phương pháp Ngẫu nhiên

Kết quả của lần chạy thứ 10, nhờ khởi tạo 3 cụm tốt mà công việc phân cụm tốt (Sai số SSE = 97.22487, xấp xỉ sai số SSE của tập dữ liệu có dán nhãn tương ứng 100.39574).



Hình 8: Lần chạy thứ 10 bằng phương pháp Ngẫu nhiên

Nhìn chung thời gian trung bình 10 lần chạy thuật toán khá bé do chi phí tính toán ít, trái lại số vòng lặp trung bình để hội tụ và sai số SSE trung bình khá lớn.

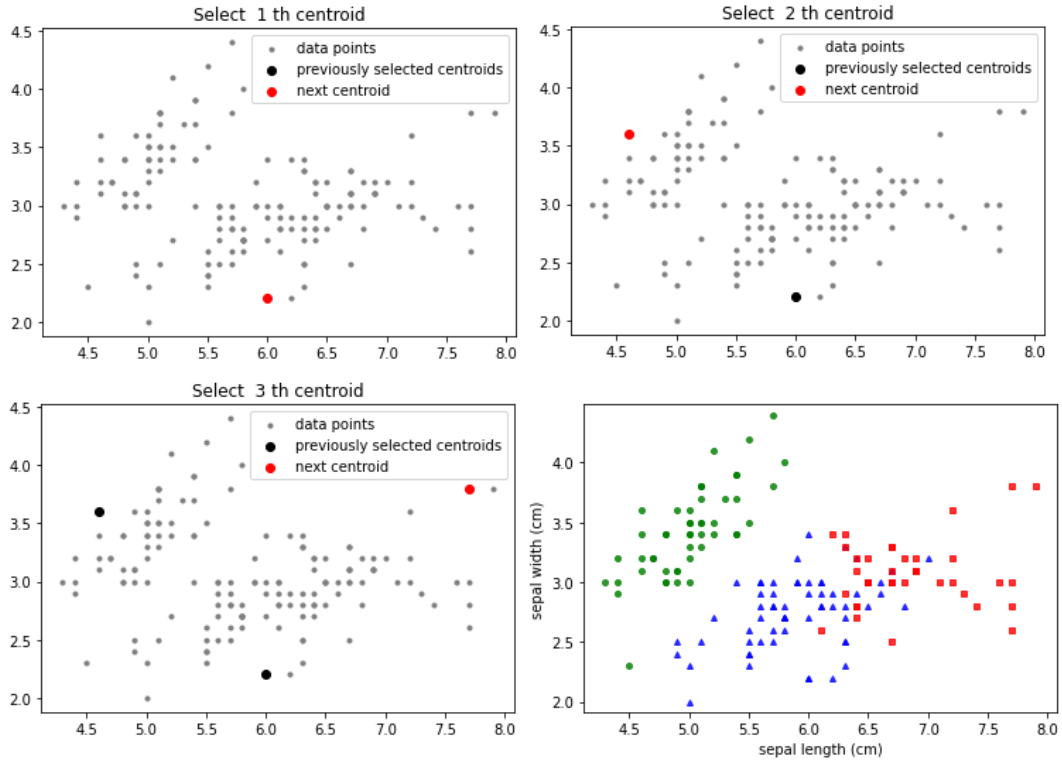
2.2.2 Phương pháp K-MEANS++

Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
K-MEANS++	Thời gian	0.52934	0.63747	0.56003	0.66934	0.54443	0.53559	0.54364	0.53747	0.62381	0.58109	**0.576221**
	Vòng lặp	5	3	5	3	10	3	11	3	10	2	**5.5**
	SSE	97.20457	97.20457	97.20457	97.20457	97.20457	97.20457	97.22487	97.20457	97.22487	97.22487	**97.21066**

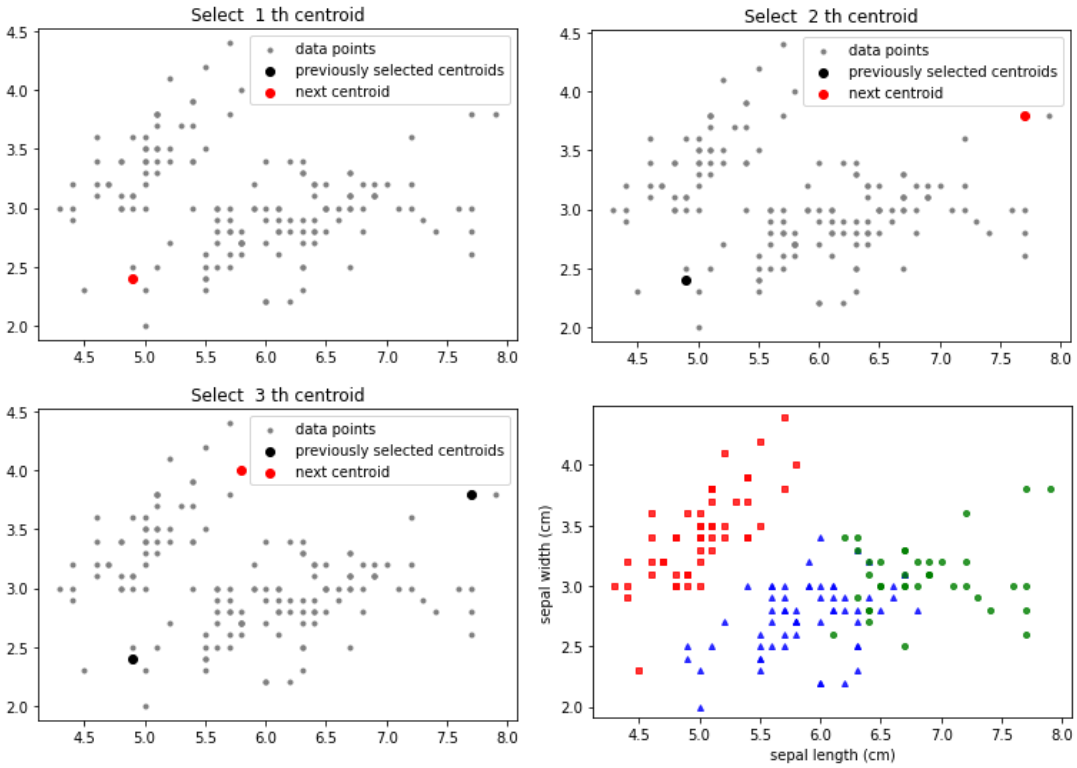
Hình 9: Bảng kết quả với phương pháp K-MEANS++

Lần chạy giải thuật thứ 3 với 5 vòng lặp để hội tụ, cùng sai số SSE là 97.20457.



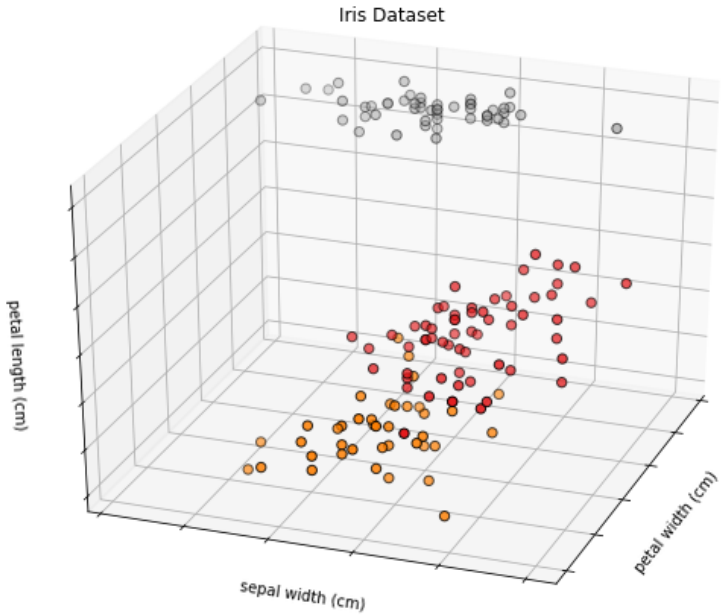
Hình 10: Lần chạy thứ 3 bằng phương pháp K-MEANS++ - các bước khởi tạo từng centroid

Ở lần chạy thứ 10, nhờ việc khởi tạo 3 centroid ở các vị trí thông minh mà chỉ cần 2 vòng lặp đã hội tụ, đồng thời đảm bảo giá trị sai số SSE thấp.



Hình 11: Lần chạy thứ 10 bằng phương pháp K-MEANS++ - các bước khởi tạo từng centroid

Nhờ việc khởi tạo centroid thông minh nên sai số SSE trung bình qua 10 lần chạy khá thấp và ổn định 97.21066, không có trường hợp 2 centroid nằm cùng 1 phân cụm thực tế. Đồng thời số vòng lặp trung bình để hội tụ khá thấp 5.5. Nhưng bù lại thời gian khởi tạo lớn vì cần nhiều chi phí tính toán.



Hình 12: Biểu diễn các điểm dữ liệu đã phân cụm trên biểu đồ 3D

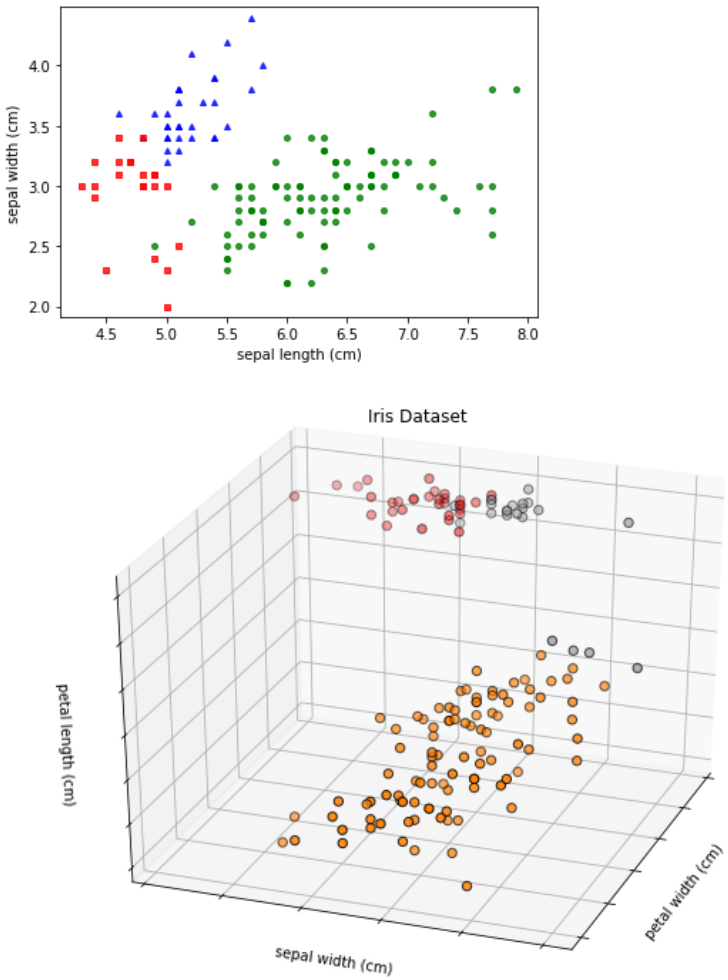
2.2.3 Phương pháp FORGY APPROACH (FA)

Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
FORGY APPROACH (FA)	Thời gian	0.00357	0.0001	0.00118	0.00034	0.0001	0.00259	0.00013	0.00005	0.00006	0.00008	**0.00082**
	Vòng lặp	3	3	10	5	11	5	4	6	10	12	**6.9**
	SSE	97.20457	97.20457	97.22487	97.20457	97.22487	124.0224	122.6001	97.20457	124.0224	97.22487	**105.113773**

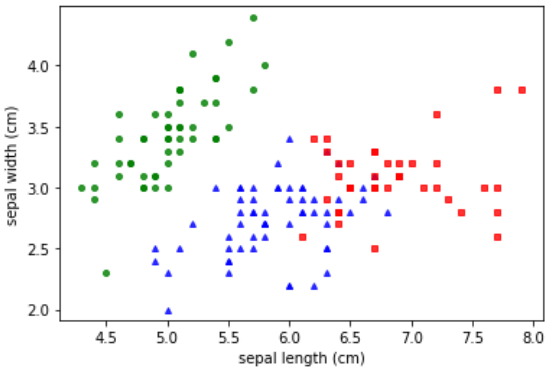
Hình 13: Bảng kết quả với phương pháp FORGY APPROACH

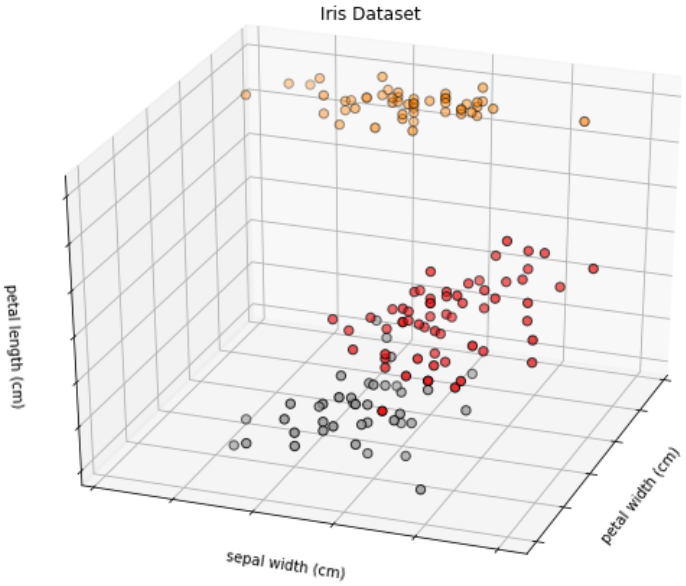
Ở lần chạy thứ 7, thời gian khởi tạo 3 trung tâm cụm rất nhanh 0.00013s và cần 4 vòng lặp để hội tụ nhưng sai số SSE khá lớn 122.6001.



Hình 14: Lần chạy thứ 7 bằng phương pháp FORGY APPROACH

Với lần chạy thứ 10, Thời gian khởi tạo các trung tâm cụm vẫn rất nhanh 0.00008s, với 12 vòng lặp để hội tụ và sai số SSE lúc này tốt hơn 97.22487 (xấp xỉ sai số SSE của tập dữ liệu có dán nhãn tương ứng 100.39574).





Hình 15: Lần chạy thứ 10 bằng phương pháp FORGY APPROACH

Dễ thấy rằng thời gian khởi tạo 3 trung tâm cụm của phương pháp FORGY APPROACH ổn định ở mức rất thấp do chỉ chọn ngẫu nhiên 3 trong số các điểm trong điểm dữ liệu, nhưng đồng thời cũng mang lại nhiều rủi ro khác như sai số SSE trung bình cao 105.113773, số vòng lặp để hội tụ cũng không ổn định lúc lớn lúc nhỏ (trung bình 6.9) và đây chỉ là tập dữ liệu gồm 150 điểm dữ liệu nên nếu ở bộ dữ liệu lớn thì độ sai biệt này càng rõ rệt hơn.
Đồng thời phương pháp này cũng mang nặng yếu tố ngẫu nhiên và cục bộ, chưa xét đến ảnh hưởng của toàn bộ tập dữ liệu khi khởi tạo nên hiệu quả của phương pháp này không ổn định hay có thể nói là không tốt.

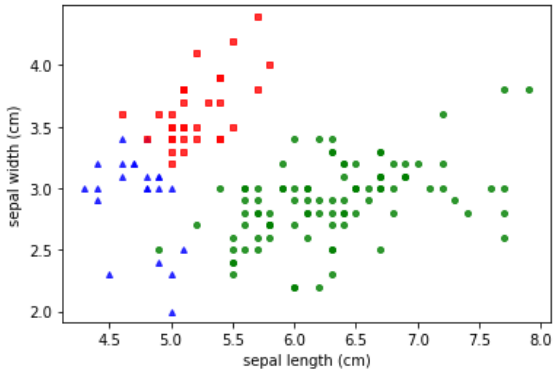
2.2.4 Phương pháp MACQUEEN APPROACH (MA)

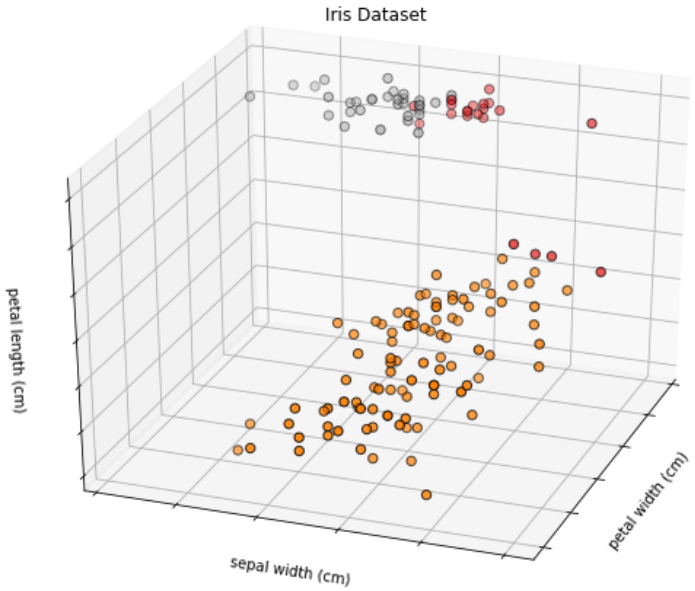
Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
MACQUEEN	Thời gian	0.00928	0.00567	0.00796	0.00918	0.00922	0.01149	0.0126	0.01705	0.01126	0.01147	**0.010518**
APPROACH	Vòng lặp	1	8	1	2	9	9	7	6	1	5	**4.9**
(MA)	SSE	97.20457	97.22487	97.20457	97.20457	124.02237	97.22487	124.02237	124.02237	97.20457	124.0224	**107.93575**

Hình 16: Bảng kết quả với phương pháp MACQUEEN APPROACH

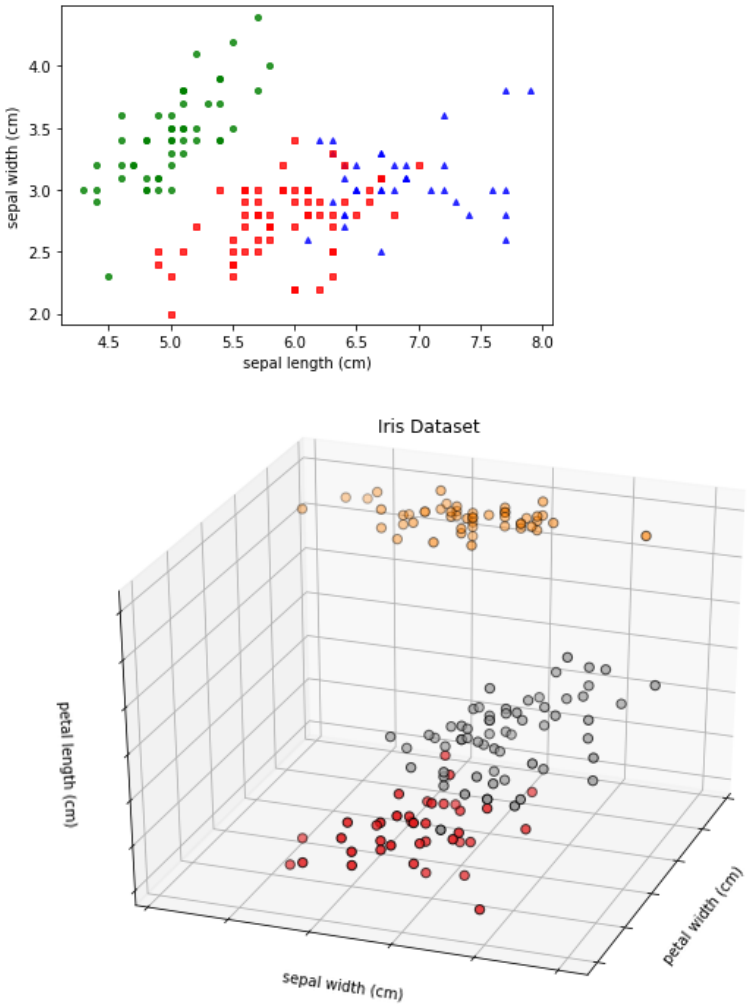
Kết quả ở lần chạy thứ 5 với sai số SSE là 124.02237, mất 0.00922s khởi tạo 3 cụm trung tâm và 9 lần lặp để hội tụ.





Hình 17: Lần chạy thứ 5 bằng phương pháp MACQUEEN APPROACH

Kết quả ở lần chạy thứ 9 với sai số SSE là 97.20457, mất 0.01126s khởi tạo 3 cụm trung tâm và chỉ cần 1 lần lặp để hội tụ.



Hình 18: Lần chạy thứ 9 bằng phương pháp MACQUEEN APPROACH

Với phương pháp MACQUEEN APPROACH số vòng lặp trung bình để hội tụ khá thấp, nhưng giá trị quan trong nhất là sai số SSE trung bình lại rất cao đồng thời yêu cầu tính toán trên toàn bộ tập dữ liệu làm giải thuật hoạt động khá chậm.

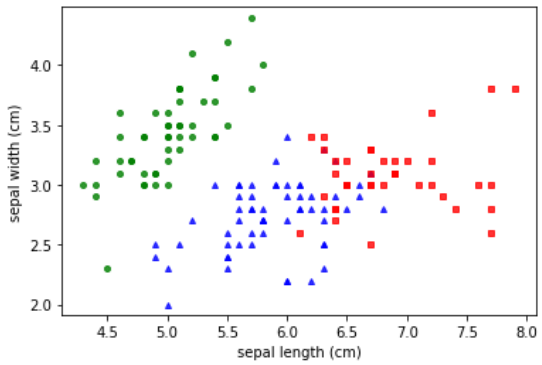
2.2.5 Phương pháp KAUFMAN APPROACH (KA)

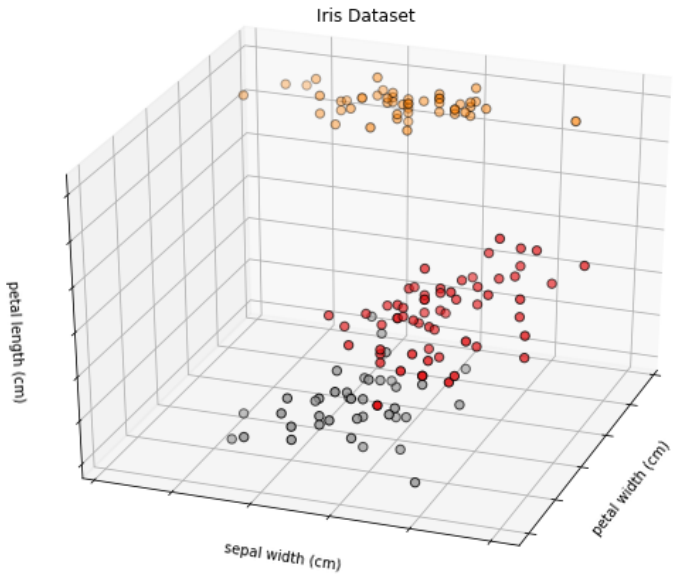
Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
KAUFMAN APPROACH (KA)	Thời gian	0.90235	0.92017	0.9026	0.98473	1.14646	0.95677	1.00863	1.04121	1.05406	0.9537	**0.987068**
	Vòng lặp	3	3	3	3	3	3	3	3	3	3	**3**
	SSE	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	**97.22487**

Hình 19: Bảng kết quả với phương pháp KAUFMAN APPROACH

Kết quả ở lần chạy th 9 với sai số SSE là 97.22487, mất 1.05406 khởi tạo 3 cụm trung tâm và chỉ cần 3 lần lặp để hội tụ.





Hình 20: Lần chạy thứ 5 bằng phương pháp KAUFMAN APPROACH

Dễ dàng nhận thấy sai số SSE và số vòng lặp hội tụ không đổi qua 10 lần chạy do không có yếu tố ngẫu nhiên trong giải thuật. Các trung tâm cụm tìm được qua n lần chạy vẫn không đổi, chỉ khác nhau thời gian chạy do ảnh hưởng của hệ điều hành trong việc sắp xếp tiến trình. Dù mang lại kết quả tốt về số vòng lặp hội tụ cũng như sai số SSE nhưng hạn chế lớn nhất của phương pháp KAUFMAN APPROACH chính là chi phí tính toán quá lớn (hàm mũ) sẽ không hoạt động tốt ở bộ dữ liệu lớn. Phần sau sẽ thực hiện trên bộ dữ liệu lớn The Mnist để thấy rõ hạn chế này.

2.3 Nhận xét

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
RANDOM K CLUSTERS	Thời gian	0.00085	0.00053	0.00029	0.00542	0.00078	0.00568	0.00039	0.0019	0.00256	0.00337	**0.002177**
	Vòng lặp	10	11	10	11	11	11	12	8	11	10	**10.5**
	SSE	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	124.02237	97.22487	97.22487	97.22487	**99.90462**
FORGY APPROACH (FA)	Thời gian	0.00357	0.0001	0.00118	0.00034	0.0001	0.00259	0.00013	0.00005	0.00006	0.00008	**0.00082**
	Vòng lặp	3	3	10	5	11	5	4	6	10	12	**6.9**
	SSE	97.20457	97.20457	97.22487	97.20457	97.22487	124.0224	122.6001	97.20457	124.0224	97.22487	**105.113773**
K-MEANS++	Thời gian	0.52934	0.63747	0.56003	0.66934	0.54443	0.53559	0.54364	0.53747	0.62381	0.58109	**0.576221**
	Vòng lặp	5	3	5	3	10	3	11	3	10	2	**5.5**
	SSE	97.20457	97.20457	97.20457	97.20457	97.20457	97.20457	97.22487	97.20457	97.22487	97.22487	**97.21066**
MACQUEEN APPROACH (MA)	Thời gian	0.00928	0.00567	0.00796	0.00918	0.00922	0.01149	0.0126	0.01705	0.01126	0.01147	**0.010518**
	Vòng lặp	1	8	1	2	9	9	7	6	1	5	**4.9**
	SSE	97.20457	97.22487	97.20457	97.20457	124.02237	97.22487	124.02237	124.02237	97.20457	124.0224	**107.93575**
KAUFMAN APPROACH (KA)	Thời gian	0.90235	0.92017	0.9026	0.98473	1.14646	0.95677	1.00863	1.04121	1.05406	0.9537	**0.987068**
	Vòng lặp	3	3	3	3	3	3	3	3	3	3	**3**
	SSE	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	97.22487	**97.22487**

Hình 21: Bảng thống kê kết quả của 5 kỹ thuật khởi tạo trung tâm cụm đầu tiên qua 10 lần chạy

Nhận xét về hiệu quả của 5 phương pháp khởi tạo trung tâm cụm ban đầu với bộ dữ liệu nhỏ Iris Flower:

- **Phương pháp Ngẫu nhiên:** dựa vào yếu tố ngẫu nhiên trên toàn bộ tập dữ liệu để chọn ra các phần tử thuộc k cụm ban đầu nhưng các trung tâm cụm này tạo ra có xu hướng nằm gần nhau nên quá trình hội tụ lâu hơn, đồng thời hiệu quả của giải thuật thông qua sai số SSE cũng không tối ưu so với các phương pháp khác.
- **Phương pháp K-MEANS++:** nhờ việc khởi tạo centroid thông minh (các trung tâm cụm cố gắng tách rời nhau, và ít bị ảnh hưởng bởi yếu tố ngẫu nhiên) nên sai số SSE thấp và ổn định, số vòng lặp để hội tụ cũng khá thấp, nhưng bù lại thời gian khởi tạo lớn vì cần nhiều chi phí tính toán (dù vậy độ phức tạp giải thuật chỉ là O(n) nên có thể áp dụng tốt cho các bộ dữ liệu vừa và nhỏ).

- **Phương pháp FORGY APPROACH:** mang nặng yếu tố ngẫu nhiên và cục bộ, chưa xét đến ảnh hưởng của toàn bộ tập dữ liệu khi khởi tạo nên chứa nhiều rủi ro như sai số SSE cao, số vòng lặp để hội tụ cũng không ổn định (với bộ dữ liệu lớn thì độ sai biệt này càng rõ rệt hơn), bù lại vì đơn giản nên thời gian khởi tạo rất nhanh.
  Hiệu quả của phương pháp này không ổn định hay có thể nói là không tốt.
- **Phương pháp MACQUEEN APPROACH:** bị ảnh hưởng bởi yếu tố ngẫu nhiên (khởi tạo ngẫu nhiên k trung tâm ban đầu), tuy có xét đến ảnh hưởng của toàn bộ tập dữ liệu xong vẫn có xu hướng hội tụ cục bộ. Kết quả là với chi phí tính toán khá lớn nhưng hiệu quả vẫn không cao (sai số SSE lớn).
- **Phương pháp KAUFMAN APPROACH:** không bị ảnh hưởng bởi các yếu tố ngẫu nhiên, số vòng lặp để hội tụ ít và sai số SSE thấp, bù lại thì chi phí tính toán quá lớn (hàm mũ theo số lượng dữ liệu) nên chỉ thích hợp với bộ dữ liệu nhỏ, số chiều ít).Thời gian khởi tạo trung bình của phương pháp này cao gấp 1204 lần phương pháp đơn giản nhất (FORGY APPROACH).

Vậy với bộ dữ liệu nhỏ như Iris Flower phương pháp khởi tạo K-MEANS++ đem lại hiểu quả tốt cả về mặt thời gian cũng như sai số SSE của toàn bộ tập dữ liệu khi đã phân cụm.

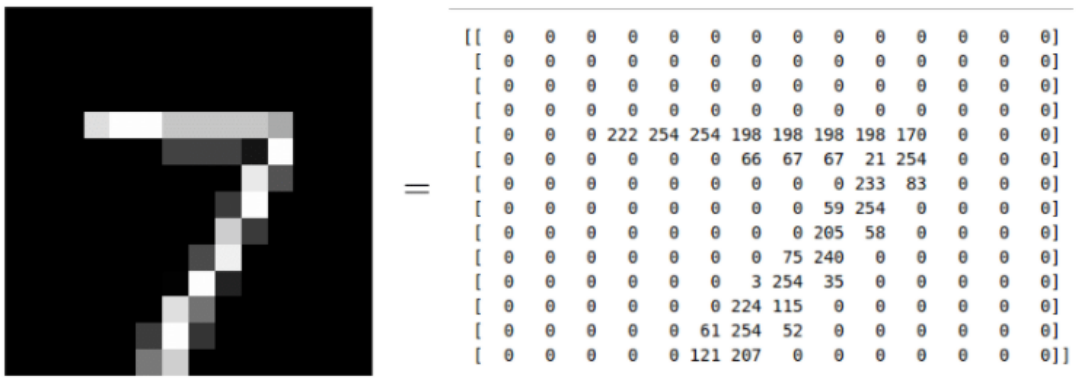
3 Ứng dụng giải thuật K-means với các kỹ thuật khởi tạo k trung tâm ban đầu trong việc phân cụm tập dữ liệu THE MNIST

3.1 Mô tả tập dữ liệu

Bộ cơ sở dữ liệu MNIST là bộ cơ sở dữ liệu lớn nhất về chữ số viết tay và được sử dụng trong hầu hết các thuật toán nhận dạng hình ảnh (Image Classification). MNIST bao gồm hai tập con: tập dữ liệu huấn luyện (training set) có tổng cộng 60k ví dụ khác nhau về chữ số viết tay từ 0 đến 9, tập dữ liệu kiểm tra (test set) có 10k ví dụ khác nhau. Tất cả đều đã được gán nhãn. Hình dưới đây là ví dụ về một số hình ảnh được trích ra từ MNIST.

Hình 22: MNIST: bộ cơ sở dữ liệu của chữ số viết tay

Mỗi bức ảnh là một ảnh đen trắng (có 1 channel), có kích thước 28x28 pixel (tổng cộng 784 pixels). Mỗi pixel mang một giá trị là một số tự nhiên từ 0 đến 255. Các pixel màu đen có giá trị bằng 0, các pixel càng trắng thì có giá trị càng cao (nhưng không quá 255). Dưới đây là một ví dụ về chữ số 7 và giá trị các pixel của nó.



Hình 23: Mô tả chữ số 7

Việc áp dụng các kỹ thuật khởi tạo k trung tâm cụm ban đầu của giải thuật K-means lên bộ dữ liệu lớn The Mnist này giúp thấy rõ các mặt hạn chế của các kỹ thuật mà Iris Flower chưa thấy được.

Bài toán phân nhóm giả định:

Bài toán: Giả sử rằng chúng ta không biết nhãn của các chữ số này, chúng ta muốn phân nhóm các bức ảnh gần giống nhau về một nhóm. Với thuật toán phân nhóm K-means clustering, chúng ta sẽ giải quyết bài toán này thế nào?

Trước khi áp dụng thuật toán K-means clustering, chúng ta cần coi mỗi bức ảnh là một điểm dữ liệu. Và vì mỗi điểm dữ liệu là 1 vector (hàng hoặc cột) chứ không phải ma trận như số 7 ở trên, chúng ta phải làm thêm một bước đơn giản trung gian gọi là vectorization (vector hóa). Nghĩa là, để có được 1 vector, ta có thể tách các hàng của ma trận pixel ra, sau đó đặt chúng cạnh nhau, và chúng ta được một vector hàng rất dài biểu diễn 1 bức ảnh chữ số. Cách làm này chỉ là cách đơn giản nhất để mô tả dữ liệu ảnh bằng 1 vector. Trên thực tế có rất nhiều kỹ thuật khác nhau để có thể tạo ra các vector đặc trưng (feature vector) giúp các thuật toán có được kết quả tốt hơn.
Sau khi vector hóa một ảnh 28x28 thành vector 784 chiều nên để đơn giản tác giả chỉ chọn 1000 ảnh của tập testing để thực hiện tiểu luận này (giảm quá trình tính toán khi hội tụ các trung tâm cụm).

Mô tả chung về dữ liệu The Mnist

```
#Code
mndata = MNIST('/content/gdrive/MyDrive/Colab Notebooks/MNIST')
mndata.load_testing()
y = np.asarray(mndata.test_labels)[:1000]
X = np.asarray(mndata.test_images)[:1000,:]/256.0
```

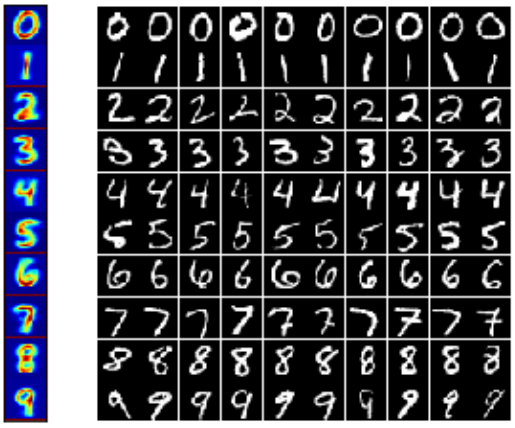
```
print(X.shape)
print(y.shape)
SSE(X, y)
centroids = []
for i in range(10):
  cluster = X[y == i]
  centroids.append(np.mean(cluster, axis = 0))
display_MNIST(X, np.array(centroids), y)

#Out
(1000, 784)
(1000,)
Error Sum of Squares:  6093.73051
```

Thể hiện các điểm dữ liệu có dán nhãn ban đầu

Để thể hiện tốt được hiệu quả phân cụm của các chữ số viết tay, ta vẽ 2 hình như sau:

- Hình bên trái: là hình vẽ giá trị trung tâm của các cụm ứng với từng hàng bên phải.
- Hình bên phải: 10 hình vẽ đầu tiên của từng cụm đã được phân loại.

Cách biểu diễn này cũng được áp dụng trong phần sau.



Hình 24: Biểu diễn tập dữ liệu The Mnist có nhãn ban đầu

Sai số SSE của tập dữ liệu ban đầu đo được là: 6093.73051. Giá trị này giúp so sánh hiệu quả của các phương pháp phân cụm được trình bày ở phần sau.

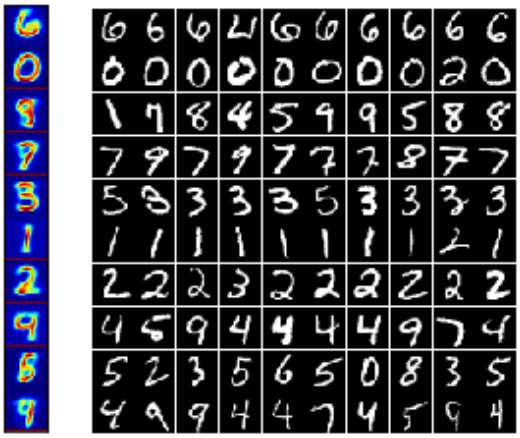
3.2 Phân loại dữ liệu với các kỹ thuật khởi tạo k trung tâm ban đầu

3.2.1 Phương pháp Ngẫu nhiên

Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
RANDOM K CLUSTERS	Thời gian	0.00855	0.00532	0.00556	0.00305	0.0051	0.00479	0.0051	0.00383	0.00299	0.00415	**0.004844**
	Vòng lặp	17	32	19	19	36	22	36	24	14	14	**23.3**
	SSE	5930.5494	5939.1930	5941.8433	5961.6496	5935.9937	5927.3018	5964.1246	5941.1095	5949.7992	5948.9486	**5944.0513**

Hình 25: Bảng kết quả với phương pháp Ngẫu nhiên



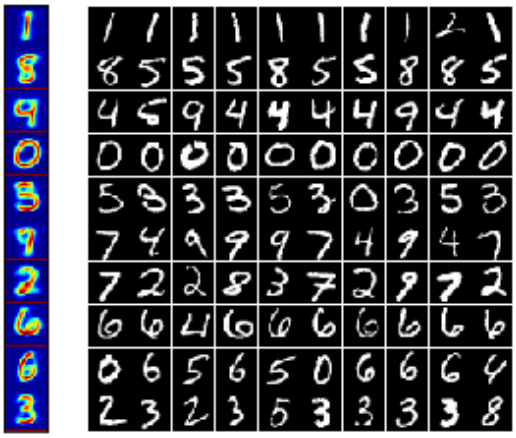
Hình 26: Biểu diễn tập dữ liệu được phân loại với phương pháp Ngẫu nhiên

3.2.2 Phương pháp K-MEANS++

Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
K-MEANS++	Thời gian	5.80787	5.8044	5.81957	5.80238	5.82219	5.77378	5.79221	5.79412	5.82603	5.8484	**5.809095**
	Vòng lặp	14	20	17	36	20	18	20	20	15	26	**20.6**
	SSE	5970.2789	5949.6351	5960.8560	5948.4690	5950.7241	5930.0117	5959.1173	5949.6351	5956.6539	5954.3616	**5952.9743**

Hình 27: Bảng kết quả với phương pháp K-MEANS++



Hình 28: Biểu diễn tập dữ liệu được phân loại với phương pháp K-MEANS++

3.2.3 Phương pháp FORGY APPROACH (FA)

Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
FORGY APPROACH (FA)	Thời gian	0.00007	0.00003	0.00003	0.00002	0.00024	0.00005	0.00013	0.00047	0.00003	0.00023	**0.00013**
	Vòng lặp	28	35	23	11	28	36	21	26	13	25	**24.6**
	SSE	5948.3647	5930.8015	5986.3016	5944.6580	5949.7843	5939.3301	5944.7821	5950.6811	5940.1967	5948.6813	**5948.3581**

Hình 29: Bảng kết quả với phương pháp FORGY APPROACH

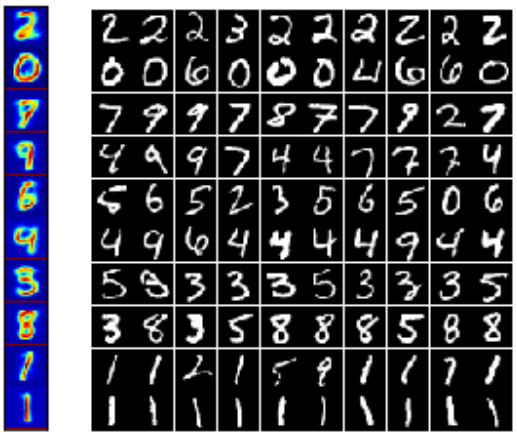
Hình 30: Biểu diễn tập dữ liệu được phân loại với phương pháp FORGY APPROACH

3.2.4 Phương pháp MACQUEEN APPROACH (MA)

Bảng kết quả sau 10 lần chạy thuật toán:

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
MACQUEEN	1.37322	1.35456	1.28899	1.29595	1.32568	1.30472	1.32406	1.3608	1.29773	1.33477	1.326048	**1.321331**
APPROACH	26	27	39	19	24	16	22	24	8	22	22.7	**22.37**
(MA)	5953.0583	6031.5797	5982.3435	5924.9864	6026.1666	5965.8175	5945.8815	5985.3529	5997.7372	5972.4951	5978.5419	**5981.09**

Hình 31: Bảng kết quả với phương pháp MACQUEEN APPROACH



Hình 32: Biểu diễn tập dữ liệu được phân loại với phương pháp MACQUEEN APPROACH

3.2.5 Phương pháp KAUFMAN APPROACH (KA)

Như đã phân tích về độ phức tạp hàm mũ của kỹ thuật KAUFMAN APPROACH khi thực hiện với bộ dữ liệu Iris flower thì với số lượng dữ liệu cũng như số chiều lớn của The Mnist, việc phân loại với số lượng dữ liệu là 1000 như với các phương pháp khác thì không thể trả về kết quả, vì vậy tác giả chỉ dùng 200 dữ liệu để phân loại.
Bảng kết quả như sau::

Phương pháp	Thông số	N	Trung bình
KAUFMAN	Thời gian	200	265.96389
APPROACH	Vòng lặp		5
(KA)	SSE		1139.26754

Hình 33: Bảng kết quả với phương pháp KAUFMAN APPROACH

Hình 34: Biểu diễn tập dữ liệu được phân loại với phương pháp KAUFMAN APPROACH

3.3 Nhận xét

Phương pháp	Thông số	Lần 1	Lần 2	Lần 3	Lần 4	Lần 5	Lần 6	Lần 7	Lần 8	Lần 9	Lần 10	Trung bình
RANDOM K CLUSTERS	Thời gian	0.00855	0.00532	0.00556	0.00305	0.0051	0.00479	0.0051	0.00383	0.00299	0.00415	**0.004844**
	Vòng lặp	17	32	19	19	36	22	36	24	14	14	**23.3**
	SSE	5930.5494	5939.1930	5941.8433	5961.6496	5935.9937	5927.3018	5964.1246	5941.1095	5949.7992	5948.9486	**5944.0513**
FORGY APPROACH (FA)	Thời gian	0.00007	0.00003	0.00003	0.00002	0.00024	0.00005	0.00013	0.00047	0.00003	0.00023	**0.00013**
	Vòng lặp	28	35	23	11	28	36	21	26	13	25	**24.6**
	SSE	5948.3647	5930.8015	5986.3016	5944.6580	5949.7843	5939.3301	5944.7821	5950.6811	5940.1967	5948.6813	**5948.3581**
K-MEANS++	Thời gian	5.80787	5.8044	5.81957	5.80238	5.82219	5.77378	5.79221	5.79412	5.82603	5.8484	**5.809095**
	Vòng lặp	14	20	17	36	20	18	20	20	15	26	**20.6**
	SSE	5970.2789	5949.6351	5960.8560	5948.4690	5950.7241	5930.0117	5959.1173	5949.6351	5956.6539	5954.3616	**5952.9743**
MACQUEEN APPROACH (MA)	Thời gian	1.37322	1.35456	1.28899	1.29595	1.32568	1.30472	1.32406	1.3608	1.29773	1.33477	**1.326048**
	Vòng lặp	26	27	39	19	24	16	22	24	8	22	**22.7**
	SSE	5953.0583	6031.5797	5982.3435	5924.9864	6026.1666	5965.8175	5945.8815	5985.3529	5997.7372	5972.4951	**5978.5419**
KAUFMAN APPROACH (KA)	Thời gian	N = 200	**265.9639**									
	Vòng lặp		5									
	SSE		**1139.268**									

Hình 35: Bảng thống kê kết quả của 5 kỹ thuật khởi tạo trung tâm cụm đầu tiên qua 10 lần chạy

Các thông số đánh giá trung bình qua 10 lần chạy của 4 kỹ thuật đều tương đồng nhau với sai số SSE chênh lệch với nhãn ban đầu rất ít (6093.73051). Với bộ dữ liệu lớn và nhiều chiều, phương pháp K-MEANS++ vẫn cho kết quả rất tốt mặc dù chênh lệnh với 3 phương pháp còn lại là không nhiều.
Vậy với bộ dữ liệu lớn và nhiều chiều như The Mnist thì ngoại trừ phương pháp KAUFMAN APPROACH đều đem lại hiểu quả phân cụm tốt.

4 Tham khảo

Tài liệu

[1] *K-means Clustering*, https://machinelearningcoban.com/2017/01/01/kmeans/

[2] *K-means Clustering: Simple Applications*, https://machinelearningcoban.com/2017/01/04/kmeans2/

[3] *An empirical comparison of four initialization methods for the K-means algorithm*

5 Phụ lục

```
#Import module
!pip install python-mnist
from google.colab import drive
drive.mount("/content/gdrive")
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
import math
import time
np.random.seed(11)
from mpl_toolkits.mplot3d import Axes3D
from mnist import MNIST
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import normalize

#K-means and other functions
# function for plot
def plot(data, centroids):
    plt.scatter(data[:, 0], data[:, 1], marker = '.',
                color = 'gray', label = 'data points')
    plt.scatter(centroids[:-1, 0], centroids[:-1, 1],
                color = 'black', label = 'previously selected
   centroids')
    plt.scatter(centroids[-1, 0], centroids[-1, 1],
                color = 'red', label = 'next centroid')
    plt.title('Select % d th centroid'%(centroids.shape[0]))
    plt.legend()
    plt.show()

def kmeans_display(X, label):
    K = np.amax(label) + 1
    X0 = X[label == 0, :]
    X1 = X[label == 1, :]
    X2 = X[label == 2, :]

    plt.plot(X0[:, 0], X0[:, 1], 'b^', markersize = 4, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'go', markersize = 4, alpha = .8)
    plt.plot(X2[:, 0], X2[:, 1], 'rs', markersize = 4, alpha = .8)
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
    plt.axis('equal')
    plt.plot()
    plt.show()

def plot_Iris_Dataset_2D_3D(X, labels):
    # plot 2D
```

```
    kmeans_display(X, labels)

    # plot 3D
    fig = plt.figure(1, figsize=(8, 6))
    ax = Axes3D(fig, elev=-150, azim=160)
    ax.scatter(X[:, 3], X[:, 1], X[:, 2], c=labels, cmap=plt.cm.Set1, edgecolor='k', s=40)
    ax.set_title("Iris Dataset")
    ax.set_xlabel(iris.feature_names[3])
    ax.w_xaxis.set_ticklabels([])
    ax.set_ylabel(iris.feature_names[1])
    ax.w_yaxis.set_ticklabels([])
    ax.set_zlabel(iris.feature_names[2])
    ax.w_zaxis.set_ticklabels([])
    plt.show()

# function support for k-means
def kmeans_assign_labels(init_function, X, centers):
    # calculate distances btw data and centers
    k = len(centers)
    while True:
        d = []
        for center in centers:
          d.append(list(map(lambda x: math.sqrt(sum(x)), (X - center)**2)))
        d = np.array(d).T
        # index of the closest center
        d = np.array(list(map(lambda x: np.argmin(x), d)))
        # sometime some centers not have any instances so init again with same init k-means
        if len(set(d)) == k:
            break
        centers = init_function(X, k)
    return d

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

def 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]))

```

```
92  def SSE(X, labels):
93      SSErr = 0
94      for i in range(len(set(labels))):
95          Xi = X[labels == i, :]
96          centeri = np.mean(Xi, axis = 0)
97          SSErr += sum(list(map(lambda x: math.sqrt(sum(x)), (Xi -
        centeri)**2)))
98      print('Error Sum of Squares: ', round(SSErr, 5))
99
100 def kmeans(init_function, X, K):
101     centers = init_function(X, K)
102     labels = []
103     it = 0
104     while True:
105         labels = kmeans_assign_labels(init_function, X, centers)
106         new_centers = kmeans_update_centers(X, labels, K)
107         if has_converged(centers, new_centers):
108             break
109         centers = new_centers
110         it += 1
111     return (centers, labels, it)
112
113 #IRIS FLOWER dataset
114 plot_Iris_Dataset_2D_3D(iris_X, iris_y)
115 SSE(iris_X, iris_y)
116
117 #Run with different init methods
118 #FORGY APPROACH (FA)
119 centers, labels, it = kmeans(kmeans_init_randomly_centers,iris_X,
        K=3)
120 print('iterator:', it)
121 plot_Iris_Dataset_2D_3D(iris_X, labels)
122 SSE(iris_X, labels)
123
124 #K-MEANS++
125 centers, labels, it = kmeans(kmeans_init_kmeansPlusPlus,iris_X, K
        =3)
126 print('iterator:', it)
127 plot_Iris_Dataset_2D_3D(iris_X, labels)
128 SSE(iris_X, labels)
129
130 #RANDOM K CLUSTERS
131 centers, labels, it = kmeans(kmeans_init_randomly_k_clusters,
        iris_X, K=3)
132 print('iterator:', it)
133 plot_Iris_Dataset_2D_3D(iris_X, labels)
134 SSE(iris_X, labels)
135
136 #MACQUEEN APPROACH (MA)
```

```
centers, labels, it = kmeans(kmeans_init_MA,iris_X, K=3)
print('iterator:', it)
plot_Iris_Dataset_2D_3D(iris_X, labels)
SSE(iris_X, labels)

#KAUFMAN APPROACH (KA)
centers, labels, it = kmeans(kmeans_init_KA,iris_X, K=3)
print('iterator:', it)
plot_Iris_Dataset_2D_3D(iris_X, labels)
SSE(iris_X, labels)

#THE MNIST dataset
#Load THE MNIST dataset and functions
def display_network(A, m = -1, n = -1):
    opt_normalize = True
    opt_graycolor = True

    # Rescale
    A = A - np.average(A)

    # Compute rows & cols
    (row, col) = A.shape
    sz = int(np.ceil(np.sqrt(row)))
    buf = 1
    if m < 0 or n < 0:
        n = np.ceil(np.sqrt(col))
        m = np.ceil(col / n)


    image = np.ones(shape=(buf + m * (sz + buf), buf + n * (sz +
   buf)))

    if not opt_graycolor:
        image *= 0.1

    k = 0

    for i in range(int(m)):
        for j in range(int(n)):
            if k >= col:
                continue

            clim = np.max(np.abs(A[:, k]))

            if opt_normalize:
                image[buf + i * (sz + buf):buf + i * (sz + buf) +
   sz, buf + j * (sz + buf):buf + j * (sz + buf) + sz] = \
                    A[:, k].reshape(sz, sz) / clim
            else:
```

```
                image[buf + i * (sz + buf):buf + i * (sz + buf) +
    sz, buf + j * (sz + buf):buf + j * (sz + buf) + sz] = \
                    A[:, k].reshape(sz, sz) / np.max(np.abs(A))
            k += 1

    return image

def display_MNIST(X, centers, labels):
    A = display_network(centers.T, 10, 1)
    f1 = plt.imshow(A, interpolation='nearest', cmap = "jet")
    f1.axes.get_xaxis().set_visible(False)
    f1.axes.get_yaxis().set_visible(False)
    plt.show()

    K = len(centers)
    N0 = 10;
    X1 = np.zeros((N0*K, 784))
    X2 = np.zeros((N0*K, 784))
    for k in range(K):
        Xk = X[labels == k, :]
        center_k = [centers[k]]
        neigh = NearestNeighbors(N0).fit(Xk)
        dist, nearest_id  = neigh.kneighbors(center_k, N0)
        X1[N0*k: N0*k + N0,:] = Xk[nearest_id, :]
        X2[N0*k: N0*k + N0,:] = Xk[:N0, :]
    plt.axis('off')
    A = display_network(X2.T, K, N0)
    f2 = plt.imshow(A, interpolation='nearest' )
    plt.gray()
    plt.show()

mndata = MNIST('/content/gdrive/MyDrive/Colab Notebooks/MNIST')
mndata.load_testing()
X = np.asarray(mndata.test_images)[:1000,:]/256.0
print(X.shape)

#Run with different init methods
#FORGY APPROACH (FA)
centers, labels, it = kmeans(kmeans_init_randomly_centers,X, K=10)
print('iterator:', it)
SSE(X, labels)
display_MNIST(X, centers, labels)

#K-MEANS++
centers, labels, it = kmeans(kmeans_init_kmeansPlusPlus,X, K=10)
print('iterator:', it)
SSE(X, labels)
display_MNIST(X, centers, labels)

```

```
232 #RANDOM K CLUSTERS
233 centers, labels, it = kmeans(kmeans_init_randomly_k_clusters,X, K
        =10)
234 print('iterator:', it)
235 SSE(X, labels)
236 display_MNIST(X, centers, labels)
237
238 #MACQUEEN APPROACH (MA)
239 centers, labels, it = kmeans(kmeans_init_MA,X, K=10)
240 print('iterator:', it)
241 SSE(X, labels)
242 display_MNIST(X, centers, labels)
243
244 #KAUFMAN APPROACH (KA)
245 centers, labels, it = kmeans(kmeans_init_KA,X[:200,:], K=10)
246 print('iterator:', it)
247 SSE(X[:200,:], labels)
248 display_MNIST(X[:200,:], centers, labels)
```